BPMS: Blockchain-Based Privacy-Preserving Multi-Keyword Search in Multi-Owner Setting

Sheng Gao, Yuqi Chen, Jianming Zhu, Zhiyuan Sui, Rui Zhang, and Xindi Ma, *Member, IEEE*

Abstract—Searchable encryption (SE) has emerged as a cryptographic primitive that allows data users to search on encrypted data. Most existing SE schemes usually delegate search operations to an intermediary such as a cloud server, which would inevitably result in single-point failure, privacy leakage, and even untrustworthy results. Several blockchain-based SE schemes have been proposed to alleviate these issues; however, they suffer from some issues, such as the support for multi-keyword multi-owner model, query privacy and data storage availability. In this paper, we propose BPMS, blockchain-based privacy-preserving multi-keyword search in multi-owner setting, which supports searching over encrypted data in trustworthy, private and efficient manners. The attribute Bloom filter has been introduced into our BPMS to build indexes, which protects query privacy and improves index generation performance. To guarantee data storage availability, our BPMS leverages the advantages of IPFS (InterPlanetary File System) to store large scale of encrypted data. Security proof and comparative analysis in theory indicate that our BPMS is more secure and efficient. A series of experiments conducted on a real-world dataset further demonstrate that our BPMS is feasible in practice.

Index Terms—Searchable encryption, blockchain, multi-keyword multi-owner, privacy preserving, IPFS

1 Introduction

With the prosperity of the Internet and cloud storage, outsourcing large-scale data to an intermediary such as a cloud server has been a popular paradigm for minimizing local data storage and maintenance burden [1]. However, due to the separation of data ownership and data control, it would suffer from various issues, such as privacy leakage, unauthorized access, illegal tampering and deletion [2]. The most common way is to encrypt the data before outsourcing, which would subsequently raise the critical issue of how to search on encrypted data. Searchable encryption (SE) as a promising, cryptographic primitive has emerged, which guarantees data confidentiality while not sacrificing searchability [3], [4]. Specifically, there are three entities in a typical SE system, where a data owner (DO) extracts a set of keywords, encrypts them into indexes and sends them with the encrypted data to a cloud server (CS), and then a data user (DU) creates a trapdoor associated with the query keywords and submits it to the CS responsible for searching. Nowadays, SE has been widely used in various fields such as healthcare [5], smart grid [6], and the Internet of Things [7].

Exisitng SE can be divided into searchable symmetric encryption (SSE) [8], [9], [10] and public key encryption with keyword search (PEKS) [11], [12], [13]. On the whole, SSE is more efficient while inevitably suffering from expensive key management and distribution issues, and PEKS is more flexible while facing keyword guessing attack [14]. Existing works that investigate privacy-preserving SE can be divided into single-keyword single-owner [8], [15], multi-keyword single-owner [16], [17], [18], single-keyword multi-owner [19], [20], and multi-keyword multi-owner [21], [22], [23]. However, these schemes usually delegate an honest-but-curious CS to take charge of storage and search management, which would cause some endogenous problems, such as single-point failure and untrustworthy results. That is, a malicious CS might deviate from the pre-defined specification and act in unauthorized tampering and deletion, which would make the retrieved results untrustworthy.

In recent years, the blockchain [24], [25] as a distributed ledger technology has been introduced to SSE, which enables SSE to be executed in decentralized, transparent and immutable manners. Specially, most of existing works [5], [26], [27], [28], [29] exploit the blockchain to achieve trustworthy and fair SSE. However, these schemes are confronted with some challenges. Firstly, all these either support single-keyword single-owner [26], [27], [29] or multi-keyword single-owner [5], [28], lacking in the support for multi-keyword multi-owner in blockchain-based SSE. Secondly, they are confronted with privacy leakage risk. Although some privacy-preserving SSE schemes have been

- Sheng Gao, Yuqi Chen, Jianming Zhu, and Zhiyuan Sui are with the School of Information, Central University of Finance and Economics, Beijing 100081, China. E-mail: {sgao, zjm}@cufe.edu.cn, yuqichen1112@163.com, suizhiyuan2010@gmail.com.
- Rui Zhang is with the Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100081, China. E-mail: zhangrui@iie.ac.cn.
- Xindi Ma is with the School of Cyber Engineering, Xidian University, Xi'an, Shaanxi 710071, China, and also with the Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin 541004, China. E-mail: xdma1989@gmail.com.

Manuscript received 16 March 2022; revised 9 July 2022; accepted 1 August 2022. Date of publication 5 August 2022; date of current version 6 September 2023.
This work was supported in part by the National Natural Science Foundation of China under Grants 62072487 and 61902290, in part by Beijing Natural Science Foundation under Grant M21036, in part by the National Statistical Science Foundation of China under Grant 2020LD01, in part by the Key Research and Development Program of Shaanxi under Grant 2020ZDLGY09-06, in part by Zhejiang Provincial Natural Science Foundation under Grant LD22F020002, and in part by Guangxi Key Laboratory of Trusted Software under Grant kx202004.
(Corresponding authors: Sheng Gao and Xindi Ma.)
Recommended for acceptance by W. Cai.
Digital Object Identifier no. 10.1109/TCC.2022.3196712

proposed, query privacy may be violated by side channel attack. Thirdly, the encrypted data are stored in a DO locally or outsourced to a CS, which inevitably impacts data availability. Any crash would make a DU cannot recover the data even if the trapdoor matches the index.

To alleviate these issues, in this paper, we propose a **B**lockchain-based **P**rivacy-preserving **M**ulti-keyword **S**earch scheme in multi-owner setting, namely BPMS, which can achieve trustworthy and private search while improving the data storage availability and system efficiency. Specifically, to support multi-keyword multi-owner in blockchain-based SSE while resisting privacy leakage, we improve MKSSMDO [23] by introducing the attribute Bloom filter [30] to guarantee query privacy in our BPMS. Moreover, we adopt the IPFS (InterPlanetary File System) [31] to achieve distributed encrypted data storage, which further enhances the efficiency and robustness of our BPMS. The main contributions are summarized as follows.

- *Trustworthy multi-keyword search in multi-owner setting.* To solve the endogenous problems in existing cloud-based SE schemes, our BPMS introduces the blockchain to take charge of search operations for trustworthy results. Moreover, our BPMS supports multi-keyword search in multi-owner setting based on MKSSMDO [23], which allows each DO to construct search indexes with owned randomly chosen keys for different data files and supports each DU to conduct multi-keyword search by using conjunctive query.
- *Index and query privacy preservation.* We found that query privacy in MKSSMDO [23] would be violated by revealing the positions of keywords in the trapdoor. Our BPMS adopts the attribute Bloom filter [30] to construct a secure index and generates a secure trapdoor by using keywords hashing to hide their positions for guaranteeing query privacy.
- *Secure and efficient storage.* To ensure data availability, our BPMS leverages the IPFS [31] to store encrypted data files in decentralization, deduplication and tamper resistance manners. The encrypted data files can be retrieved with content addressing from multiple nodes at once, which can provide a high throughput. Compared with existing IPFS-based storage paradigm for SE, our BPMS expounds the detailed process of uploading the identifiers in the IPFS to the blockchain instead of the encrypted data files.
- *Efficiency with improved security.* Our BPMS ensures enhanced security against query privacy leakage without sacrificing performance. Theoretical analysis and experimental evaluation using a real-world dataset demonstrate that our BPMS and MKSSMDO [23] achieve comparable time cost for trapdoor generation and search. Especially, the time cost for index construction of our BPMS is lower.

The remainder of this paper is organized as follows. In Section 2, we review related work. In Section 3, we introduce some preliminaries required for the design of our BPMS. In Section 4, we present system overview of our BPMS and in Section 5, we detail the design of our BPMS. Theoretical analysis and experimental evaluation are conducted in Section 6 and 7, respectively. Finally, we conclude the paper in Section 8.

2 Related Work

SSE has been regarded as an effective primitive for searching over encrypted data, which is first proposed by Song et al. [8] in single-keyword single-owner setting. Subsequently, the security models have been formalized by Goh [32] and Curtmola et al. [9]. In this section, we briefly summarize the state-of-the-art works on SSE that are related to our BPMS.

2.1 Cloud-Based SSE

Cloud-based SSE schemes with various characteristics summarized in [4], [9], [10] have been proposed for supporting different search types, such as conjunctive search [16], [33], [34] and ranked search [15], [17], [23], [35], [36].

Golle et al. [33] first formalized a security model for conjunctive Boolean search, and then proposed two schemes under the security model with different communication costs. Ballard et al. [16] used bilinear pairing to achieve conjunctive Boolean search with constant size trapdoors and less storage overhead. Cash et al. [34] extended conjunctive Boolean search to support very large databases and text search at the cost of revealing data access patterns. To further reduce the communication costs, ranked search arises. Wang et al. [15] first defined ranked SSE, and then utilized order-preserving symmetric encryption to achieve single-keyword ranked search while preserving keyword privacy against frequency relevance attack. Cao et al. [17] first investigated multi-keyword ranked search. They measured the similarity between a search query and a document based on secure inner product computation, and then proposed two multi-keyword ranked search schemes for different levels of privacy under the defined threat models. Sun et al. [35] improved the search result accuracy and efficiency in [17] by the vector space model and a proposed tree-based index. To reduce the cost of search, Ding et al. [36] proposed a privacy-preserving and efficient multi-keyword top-k search based on group partition. However, these schemes only support single-owner setting, which suffer from complicated key distribution and considerable communication costs across multi-owner.

Several SSE schemes in multi-owner setting have been proposed [21], [22], [23]. Zhang et al. [21] used an additive order and privacy-preserving function family to achieve ranked search in multi-keyword multi-owner setting. Guo et al. [22] introduced a trusted third party to distribute keys for generating encrypted index and trapdoor, and then proposed a heuristic weight generation algorithm by taking into account the data quality to achieve a reasonable multi-keyword search in multi-owner setting. However, it would incur the risk of single-point failure and increase the computational cost. Yin et al. [23] enabled the CS to compute the similarity of a multi-keyword conjunctive query to a data file satisfying the query for achieving ranked search without the involvement of an extra intermediary. However, the reveal of the positions of keywords in the trapdoor would cause the leakage of query privacy. On the whole, the trustworthiness of search results depends on the CS in these schemes, which inevitably causes some endogenous security issues as mentioned above.

