Query Integrity Meets Blockchain: A Privacy-Preserving Verification Framework for Outsourced Encrypted Data

Shunrong Jiang, Jianqing Liu, Jingwei Chen, Yiliang Liu, Liangmin Wang, *Member, IEEE*, and Yong Zhou

Abstract—Cloud outsourcing provides flexible storage and computation services for data users in a low cost, but it brings many security threats as the cloud server may not be fully trusted. Previous secure outsourcing solutions mostly assume that the server is honest-but-curious while the adversary model of a malicious server that may return incorrect results is rarely explored. Moreover, with the increasing popularity of verifiable computations, existing verification schemes are yet not efficient and cannot cater to different scenarios in practice. In this paper, we propose a blockchain-based verifiable search framework for the adversarial cloud outsourcing context. When outsourcing the encrypted data to the cloud or Interplanetary File System (IPFS), we also store the encrypted data index in a decentralized blockchain (i.e., Ethereum in this paper) which is public and cannot be modified. Once a user is authorized, he/she can flexibly obtain the query results and efficiently check the query integrity via the pre-deployed smart contract, without the need of the data owner being online. Moreover, for user's privacy protection, we construct a stealth authorization scheme to deliver the access authorization without any identity disclosure. Finally, theoretical analysis and performance evaluation validate the security and efficiency of our proposed framework.

Index Terms—Outsourced encryption, blockchain, query integrity, access authorization, privacy protection

1 Introduction

Nowadays, the development of cloud computing has revolutionized the traditional computation and storage paradigms. With great flexibility and low cost, data owners can outsource their personal data to the cloud and rely on it to provide diverse services (e.g., storage and query). Despite the great benefits, security and privacy concerns still exist and may discourage the wide participation of users. For data confidentiality, data encryption is a straightforward and efficient way, in which cloud can apply the searchable symmetric encryption (SSE) schemes [2], [3] to perform efficient search functions on encrypted data according to the encrypted query from data users. Essentially, SSE ensures the query availability on the ciphertext domain, and both the data privacy of data owners and the query privacy of data users are not compromised.

To give an example, a pharmaceutical company may encrypt a set of clinical trial records and outsource them to the cloud server which could be in another trusted domain (e.g., owned by an IT company). Whenever needed, its employee or external parties can pose queries with certain keywords [4] to get these records for research, diagnosis and treatment. However, as data outsourcing causes data owners to lose control of their own private data, it leaves to the mercy of the cloud server to return legit SSE query results. Apparently, the query results may be incorrect (incomplete or falsified) as the cloud server may be malicious (intentionally or compromised by attackers) and cannot be fully trusted. For reliable query services, it is necessary to enable the data users to verify the correctness of query results returned by the cloud.

For query integrity verification, plenty of researches have been proposed for different types of data, including structured attribute-valued data [5], [6] and streaming data [7], [8]. However, only a few works consider the query integrity verification for encrypted data [9], [10]. Among these limited works, the index and encrypted data are both outsourced to the untrusted third party, making the query results uncontrollable. Moreover, large computation overhead is incurred and hence the centralized cloud platforms are barely practical. To address these issues, intuitively, we can store the index in a public decentralized system, in which the query correctness can be guaranteed via the trust maintained by the distributed multiple entities. However, performing secure queries and verification in such a decentralized system remains under-explored and needs further investigation.

- *Shunrong Jiang, Jingwei Chen, and Yong Zhou are with the School of Computer Science & Technology, China University of Mining and Technology, Xuzhou 221116, China. E-mail: {jsywow, cjwcier}@gmail.com, yzhou@cumt.edu.cn.*
- *Jianqing Liu is with the Department of Computer Science, North Carolina State University, Raleigh, NC 27695 USA. E-mail: jliu96@ncsu.edu.*
- *Yiliang Liu is with the School of Cyber Science and Engineering, Xi'an Jiaotong University, Xi'an 710049, China. E-mail: alanliuyiliang@gmail.com.*
- *Liangmin Wang is with the School of Cyber Science and Engineering, Southeast University, Nanjing 211100, China. E-mail: wanglm@ujs.edu.cn.*

Manuscript received 13 April 2022; accepted 3 August 2022. Date of publication 16 August 2022; date of current version 12 June 2023.
This work was supported in part by National Key R&D Program of China under Grant 2020YFB1005500, in part by the Fundamental Research Funds for the Central Universities of Ministry of Education of China under Grant 2020ZDPY0306, in part by the Xuzhou Science and Technology Program under Grant kc21044, and in part by the National Natural Science Foundation of China under Grant 62101429. The work of Jianqing Liu was supported in part by National Science Foundation under Grant CNS-2211214.
(Corresponding author: Yong Zhou.)
Digital Object Identifier no. 10.1109/TSC.2022.3199111

Recently, with the prosperity of crypto currencies [11], [12], [13], [14], blockchain, as its underlying technology, is featured by decentralization and perfectly matches the requirement of our design goal. By leveraging blockchain, without trusting the cloud server, the query can be decoupled from cloud servers with ensured integrity. Besides, potential data unavailability in traditional cloud outsourcing can be prevented as well. As any data recorded on the blockchain cannot be tampered with and the smart contract can ensure the correctness of predefined functionalities, we incline to record the encrypted index on the blockchain which can then be efficiently retrieved with correctness guarantee by the data users. Despite its promise, realizing the blockchain-based verifiable query with privacy preservation and access authorization is still challenging for its characteristics that each transaction and data are publicly visible. Even there are several privacy-enhanced technologies in blockchain, e.g., zk-SNARKs [15] and CryptoNote [16], they are not suitable in our application scenario because of high overhead and application limitation. Besides, although there are some blockchain-based access control schemes [17], [18], [19], few of them consider the access authorization problem [20] in the blockchain. This implies that these schemes [17], [18], [19] need other secure channels to deliver the authorized content which is neither convenient nor flexible. Therefore, in this paper, we aim to provide an Ethereum-based [21] verifiable query scheme with privacy-preservation and flexible access authorization delivery, in which the data owners can be offline. Specifically, we have the following three main contributions.

- We propose a blockchain-based verifiable search framework[1], which shifts the search index to Ethereum and returns the trusted hash value of matching results. Thus, query integrity and data privacy of both data owners and data users can be guaranteed.
- We construct a stealth authorization scheme to support privacy-aware access authorization delivery in Ethereum. Specifically, we hide the receiver's address of each transaction while ensuring privacy about the access authorization by cryptographic primitives.
- We implement our proposed framework on a locally simulated network (Ganache) of Ethereum and an Ethereum-based Layer 2 extension test network Mumbai of Polygon [22]. The results validate the practicability of our proposed blockchain-based framework.

The rest parts are presented as follows: The system model and preliminaries are illustrated in Section 2. Section 3 describes our framework. Sections 4 and 5 demonstrate the security analysis and the performance evaluation, respectively. Section 6 summarizes the related works. Finally, we conclude our paper in Section 7.

2 System Model and Preliminaries

2.1 System Model and Design Goals

As shown in Fig. 1, the system contains four entities: a data owner DO, who outsources a large-scale collection of $\widetilde{d}$ documents to the remote cloud server S or IPFS [23], and stores the corresponding indices in Ethereum; the cloud server S is the entity who provides storage and query services; a set of data users DUs, who can access the documents stored in the cloud with the help of Ethereum; and a public blockchain-Ethereum which stores indices of DO and offers automatic and trustworthy search service by the smart contract (*searchVerify contract*). Notice that DO/DU can be an organization or individual, and they join Ethereum as the blockchain users. In our framework, DO does not need to know DU in advance. When DU sends the stealth transmission to DO, she/he only needs to bind a certificate (such as blind signature or the anonymous authentication technology in zebralancer [24]) to prove she/he is a valid user. Only when DU passes the validity verification, DO agrees on the request and responds the encrypted keywords to DU according to the access profile. For secure and efficient information retrieval, the blockchain-based decentralized data storage and sharing are described as follows:

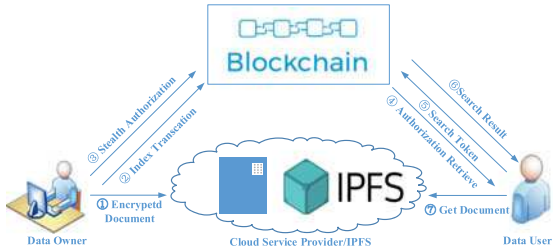


Fig. 1. The system model.

- Without loss of generality, we assume DO has a database DB=(W_i, ind_i), which is a set of keyword and identifier pairs. Specifically, W_i is a keyword set and ind_i is the set of document identifiers that contain the keyword[2]. After DO extracts the keywords of each document and builds a keyword index, he/she encrypts documents as well as the keyword index. As we can see in step ① from Fig. 1, the encrypted files are sent to the cloud while the keyword index is recorded in Ethereum (step ②). Besides, DO deploys **searchVerify contract** to execute different operations, which can be executed automatically and efficiently.
- After performing step ① and step ②, a stealth authorization scheme is executed between DO and DU to achieve the anonymous access authorization as described in step ③. Upon recovering the authorization information in step ④, DUs take the search token as the data part of the transaction and send it to **searchVerify contract** by the address $\mathcal{B}_j$ and get the corresponding index results through the smart contract as illustrated in step ⑤ and ⑥, respectively.
- After obtaining the index results, DUs send the index to the cloud (step ⑦) to obtain the corresponding encrypted documents from the cloud server. Besides, DUs can verify the returned documents by these index results. Finally, DUs can get the decrypted documents with authorization information.

1. According to different access orders about blockchain during the search process, the verifiable search frameworks can be categorized: *search-then-verify* and *verify-then-search*. The main difference is that the search index should be stored in the outsourced cloud for *search-then-verify*.