2.2 Blockchain-Based SSE

To alleviate the endogenous security issues, blockchain-based SSE attracts widespread attention. Hu et al. [26]

TABLE 1
Summary of Notations

Notation	Description
$\mathcal{DO}$	The set of data owners, denoted as $\mathcal{DO} = \{DO_1, DO_2, \ldots, DO_{\vert\mathcal{DO}\vert}\}$
$\mathcal{F}_i$	The data file set owned by DO_i, denoted as $\mathcal{F}_i = \{F_{i,1}, F_{i,2}, \ldots, F_{i,\vert\mathcal{F}_i\vert}\}$
$W_{i,j}$	The keyword set of $F_{i,j}$, denoted as $W_{i,j} = \{w_{i,j,1}, w_{i,j,2}, \ldots, w_{i,j,\vert W_{i,j}\vert}\}$
$\mathcal{I}_{F_{i,j}}$	The secure index of $F_{i,j}$
$sk_{i,j}$	The temporary key used to generate $\mathcal{I}_{F_{i,j}}$
u	An authenticated data user
$\mathcal{Q}$	The query keyword set of u, denoted as $\mathcal{Q} = \{w_1, w_2, \ldots, w_{\vert\mathcal{Q}\vert}\}$
r_u, s_u	Two temporary keys chosen by u to generate trapdoor
$\mathcal{T}(\mathcal{Q})$	The trapdoor generated by u for query $\mathcal{Q}$
$\mathcal{D}$	A pre-defined keyword dictionary that is the collection of keyword sets of all data files
$id(F_{i,j})$	The CID of $F_{i,j}$
$\mathcal{L}$	The list that contains the identifiers of matched data files

first exploited the well-designed smart contract to replace the CS for achieving trustworthy and financially-fair search. Tahir et al. [37] proposed to store the encrypted data on a permissioned blockchain and use probabilistic trapdoor to hide the search pattern. To resist threats that come from both the CS and DUs in decentralized storage, Li et al. [27] and Cai et al. [29] utilized the blockchain to guarantee the fairness when dispute happens. However, all these schemes are proposed in single-keyword single-owner setting. Chen et al. [5] extended the single-keyword search in [26] to support Boolean search. They respectively store the search index on the blockchain and the encrypted data in the CS, and only those DUs authenticated by the DO could obtain the trapdoor for searching. Obviously, it cannot protect query privacy and causes considerable communication cost. Jiang et al. [28] used the Bloom filter [38] to get the low-frequency keywords for improving the efficiency of SSE in multi-keyword single-owner setting. To the best of our knowledge, the blockchain-based SSE in multi-keyword multi-owner setting has not been deeply investigated, which limits diverse search patterns. Moreover, due to the use of the centralized model, storing encrypted data in DOs locally or outsourcing to a CS will inevitably reduce the data availability.

3 Preliminaries

For the sake of description, we first introduce the main notations used in this paper, as shown in Table 1, and then present the relevant background knowledge of our BPMS.

3.1 Basic Cryptography

Definition 1. ***Bilinear Mapping*** *[23], [39]. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two cyclic multiplicative groups with the same composite order $q = q_1 \cdot q_2$, where $q_1, q_2 \in \mathbb{Z}_q$ are two large primes. A bilinear mapping $\hat{e}$ is defined as $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$, which satisfies the following properties.*

- ***Bilinearity****: $\forall u, v \in \mathbb{G}_1, \forall a, b \in \mathbb{Z}_q^*$, then we have $\hat{e}(u^a, v^b) = \hat{e}(u, v)^{ab}$.*
- ***Computability****: $\forall u, v \in \mathbb{G}_1$, there is a polynomial time algorithm to compute $\hat{e}(u, v)$.*
- ***Non-degeneracy****: If g_1, g_2 are the generators of $\mathbb{G}_1$, then $\hat{e}(g_1, g_2)$ is a generator of $\mathbb{G}_2$.*

Definition 2. ***Discrete Logarithm Problem (DLP) Assumption*** *[23], [40]. Let g be a generator of cyclic multiplicative group $\mathbb{G}$ with order q. Given g and g^a, the DLP assumption is that there is no probabilistic polynomial time (PPT) adversary, who can output $a \in \mathbb{Z}_q^*$ with a non-negligible advantage.*

Definition 3. ***Decisional Diffie-Hellman (DDH) Assumption*** *[23], [41]. Let g be a generator of cyclic multiplicative group $\mathbb{G}$ with order q. Given three random elements $a, b, c \in \mathbb{Z}_q^*$, the DDH assumption is that there is no PPT adversary, who can distinguish the tuple (g, g^a, g^b, g^{ab}) from the tuple (g, g^a, g^b, g^c) with a non-negligible advantage.*

3.2 Blockchain and IPFS

The Blockchain is first introduced into Bitcoin [42], which achieves trustworthy transactions among those mutually distrusting nodes. By integrating these technologies such as distributed ledger, cryptography, consensus mechanism and smart contract, it can store transactions in consistent, immutable, and traceable manners [24]. Considering the demands of large storage space for data volume expansion and high network bandwidth for data synchronization, it is improper to directly store large scale of data on blockchains [43]. To strengthen system scalability and guarantee data availability, IPFS (InterPlanetary File System) [31] has emerged, which exploits content-based addressing instead of location-based addressing to achieve decentralized data storage. Specifically, a large file is split into blocks and each block has a content identifier (CID) generated by a cryptographic hash function. Then, a Merkle directed acyclic graph that represents the file as a whole can be formed by linking these CIDs, which can help to achieve immutability, deduplication and tamper proof. Finally, the file can be recovered by fetching those blocks across network nodes at once, which is implemented by a distributed hash table (DHT) [44]. The DHT that stores the key-value pairs is dispersed over all network nodes, where each node only hosts a part of them. It is used for locating the nodes that store the required blocks. When looking for a specific value, the request node communicates with a node whose identifier is close to the key. It either returns the corresponding value or forward it to other nodes with closer identifiers [45].

3.3 Attribute Bloom Filter

Bloom filter (BF) is first proposed by Bloom [38], which is used to check whether or not an element is included in a set in the manner of space and time trade-offs. However, it only provides membership query and suffers from a certain false positive rate. Yang et al. [30] extended BF to attribute Bloom filter (ABF) to achieve both the membership check and the attribute localization in much lower false positive rate. To insert an element e into ABF, e is first shared with (m, m) secret sharing scheme by randomly generating $m-1$ elements $r_{l,e}$, and the m-th element is set as $r_{m,e} = r_{1,e} \oplus r_{2,e} \oplus \cdots \oplus r_{m-1,e} \oplus e$. The insert position of $r_{l,e}$ in the ABF, denoted as $H_l(att_e)$, is computed by the hash function $H_l(\cdot)$, where $l = 1, 2, \ldots, m$ and att_e is the attribute associated with e. When the conflict happens, a prior element occupied in the same position would be reused.

3.4 Review of Yin et al's Scheme

Yin et al. [23] proposed MKSSMDO to support multi-keyword search in multi-owner setting, where DOs are allowed to choose different keys to construct indexes and authenticated DUs can also randomly choose keys to generate trapdoors without knowing the keys of different DOs. MKSSMDO [23] consists of multiple DOs, multiple DUs and a CS. It is worth noting that the scheme mainly includes four phases, namely system initialization ($Init$), secure index construction ($IndexCon$), trapdoor generation ($TrapGen$) and search ($ServerSearch$). We breifly review the design of MKSSMDO in Yin et al.'s scheme [23] as follows.

- $Init(1^\lambda) \rightarrow (\mathbb{G}_1, \mathbb{G}_2, \hat{e}, q, g, h, q_1, q_2)$: Taking a security parameter λ as input, bilinear parameters $\{\mathbb{G}_1, \mathbb{G}_2, \hat{e}, q, g\}$, a hash function h and two important parameters q_1, q_2 are generated.
- $IndexCon(W_{i,j}, sk_{i,j}, h, q_1) \rightarrow \mathcal{I}_{F_{i,j}}$: The index construction algorithm is run by DOs. For the data file $F_{i,j} \in \mathcal{F}_i$ owned by DO_i, it takes the keyword set $W_{i,j}$, a temporary key $sk_{i,j}$, the hash function h and the parameter q_1 as inputs and outputs the corresponding index $\mathcal{I}_{F_{i,j}}$.
- $TrapGen(\mathcal{Q}, s_u, r_u, h, q_2) \rightarrow \mathcal{T}(\mathcal{Q})$: The trapdoor generation algorithm is run by an authenticated data user u. Taking a query $\mathcal{Q}$ of u, two randomly chosen keys s_u, r_u, the hash function h and the parameter q_2 as inputs, the algorithm outputs the trapdoor $\mathcal{T}(\mathcal{Q})$.
- $ServerSearch(\mathcal{I}_{F_{i,j}}, \mathcal{T}(\mathcal{Q})) \rightarrow$ encrypted file: The search algorithm is run by a CS. Taking the index $\mathcal{I}_{F_{i,j}}$ and the trapdoor $\mathcal{T}(\mathcal{Q})$ as inputs, the algorithm outputs the corresponding encrypted file if $\mathcal{I}_{F_{i,j}}$ matches $\mathcal{T}(\mathcal{Q})$.

Though MKSSMDO is a more practical SE scheme, it cannot guarantee query privacy in essence. Taking a query of u for example, the trapdoor for $\mathcal{Q}$ is denoted as

$$\mathcal{T}(\mathcal{Q}) = \begin{cases} T_1 = \{g^{s_u \cdot \sum_{k=1}^{|\mathcal{Q}|} h(w_k)}, p_{w_1}, p_{w_2}, \ldots, p_{w_{|\mathcal{Q}|}}\} \\ T_2 = g^{q_2 \cdot r_u \cdot (\hat{s}_u + 1)} \\ T_3 = g^{r_u \cdot q_2} \end{cases}.$$

It is worth noting that T_1 contains the positions of query keywords in keyword dictionary $\mathcal{D}$. Since $\mathcal{D}$ is public and the trapdoor is transmitted on public channel, anyone can uncover query keywords by simply checking the keywords at positions indexed by $p_{w_1}, p_{w_2}, \ldots, p_{w_{|\mathcal{Q}|}}$ in $\mathcal{D}$.

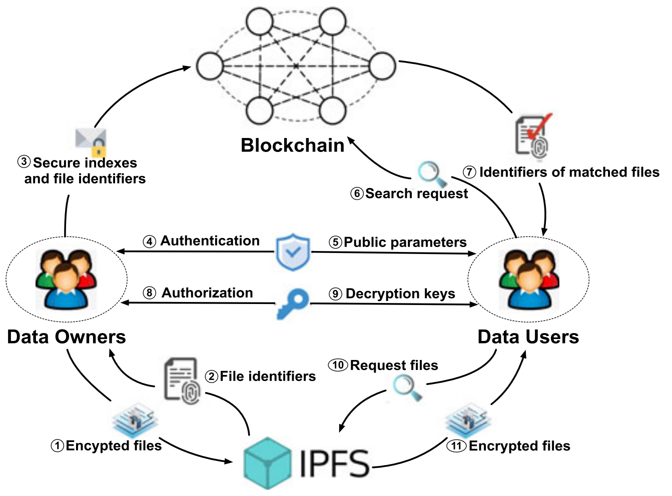

Fig. 1. System model of BPMS.

4 System Overview

In this section, we first present the system model of our BPMS, and then respectively define the key algorithms and security model for our BPMS. Finally, we put forward the overall design goals.

4.1 System Model

As shown in Fig. 1, we consider a blockchain-based SSE system, which consists of four entities as Data Owners (DOs), Data Users (DUs), Blockchain and IPFS.

- **Data Owners.** DOs mainly take charge of secure index construction. For each data file $F_{i,j} \in \mathcal{F}_i$, DO_i first builds the keyword set $W_{i,j} \in \mathcal{D}$ and then performs encryption operations to construct the secure search index $\mathcal{I}_{F_{i,j}}$. To achieve more robust and efficient storage, DO_i encrypts $F_{i,j}$ with a symmetric key and gets the corresponding identifier by sending it to the IPFS. To achieve trustworthy search, DO_i packages $\mathcal{I}_{F_{i,j}}$ and the identifier into a transaction and sends it to the blockchain.
- **Data Users.** DUs have to be strictly authenticated by DOs before entering the system, and the authenticated data user u will generate the trapdoor according to query keywords of interest. Rather than searching for only one keyword at a time, u can perform some certain operations on a query keyword set $\mathcal{Q} \in \mathcal{D}$ to achieve multi-keyword search. To prevent unauthorized access to the data files, u requests the decryption keys from DOs after receiving the identifiers of encrypted files from the blockchain.
- **Blockchain.** Our BPMS employs the consortium blockchain that is composed by a set of pre-defined nodes for achieving trustworthy search. It stores the encrypted indexes and conducts the accurate match operations by using smart contract. Specifically, after receiving the trapdoor $\mathcal{T}(\mathcal{Q})$ from u, each node of the consortium blockchain traverses indexes and performs search operation to match them with $\mathcal{T}(\mathcal{Q})$. Once consensus achieved, the identifiers of matched data files are sent back to u. Note that our BPMS

adopts voting-based consensus mechanism, which can improve the system efficiency.

- **IPFS.** The IPFS is responsible for the storage of encrypted data files. The large file would be split and stored in different nodes across the network. After receiving the identifiers of target files from u, the IPFS retrieves encrypted files from different nodes at once and sends them to u.

More specifically, multiple DOs have certain data files and have demand of outsourcing them to share with authenticated DUs [46]. The authentication method can refer to the scheme proposed in [47]. Specifically, in step ①, DO_i uses a symmetric encryption algorithm to encrypt owned files, sends them to the IPFS and records the identifier of each file returned by the IPFS in step ②. As shown in step ③, DO_i then utilizes a self-chosen temporary key to encrypt the pre-extracted keyword set into an index for each data file. The index along with the identifier of the corresponding file is sent to the blockchain. Because our BPMS supports multi-owner setting, different DOs are allowed to use independent and distinct keys to generate indexes.

A data user who wants to search encrypted data sends authentication request to corresponding DO_i in step ④, and DO_i who grants the permission shares public parameters with the authenticated data user u in step ⑤. In step ⑥, u uses the public parameters and two randomly selected keys to convert the query keyword set of interest into a trapdoor and sends it to the blockchain. It should be pointed that in our BPMS, u can generate the trapdoor without knowing the keys that are owned by different DOs for index generation. According to the trapdoor, the operation of matching indexes is implemented by smart contract on the blockchain and the identifiers of matched files are sent back to u as shown in step ⑦. u then requests DO_i for the access authorization of the corresponding encrypted files and obtains the decryption keys in step ⑧ and step ⑨. The identifiers of matched files are sent to the IPFS by u in step ⑩, and finally the encrypted files that can be decrypted by u are returned in step ⑪.

4.2 Definition of BPMS

For the sake of presentation, we mainly define four key algorithms in our BPMS and the search in the IPFS will be descried separately.

- $SystemSetup(1^{\lambda}) \rightarrow (\mathbb{G}_1, \mathbb{G}_2, \hat{e}, q, g, h, H_t, q_1, q_2)$: Taking a security parameter λ as input, bilinear parameters $\{\mathbb{G}_1, \mathbb{G}_2, \hat{e}, q, g\}$, a hash function h, n hash functions H_t $(t = 1, 2, \ldots, n)$ for the ABF and two important parameters q_1, q_2 are generated.
- $GenIndex(W_{i,j}, sk_{i,j}, h, H_t, q_1) \rightarrow \mathcal{I}_{F_{i,j}}$: The index generation algorithm is run by a DO. For each data file $F_{i,j} \in \mathcal{F}_i$ owned by DO_i, it takes the keyword set $W_{i,j}$, a temporarily chosen key $sk_{i,j}$, the hash function h, these n hash functions H_t in the ABF and the parameter q_1 as inputs, and outputs the secure search index $\mathcal{I}_{F_{i,j}}$.
- $GenTrapdoor(\mathcal{Q}, s_u, r_u, h, H_t, q_2) \rightarrow \mathcal{T}(\mathcal{Q})$: The trapdoor generation algorithm is run by a DU. Taking a query $\mathcal{Q}$, two randomly chosen keys s_u, r_u, the hash function h, these n hash functions H_t in the ABF and the parameter q_2 as inputs, the algorithm outputs the corresponding trapdoor $\mathcal{T}(\mathcal{Q})$.
- $Search(\mathcal{I}_{F_{i,j}}, \mathcal{T}(\mathcal{Q})) \rightarrow id(F_{i,j})$: This algorithm is executed by the consensus committee in the consortium blockchain. It takes the index $\mathcal{I}_{F_{i,j}}$ and the trapdoor $\mathcal{T}(\mathcal{Q})$ as inputs and outputs $id(F_{i,j})$ if $\mathcal{I}_{F_{i,j}}$ matches $\mathcal{T}(\mathcal{Q})$.

4.3 Security Model

In our BPMS, any DO or DU that has been authenticated [47] can join the consortium blockchain and access the encrypted indexes and the submitted trapdoors. Each node in the consortium blockchain is considered to be honest-but-curious. That is, it will perform all blockchain operations as predefined protocols, but it is curious about the privacy of the encrypted data files and the queries.

With the above considerations, we consider two interactive games between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$, which are used to prove that our BPMS is secure in indistinguishability against chosen keyword attack (IND-CKA) under the random oracle model.

Game$_1$: $\mathcal{A}$ is allowed to get access to index construction oracle for polynomial times and continuously generate indexes with self-selected keyword sets.

- **Setup** The challenger $\mathcal{C}$ runs $SystemSetup$ and sends public parameters to $\mathcal{A}$.
- **Phase 1** The adversary $\mathcal{A}$ is allowed to query the index construction oracle for polynomial times with different keyword sets. Upon receiving a keyword set from $\mathcal{A}$, the challenger runs $GenIndex$ to produce the corresponding index and returns it to $\mathcal{A}$.
- **Challenge** $\mathcal{A}$ chooses two sets of keywords W_0 and W_1 with the same size and sends them to $\mathcal{C}$. $\mathcal{C}$ randomly selects a bit $b \in \{0, 1\}$, encrypts the corresponding keyword set W_b into the index and sends it to $\mathcal{A}$.
- **Phase 2** Same as **Phase 1**, $\mathcal{A}$ continues to query the index construction oracle with different keyword sets.
- **Guess** $\mathcal{A}$ outputs the guess $b' \in \{0, 1\}$ on b, if $b = b'$, then $\mathcal{A}$ wins the game.

Game$_2$: $\mathcal{A}$ is allowed to get access to trapdoor generation oracle for polynomial times with self-selected query keyword sets.

- **Setup** The challenger $\mathcal{C}$ runs $SystemSetup$ and sends public parameters to $\mathcal{A}$.
- **Phase 1** $\mathcal{A}$ is allowed to query the trapdoor generation oracle for polynomial times with different search keyword sets. Upon receiving a search keyword set from $\mathcal{A}$, the challenger runs $GenTrapdoor$ to produce the corresponding trapdoor and return it to $\mathcal{A}$.
- **Challenge** $\mathcal{A}$ chooses two sets of query keywords $\mathcal{Q}_0$ and $\mathcal{Q}_1$ with same size and sends them to $\mathcal{C}$. $\mathcal{C}$ randomly selects a bit $b \in \{0, 1\}$, encrypts the corresponding keyword set $\mathcal{Q}_b$ into the trapdoor and sends it to $\mathcal{A}$.
- **Phase 2** Same as **Phase 1**, $\mathcal{A}$ continues to query the trapdoor generation oracle with different search keyword sets.

- **Guess** $\mathcal{A}$ outputs the guess $b' \in \{0,1\}$ on b, if $b = b'$, then $\mathcal{A}$ wins the game.

The advantage that the adversary $\mathcal{A}$ wins the above games are defined as $Adv_{\mathcal{A}}^{Index} = |Pr[b = b'] - \frac{1}{2}|$ and $Adv_{\mathcal{A}}^{Trapdoor} = |Pr[b = b'] - \frac{1}{2}|$, respectively. For any PPT adversary $\mathcal{A}$, if the advantage $Adv_{\mathcal{A}}^{Index}$ and $Adv_{\mathcal{A}}^{Trapdoor}$ are negligible, then our BPMS achieves IND-CKA security under the random oracle model.

Definition 4. *Our BPMS satisfies IND-CKA security under the random oracle model.*

4.4 Design Goals

To achieve trustworthy and privacy-preserving multi-keyword search in multi-owner setting, our BPMS should simultaneously meet the following design goals.

- *Effective Search.* Our BPMS should provide trustworthy and accurate search results when DUs conduct conjunctive keywords search on encrypted data of multiple DOs.
- *Privacy Preservation.* Our BPMS should prevent adversaries from knowing the plaintext of encrypted keywords derived from indexes or trapdoors.
- *Reliable Storage.* Our BPMS should provide high availability of encrypted data storage, which can help DUs keep from suffering from issues like single-point failure.
- *Running Efficiency.* Our BPMS should guarantee the efficiency be acceptable in evaluation of index construction, trapdoor generation and search algorithm.

5 Design of Our BPMS

In this section, we detail the design of our BPMS, including system initialization, secure index construction, query trapdoor generation, search on the blockchain and search in the IPFS. The notions used have been presented in Table 1.