2. Note that ind_i is the hash value of the file in some file storage systems in IPFS [23].

2.2 Attack Model

In this paper, we assume that DO is fully trusted, while authorized and unauthorized DUs are semi-trusted[3]. In other words, DUs will strictly follow the designated protocol, but may try to infer some sensitive information from the search results or obtain documents without being authorized. Ethereum is trusted, while the cloud server is not as it may manipulate the outsourced encrypted data. Besides, we consider potential entity collusions such as collusion among DUs [7]. Thus, our framework should support public verification. Moreover, all Ethereum users are honest to maintain the recorded data and verify each transaction, but are curious about the keyword indexes of DOs, the searched keywords/results of DUs, and the authorization relationship.

2.3 Design Goals

Considering these security threats, our design is required to achieve the design goals as follows:

- *Data privacy*. For DO, the data confidentiality of documents and keyword indices should be well protected. Besides, since each transaction information transmitted and stored on the blockchain is public in Ethereum, we should protect the privacy of the authorization information and the encrypted indices, and conceal any side-channel yet private information about the documents.
- *Anonymous access authorization*. To ensure privacy-aware access authorization delivery, we should achieve *unlinkability* in Ethereum, i.e., for any two outgoing transactions, it is unknown whether they are sent to the same receiver. Moreover, only the authorized user can recover the authorization information.
- *Query integrity*. We should guarantee the query integrity: *Soundness* means all the query results are originated from the DO and satisfy the query criteria; *Completeness* means no valid answer in the query results is missing.

2.4 Preliminaries

2.4.1 Blockchain

As the first application of blockchain, Bitcoin [11] was proposed as a decentralized cryptocurrency and has received dramatic attention in the past few years. Blockchain is a distributed database that records all transactions conducted in a peer-to-peer network. Each participant in the network holds a copy of the database. Compared with centralized storage, there is no central authority, and no single entity that can control the entire network. Actually, blockchain is a series of linked data blocks which are confirmed by a consensus mechanism among most nodes in the network. For each block, it contains a series of transactions and a block header (composed of a pointer linked to the block header of the previous block, the merkle hash tree root, and a timestamp). With the block header, each block is linked together in chronological order. The cryptographic hash function ensures that the transaction data in each block cannot be tampered with.

3. Semi-trusted means that a user follows protocols but will try to learn as much information as possible, without actively "cheating".

TABLE 1
Notations Used in the Scheme

Notations	Descriptions
μ_i/μ_j	DO/DU
(a_i, b_i)	The private secp256k1 key pair of μ_i
(A_i, B_i)	The public secp256k1 key pair of μ_i
$(\mathcal{A}_i, \mathcal{B}_i)$	The Ethereum addresses pair of of μ_i
$\mathcal{H}_1(\cdot), \mathcal{H}_2(\cdot)$	$\mathcal{H}_1 : E \rightarrow \mathbb{Z}_p^*, \mathcal{H}_2 : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$
$\mathsf{Enc}(\cdot)/\mathsf{Dec}(\cdot)$	The AES-128 encryption/decryption
$\mathcal{ENC}_K(\cdot)/\mathcal{DEC}_K(\cdot)$	The ECC encryption/decryption with K

2.4.2 Ethereum and Gas

As a novel decentralized application platform, Ethereum supports smart contract [21]. The main advantage of Ethereum lies in its feature of programmability, and in support of Turing complete applications, in which users are able to build, deploy, and run smart contracts on Ethereum. Upon deployment, the contracts will be identified by a special address and automatically implemented with the agreed logic. Similarly, further modification on the contract's code is not feasible even for its creator.

In the Ethereum platform, $ether$ (ETH) is a token as its cryptocurrency. The base unit of $ether$ is called wei and $1Gwei = 10^9 wei = 10^{-9} ether$ [21]. For the Ethereum blockchain, the smart contract runs based on the Ethereum Virtual Machine (EVM), in which transactions are received, broadcast, verified, and executed by miners. Once the code in the contract is triggered by a message or a transaction, EVM will run on each miner side as part of the block validation protocol. The miners will authenticate every transaction contained in the block and execute the codes in smart contract, having the same calculations and storing the same values. Along with each operation in the EVM, there is a specific consumption which is counted by the unit gas [21]. To perform the required data storage or computations, the sender of the transaction needs to pay some gas in the form of $ether$.

3 The Proposed Search Framework

In this section, we construct our verifiable search framework. The notations used throughout this paper are listed in Table 1.

3.1 Overview

In our framework, after outsourcing the encrypted documents to the cloud, DO sends the corresponding encrypted index to Ethereum and deploys the *searchVerify contract* to execute the search, addition, and deletion operations. After that, all the operations can be carried out for data update automatically, efficiently and trustfully. To achieve access control and ensure the privacy of DU, we construct a stealth authorization scheme based on the concept of stealth address in Monero [16]. However, when the authorized user uses the same address to access the *searchVerify contract*, the relationship between the stealth authorized parties is revealed. To deal with this drawback, we adopt another address of the authorized user to access the *searchVerify contract*. Specifically, our framework consists of the following steps:

System initialization: E is a ECC-secp256k1 elliptic curve [25] over a finite field GF_p of prime order p with a base point G. Let $\mathcal{H}_1$ be a hash function where $\mathcal{H}_1 : E \rightarrow \mathbb{Z}_p^*$. F_p and F_τ are pseudo-random functions (PRFs) output strings in $\mathbb{Z}_p^*$ and $\{0,1\}^\tau$, respectively. A user μ_i (DO/DU) executes the following operations.

- μ_i chooses a private ECC-secp256k1 key pair (a_i, b_i), where $a_i, b_i \in \mathbb{Z}_p^*$;
- μ_i calculates the corresponding public ECC-secp256k1 key pair (A_i, B_i), where $A = a_iG$ and $B = b_iG$;
- μ_i gets the corresponding addresses of Ethereum as $(\mathcal{A}_i, \mathcal{B}_i)$ according to (A_i, B_i) [21].

EDBSetup(DB)

(1) μ_i constructs the index (w_i, ind_i) according to DB;
(2) μ_i randomly chooses a key K_E to encrypt DB as EDB by AES-128 and outsources it to the cloud;
(3) Before sending the index to Ethereum, μ_i chooses three keys K_e, K_a, K_d, and an empty list L;
(4) For each keyword $w_i \in \mathsf{W}$:

- $K_{1i} = F_p(K_e, 1||w_i), K_{2i} = F_p(K_e, 2||w_i)$, where "||" is denoted as the concatenation operation.
- For each ind_i containing w_i:
  - μ_i randomly chooses $r \in \{0,1\}^\tau$, calculates $d = \mathsf{ind}_i \oplus F_\tau(K_{2i}, r)$, $l = F_p(K_{1i}, c)$, $c++$,where c is a counter increased form 0 to d_{max}.
  - μ_i adds (l, d, r) to L.

(5) μ_i takes L as the data part of the transaction and sends it to the address of **searchVerify contract** by the address $\mathcal{A}_i$ and records the corresponding transaction id $txid_L$;
(6) μ_i stores $(K_E, K_e, K_a, K_d, w_i, txid_L)$;
(7) μ_i takes $\mathcal{A}_i$, the description of sharing data and requirements as the data part of the transaction and sends it to the address of **searchVerify contract** by the address $\mathcal{A}_i$;

Fig. 2. The initialization process of our framework.

- $(\mathsf{EDB}, K, txid_L) \leftarrow \mathsf{EDBSetup}(\mathsf{DB})$: The algorithm is run by DO. He/she takes a database DB as input and outputs three tuples $(\mathsf{EDB}, K, txid_L)$ where EDB is the encrypted database, K is the corresponding secret key, and $txid_L$ is the transaction id of the encrypted index in Ethereum.
- $L_R \leftarrow \mathsf{Search}(txid_L, \{K_{1i}, K_{2i}, K_{1i}^A, K_{2i}^A\})$: The algorithm is run by DO or DU. The algorithm can be divided in two phases: DO/DU takes $\{K_{1i}, K_{2i}, K_{1i}^A, K_{2i}^A\}$ to generate the search token l, l_A. Then, he/she sets l, l_A with the transaction id $txid_L$ as the input to the *searchVerify contract*. The *searchVerify contract* outputs a set of identifiers L_R. Finally, he/she can get the corresponding result according to L_R from the cloud server. *Before* DU*runs this algorithm, a stealth authorization scheme should be performed between* DO *and* DU *through Ethereum.*
- $L_A \leftarrow \mathsf{Add}(\mathsf{ind}_j, \mathsf{W}_j)$: The algorithm can only be executed by DO. He/she takes the newly added ind_j with the corresponding keyword set W_j as input, and the *searchVerify contract* updates L_A. Besides, DO also updates the corresponding documents stored in the cloud server.
- $L_D \leftarrow \mathsf{Delete}(\mathsf{ind}_i)$: The algorithm can only be executed by DO. He/she takes ind_i as input, the *searchVerify contract* outputs L_D. Moreover, DO also deletes the corresponding documents from the cloud server.

3.2 The Proposed Scheme

3.2.1 Initialization

This phase mainly contains two parts: system initialization and outsourcing content generation. As shown in Fig. 2, we first set the system parameters. Since the private/public key and address of Ethereum are generated by ECC-secp256k1 algorithm [26], for each user μ_i, he/she chooses the private secp256k1 key pair (a_i, b_i), calculates the public secp256k1-key pair (A_i, B_i), and gets the corresponding addresses of Ethereum as $(\mathcal{A}_i, \mathcal{B}_i)$. For the ease of presentation, we denote μ_i as the data owner DO and μ_j as the data user DU.

Before outsourcing documents, the data owner μ_i calculates the encrypted documents and the corresponding index by EDBSetup(DB). Then, μ_i outsources the encrypted documents EDB to the cloud while the encrypted index is stored in Ethereum. Here, we assume that the maximum number of documents containing w_i is d_{max}. Thus, the maximum value of c is d_{max}. During the search process, DU only executes d_{max} times to compute l which can reduce the computation overhead.