5.1 System Initialization

In the system initialization phase of our BPMS, given a large security parameter λ, DO_i that is strictly authenticated before entering the system can run *SystemSetup* to generate global parameters $\{\mathbb{G}_1, \mathbb{G}_2, \hat{e}, q, g, h, H_t, q_1, q_2\}$. The bilinear parameters $\{\mathbb{G}_1, \mathbb{G}_2, \hat{e}, q, g\}$ are posted to the bulletin board, where $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ is a bilinear map, $\mathbb{G}_1$ and $\mathbb{G}_2$ are two multiplicative cyclic groups with the same composite order q and g is a generator of $\mathbb{G}_1$. A cryptographic one-way hash function $h : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$ and n independent and unified hash functions H_t are generated for the ABF, where $H_t : \{0,1\}^* \rightarrow \mathbb{Z}_q^*,\ t = 1, 2, \ldots, n$. Note that they are all used to map an arbitrary string to an element in $\mathbb{Z}_q^*$. In addition, DO_i also generates two λ-bit large primes $q_1, q_2 \in \mathbb{Z}_q$ that are shared among those authenticated DOs in a secure manner, where $q = q_1 \cdot q_2$. Any DU that is authenticated by DO_i can access the parameter q_2 via a secure communication channel, which is used to generate a query trapdoor [23]. It should be pointed that although the authenticated DUs can also obtain q_1 due to the fact that q is public, it would not affect the security of our BPMS. That is because the temporary key randomly generated by each DO independently is also needed to encrypt the keywords of a data file.

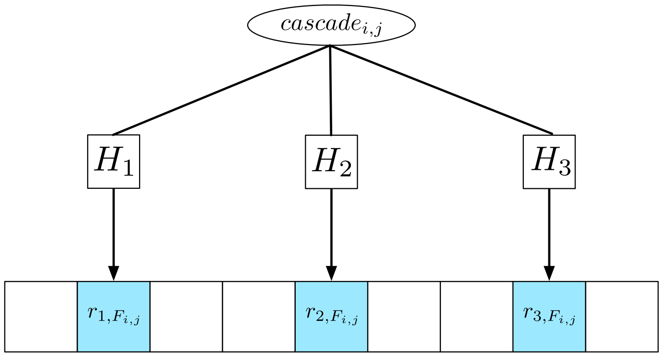

Fig. 2. An example of secure index construction.

5.2 Secure Index Construction

In Section 3, we have briefly introduced the ABF [30], which would be used for secure index construction and match search. Specifically, in our BPMS, DOs run *GenIndex* to generate the secure index for a data file, as shown in Algorithm 1.

Algorithm 1. *GenIndex*

```
Input: the keyword set $W_{i,j}$, a temporary key $sk_{i,j} \in \mathbb{Z}_q^*$, the hash function $h$, $n$ hash functions $H_t,\ t = 1, 2, \ldots, n$, the parameter $q_1 \in \mathbb{Z}_q$
Output: the secure index $\mathcal{I}_{F_{i,j}}$
 1: $\mathcal{I}_{F_{i,j}} \leftarrow \emptyset$;
 2: Compute $\prod_{k=1}^{|W_{i,j}|} g^{h(w_{i,j,k}) + q_1 \cdot sk_{i,j}}$;
 3: for $l = 1$ to $n - 1$ do
 4:     Generate $r_{l,F_{i,j}} \in \mathbb{G}_1$ at random;
 5: end for
 6: Compute $r_{n,F_{i,j}}$ as $r_{n,F_{i,j}} \leftarrow r_{1,F_{i,j}} \oplus r_{2,F_{i,j}} \oplus \ldots \oplus r_{n-1,F_{i,j}} \oplus \prod_{k=1}^{|W_{i,j}|} g^{h(w_{i,j,k}) + q_1 \cdot sk_{i,j}}$;
 7: for $t = 1$ to $n$ do
 8:     Compute the inserted position in $\mathcal{I}_{F_{i,j}}$ as $H_t(w_{i,j,1} || w_{i,j,2} || \ldots || w_{i,j,|W_{i,j}|})$;
 9:     Insert the element $r_{t,F_{i,j}}$ into $\mathcal{I}_{F_{i,j}}$;
10: end for
11: return $\mathcal{I}_{F_{i,j}}$
```

Taking the file $F_{i,j}$ as example, DO_i first initializes an empty array, and uses a temporarily chosen key $sk_{i,j}$ to compute $\prod_{k=1}^{|W_{i,j}|} g^{h(w_{i,j,k}) + q_1 \cdot sk_{i,j}}$ according to the keyword set $W_{i,j}$. To pad the array, DO_i then randomly generates $n-1$ elements $r_{l,F_{i,j}} \in \mathbb{G}_1$, where $l = 1, 2, \ldots, n-1$ and sets $r_{n,F_{i,j}}$ as

$$r_{n,F_{i,j}} = r_{1,F_{i,j}} \oplus r_{2,F_{i,j}} \oplus \ldots \oplus r_{n-1,F_{i,j}} \oplus \prod_{k=1}^{|W_{i,j}|} g^{h(w_{i,j,k}) + q_1 \cdot sk_{i,j}}.$$

To determine where to insert the elements, DO_i further hashes $w_{i,j,1} || w_{i,j,2} || \ldots || w_{i,j,|W_{i,j}|}$ with these n independent and unified hash functions H_t to obtain $H_t(w_{i,j,1} || w_{i,j,2} || \ldots || w_{i,j,|W_{i,j}|})$ and insert element $r_{t,F_{i,j}}$ into the array at the position indexed by $H_t(w_{i,j,1} || w_{i,j,2} || \ldots || w_{i,j,|W_{i,j}|})$, where $t = 1, 2, \ldots, n$ and "$||$" denotes the conjunction operator.

To make it more clearly, we present a simple example of the secure index construction, which is shown in Fig. 2. Suppose there are three hash functions, and the conjunction of keywords of data file $F_{i,j}$, denoted as $cascade_{i,j}$ that represents $w_{i,j,1} || w_{i,j,2} || \ldots || w_{i,j,|W_{i,j}|}$. Given three elements $r_{1,F_{i,j}}, r_{2,F_{i,j}}, r_{3,F_{i,j}}$, where $r_{3,F_{i,j}} = r_{1,F_{i,j}} \oplus r_{2,F_{i,j}} \oplus \prod_{k=1}^{|W_{i,j}|} g^{h(w_{i,j,k}) + q_1 \cdot sk_{i,j}}$, we can compute the inserted position for

$r_{t,F_{i,j}}$ by hashing $cascade_{i,j}$ with H_t $(t = 1, 2, 3)$. That is to compute $H_t(w_{i,j,1}||w_{i,j,2}||\ldots||w_{i,j,|W_{i,j}|})$. Because the $cascade_{i,j}$ is considered to be fixed for each data file with an independent ABF, the hash collision in our BPMS can be avoided during the secure index construction.

5.3 Query Trapdoor Generation

To protect query privacy with the search keyword set $\mathcal{Q}$, we should generate the trapdoor $\mathcal{T}(\mathcal{Q})$ in a secure manner against side channel attack.

Algorithm 2. *GenTrapdoor*

Input: the query keyword set $\mathcal{Q}$, two secret keys $s_u, r_u \in \mathbb{Z}_q^*$, the hash function h, these n hash functions $H_t,\ t = 1, 2, \ldots, n$, the parameter $q_2 \in \mathbb{Z}_q$

Output: the trapdoor $\mathcal{T}(\mathcal{Q})$

```
1: for each $w_k \in \mathcal{Q}$ do
2:     Compute $h(w_k)$;
3: end for
4: Compute $g^{s_u \cdot \sum_{k=1}^{|\mathcal{Q}|} h(w_k)}$;
5: for $t = 1$ to $n$ do
6:     Compute $H_t(w_1||w_2||\ldots||w_{|\mathcal{Q}|})$;
7: end for
8: Generate $T_1$ as $T_1 \leftarrow \{g^{s_u \cdot \sum_{k=1}^{|\mathcal{Q}|} h(w_k)}, (H_1(w_1||w_2||\ldots||w_{|\mathcal{Q}|}), \ldots, H_n(w_1||w_2||\ldots||w_{|\mathcal{Q}|}))\}$;
9: Compute $\hat{s}_u$ as $\hat{s}_u \cdot s_u = 1\ mod\ q$;
10: Compute $T_2$ as $T_2 \leftarrow g^{q_2 \cdot r_u \cdot (\hat{s}_u + 1)}$;
11: Compute $T_3$ as $T_3 \leftarrow g^{r_u \cdot q_2}$;
12: Get $\mathcal{T}(\mathcal{Q})$ as $\mathcal{T}(\mathcal{Q}) \leftarrow \{T_1, T_2, T_3\}$;
13: return $\mathcal{T}(\mathcal{Q})$
```

As shown in Algorithm 2, u first uses a temporary key $s_u \in \mathbb{Z}_q^*$ to compute $g^{s_u \cdot \sum_{k=1}^{|\mathcal{Q}|} h(w_k)}$. To indicate the positions for getting the elements in each index during search process while realizing query privacy preservation, u also computes n hash values $H_t(w_1||w_2||\ldots||w_{|\mathcal{Q}|}),\ t = 1, 2, \ldots, n$. To implement the query, u further chooses another temporary key $r_u \in \mathbb{Z}_q^*$ to compute $g^{q_2 \cdot r_u \cdot (\hat{s}_u + 1)}$ and $g^{r_u \cdot q_2}$, where $\hat{s}_u$ is the inverse of s_u such that $s_u \cdot \hat{s}_u = 1\ mod\ q$. Thus, inspired by [23], for the query keyword set $\mathcal{Q} = \{w_1, w_2, \ldots, w_{|\mathcal{Q}|}\}$, the corresponding trapdoor $\mathcal{T}(\mathcal{Q})$ is computed as

$$\mathcal{T}(\mathcal{Q}) = \begin{cases} T_1 = \{g^{s_u \cdot \sum_{k=1}^{|\mathcal{Q}|} h(w_k)}, (H_1(w_1||w_2||\ldots||w_{|\mathcal{Q}|}), \ldots, \\ \qquad H_n(w_1||w_2||\ldots||w_{|\mathcal{Q}|}))\} \\ T_2 = g^{q_2 \cdot r_u \cdot (\hat{s}_u + 1)} \\ T_3 = g^{r_u \cdot q_2} \end{cases}$$

5.4 Search on the Blockchain

Each node in the consortium blockchain runs the *Search* algorithm that can be implemented using smart contract to match the trapdoor with the index. The consensus committee adopts voting-based consensus mechanism to make the matching process trustworthy and efficient. Once the consensus is achieved, it will return the list $\mathcal{L}$ that contains $id(F_{i,j})$ to u, where $id(F_{i,j})$ is the CID of $F_{i,j}$ in the IPFS. Finally, according to $\mathcal{L}$, u requests decryption keys from DOs. Note that the *Search* algorithm only outputs $id(F_{i,j})$, the retrieval of encrypted files is conducted in the IPFS.

The process of search on the blockchain is presented in Algorithm 3. Upon receiving $\mathcal{T}(\mathcal{Q})$, given the index $\mathcal{I}_{F_{i,j}}$, the node first fetches the elements from $\mathcal{I}_{F_{i,j}}$ at the positions denoted by $H_t(w_1||w_2||\cdots||w_{|\mathcal{Q}|}),\ t = 1, 2, \ldots, n$ and computes $\prod_{k=1}^{|W_{i,j}|} g^{h(w_{i,j,k}) + q_1 \cdot sk_{i,j}}$ as

$$\prod_{k=1}^{|W_{i,j}|} g^{h(w_{i,j,k}) + q_1 \cdot sk_{i,j}} = r_{1,F_{i,j}} \oplus r_{2,F_{i,j}} \oplus \cdots \oplus r_{n,F_{i,j}}.$$

Subsequently, the node determines whether or not the following equality holds. That is,

$$\hat{e}\left(\prod_{k=1}^{|W_{i,j}|} g^{h(w_{i,j,k}) + q_1 \cdot sk_{i,j}}, T_3\right) \overset{?}{=} \hat{e}(T_1, T_2)/\hat{e}(T_1, T_3).$$

If the equality holds, then $F_{i,j}$ is the data file that u wants to search, and the node adds the CID of $F_{i,j}$ to the list $\mathcal{L}$. The correctness of the equality can be verified as follows.

Algorithm 3. *Search*

Input: the index $\mathcal{I}_{F_{i,j}}$, the trapdoor $\mathcal{T}(\mathcal{Q})$

Output: $id(F_{i,j})$

```
1: for $l = 1$ to $n$ do
2:     if $r_{l,F_{i,j}} = NULL$ then
3:         return False;
4:     end if
5: end for
6: Compute $\prod_{k=1}^{|W_{i,j}|} g^{h(w_{i,j,k}) + q_1 \cdot sk_{i,j}}$ as $r_{1,F_{i,j}} \oplus r_{2,F_{i,j}} \oplus \cdots \oplus r_{n-1,F_{i,j}} \oplus r_{n,F_{i,j}}$;
7: Compute $\hat{e}(\prod_{k=1}^{|W_{i,j}|} g^{h(w_{i,j,k}) + q_1 \cdot sk_{i,j}}, T_3)$;
8: Compute $\hat{e}(T_1, T_2)/\hat{e}(T_1, T_3)$;
9: if $\hat{e}(\prod_{k=1}^{|W_{i,j}|} g^{h(w_{i,j,k}) + q_1 \cdot sk_{i,j}}, T_3) = \hat{e}(T_1, T_2)/\hat{e}(T_1, T_3)$ then
10:     return $id(F_{i,j})$;
11: else
12:     return False;
13: end if
```

On the right-hand side, we can compute $\hat{e}(T_1, T_2)/\hat{e}(T_1, T_3)$ as

$$\begin{aligned} \hat{e}(T_1, T_2)/\hat{e}(T_1, T_3) &= \frac{\hat{e}(g^{s_u \cdot \sum_{k=1}^{|\mathcal{Q}|} h(w_k)}, g^{q_2 \cdot r_u \cdot (\hat{s}_u + 1)})}{\hat{e}(g^{s_u \cdot \sum_{k=1}^{|\mathcal{Q}|} h(w_k)}, g^{r_u \cdot q_2})} \\ &= \frac{\hat{e}(g^{\sum_{k=1}^{|\mathcal{Q}|} h(w_k)}, g^{q_2 \cdot r_u})^{s_u \cdot \hat{s}_u} \cdot \hat{e}(g^{s_u \cdot \sum_{k=1}^{|\mathcal{Q}|} h(w_k)}, g^{r_u \cdot q_2})}{\hat{e}(g^{s_u \cdot \sum_{k=1}^{|\mathcal{Q}|} h(w_k)}, g^{r_u \cdot q_2})} \\ &= \hat{e}(g^{\sum_{k=1}^{|\mathcal{Q}|} h(w_k)}, T_3). \end{aligned}$$

On the left-hand side, we can compute $\hat{e}(\prod_{k=1}^{|W_{i,j}|} g^{h(w_{i,j,k}) + q_1 \cdot sk_{i,j}}, T_3)$ as

$$\begin{aligned} &\hat{e}\left(\prod_{k=1}^{|W_{i,j}|} g^{h(w_{i,j,k}) + q_1 \cdot sk_{i,j}}, T_3\right) \\ &= \hat{e}(g^{\sum_{k=1}^{|W_{i,j}|} h(w_{i,j,k})} \cdot g^{\sum_{k=1}^{|W_{i,j}|} q_1 \cdot sk_{i,j}}, g^{r_u \cdot q_2}) \\ &= \hat{e}(g^{\sum_{k=1}^{|W_{i,j}|} h(w_{i,j,k})}, g^{r_u \cdot q_2}) \cdot \hat{e}(g^{\sum_{k=1}^{|W_{i,j}|} q_1 \cdot sk_{i,j}}, g^{r_u \cdot q_2}) \\ &= \hat{e}(g^{\sum_{k=1}^{|W_{i,j}|} h(w_{i,j,k})}, g^{r_u \cdot q_2}) \cdot \hat{e}(g, g)^{|W_{i,j}| \cdot q_1 \cdot sk_{i,j} \cdot r_u \cdot q_2} \\ &= \hat{e}(g^{\sum_{k=1}^{|W_{i,j}|} h(w_{i,j,k})}, g^{r_u \cdot q_2}) \\ &= \hat{e}(g^{\sum_{k=1}^{|W_{i,j}|} h(w_{i,j,k})}, T_3). \end{aligned}$$

Obviously, if $|\mathcal{Q}| = |W_{i,j}|$ and $\mathcal{Q}$ contains the same keywords as $W_{i,j}$, then the equality holds.

5.5 Search in the IPFS

After receiving $\mathcal{L}$ from the blockchain, the data user u sends it to any network node in the IPFS. To obtain an encrypted data file, the node first needs to determine which nodes that store the blocks corresponding to the data file. As we detailed in Section 3.2, the DHT is used for implementing this operation. That is, we can identify those nodes that contain the required blocks.

Whereafter, it is necessary to establish connections with these nodes where the blocks are stored. In the IPFS, a block exchange protocol called BitSwap that supports connecting nodes is used to request data from other nodes. The node can send the list $\mathcal{L}$ that contains $id(F_{i,j})$ to these connected nodes and obtain the blocks at once. It has been proved that the IPFS can provide a high throughput [31], which makes u get the required data file in an efficient manner.

6 Theoretical Analysis

In this section, we first analyze the security of BPMS and then theoretically compare our BPMS with those related state-of-the-art schemes.

6.1 Security Analysis

Our BPMS satisfies IND-CKA security under the random oracle model if both Theorem 1 and Theorem 2 hold.

Theorem 1. *The index of our BPMS is indistinguishable against adaptive chosen keyword attack under the random oracle model.*

Proof. We prove the security of index in our BPMS in $Game_1$. We assume that there is a PPT adversary who has a non-negligible advantage ϵ to break the index encryption algorithm, so we can also construct a simulator $\mathcal{B}$ who can solve the DDH problem with a non-negligible advantage. □

$Game_1$: $\mathcal{C}$ first randomly selects $\mu \in \{0,1\}$, if $\mu = 0$, $\mathcal{C}$ sets tuple $t_0 : (g, A = g^a, B = g^b, C = g^{ab})$, if $\mu = 1$, $\mathcal{C}$ sets tuple $t_1 : (g, A = g^a, B = g^b, C = g^c)$, where a, b, c are three elements randomly chosen from $\mathbb{Z}_p^*$. Then tuple t_μ is sent to $\mathcal{B}$, and $\mathcal{B}$ plays a game with the adversary $\mathcal{A}$ on behalf of the challenger $\mathcal{C}$.

- **Setup.** $\mathcal{B}$ runs $SystemSetup$ and sends public parameters to $\mathcal{A}$.
- **Phase 1.** $\mathcal{A}$ adaptively chooses some keyword sets and queries the index construction oracle to obtain the corresponding ciphertexts. Then $\mathcal{A}$ outputs two keyword sets W_0, W_1 and sends them to $\mathcal{B}$.
- **Challenge.** $\mathcal{B}$ randomly selects $\gamma \in \{0,1\}$ and encrypts W_γ as $\tau = g^{\sum_{k=1}^{|W_\gamma|} h(w_{\gamma,k})} \cdot C$. If $\mu = 0$, then $C = g^{ab}$. Since sk is a randomly chosen element, $|W_\gamma| \cdot sk \cdot q_1$ is also a random element. We let $ab = |W_\gamma| \cdot sk \cdot q_1$, thus $\tau = g^{\sum_{k=1}^{|W_\gamma|} h(w_{\gamma,k})} \cdot C = g^{\sum_{k=1}^{|W_\gamma|} h(w_{\gamma,k})}$. $g^{ab} = g^{\sum_{k=1}^{|W_\gamma|} h(w_{\gamma,k}) + |W_\gamma| \cdot sk \cdot q_1}$, where τ is a valid element of the index construction oracle. If $\mu = 1$, $C = g^c$, thus $\tau = g^{\sum_{k=1}^{|W_\gamma|} h(w_{\gamma,k}) + c}$. Since c is a random element, τ is a random element in $\mathbb{G}_1$ and contains no information about W_γ.
- **Phase 2.** $\mathcal{A}$ continues to use self-selected keyword sets to obtain corresponding indexes.
- **Guess.** $\mathcal{A}$ outputs the guess γ' of γ. If $\mathcal{C}$ chooses $\mu = 0$, which means $\mathcal{C}$ sends a valid tuple $t_0 : (g, A = g^a, B = g^b, C = g^{ab})$ to $\mathcal{B}$. Since $\mathcal{A}$ has a non-negligible advantage ϵ to break the index generation algorithm, the probability that the guess γ' of γ satisfying $\gamma' = \gamma$ is $\frac{1}{2} + \epsilon$ and the probability that the guess μ' of μ satisfying $\mu' = \mu = 0$ is $\frac{1}{2} + \epsilon$. If $\mathcal{C}$ chooses $\mu = 1$, which means $\mathcal{C}$ sends an invalid tuple $t_1 : (g, A = g^a, B = g^b, C = g^c)$ to $\mathcal{B}$. Thus, the probability that the guess γ' of γ satisfying $\gamma' = \gamma$ is $\frac{1}{2}$, and correspondingly the probability that the guess μ' of μ satisfying $\mu' = \mu = 1$ is $\frac{1}{2}$.