3.2.2 Stealth Authorization

After the initialization phase, μ_i should deploy the search smart contract in Ethereum. Then, μ_i can deliver the access authorization to other users to realize data sharing through Ethereum. To achieve privacy-preserving access authorization delivery and prevent inferring the relationship between the authorized parties during the search operation, we design a privacy-preserving and stealth authorization scheme as follows:

Stealth Transmission. As shown in Fig. 3, in order to apply for sharing the data of μ_i, μ_j first sends the secure information to μ_i according to A_i:

- μ_j chooses a random number $r_1 \in \mathbb{Z}_p^*$, calculates the transaction public key $R_1 = r_1G$, and the stealth tag $ST_1 = \mathcal{H}_2(T_1||\mathcal{H}_1(r_1A_i))G$, where T_1 is the current timestamp;
- μ_j encrypts (A_j, B_j) by A_i as $c_1 = \mathcal{ENC}_{A_i}(A_j, B_j)$ by ECC-secp256k1;
- μ_j adds $T_1||R_1||ST_1||c_1$ as the transaction and broadcasts it in Ethereum address $\mathcal{A}_j$.

To recover the secure information of μ_j, μ_i checks the recent transactions in the newly generated blocks, and extracts the stealth transmission information as shown in Fig. 4:

- μ_i calculates $ST_1' = \mathcal{H}_2(T_1||\mathcal{H}_1(a_iR_1))G$, and checks whether $ST_1' = ST_1$ or not. If the equation holds, it means μ_i is the receiver, then;

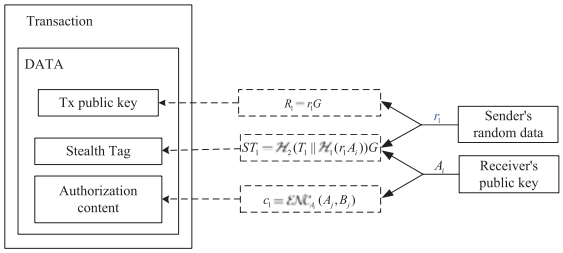


Fig. 3. The stealth transmission generation.

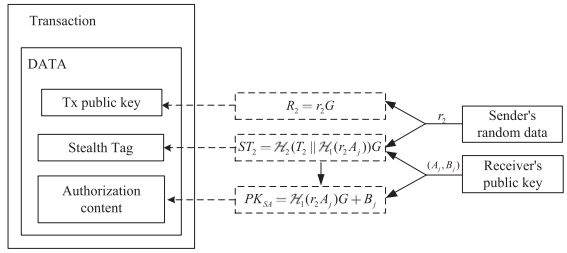


Fig. 5. The stealth authorization generation.

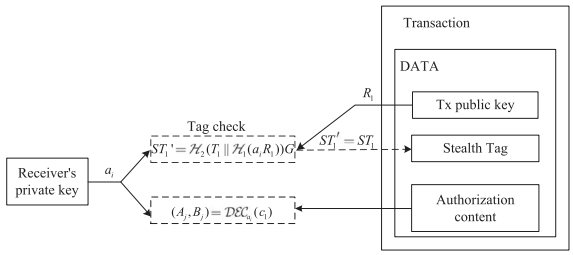


Fig. 4. The stealth transmission regain.

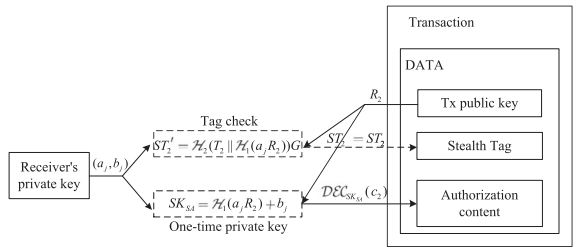


Fig. 6. The stealth authorization regain.

- μ_i calculates $\mathcal{DEC}_{a_i}(c_1)$ by a_i to obtain the secure information (A_j, B_j) of μ_j.

Stealth Authorization. After checking the validity of μ_j, μ_i will execute the stealth authorization to μ_j. The transaction of the stealth authorization is generated as shown in Fig. 5:

- μ_i gets the public key pair (A_j, B_j) of μ_j and randomly selects data $r_2, r_i \in \mathbb{Z}_p^*$;
- μ_i calculates the stealth tag as $ST_2 = \mathcal{H}_2(T_2||\mathcal{H}_1(r_2 A_j))G$, the transaction public key $R_2 = r_2 G$, and the public key $PK_{SA} = \mathcal{H}_1(r_2 A_j)G + B_j$ for encryption,where T_2 is the current timestamp.
- μ_i calculates $\mathbf{AccTk} = r_i||T_{ei}$ as the access control token of μ_j and $\mathbf{AccPr} = \mathcal{H}_2(B_j||\mathbf{AccTk})$ as the access control information, where T_{ei} is the expiration time.
- μ_i encrypts the authorization content as $c_2 = \mathcal{ENC}_{PK_{SA}}(txid_L, \{K_{1i}, K_{2i}, K_{1i}^A, K_{2i}^A\}, \mathbf{AccTk})$.
- μ_i adds $T_2||R_2||ST_2||c_2$ as the transaction and broadcasts it in Ethereum by address $\mathcal{A}_i$ and updates **AccPr** as the access control authorized information of μ_j to *searchVerify contract*.

The receiver μ_j checks the recent transactions in the newly generated blocks, and extracts the stealth authorization information as shown in Fig. 6:

- μ_j computes $ST_2' = \mathcal{H}_2(T_2||\mathcal{H}_1(a_j R_2))G$, and checks whether $ST_2' = ST_2$ or not. If the equation holds, it means μ_j is authorized to the receiver, then
- μ_j recovers the corresponding decryption key $SK_{SA} = \mathcal{H}_1(a_j R_2) + b_j$;
- μ_j decrypts $\mathcal{ENC}_{SK_{SA}}(c_2)$ to get the authorized information $(txid_L, \{K_{1i}, K_{2i}, K_{1i}^A, K_{2i}^A\}, \mathbf{AccTk})$.

Finally, μ_i adds **AccPr** into the authorizedInfo list of *searchVerify contract* in batch.

3.2.3 Search

After recovering the authorized information, μ_j can execute the verifiable search operation. He/she first runs the algorithm to generate the search token l, l_A. Then, μ_j sends l, l_A, $AccTk$ and $txid_L$ to the *searchVerify contract* in Ethereum by the address $\mathcal{B}_j$ and gets the corresponding index result. The search process is designed to *verifySearch* function locally called to get the search result. As shown in Fig. 7, the detailed search operation is performed by the *searchVerify contract*. Here, w_{max} is the max number of keywords in ind_i. After getting the returned result (d, r), (d_A, r_A), and L_D from *searchVerify contract*, μ_j could recover ind_i by further processing as shown in Fig. 7. Finally, he/she can get the corresponding encrypted documents from the cloud server and recover the plaintext. Obviously, ind_i can also be regarded as the proof for the search integrity verification.

3.2.4 Dynamic Update

Our framework also supports dynamic update operation as shown in Figs. 8 and 9, respectively. To add the new documents, DO first executes the operation as $\mathsf{EDBSetup}(\mathsf{DB})$ to get the corresponding (l_A, d_A, r_A). Then, we adopt the *searchVerify contract* to achieve the added operation. Here, we just build another list L_A to store new content. For the deletion operations, we build a deletion list L_D to store the deleted documents by the *searchVerify contract*. Both operations can only be run by the creator of the smart contract.

3.3 Smart Contract Design

In this section, we present the detailed operation of our smart contract. We adopt the solidity language [27] to program the smart contract running on Ethereum. There are several special variables and functions in the global namespace as follows:

- $msg.sender$: the sender of a message or transaction. When the smart contract is deployed, it is the address of the contract creator. When the smart contract is called, it is the address of the smart contract caller.
- $msg.value$: the number of wei sent with a message. For a subsequent user, we use $msg.value$ to represent

Search($txid_L, \{K_{1i}, K_{2i}, K_{1i}^A, K_{2i}^A\}$, $\mathbf{AccTk}$)

After μ_j recovers the authorization information, μ_j creates an empty list L_R and executes the following search operation:

- For $c = 0$ until d_{max}:
  - μ_j calculates $l = F_p(K_{1i}, c)$;
- For $c = 0$ until w_{max}:
  - μ_j calculates $l_A = F_p(K_{1i}^A, c)$;
- μ_j locally calls the **verifySearch** function of **searchVerify contract** with parameters (l, l_A and $\mathbf{AccTk}$).

The smart contract asserts that the present time within the expiration time and corresponding $\mathbf{AccPr}$ is in the authorizedInfo list then:

- get (d, r) according to l in L;
- get (d_A, r_A) according to l_A in L_A;
- return (d, r), (d_A, r_A) and L_D to μ_j.

After getting (d, r) and (d_A, r_A), μ_j

- For each (d, r), μ_j calculates $\mathsf{ind}_i = d \oplus F_\tau(K_{2i}, r)$ and adds ind_i into L_R;
- For each (d_A, r_A), μ_j calculates $\mathsf{ind}_i = d_A \oplus F_\tau(K_{2i}^A, r)$ and adds ind_i into L_R;

At last, μ_j checks L_D as follows:

- For each ind_i in the L_R, μ_j calculates $\mathsf{ind}_{del} = F_\tau(K_d, \mathsf{ind}_i)$ and checks $\mathsf{ind}_{del} \in L_D$ or not. Finally, μ_j returns ind_i as the search result L_R where $\mathsf{ind}_{del} \notin L_D$.

Fig. 7. The search process of our framework.

Add(ind_j, W_j)

For the newly added document ind_j with the corresponding keyword set W_j, the data owner μ_i performs the following operations:

- μ_i uses the encryption key K_E to encrypt document as EDB' and outsources it to the cloud;
- For each keyword $w_i \in \mathsf{W}_j$:
  1) μ_i calculates $K_{1i}^A = F_p(K_a, 1||w_i)$, $K_{2i}^A = F_p(K_a, 2||w_i)$;
  2) μ_i randomly chooses $r_A \in \{0,1\}^\tau$, calculates $d_A = \mathsf{ind}_j \oplus F_\tau(K_{2i}^A, r)$, $l_A = F_p(K_{1i}^A, c)$, $c++$;
  3) Add (l_A, d_A, r_A) to L_A.
- Finally, DO sends L_A to the **searchVerify contract** to update L_A by address $\mathcal{A}_i$.

Fig. 8. The added operation of our framework.

Delete(ind_i)

An empty list L_D was established at the initialized phase. When the data owner μ_i wants to delete the document ind_i, he/she sends the request to cloud to delete the corresponding document.

- For each ind_i
  - μ_i calculates $\mathsf{ind}_{del} = F_\tau(K_d, \mathsf{ind}_i)$.
- Finally, μ_i sends ind_{del} to the **searchVerify contract** to update L_D by address $\mathcal{A}_i$.

Fig. 9. The deletion operation of our framework.

the number of *wei* attached to a message and *cost* to represent a fixed number of *wei*.

Specifically, the smart contract is deployed by DO (e.g., μ_i in our framework) as the *searchVerify contract* which includes five functions (*searchInit*, *addUser*, *addIndex*, *deleteIndex*, *verifySearch*). In the initialization process, we define several variables required for the *searchVerify contract*.

- The contract creator variable is the address type, which is also the address of μ_i.
- The variable of the mapping type l, which defines a collection from the encrypted keyword indices to a structure L set. The encrypted keyword index is stored as a byte type. The structure L contains three variables as follows:
  - l is the index calculated as $F_p(K_{1i}, c)$;
  - r is a random number;
  - d is used to transform ind_i into $\mathsf{ind}_i \oplus F_\tau(K_{2i}, r)$;

The *searchVerify contract* mainly provides the following five functional interfaces:

searchInit(l, d, r): This function can only be executed by the contract creator μ_i. As described in the step (5) of EDBSetup(DB) in Fig. 2, μ_i initializes the trust index by L as *Algorithm 1*. It builds an index set L according to index l by the solidity mapping function.

Algorithm 1. searchInit

Input: l, d, r
Output: Bool
1: **if** *msg.sender* is not μ_i **then**
2: throw;
3: **end if**
4: mapping l to (d, r), and add it to the initialization index set L;
5: **return** true.

addUser($newAccPr$): This function can only be run by the contract creator μ_i. After delivering the access authorization to μ_j, μ_i should add the access control authorization information of $\mathcal{B}_j$ into the authorizedInfo list as *Algorithm 2*. Notice that, this operation should be in batch operations.

Algorithm 2. addUser

Input: **AccPr**
Output: bool
1: **if** $msg.sender$ is not μ_i **then**
2: throw;
3: **end if**
4: **if** **AccPr** has existed **then**
5: **return** false;
6: **else**
7: authorizedInfo[**AccPr**] $\Leftarrow$ true;
8: **return** true;
9: **end if**

addIndex(l_A, d_A, r_A): Only the contract creator μ_i could call this function. As shown in Fig. 8, for the newly added document ind_i with the corresponding keyword set W_j, μ_i calculates the corresponding l_A, d_A, and r_A, then he/she stores it to the smart contract. He/she builds an add index set L_A according to index l_A by the solidity mapping function as *Algorithm 3*.

Algorithm 3. addIndex

Input: l_A, d_A, r_A
Output: Bool
1: **if** $msg.sender$ is not μ_i **then**
2: throw;
3: **end if**
4: mapping l_A to (d_A, r_A), and add it to the add index set L_A;
5: **return** true.

deleteIndex(ind_i): This function can only be performed by the creator μ_i of the contract. When μ_i deletes a certain document ind_i from the cloud server, it is necessary to add ind_i to the deletion list L_D in the smart contract as *Algorithm 4*.

Algorithm 4. deleteIndex

Input: ind_i
Output: Bool
1: **if** $msg.sender$ is not μ_i **then**
2: throw;
3: **end if**
4: add ind_i to the delete index set L_D;
5: **return** true.

verifySearch(l, l_A, *AccTk*): This function can only be executed by the user in the authorized set and the creator μ_i of the contract. As shown in Fig. 7, after μ_j calculates the search token, he/she sends l, l_A, and **AccTk** to this function and gets the corresponding result L_R and L_D by the address $\mathcal{B}_j$ as and *Algorithm 5*.

In our scheme, we consider the local call of the search process. DU locally invokes the *verifySearch* function by sending l, l_A and *AccTk* to the smart contract that will check the access control status of the current account address before returning the corresponding search results. Since the local call does not change any data state, only DU gets the call results by *verifySearch* function. Thus, we can ensure access control and prevent privacy leakage.

Algorithm 5. verifySearch

Input: l, l_A, **AccTk**
Output: SearchResult, L_R
1: calculate **AccPr** $= \mathcal{H}_2(msg.sender||$**AccTk**$)$
2: **if** authorizedInfo[**AccPr**]==false
3: or **AccTk**.T_{ei} $<$ time.now **then**
4: throw;
5: **end if**
6: get $L[l]$ array's length len;
7: **if** len equal 0 **then**
8: $L_R \Leftarrow$ NULL;
9: **else**
10: $L_R \Leftarrow L[l]$;
11: **end if**
12: get $L_A[l_A]$ array's length len_A;
13: **if** len_A equal 0 **then**
14: $L_R \Leftarrow L[l]$;
15: **else**
16: $L_D \Leftarrow L_A[l_A]$;
17: **end if**
18: **return** L_R, L_D.

4 Security Analysis

In this section, we will discuss the security and privacy of our framework.

4.1 Data Privacy

Before outsourcing documents, DO encrypts DB as EDB by AES-128 algorithm. Thus, the cloud can only see ciphertexts without obtaining the plaintext of documents. After that, only the authorized DUs can obtain ciphertexts through the search operation. For the corresponding index, DO transforms (w_i, ind_i) into (l, d, r) and stores them in Ethereum as shown in Fig. 2. Each keyword w_i is encrypted as $K_{1i} = F_p(K_e, 1||w_i)$ and $K_{2i} = F_p(K_e, 2||w_i)$ by pseudo-random functions, while each ind_i is hidden by $d = \mathsf{ind}_i \oplus F_\tau(K_{2i}, r)$ where r is a random number. Thus, even two documents have the same keyword w_i, attackers cannot identify them according to (l, d, r). For the similar reason, it is impossible to distinguish whether the two newly added documents contain the keywords in existing documents. Moreover, only $\{K_{1i}, K_{2i}, K_{1i}^A, K_{2i}^A\}$ are sent to DU during the access authorization delivery. This means that even the authorized users cannot learn w_i except for the authorized documents.

In Ethereum, transactions are visible by anyone, which could result in data leakage. During the access authorization, we construct the stealth authorization to hide the receiver's address while ensuring the confidentiality of authorized information as shown in Fig. 5. During this process, we construct a public key PK_{SA} to encrypt the authorized information ($txid_L$, $\{K_{1i}, K_{2i}, K_{1i}^A, K_{2i}^A\}$, **AccTk**) by ECC-secp256k1 encryption/decryption. Thus, only the authorized DU can recover the corresponding private key SK_{SA} and obtain the detailed information. Moreover, the key pair PK_{SA}/SK_{SA} is a one-time session key.

4.2 Anonymous Access Authorization

In our framework, the access authorization can only be delivered by DO. To achieve the *untraceability* and *unlinkability*

during the access authorization, we adopt the stealth authorization scheme to hide the receiver's address as shown in Figs. 3 and 5. Without the receiver's address, μ_i can recover the application information $T_1||R_1||ST_1||c_1$ by his/her private key a_i as shown in Fig. 4. Then, μ_i constructs a stealth authorization as $T_2||R_2||ST_2||c_2$ in Ethereum as shown in Fig. 5. To recover the corresponding authorized information, μ_j should check recent transactions with his/her private key pair (a_j, b_j) to generate the corresponding decryption key SK_{SA} as shown in Fig. 6. Without the private key, it is an Elliptic Curve Discrete Logarithm Problem (ECDLP) to recover authorized information. Only after decrypting the authorized information, μ_j can execute the search smart contract and get the correct results.

Besides, DO adds adds **AccPr** into the authorizedInfo list of *searchVerify contract*. Thus, when DUs call the *searchVerify contract*, an identity check will be performed at first to check the user legitimacy to ensure access control.

4.3 Query Integrity

It is obvious that our framework can ensure query integrity as long as the security of Ethereum is guaranteed. This is because the smart contracts can perform search operations honestly according to predefined logic, and return the corresponding results $\mathcal{L}_R$. Besides, each node in the Ethereum network can verify the correctness of operations. The consensus mechanism of Ethereum guarantees the correct execution of each search operation. Although the returned results by the smart contract may contain the deleted documents, it is easy to filter them by checking $\mathcal{L}_R$. Thus, query integrity can be guaranteed which means that all the query results are originated from the DO and satisfy the query criteria (*soundness*). Besides, no valid answer in the query results is missing (*completeness*).

5 Performance Analysis

5.1 Complexity Analysis

Let *E.M.*, *E.H.*, *E.E.*, and *E.D.* denote one multiplication operation, one hash operation ($\mathcal{H}_1(\cdot)$), one encryption operation, and one decryption operation in ECC-secp256k1 elliptic curve, respectively. *P.F.* indicates one pseudo-random function operation (F_p and F_τ) and *H.* means one hash operation ($\mathcal{H}_2(\cdot)$). $|\mathbf{E}|$, $|\mathbf{PF}|$, $|\mathbf{R}|$, $|\mathbf{C_2}|$, and $|\mathbf{H}|$ denotes the size of E, the size of F_p/F_τ, the size of r/r_A, the size of c_2, and the size of $\mathcal{H}_2(\cdot)$, respectively. During the analysis, we focus on the overhead about Ethereum.

Initialization. For each keyword w_i, μ_i first encrypts K_{1i} and K_{2i} by pseudo-random functions, then calculates the d and l which need two pseudo-random function operations. Thus, the total computation is about $(2w_{max}+2O(w_{max}*d_{max}))$**P.F.**. After that, μ_i should upload (l, d, r) to Ethereum, the corresponding communication overhead is about $O(w_{max}*d_{max})(2|\mathbf{PF}|+|\mathbf{R}|)$ bytes.