Based on the game, the advantage $Adv_{\mathcal{B}}^{DDH}$ that $\mathcal{B}$ solves the DDH problem is computed as

$$Adv_{\mathcal{B}}^{DDH} = \left| \frac{1}{2} Pr[\mu = \mu' | \mu = 0] + \frac{1}{2} Pr[\mu = \mu' | \mu = 1] - \frac{1}{2} \right| = \left| \left[\frac{1}{2}\left(\frac{1}{2} + \epsilon\right) + \frac{1}{2} \cdot \frac{1}{2} \right] - \frac{1}{2} \right| = \frac{\epsilon}{2}.$$

Therefore, if the adversary $\mathcal{A}$ has a non-negligible advantage ϵ to break the index generation algorithm, then the simulator $\mathcal{B}$ is able to solve the DDH problem with a non-negligible advantage $\frac{\epsilon}{2}$ which is contradict with the DDH assumption. Thus, the advantage $Adv_{\mathcal{A}}^{Index}$ that $\mathcal{A}$ wins the game satisfies $Adv_{\mathcal{A}}^{Index} = |Pr[\gamma' = \gamma] - \frac{1}{2}| \leq \zeta$ where ζ is a negligible probability, which means that the encryption for constructing index in our BPMS satisfies indistinguishability against adaptive chosen keyword attack under the random oracle model.

Theorem 2. *The trapdoor of our BPMS is indistinguishable against adaptive chosen keyword attack under the random oracle model.*

Proof. We prove the indistinguishability of trapdoor in our BPMS in $Game_2$, the security of trapdoor is based on the DLP assumption. □

$Game_2$: In the game, $\mathcal{A}$ is allowed to get access to $GenTrapdoor$ multiple times and obtain trapdoors according to self-selected query keyword sets.

- **Setup.** $\mathcal{C}$ runs $SystemSetup$ and sends public parameters to $\mathcal{A}$.
- **Phase 1.** $\mathcal{A}$ adaptively chooses some query keyword sets and queries the trapdoor generation oracle to issue corresponding trapdoors. Then $\mathcal{A}$ outputs two query keyword sets $\mathcal{Q}_0$ and $\mathcal{Q}_1$ and sends them to $\mathcal{C}$.
- **Challenge.** $\mathcal{C}$ randomly selects $\gamma \in \{0,1\}$ and encrypts $\mathcal{Q}_\gamma$ as $g^{s_u \cdot \sum_{k=1}^{|\mathcal{Q}_\gamma|} h(w_k)}$.
- **Phase 2.** With the same as **Phase 1**, $\mathcal{A}$ continues to get access to $GenTrapdoor$ and issue adaptively generated trapdoors based on different self-selected query keyword sets.
- **Guess.** $\mathcal{A}$ outputs the guess γ' of γ. Due to the DLP assumption, given the trapdoor $\mathcal{T}(\mathcal{Q}_\gamma)$ of $\mathcal{Q}_\gamma$, the query keywords cannot be recovered by

TABLE 2
Functionality Comparison With Related State-of-the-Art Schemes

Scheme	Trustworthy Search	Query Privacy	Multi-owner	Multi-keyword	IPFS Storage
MKSSMDO [23]	✗	✗	✓	✓	✗
Chen et al. [5]	✓	✗	✗	✓	✗
Hu et al. [26]	✓	✓	✗	✗	—
Li et al. [27]	✓	✓	✗	✗	✗
Jiang et al. [28]	✓	✓	✗	✓	✗
Cai et al. [29]	✓	✓	✗	✗	✗
Our BPMS	✓	✓	✓	✓	✓

analyzing $g^{s_u \cdot \sum_{k=1}^{|\mathcal{Q}_\gamma|} h(w_k)}$, the one-way property of h further ensures the privacy of query keywords. Thus, the probability that the guess γ' of γ satisfying $\gamma' = \gamma$ is $\frac{1}{2}$.

Based on the game, the advantage $Adv_{\mathcal{A}}^{Trapdoor}$ that $\mathcal{A}$ wins the game satisfies $Adv_{\mathcal{A}}^{Trapdoor} = |Pr[\gamma' = \gamma] - \frac{1}{2}| \leq \zeta$, where ζ is a negligible probability. Thus, the trapdoor generation in our BPMS satisfies indistinguishability against adaptive chosen keyword attack under the random oracle model.

6.2 Comparative Analysis

In terms of functionality and complexity, we make a comprehensive comparison between our BPMS and previous related schemes.

6.2.1 Functionality Comparison

We mainly compare our BPMS with those schemes [5], [23], [26], [27], [28], [29], focusing on trustworthy search, query privacy, multi-owner, multi-keyword and IPFS storage, as shown in Table 2.

As for trustworthy search, these schemes [5], [26], [27], [28], [29] introduce the blockchain to guarantee trustworthiness except MKSSMDO that relies on a centralized CS. For query privacy, MKSSMDO leaks the positions of keywords in dictionary $\mathcal{D}$ during the process of trapdoor generation. Chen et al. [5] proposed to use complex Boolean expressions to build indexes for supporting multi-keyword search. However, a DU needs to reveal query expressions to obtain trapdoors returned by a DO, which is undoubtedly an invasion of the DU's query privacy. Both MKSSMDO and our BPMS take multi-keyword search for multiple data owners into consideration. Those schemes [5], [26], [27], [28], [29] essentially consider single-owner model, which would have problems of complex key distribution and heavy communication overhead in multi-owner setting. Jiang et al. [28] proposed blockchain-based multi-keyword search, which improves the efficiency by narrowing down the searching space with a Bloom filter [38]. However, they would suffer from high communication and storage costs by directly uploading encrypted files to the blockchain. For the storage concerns, some schemes [5], [23], [27] store encrypted data files in a centralized model, which inevitably reduces the data availability. Cai et al. [29] proposed to achieve efficient encrypted keyword search over decentralized storage like Storj [1] and Sia [2] by preserving data locality. Hu et al. [26] only mentioned the use of IPFS storage while nothing is given. Only our BPMS presents how to enhance the data availability by using the IPFS storage.

1. https://www.storj.io
2. https://sia.tech

6.2.2 Complexity Comparison

We mainly make a complexity comparison between our BPMS and MKSSMDO [23], both of which are the type of multi-keyword multi-owner searchable encryption. Table 3 presents the comparison including computation cost and output size in predominant phases. It should be pointed out that the cost of randomly generating elements in $\mathbb{G}_1$ during index construction phase and the cost of multiplication in $\mathbb{G}_1$ during search phase are not included in MKSSMDO. Nevertheless, we have included them in the comparison to ensure fairness.

As for index construction, in MKSSMDO, $g^{h(w_{i,j,k}+q_1 \cdot sk_i)}$ is computed for each keyword which causes the computation cost of keywords encryption in a data file to be $|W_{i,j}|(h + A_{\mathbb{Z}_q} + E_{\mathbb{G}_1})$. In addition, $|\mathcal{D}| - |W_{i,j}|$ elements in $\mathbb{G}_1$ are also needed to be generated to pad the index. Besides the computation cost of keywords encryption, our BPMS needs $|W_{i,j}| - 1$ multiplications in $\mathbb{G}_1$, $n-1$ elements generation operations in $\mathbb{G}_1$ and $n-1$ XOR operations to generate $r_{n,F_{i,j}}$, $|W_{i,j}| - 1$ cascade operations and n hash operations of H in the ABF to insert elements. Apart from the time cost of keywords encryption, the remaining time cost of index in the two schemes are mainly affected by $R_{\mathbb{G}_1}$, which is the operation of randomly generating elements in $\mathbb{G}_1$. Compared with MKSSMDO, the time cost of $R_{\mathbb{G}_1}$ in our BPMS would be greatly reduced. That is because the number of operations needed only depends on the number of hash functions. As for trapdoor generation, in MKSSMDO, $|\mathcal{Q}|$ hash operations of h, $|\mathcal{Q}| - 1$ additions in $\mathbb{Z}_q$, one multiplication in $\mathbb{Z}_q$ and one exponentiation in $\mathbb{G}_1$ are needed for T_1 while two multiplications in $\mathbb{Z}_q$, two exponentiations in $\mathbb{G}_1$ and one addition in $\mathbb{Z}_q$ are required for the generation of T_2 and T_3. This phase in our BPMS is similar to MKSSMDO except that our BPMS requires additional $|\mathcal{Q}| - 1$ cascade operations and n hash operations of H. As for search, the CS in MKSSMDO needs three pairing operations, $|\mathcal{Q}| - 1$ multiplications in $\mathbb{G}_1$ and one multiplication in $\mathbb{G}_2$ to complete the phase while three pairing operations, $n-1$ XOR operations and one multiplications in $\mathbb{G}_2$ are required in our BPMS. The index size of MKSSMDO and our BPMS is $|\mathcal{D}||\mathbb{G}_1|$ and $n|\mathbb{G}_1|$, respectively. It is obvious that our BPMS has less storage cost of the index than that of MKSSMDO for $n \ll |\mathcal{D}|$ in practice. The trapdoor size of these two schemes is the same, which is $3|\mathbb{G}_1|$.

TABLE 3
Complexity Comparison of our BPMS With MKSSMDO [23]

Scheme	MKSSMDO [23]	BPMS
Computation cost of index	$\lvert W_{i,j}\rvert(h + A_{\mathbb{Z}_q} + E_{\mathbb{G}_1}) + M_{\mathbb{Z}_q} + (\lvert\mathcal{D}\rvert - \lvert W_{i,j}\rvert)R_{\mathbb{G}_1}$	$\lvert W_{i,j}\rvert(h + A_{\mathbb{Z}_q} + E_{\mathbb{G}_1}) + M_{\mathbb{Z}_q} + (\lvert W_{i,j}\rvert - 1)(Cas + M_{\mathbb{G}_1}) + (n-1)(XOR + R_{\mathbb{G}_1}) + nH$
Computation cost of trapdoor	$\lvert\mathcal{Q}\rvert(A_{\mathbb{Z}_q} + h) + 3(E_{\mathbb{G}_1} + M_{\mathbb{Z}_q})$	$\lvert\mathcal{Q}\rvert(A_{\mathbb{Z}_q} + h) + 3(E_{\mathbb{G}_1} + M_{\mathbb{Z}_q}) + (\lvert\mathcal{Q}\rvert - 1)Cas + nH$
Computation cost of search	$3P + (\lvert\mathcal{Q}\rvert - 1)M_{\mathbb{G}_1} + M_{\mathbb{G}_2}$	$3P + (n-1)XOR + M_{\mathbb{G}_2}$
Output size of index	$\lvert\mathcal{D}\rvert\lvert\mathbb{G}_1\rvert$	$n\lvert\mathbb{G}_1\rvert$
Output size of trapdoor	$3\lvert\mathbb{G}_1\rvert$	$3\lvert\mathbb{G}_1\rvert$

P is a bilinear pairing operation, $R_{\mathbb{G}_1}$ is an operation of randomly generating an element in $\mathbb{G}_1$, h is a hash operation and H is a hash operation in the ABF. $E_{\mathbb{G}_1}$ is an exponentiation operation in group $\mathbb{G}_1$, $A_{\mathbb{Z}_q}$, $M_{\mathbb{Z}_q}$ denote an addition operation and a multiplication operation in $\mathbb{Z}_q$, $M_{\mathbb{G}_1}$ and $M_{\mathbb{G}_2}$ denote a multiplication operation in group $\mathbb{G}_1$ and $\mathbb{G}_2$. XOR denotes an exclusive OR operation and Cas denotes a cascade operation. $\lvert\mathbb{G}_1\rvert$ denotes the element-size in $\mathbb{G}_1$.