Stealth Transmission. In order to apply for sharing the data, μ_j calculates the transaction public key $R_1 = r_1 G$, the stealth tag $ST_1 = \mathcal{H}_2(T_1||\mathcal{H}_1(r_1 A_i))G$, and encrypts $c_1 = \mathcal{ENC}_{A_i}(A_j, B_j)$ by ECC encryption. The elapsed time is *E.M.*, 2*E.M.*+*E.H.*+*H.*, and *E.E.*, respectively. After that, μ_j broadcasts $T_1||R_1||ST_1||c_1$ to Ethereum. The corresponding communication overhead is about $4|\mathbf{E}|$. Thus, the overall computation overhead and communication overhead are 3*E.M.*+*E.E.*+*E.H.*+*H.* and $4|E|$, respectively.

Then μ_i checks the transactions in the newly generated block by calculating $ST_1' = \mathcal{H}_2(T_1||\mathcal{H}_1(a_i R_1))G$ using his/her own private key a_i and R_1. Let μ_i match N_m times to find the right transaction that makes the equation hold. The computation overhead is about (2*E.M.*+*E.H.*+*H.*)N_m. Next, μ_i decrypts the message c_1 to get the plaintext. The corresponding computation overhead is *E.D.*. As a result, the total computation overhead of the receiver is (2*E.M.*+*E.H.* +*H.*)N_m+*E.D.*.

Stealth Authentication. To authorize the access information, μ_i calculates $ST_2 = \mathcal{H}_2(T_2||\mathcal{H}_1(r_2 A_j))G$, $R_2 = r_2 G$, $PK_{SA} = \mathcal{H}_1(r_2 A_j)G + B_j$, and encrypts $c_2 = \mathcal{ENC}_{PK_{SA}}(txid_L, \{K_{1i}, K_{2i}, K_{1i}^A, K_{2i}^A\}, AccTk)$. The corresponding computation overhead are about 2*E.M.*+*E.H.*+*H.*, *E.M.*, 2*E.M.*+*E.H.*, and *E.E.*+*H.*, respectively. Finally, μ_i sends the transaction $T_2||R_2||ST_2||c_2$ to Ethereum. The corresponding communication overhead is about $2|\mathbf{E}|+|\mathbf{C_2}|$. Thus, the overall computation overhead and communication overhead are 3*E.M.* +2*E.H.*+*E.E.*+2*H.* and $2|\mathbf{E}|+|\mathbf{C_2}|$, respectively.

μ_j checks the transactions in the newly generated block according to the transaction public key $\mathcal{R}_2$. μ_j uses his/her own private key a_j to calculate $ST_2' = \mathcal{H}_2(T_2||\mathcal{H}_1(a_j R_2))G$. We assume μ_j needs N_m times to find the right transaction that makes the equation hold. The computation overhead is about (2*E.M.*+*E.H.*+*H.*)N_m. Next, he/she calculates the corresponding decryption key $SK_{SA} = \mathcal{H}_1(a_j R_2) + b_j$ and decrypts $\mathcal{ENC}_{SK_{SA}}(c_2)$ to get $(txid_L, \{K_{1i}, K_{2i}, K_{1i}^A, K_{2i}^A\}$, **AccTk**). The corresponding calculation overhead are *E.M.* +*E.H.* and *E.D.*, respectively. As a result, the total computation overhead is about (2*E.M.*+*E.H.*+*H.*)N_m+*E.H.*+*E.D.*.

Search: In the search operation, μ_j first calculates d_{max} and w_{max} times pseudo-random function F_p, which consumes $(d_{max}+w_{max})$*P.F.*. Then, μ_j sends l and l_A to the *searchVerify contract* to get (d, r), (d_A, r_A), and L_D. After that, he/she recovers ind_i by calculates $\mathsf{ind}_i = d \oplus F_\tau(K_{2i}, r)$ and $\mathsf{ind}_i = d_A \oplus F_\tau(K_{2i}^A, r)$. The elapsed time is about $2O(d_{max}+w_{max})$*P.F.*. Finally, μ_j should check whether each ind_i is in L_D or not by calculating $\mathsf{ind}_{del} = F_\tau(K_d, \mathsf{ind}_i)$. Thus, the overhead is about $O(d_{max}+w_{max})$*P.F.*.

Dynamic Update: As shown in Fig. 8, to add a document ind_j, for each w_i in ind_j, μ_i should calculate $K_{1i}^A = F_p(K_a, 1||w_i), K_{2i}^A = F_p(K_a, 2||w_i)$, $d_A = \mathsf{ind}_j \oplus F_\tau(K_{2i}^A, r)$, and $l_A = F_p(K_{1i}^A, c)$. The corresponding computation overhead is about $4O(w_{max})$*P.F.*. Finally, he/she sends L_A to the *searchVerify contract* which needs $2|\mathbf{PF}|$ communication overhead for each added document. To delete ind_i, μ_i computes $\mathsf{ind}_{del} = F_\tau(K_d, \mathsf{ind}_i)$ and sends ind_{del} to the *searchVerify contract* to update L_D, the corresponding computation and communication overhead are about *P.F.* and $|\mathbf{PF}|$, respectively.

Finally, we summarize the computation and communication overhead in Table 2.

5.2 Local Implementation

During the implementation, we instantiate $\mathcal{H}_2(\cdot)$ as the SHA-256 hash function. F_p and F_τ are instantiated with Keccak256 functions. We carry out $\mathcal{ENC}_K(\cdot)/\mathcal{DEC}_K(\cdot)$ by the ECC-secp256k1 encryption/decryption algorithm. $\mathsf{Enc}(\cdot)/\mathsf{Dec}(\cdot)$ uses the AES-128 encryption/decryption algorithm. The size of $\mathcal{A}_i/\mathcal{B}_i$ is 20 bytes as the Ethereum address. The size of the

TABLE 2
The Overhead of our Scheme

	Entity	Computation Overhead[1]	Communication Overhead (bits)
Initialization	DO	$(2w_{max} + 2O(w_{max} * d_{max}))$**P.F.**	$O(w_{max} * d_{max})(2\|\mathbf{PF}\| + \|\mathbf{R}\|)$
Stealth Authorization	DU($\mathsf{DU} \rightarrow *$)	3**E.M.**+**E.E.**+**E.H.**+**H.**	$4\|\mathbf{E}\|$
	DO	(2**E.M.**+**E.H.**+**H.**)N_m+**E.D.**	N/A
	DO($\mathsf{DO} \rightarrow *$)	3**E.M.**+2**E.H.**+**E.E.**+2**H.**	$2\|\mathbf{E}\| + \|\mathbf{C_2}\|$
	DU	(2**E.M.**+**E.H.**+**H.**)N_m+**E.H.**+**E.D.**	N/A
Search	DU	$4O(d_{max} + w_{max})$**P.F.**	$O(d_{max} + w_{max})$**P.F.**
Update	DO (Addition)	$4O(w_{max})$**P.F.**	N/A
	DO (Deletion)	**P.F.**	$\|\mathbf{PF}\|$

[1] *N_m is the attempt time to find the matched transactions for stealth authorization.*

TABLE 3
Initialization Phase Performance

DB name	(w_i, ind_i) pairs	Encryption	TX	Our		Hu	
				Time	Gas[1]	Time	Gas
DB_1	39975	3.66 MB	195	50.68 s	91804701	8.55 s	220331085
DB_2	91083	8.34 MB	291	107.91 s	206088062	17.14 s	502387638
DB_3	157976	14.47 MB	392	184.24 s	355387121	28.97 s	860519184
DB_4	249488	22.85 MB	496	286.11 s	559291404	45.53 s	1342468640

[1] $1\ gas = 32\ Gwei = 3.2 \times 10^{10} wei = 3.2 \times 10^{-8} ether.$

secret key a_i/b_i is 32 bytes and the corresponding public key A_i/B_i is 65 bytes in Ethereum. Thus, $SK_{SA} = 32$ bytes and $PK_{SA} = 64$ bytes, respectively. Besides, $\mathsf{ind}_i = w_i = 32$ bytes[4]. To illustrate the performance, we run our framework on the locally simulated network Ethereum (Ganache [29]) and set the gasPrice as $32Gwei$ where $1Gwei = 10^9 wei = 10^{-9} ether$.

In our implementation, DO/DU side is programmed with Python [30] and the smart contract is written in Solidity. Besides, both of them run our framework on a desktop with the Intel(R) Core (TM) i5-10400F (2.9GHz) processor, 8 GB RAM, and Ubuntu 18.04 system. We implement four different databases with different pairs of (w_i, ind_i) as shown in Table 3. We compare our scheme with Hu's scheme [28]. Notice that during the comparison, we set the size of ind_i as 32 bytes without packing operation[5]. The detailed performance on Ganache in different phases is demonstrated as follows:

To initialize the database, DO needs to perform $\mathsf{EDBSetup}(\mathsf{DB})$ and sends L to the smart contract. Table 3 presents the detailed consumption with regard to different databases. TX means the number of transactions required for sending the corresponding (l, d, r) (generated by (w_i, ind_i) pairs) of different databases to Ganache. To finish the setup operation, it needs about 50.68 s for DB_1, 107.91 s for DB_2, 184.24 s for DB_3, and 286.11 s for DB_4, respectively. The corresponding gas consumption is about 91804701, 192296892, 355387121, and 559291404, respectively. Specifically, both setup time and gas cost linearly increase with the increasing size of (w_i, ind_i) pairs. The main reason is that we need more Keccak256 operations to hide w_i/ind_i and thousands of transactions (TX) to store L in Ganache. Thus, the average overhead is about 259.83 ms and 470793.33 gas for each transaction. While for Hu's scheme, the corresponding time and gas costs are about 43.77 ms and 1129903, respectively. Comparing with Hu's framework, ours incurs more computation overhead but much less gas. Because the hashlib library used by Hu's framework is almost twice as fast as the Keccak256 library we used in Python and the data size of Hu's framework is much smaller by packing operation. But we take the index as the data of a transaction and send it to the Ethereum instead of storing the data in the contract as by Hu's framework, which optimizes the gas consumption.

To achieve the access authorization, a stealth authorization scheme is implemented between DO and DU. We present the corresponding performance in different phases over 100 independent trials as shown in Fig. 10. Notice that, we only consider one time matching operation for the stealth tag. Thus, it needs at least 45.22 ms and 85.94 ms to execute the stealth transmission and stealth authorization operation, respectively. The corresponding message size is 399 bytes (261 bytes for c_1) and 537 bytes (401 bytes for c_2), respectively. Moreover, to transmit messages via Ethereum, it needs 34132 gas and 38608 gas, respectively. As depicted in Fig. 10, the time overhead for stealth authorization is much more than that for stealth transmission. This is reasonable due to the fact that two time-consuming keys (PK_{SA}/SK_{SA}) are calculated during this process. Moreover, we can observe that both operations show a fluctuated computation overhead over time, with an overall declining trend. Besides, we also study the time cost for the stealth tag matching operation. As shown in Fig. 11, we perform 100 trials and the corresponding overhead fluctuates between 19 ms and 23 ms for each matching operation. Thus, the

4. Notice that we can reduce the size of the keywords and the index to improve the performance as in [28].

5. We can pack the document identifiers to improve the performance as in [28].

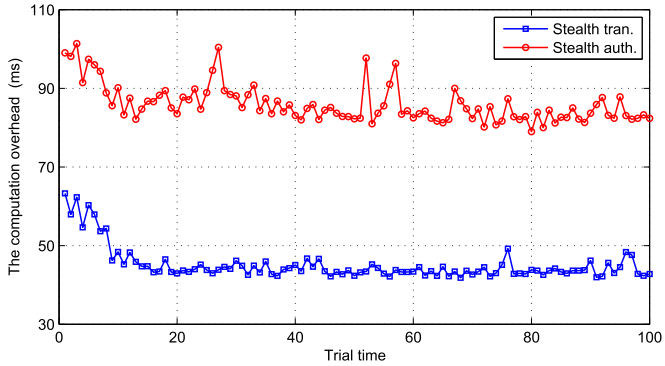


Fig. 10. The overhead of stealth transmission/authorization.

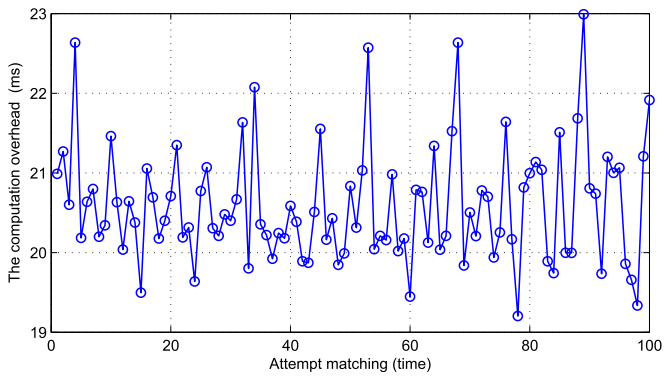


Fig. 11. The overhead of each label attempt matching.

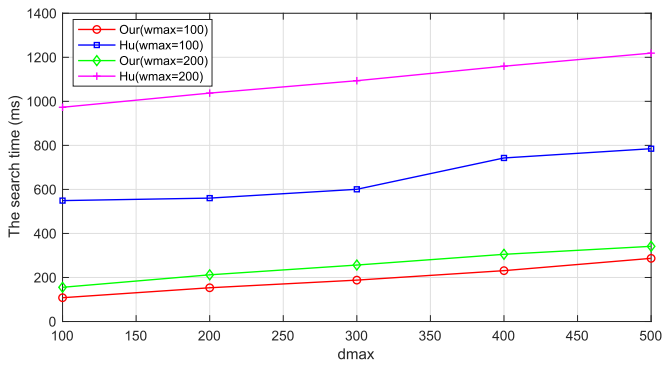


Fig. 12. The computation overhead of search operation.

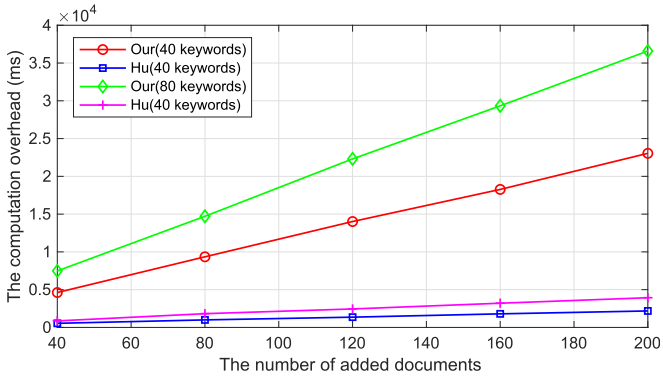


Fig. 13. The computation overhead of added operation.

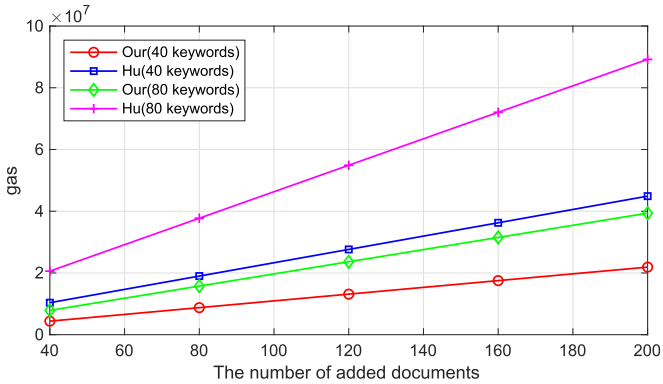


Fig. 14. The gas consumption of added operation.

average computation overhead for one matching is about 20.57 ms. Obviously, the total matching time will linearly increase with the number of matching times.

To demonstrate the efficiency of the search operation, we execute the smart contract with $\mathbf{DB}_4$. We repeat each operation for 10 times and get the average running time. Fig. 12 presents the time cost for DU with different numbers of d_{max} and w_{max}. Even for the time-consuming situation, it only takes 188.07 ms to get 300 matching documents. Obviously, the corresponding overhead linearly increased with the number of d_{max} or w_{max}. Hu's scheme adopts the mapping function to store the (l, d, r) in contract, which causes time cost exponentially to increase depending on the number of the call request. Our framework takes (l, d, r) as the transaction data in batches to send them to Ethereum instead of storing it in the contract, which greatly reduces the gas cost. According to the property of Ethereum, when we read the data from Ganache, we do not need to consume gas in the call function, since this is a local call without modifying the state of the blockchain.

We also present the performance of added operations. Figs. 13 and 14 show the impact of the number of the added documents on time and gas cost, respectively. In our added operations, we assume that each document contains 40 or 80 keywords, respectively. As shown in Fig. 13, for 40 keywords included in each document, it takes about 4620.55 ms and 23046.51 ms to add documents when there are 40 and 200 documents added, respectively. While for 80 keywords, the corresponding delay are about 7481.58 ms and 36592.64 ms, respectively. The average delay is about 115.50 ms for each document with 40 keywords and 187.08 ms for each document with 80 keywords. In contrast, the average delay in Hu's scheme are about 13.66 ms and 21.52 ms, respectively. Notice that, it consumes about 14.78 ms for each transaction to send the data to Ganache. Hu's scheme has a less calculation cost mainly due to the performance of the chosen hash library in Python and smaller packed data. It is obvious that the added cost almost linearly increases with the number of added documents. Besides, we present the corresponding gas consumption in Fig. 14. Similarly, we capture a linear increase in the gas cost when more documents with fixed keywords are added. Moreover, our scheme shows much less gas consumption compared to Hu's work, which is due to the optimized data storage mode we used.

Finally, we list the overhead of deletion operations in Table 4. During the implementation, each document contains about 40 keywords and we pack at most 120 ind_{del} in one transaction. As shown in Table 4, it takes about 50.89

TABLE 4
Document Deletion Performance

n_{ind}^1	Our		Hu	
	Time (ms)	*Gas*	Time (ms)	*Gas*
40	50.89	50524	254.33	772110
80	77.14	78910	330.83	1399535
120	90.62	109506	472.91	2026960
160	99.37	137889	613.38	2654385
200	113.50	166260	681.06	3281810

[1] *n_{ind} is the number of deleted documents (each contains 40 keywords).*

ms and 50524 gas to delete 40 documents in our scheme. In Hu's scheme, the corresponding overhead is about 254.33 ms and 772110 gas, respectively. Since the deletion operation linearly increases with the number of keywords in Hu's scheme, the overhead is much more expensive. Obviously, the overhead of both schemes increases with the number of deleted documents.

5.3 Testnet Implementation

To present the practicability, we deploy our scheme in Mumbai, the test network of Polygon, a proof-of-stake Ethernet-based Layer 2 extension network. Mumbai allows us to deploy and interact with smart contracts at a much lower cost than the Ethereum mainnet, which can reduce overhead while guaranteeing our scheme's performance and practicability. Again, we compare our scheme with Hu's scheme, with contract addresses 0xe69b537ba8A476E36FB4d92cB759b7D7E1fc6085 and 0x95Daa08e0a49faC4a1DDb4DCCA26787C8F9712DB, respectively.

Table 5 shows the time cost and gas consumption for the initialization phase with different databases in Mumbai. Compared with Table 3, the time cost for DB_4 of our scheme increases from 286.11 s to 341.25 s, and the gas consumption decreases from 559291404 to 415597904. While for Hu's scheme, the corresponding time grows from 8.55 s to 85.25 s and the gas consumption increases from 1342468640 to 1492963968. The increased time is due to the delay of Mumbai network and the block generation time (about 5 s per block in Mumbai). Besides, the difference in the EVM between Mumbai and Ganache leads to a slight decrease and increase in the gas consumption for our and Hu's schemes, respectively. However, it is obvious that the initialization cost in Mumbai is much lower than that of Ganache. For our scheme, it needs about 14.54 $MATIC$ for DB_4 and the corresponding overhead is 17.89 ETH in Ganache ($1.49/$MATIC$ while $3265.01/$ETH$).