7 Experimental Evaluation

In this section, we conduct comprehensive experiments to evaluate the practical performance of our BPMS and make a comparison with MKSSMDO [23] under various settings.

7.1 Evaluation Setup

Dataset. We leverage the real-world Enron email dataset [48] for evaluation. The raw dataset contains 619446 emails collected from 158 users, from which we randomly select 1000 data files as the experimental dataset, and then use Scikit-learn [49] to extract keywords that would be inserted into the dictionary $\mathcal{D}$ for each data file. For the sake of performance analysis on our BPMS, we set the number of keywords in a data file from 10 to 80 and the number of data files from 100 to 800, respectively.

Implementation. All experiments are implemented in Java by using JPBC library [50] configured on Ubuntu 16.04 desktop system with 3.20 GHz Intel Core (TM) i7-8700 and 8 GB RAM. We independently run every experiment 20 times to obtain the average result. The Type A1 elliptic curve used is $y^2 = x^3 + x$ and the length of q_1, q_2 is set to be 128 bits. The experiments on the blockchain are conducted on Hyperledger Fabric v1.4.4 [51], where 4 peers are configured, and the consensus algorithm is set as Raft [52]. The performance is tested by Hyperledger Caliper v1.4.0 [53].

7.2 Evaluation of Index Construction Algorithm

We conduct two kinds of experiments to evaluate the time cost of index construction algorithm $GenIndex$ in our BPMS under different number of hash functions and make the comparison with MKSSMDO. Specifically, given the fixed size of the dictionary $\mathcal{D}$ as 200, Fig. 3(a) shows the time cost of generating index with varying the number of keywords in a data file and Fig. 3(b) depicts the time cost of secure index generation for different number of data files with the fixed number of keywords as 50 in a data file. It should be pointed out the time cost of MKSSMDO is also affected by the size of $\mathcal{D}$ while our BPMS is not. The time cost of randomly generating elements in $\mathbb{G}_1$ is not included in MKSSMDO. When $\mathcal{D}$ is larger, it would be considerable. Here we have included this part of time cost in the comparison.

As shown in Fig. 3(a), the time cost of index construction in our BPMS linearly increases when the number of keywords contained in a data file varies from 10 to 80. That is because a hash operation of h, an addition in $\mathbb{Z}_q$ and an exponentiation in $\mathbb{G}_1$ have to be performed for each keyword, the time taken by cascade operation and multiplication in $\mathbb{G}_1$ also depends on the number of keywords. When the number of hash functions is set of 5, 10 and 15, the time cost of BPMS grows with the increasing number of hash functions. The reason is that each hash function would be executed in BPMS to compute the position for element insertion. Compared with MKSSMDO, given the fixed size of $\mathcal{D}$ as 200, our BPMS takes considerable less time even when the number of hash functions increases to 15. That is because in our BPMS, the time cost of randomly generating elements in $\mathbb{G}_1$ is greatly reduced. The time cost of MKSSMDO does not vary with the number of keywords. Because although more encryption operations are required as the number of keywords increases, fewer operations of generating elements in $\mathbb{G}_1$ to pad the index are needed. Fig. 3(b) further illustrates that the time cost linearly grows with the increasing number of data files, where the fixed number of keywords in a data file is 50 and the fixed size of $\mathcal{D}$ is 200. Given a fixed number of data files, the time cost of our BPMS increases with the number of hash functions while is less than that of MKSSMDO. That is because the operations for index construction that depend on a data file would be repeated with varying the number of data files.

7.3 Evaluation of Trapdoor Generation Algorithm

We perform two kinds of experiments to evaluate the time cost of secure trapdoor generation algorithm $GenTrapdoor$ under the same hash function settings in our BPMS and make the comparison with MKSSMDO. Fig. 4(a) shows the time cost of generating trapdoor with varying the number

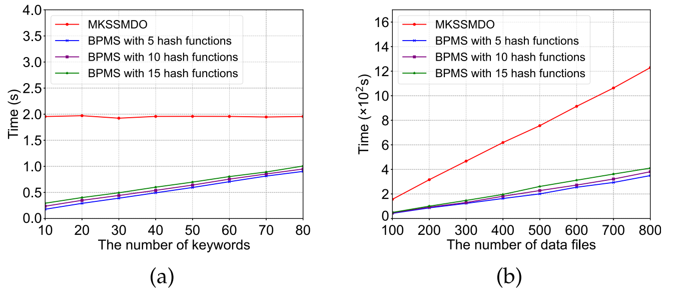

Fig. 3. Performance of $GenIndex$. (a) The time cost of secure index generation for different number of keywords in a data file. (b) The time cost of secure index generation for different number of data files with the fixed number of keywords as 50 in a data file.

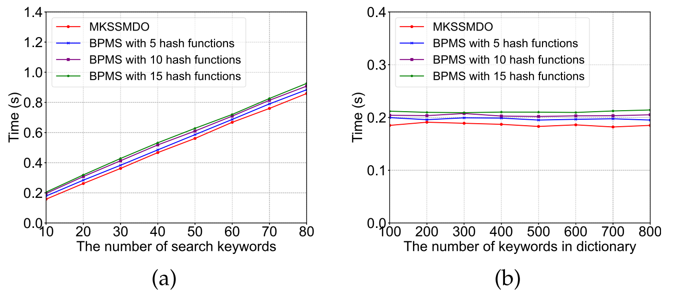

Fig. 4. Performance of $GenTrapdoor$. (a) The time cost of trapdoor generation for different number of search keywords. (b) The time cost of trapdoor generation for different size of dictionary $\mathcal{D}$ with the fixed number of search keywords as 12.

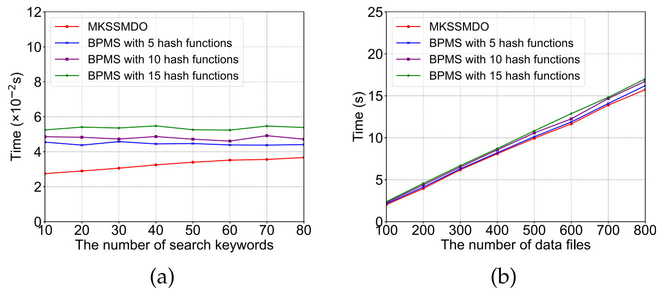

Fig. 5. Performance of $Search$. (a) The time cost of search with varying the number of search keywords in a data file. (b) The time cost of search for different number of data files with the fixed number of search keywords as 50.

of search keywords and Fig. 4(b) depicts the time cost of generating trapdoor with varying the size of $\mathcal{D}$, where the number of search keywords is fixed as 12. It should be noted that the implementation of MKSSMDO only considers the time taken by exponentiation in $\mathbb{G}_1$ while ignoring the time cost of the hash operation h for each keyword. Here, we have taken the time cost of all operations in trapdoor generation into consideration.

As shown in Fig. 4(a), the time cost of our BPMS linearly increases with the number of search keywords that varies from 10 to 80. That is because $|\mathcal{Q}|$ addition operations in $\mathbb{Z}_q$, $|\mathcal{Q}|$ hash operations of h and $|\mathcal{Q}| - 1$ cascade operations are required for each search in our BPMS. It can be seen that the time cost slightly grows with the increasing number of hash functions. The reason is that these hash functions would be only used to compute the positions for obtaining the elements in the index. Compared with MKSSMDO, our BPMS takes a little more time. That is because our BPMS requires additional $|\mathcal{Q}| - 1$ cascade operations and n hash operations of H to generate trapdoor. As illustrated in Fig. 4(b), we can observe that the time costs of our BPMS and MKSSMDO are nearly constant and do not vary with the size of $\mathcal{D}$. This is due to the fact that the time cost is only affected by $|\mathcal{Q}|$ as analyzed above. Moreover, the time cost of our BPMS is very slow to grow with the increasing number of hash functions. In general, the time cost of trapdoor generation in our BPMS is almost the same with MKSSMDO.

7.4 Evaluation of Search Algorithm

We conduct two kinds of experiments to evaluate the time cost of search algorithm $search$ in our BPMS under the same function settings in our BPMS and make the comparison with MKSSMDO. Fig. 5(a) shows the time cost of search with varying the number of search keywords in a data file and Fig. 5(b) illustrates the time cost of search with varying the number of data files, where the number of search keywords is fixed as 50.

As we can see from Fig. 5(a), the time cost of our BPMS is not affected by the number of search words. It grows slowly with the increasing number of hash functions. The reason is that besides a multiplication and three pairing operations, our BPMS also needs additional $n - 1$ XOR operations that are related to the number of hash functions. Compared with MKSSMDO, given a fixed number of search keywords, our BPMS takes only a little more time when the number of hash functions is respectively set to be 5, 10, 15. The reason is that the time cost of XOR operations is affected by the number of elements from the index. Obviously, the more the number of hash functions is, the more time cost it takes. As shown in Fig. 5(b), because the search operations of MKSSMDO and our BPMS are implemented in a data file, given the fixed number of search keywords as 50, it is obvious that the time cost of which takes is linear to the number of data files. We also observe that the difference of time cost between MKSSMDO and our BPMS in a data file is very limited, which makes the total time cost be close with varying the number of data files.

7.5 Evaluation of Operations on the Blockchain

In our BPMS, the $Search$ algorithm implemented by using smart contract is run on the blockchain. To quantify the throughput and average latency of the $Search$ algorithm, we set the search send rate of 100 transactions per second (TPS). Fig. 6(a) shows the throughput of $Search$ with varying the number of search keywords in a data file and Fig. 6(b) depicts the average latency of $search$ with different number of search keywords in a data file.