Fig. 15 shows the time cost of search operations in Mumbai. Compared with Fig. 12, we can see that the time for our scheme grows from 188.07 ms to 375.05 ms when d_{max} is 300. The corresponding overhead still tends to linearly increase with d_{max}. For Hu's scheme, the time overhead increases faster and is more volatile than that of Ganache. This is because of multiple call operations Hu's scheme used. Thus, our scheme achieves lower and more stable search operation overhead in Mumbai.

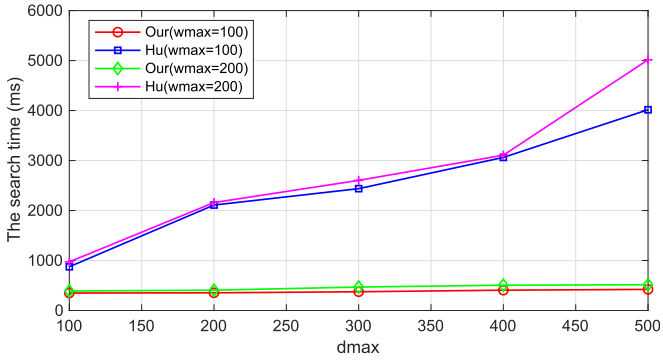


Fig. 15. The computation overhead of search operation in Mumbai.

The time cost and gas consumption of the add operation in Mumbai are shown in Figs. 16 and 17, respectively. Compared with Fig. 13, the average time cost increases from 115.50 ms to 187.53 ms for 40 keywords included in each document and from 187.08 ms to 328.13 ms for 80 keywords included in each document, respectively. Obviously, the increased time is mainly caused by the delay of the test network and the increasing number of transactions. In addition, Fig. 17 shows the gas consumption of the add operation, which has the same linear growth trend as Fig. 14. Besides, we find that both our scheme and Hu's scheme consume less gas than that of Ganache, which is related to its EVM's gas calculation rules in Mumbai.

Finally, we list the overhead of the deletion operation in Table 6, where each document contains 40 keywords. The gas consumption for deleting 200 documents is 123020 and 2530632 for our scheme and Hu's scheme, respectively. Compared to Table 4, the gas consumption still maintains an increasing trend with the number of deleted documents, while the gas consumption is reduced by 23.32% on average in Mumbai. The corresponding time overhead is 203.12 ms and 2781.25 ms, which are 1.34 times and 2.74 times higher than that of Ganache, respectively. Due to the difference in deletion algorithms, Hu's scheme requires more contract interactions, resulting in a larger increase in time overhead as well.

Deployment results from the Polygon test network in Mumbai show a corresponding increase in time overhead for our scheme compared to local deployments, which

TABLE 5
Initialization Phase Performance in Mumbai

DB name	(w_i, ind_i) pairs	Encryption	TX	Our		Hu	
				Time	Gas[1]	Time	Gas
DB_1	39975	3.66 MB	195	70.07 s	74322885	24.37 s	244787790
DB_2	91083	8.34 MB	291	154.59 s	159067293	36.37 s	558504369
DB_3	157976	14.47 MB	392	202.12 s	268362808	59.26 s	956837504
DB_4	249488	22.85 MB	496	341.00 s	415597904	85.25 s	1492963968

[1] *$1\ gas = 35\ Gwei = 3.5 \times 10^{10} wei = 3.5 \times 10^{-8} MATIC$.*

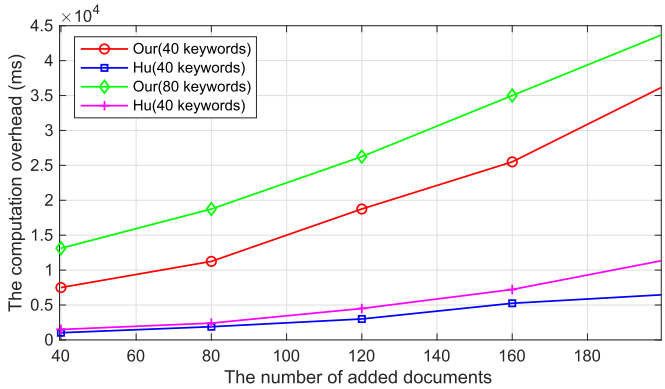


Fig. 16. The computation overhead of added operation in Mumbai.

realistically reflected the practical performance of our scheme. It is because the impact of the delay of blockchain network and block generation time on any smart contract interaction is inevitable and dominant. With the lower token price, the Polygon can dramatically reduce the cost of the contract deployment and interaction-related operations, while delivering greater transaction processing capability than the Ethereum main network.

6 Related Work

With the emergence and great potential of blockchain technology, some researchers have combined blockchain with many existing technologies such as searchable encryption. Different from previous work based on a centralized cloud server, blockchain-assisted searchable encryption enables the data owners to control their data against the untrusted platform. In our preliminary work [1], we presented a blockchain-based privacy-preserving framework to realize query integrity and data privacy for outsourced encrypted data without considering the specific access control operation. In [31], an efficient dynamic searchable encryption scheme was proposed by integrating blockchain. The authors established a trustworthy keyword search scheme in the decentralized storage. Do et al. [32] and Jiang et al. [33] focused on how to implement the distributed data storage, and how to use smart contracts to manage data. In these decentralized schemes, the data owners are responsible for authorizing users a search right, which hence cannot be applied to public task matching in crowdsourcing. Leveraging the smart contract in Ethereum, Hu et al. [28] proposed a

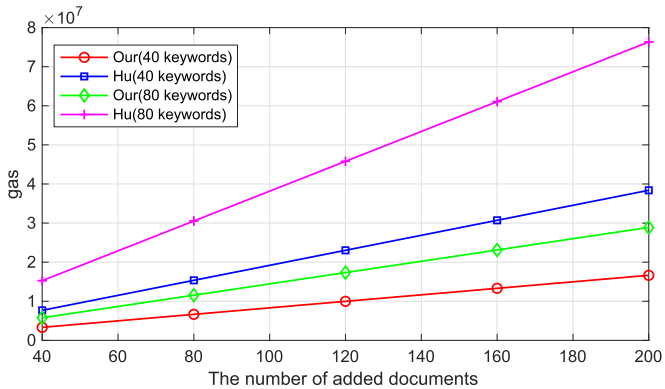


Fig. 17. The gas consumption of added operation in Mumbai.

TABLE 6
Document Deletion Performance in Mumbai

n_{ind}^1	Our		Hu	
	Time (ms)	*Gas*	Time (ms)	*Gas*
40	62.50	41816	12875.00	583992
80	90.62	61748	30781.25	1070652
120	143.75	83228	63046.87	1557312
160	234.37	103136	120687.50	2043972
200	203.12	123020	213281.25	2530632

[1] n_{ind} *is the number of deleted documents (each contains 40 keywords).*

searchable encryption scheme with integrity guarantee via blockchain. However, the access authorization was not investigated. Thus, data owners should always be online to carry out the search operation for data users. To further improve the efficiency, they employed the private blockchain to enable the user to search keywords efficiently and independently, which also makes a trade-off between security and efficiency [34]. In [35], a novel vChain framework was developed to provide verifiable queries over blockchain databases. The authors proposed an accumulator-based authenticated data structure (ADS) scheme to support dynamic data aggregation over arbitrary query attributes. Zhang et al. [36] studied authenticated range queries for database stored in the hybrid-storage blockchain. A novel gas-efficient ADS GEM^2-tree was developed to significantly reduce the storage and computation cost of the smart contract. Li et al. [37] proposed a searchable encryption scheme based on the Bitcoin system, in which the encrypted data and encrypted indices are split and stored at a set of bitcoin transactions. Nevertheless, as the data grows, a large number of transactions need to be maintained in the blockchain, thereby resulting in a high overhead and low efficiency. Tang [38] proposed two Blockchain-based frameworks which could be applied to most existing symmetric searchable encryption schemes with fairness and privacy guarantees. Recently, a blockchain-based decentralized data sharing framework was presented [39] to achieve fine-grained access control. They combined the attribute-based encryption technology and smart contract of Ethereum. Zhang et al. [14] designed a blockchain-based outsourcing service payment framework named as BPay, which can enable soundness and robust fairness. Moreover, an SSE scheme is introduced based on their BPay. Jiang et al. [40] constructed a bloom filter-enabled multi-keyword search scheme over the encrypted data on the blockchain. To improve the efficiency, they adopted bloom filter to determine the low-frequency keyword and filter the encrypted data according to the keyword. Zhang et al. [41] designed a privacy-preserving personal health information (PHI) sharing scheme based on two kinds of blockchains (consortium and private), which were applied to store encrypted indices and PHI, respectively. Chen et al. [42] considered the searchable sharing application for encrypted electronic health records (EHRs) based on blockchain. In their design, they only stored the index for EHRs in the blockchain which consisted of complex logic expressions. Besides, Jiang et al. [43] constructed a patients-controlled secure and privacy-preserving scheme based on consortium blockchain to achieve electronic health records sharing among different medical institutions.

7 Conclusion

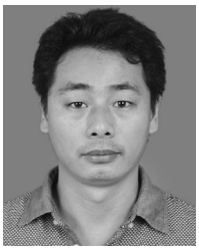
In this paper, we study the query integrity for outsourced encrypted data based on blockchain and propose a verifiable search framework in Ethereum. In our framework, we can efficiently get the correct search result by the designed smart contracts without the requirement of trust of the outsourced cloud platform. Moreover, we construct the stealth authorization scheme to achieve privacy-preserving access authorization delivery. The security analysis shows that our framework can meet the security requirements. The experimental performances according to local implementation and testnet implementation demonstrate the practicability and feasibility of our framework.

Acknowledgments

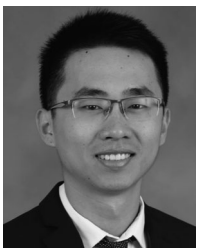
A part of this work [1] was accepted by IEEE ICC 2019.

References

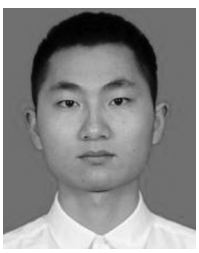
[1] S. Jiang, J. Liu, L. Wang, and S. Yoo, "Verifiable search meets blockchain: A privacy-preserving framework for outsourced encrypted data," in *Proc. IEEE Int. Conf. Commun.*, 2019, pp. 1–6.
[2] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 965–976.
[3] S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Outsourced symmetric private information retrieval," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2013, pp. 875–888.
[4] M. Azraoui, K. Elkhiyaoui, M. Onen, and R. Molva, "Publicly verifiable conjunctive keyword search in outsourced databases," in *Proc. IEEE Conf. Commun. Netw. Secur.*, 2015, pp. 619–627.
[5] H. Pang, J. Zhang, and K. Mouratidis, "Scalable verification for outsourced dynamic databases," *Proc. VLDB Endowment*, vol. 2, no. 1, pp. 802–813, 2009.
[6] F. Li, M. Hadjieleftheriou, K. Kollios, and L. Reyzin, "Dynamic authenticated index structures for outsourced databases," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2006, pp. 121–132.
[7] S. Nath and R. Venkatesan, "Publicly verifiable grouped aggregation queries on outsourced data streams," in *Proc. 29th Int. Conf. Data Eng.*, 2013, pp. 517–528.
[8] F. Li, K. Yi, M. Hadjieleftheriou, and G. Kollios, "Proof-infused streams: Enabling authentication of sliding window queries on streams," in *Proc. 33rd Int. Conf. Very large Data Bases VLDB Endowment*, 2007, pp. 147–158.
[9] R. Canetti, O. Paneth, D. Papadopoulos, and N. Triandopoulos, "Verifiable set operations over outsourced databases," in *Proc. Int. Workshop Public Key Cryptography*, 2014, pp. 113–130.
[10] S. Jiang, X. Zhu, L. Guo, and J. Liu, "Publicly verifiable boolean query over outsourced encrypted data," *IEEE Trans. Cloud Comput.*, vol. 7, no. 3, pp. 799–813, Mar. 2019.
[11] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Bus. Rev.*, p. 21260, 2008.
[12] Y. Niun, L. Wei, C. Zhang, J. Liu, and Y. Fang, "An anonymous and accountable authentication scheme for Wi-Fi Hotspot access with the bitcoin blockchain," in *Proc. IEEE/CIC Int. Conf. Commun.*, 2017, pp. 1–6.
[13] Y. Zhang, C. Xu, C. Nan, H. Li, H. Yang, and X. Shen, "Chronos+: An accurate blockchain-based time-stamping scheme for cloud storage," *IEEE Trans. Serv. Comput.*, vol. 13, no. 2, pp. 216–229, Mar./Apr. 2020.
[14] Y. Zhang, R. H. Deng, X. Liu, and D. Zheng, "Outsourcing service fair payment based on blockchain and its applications in cloud computing," *IEEE Trans. Serv. Comput.*, vol. 14, no. 4, pp. 1152–1166, Jul./Aug. 2021.
[15] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox, "Zcash protocol specification," 2016. [Online]. Available: https://github.com/zcash/zips/blob/master/protocol/protocol.pdf
[16] N. Saberhagen, "Cryptonote v 2.0," 2022. [Online]. Available: https://decred.org/research/saberhagen2013.pdf
[17] A. Ouaddah, A. A. Elkalam, and A. A. Ouahman, "FairAccess: A new blockchain-based access control framework for the internet of things," *Secur. Commun. Netw.*, vol. 9, no. 18, pp. 5943–5964, 2016.
[18] O. J. A. Pinno, A. Gregio, and L. C. E. Bona, "Controlchain: Blockchain as a central enabler for access control authorizations in the IoT," in *Proc. IEEE Glob. Commun. Conf.*, 2017, pp. 1–6.
[19] D. D. Maesa, P. Mori, and L. Ricci, "A blockchain based approach for the definition of auditable access control systems," *Comput. Secur.*, vol. 84, pp. 93–119, 2019.
[20] R. Alcarria, B. Bordel, T. Robles, D. Martin, and M. Mansocallejo, "A blockchain-based authorization system for trustworthy resource monitoring and trading in smart communities," *Sensors*, vol. 18, no. 10, pp. 3561–3590, 2018.
[21] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, no. 2014, pp. 1–32, 2014.
[22] 2022. [Online]. Available: https://polygon.technology/
[23] J. Benet, "Ipfs-content addressed, versioned, P2P file system," 2014, *arXiv:1407.3561*.
[24] Y. Lu, Q. Tang, and G. Wang, "Zebralancer: Private and anonymous crowdsourcing system atop open blockchain," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst.*, 2018, pp. 853–865.
[25] D. Hankerson, S. Vanstone, and A. Menezes, *Guide to Elliptic Curve Cryptography*, vol. 22, Berlin, Germany: Springer, 2004, Art. no. 311.
[26] Sec 2: Recommended elliptic curve domain parameters, version 2.0, 2022. [Online]. Available: http://www.secg.org/sec2–v2.pdf
[27] 2022. [Online]. Available: https://github.com/ethereum/solidity
[28] 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 Conf. Comput. Commun.*, 2018, pp. 792–800.
[29] 2022. [Online]. Available: https://www.trufflesuite.com/ganache
[30] M. Lutz, *Programming Python*. Philadelphia, PA, USA: O'Reilly Media, Inc., 2001.
[31] C. Cai, X. Yuan, and C. Wang, "Towards trustworthy and private keyword search in encrypted decentralized storage," in *Proc. IEEE Int. Conf. Commun.*, 2017, pp. 1–7.
[32] H. Do and W. K. Ng, "Blockchain-based system for secure data storage with private keyword search," in *Proc. IEEE World Congr. Serv.*, 2017, pp. 90–93.
[33] P. Jiang, F. Guo, K. Liang, J. Lai, and Q. Wen, "Searchain: Blockchain-based private keyword search in decentralized storage," *Future Gener. Comput. Syst.*, vol. 107, pp. 781–792 2017.
[34] S. Hu et al., "Augmenting encrypted search: A decentralized service realization with enforced execution," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 6, pp. 2569–2581, Nov.-Dec. 2019.
[35] C. Xu, C. Zhang, and J. Xu, "vChain: Enabling verifiable boolean range queries over blockchain databases," 2018, *arXiv:1812.02386*.
[36] C. Zhang, C. Xu, J. Xu, Y. Tang, and B. Choi, "GEM^2303 2-tree: A gas-efficient structure for authenticated range queries in blockchain," in *Proc. IEEE 35th Int. Conf. Data Eng.*, 2019, pp. 842–853.
[37] H. Li, H. Tian, F. Zhang, and J. He, "Blockchain-based searchable symmetric encryption scheme," *Comput. Elect. Eng.*, vol. 73, pp. 32–45, 2019.
[38] Q. Tang, "Towards blockchain-enabled searchable encryption," in *Proc. Int. Conf. Inf. Commun. Secur.*, 2019, pp. 482–500.
[39] 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. 38 437–38 450, 2018.
[40] S. Jiang et al., "Privacy-preserving and efficient multi-keyword search over encrypted data on blockchain," in *Proc. IEEE Int. Conf. Blockchain*, 2019, pp. 1–6.
[41] A. Zhang and X. Lin, "Towards secure and privacy-preserving data sharing in e-health systems via consortium blockchain," *J. Med. Syst.*, vol. 42, no. 8, pp. 1–18, 2018.
[42] L. Chen, W. Lee, C. Chang, K. R. Choo, and N. Zhang, "Blockchain based searchable encryption for electronic health record sharing," *Future Gener. Comput. Syst.*, vol. 95, pp. 420–429, 2019.
[43] S. Jiang, H. Wu, and L. Wang, "Patients-controlled secure and privacy-preserving EHRs sharing scheme based on consortium blockchain," in *Proc. IEEE Glob. Commun. Conf.*, 2019, pp. 1–6.



Shunrong Jiang received the BE degree in information engineering from Chongqing University, Chongqing, China, in 2008, and the PhD degree in communication and information systems from the School of Telecommunications Engineering, Xidian University, Xi'an, China, 2016. He joined the School of Computer Science & Technology, China University of Mining and Technology as an associate professor in May 2019. His research interests include security and privacy for VANETs, cloud computing, and blockchain, etc.



Jianqing Liu received the BEng degree from the University of Electronic Science and Technology of China, in 2013, and the PhD degree from the University of Florida, in 2018. He is currently an assistant professor with the Department of Computer Science, NC State University. His research interests include wireless communications and networking, security and privacy. He received the U.S. National Science Foundation Career Award in 2021.



Jingwei Chen received the BE degree in computer science and technology from the China University of Mining and Technology, Xuzhou, in 2020. He is currently working toward the master's degree in software engineering with the School of China University of Mining and Technology, Xuzhou, China. His research interests include security and privacy for blockchain, etc.

Yiliang Liu received the BE and MSc degrees in computer science and communication engineering from Jiangsu University, Zhenjiang, China, in 2012 and 2015, respectively, and the PhD degree from the School of Electronics and Information Engineering, Harbin Institute of Technology, Harbin, China, in 2020. He is currently a lecturer with the School of Cyber Science and Engineering, Xi'an Jiaotong University, Xi'an, China. His research interests include the security of wireless communications, physical layer security, and intelligent connected vehicles.

Liangmin Wang (Member, IEEE) received the BS degree in computational mathematics from Jilin University, Changchun, China, in 1999, and the PhD degree in cryptology from Xidian University, Xi'an, China, in 2007. He is a full professor with the School of Cyber Science and Engineering, Southeast University, Nanjing, China. He has been honored as a "Wan-Jiang Scholar" of Anhui Province since Nov. 2013. His research interests include data security and privacy. He is an associate editor of Security and Communication Networks, a senior member of CCF.

Yong Zhou received the BE degree in industrial automation from Hohai University, Nanjing, China, in 1997, and the MS and PhD degrees in control theory and control engineering from the China University of Mining and Technology, Xuzhou, China, in 2003 and 2006, respectively. He is a professor with the China University of Mining and Technology. His research interests include artificial intelligence, deep learning, computer vision, and blockchain, etc.