As shown in Fig. 6(a) and Fig. 6(b), we can observe that the performance of $Search$ in terms of throughput and average latency does not vary with the increasing number of search keywords in a data file. That is because both the throughput and average latency are mainly affected by the used consensus algorithm. According to the analysis in Section 7.4, the time cost of search in a data file is independent of the number of search keywords, which means it is relatively stable. Therefore, once the blockchain has been configured, both the throughput and average latency will also not be affected with varying the number of search keywords

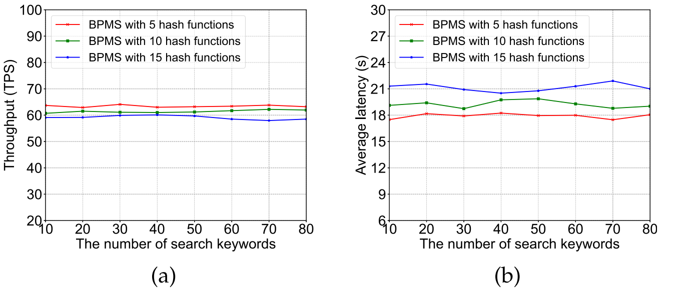

Fig. 6. Performance of $Search$ on the blockchain. (a) The transaction throughput for different number of search keywords. (b) The average latency with different number of search keywords.

in a data file. Moreover, as illustrated by Fig. 6(a), we can see that the throughput of our BPMS slightly decreases with the increasing number of hash functions. That is because the more the number of hash functions is, the more the time cost of search in a data file takes. For the same reason, the average latency of our BPMS slightly increases with the increasing number of hash functions, as shown in Fig. 6(b). All these results also further demonstrate that the higher the throughput is, the lower the average latency is.

8 Conclusion

In this paper, we have proposed BPMS, a blockchain-based privacy-preserving multi-keyword search scheme, which supports trustworthy search in multi-keyword multi-owner setting. Our BPMS exploits the attribute Bloom filter to guarantee query privacy in an efficient manner. By introducing IPFS in our BPMS, we can achieve more secure and efficient storage, which improves the data availability and practical feasibility. Theoretical analysis and experimental evaluation demonstrate that our BPMS can enrich the functionality and enhance the privacy and efficiency.

References

[1] S. Gao et al., "TrPF: A trajectory privacy-preserving framework for participatory sensing," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 6, pp. 874–887, Jun. 2013.

[2] Z. Xiao and Y. Xiao, "Security and privacy in cloud computing," *IEEE Commun. Surv. Tut.*, vol. 15, no. 2, pp. 843–859, Apr.–Jun. 2013.

[3] C. Bösch et al., "A survey of provably secure searchable encryption," *ACM Comput. Surv.*, vol. 47, no. 2, pp. 1–51, 2015.

[4] R. Zhang, R. Xue, and L. Liu, "Searchable encryption for healthcare clouds: A survey," *IEEE Trans. Services Comput.*, vol. 11, no. 6, pp. 978–996, Nov./Dec. 2018.

[5] L. Chen et al., "Blockchain based searchable encryption for electronic health record sharing," *Future Gener. Comput. Syst.*, vol. 95, pp. 420–429, 2019.

[6] K. Fan et al., "MSIAP: A dynamic searchable encryption for privacy-protection on smart grid with cloud-edge-end," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2021.3134015.

[7] L. Wu et al., "Efficient and secure searchable encryption protocol for cloud-based Internet of Things," *J. Parallel Distrib. Comput.*, vol. 111, pp. 152–161, 2018.

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

[9] R. Curtmola et al., "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. 13th ACM Conf. Comput. Commun. Secur.*, 2006, pp. 79–88.

[10] G. S. Poh et al., "Searchable symmetric encryption: Designs and challenges," *ACM Comput. Surv.*, vol. 50, no. 3, pp. 1–37, 2017.

[11] D. Boneh et al., "Public key encryption with keyword search," in *Proc. Int. Conf. Theory Appl. Cryptographic Techn.*, 2004, pp. 506–522.

[12] R. Chen et al., "Server-aided public key encryption with keyword search," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 12, pp. 2833–2842, Dec. 2016.

[13] B. Wang et al., "Inverted index based multi-keyword public-key searchable encryption with strong privacy guarantee," in *Proc. IEEE Conf. Comput. Commun.*, 2015, pp. 2092–2100.

[14] J. W. Byun et al., "Off-line keyword guessing attacks on recent keyword search schemes over encrypted data," in *Proc. 3rd VLDB Workshop Secure Data Manage.*, 2006, pp. 75–83.

[15] C. Wang et al., "Secure ranked keyword search over encrypted cloud data," in *Proc. 30th Int. Conf. Distrib. Comput. Syst.*, 2010, pp. 253–262.

[16] L. Ballard, S. Kamara, and F. Monrose, "Achieving efficient conjunctive keyword searches over encrypted data," in *Proc. Int. Conf. Inf. Commun. Secur.*, 2005, pp. 414–426.

[17] N. Cao et al., "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 1, pp. 222–233, Jan. 2014.

[18] B. Wang et al., "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud," in *Proc. IEEE Conf. Comput. Commun.*, 2014, pp. 2112–2120.

[19] H. Yin et al., "A query privacy-enhanced and secure search scheme over encrypted data in cloud computing," *J. Comput. Syst. Sci.*, vol. 90, pp. 14–27, 2017.

[20] P. Chaudhari and M. L. Das, "Privacy preserving searchable encryption with fine-grained access control," *IEEE Trans. Cloud Comput.*, vol. 9, no. 2, pp. 753–762, Apr.–Jun. 2021.

[21] W. Zhang et al., "Privacy preserving ranked multi-keyword search for multiple data owners in cloud computing," *IEEE Trans. Comput.*, vol. 65, no. 5, pp. 1566–1577, May 2016.

[22] Z. Guo et al., "Secure multi-keyword ranked search over encrypted cloud data for multiple data owners," *J. Syst. Softw.*, vol. 137, pp. 380–395, 2018.

[23] H. Yin et al., "Secure conjunctive multi-keyword ranked search over encrypted cloud data for multiple data owners," *Future Gener. Comput. Syst.*, vol. 100, pp. 689–700, 2019.

[24] S. Gao et al., "T-pbft: An eigentrust-based practical byzantine fault tolerance consensus algorithm," *China Commun.*, vol. 16, no. 12, pp. 111–123, 2019.

[25] S. Gao et al., "Trustaccess: A trustworthy secure ciphertext-policy and attribute hiding access control scheme based on blockchain," *IEEE Trans. Veh. Technol.*, vol. 69, no. 6, pp. 5784–5798, Jun. 2020.

[26] S. Hu et al., "Searching an encrypted cloud meets blockchain: A decentralized, reliable and fair realization," in *Proc. IEEE INFOCOM*, 2018, pp. 792–800.

[27] H. Li et al., "Blockchain-based searchable symmetric encryption scheme," *Comput. Elect. Eng.*, vol. 73, pp. 32–45, 2019.

[28] S. Jiang et al., "Privacy-preserving and efficient multi-keyword search over encrypted data on blockchain," in *Proc. IEEE Int. Conf. Blockchain*, 2019, pp. 405–410.

[29] C. Cai et al., "Enabling reliable keyword search in encrypted decentralized storage with fairness," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 1, pp. 131–144, Jan./Feb. 2021.

[30] K. Yang et al., "An efficient and fine-grained Big Data access control scheme with privacy-preserving policy," *IEEE Internet Things J.*, vol. 4, no. 2, pp. 563–571, Apr. 2016.

[31] J. Benet, "IPFS - content addressed, versioned, p2p file system (draft 3)," 2014. [Online]. Available: https://raw.githubusercontent.com/ipfs-inactive/papers/master/ipfs-cap2pfs/ipfs-p2p-file-system.pdf

[32] E.-J. Goh, "Secure indexes," *Cryptology ePrint Archive*, Tech. Rep. 2003/216, 2003. [Online]. Available: http://eprint.iacr.org/2003/216/

[33] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over-encrypted data," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur.*, 2004, pp. 31–45.

[34] D. Cash et al., "Highly-scalable searchable symmetric encryption with support for boolean queries," in *Proc. 33rd Annu. Int. Cryptol. Conf.*, 2013, pp. 353–373.

[35] W. Sun et al., "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," in *Proc. 8th ACM SIGSAC Symp. Inf. Comput. Commun. Secur.*, 2013, pp. 71–82.

[36] X. Ding, P. Liu, and H. Jin, "Privacy-preserving multi-keyword top-k similarity search over encrypted data," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 2, pp. 344–357, Mar./Apr. 2019.

[37] S. Tahir and M. Rajarajan, "Privacy-preserving searchable encryption framework for permissioned blockchain networks," in *Proc. iThings GreenCom CPSCom SmartData*, 2018, pp. 1628–1633.

[38] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[39] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Proc. 21st Annu. Int. Cryptol. Conf.*, 2001, pp. 213–229.

[40] K. S. McCurley, "The discrete logarithm problem," in *Proc. Symposia Appl. Math.*, 1990, pp. 49–74.

[41] D. Boneh, "The decision diffie-hellman problem," in *Proc. Int. Algorithmic Number Theory Symp.*, 1998, pp. 48–63.

[42] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[43] Q. Zheng et al., "An innovative IPFS-based storage model for blockchain," in *Proc. IEEE/WIC/ACM Int. Conf. Web Intell.*, 2018, pp. 704–708.

[44] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the XOR metric," in *Proc. Int. Workshop Peer-to-Peer Syst.*, 2002, pp. 53–65.

[45] M. Steichen et al., "Blockchain-based, decentralized access control for IPFS," in *Proc. IEEE Int. Conf. Internet Things (iThings) IEEE Green Comput. Commun. (GreenCom) IEEE Cyber, Phys. Social Comput. IEEE Smart Data*, 2018, pp. 1499–1506.

[46] S. Gao et al., "TrustWorker: A trustworthy and privacy-preserving worker selection scheme for blockchain-based crowdsensing," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2021.3103938.

[47] S. Gao et al., "A privacy-preserving identity authentication scheme based on the blockchain," *Secur. Commun. Netw.*, vol. 2021, 2021, Art. no. 9992353.

[48] B. Klimt and Y. Yang, "The enron corpus: A new dataset for email classification research," in *Proc. Eur. Conf. Mach. Learn.*, 2004, pp. 217–226.

[49] F. Pedregosa et al., "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[50] A. De Caro and V. Iovino, "jPBC: Java pairing based cryptography," in *Proc. IEEE Symp. Comput. Commun.*, 2011, pp. 850–855.

[51] E. Androulaki et al., "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proc. 13th EuroSys Conf.*, 2018, pp. 1–15.

[52] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proc. USENIX Annu. Tech. Conf.*, 2014, pp. 305–319.

[53] "Hyperledger Caliper." Accessed: Sep. 2021. [Online]. Available: https://github.com/hyperledger/caliper

Sheng Gao received the PhD degree in computer science and technology from Xidian University, Xi'an, China, in 2014. He is now a professor and the PhD supervisor with the School of Information, Central University of Finance and Economics. He has authored or coauthored more than 5 books and published more than 50 papers in refereed international journals and conferences, such as *IEEE Transactions on Information Forensics and Security*, *IEEE Transactions on Vehicular Technology*, *IEEE Transactions on Services Computing*, and *IEEE Transactions on Wireless Communications*. His current research interests include blockchain, data security, and privacy computing.

Yuqi Chen received the BE degree from the School of Information Science and Technology from University of International Relations, Beijing, China, in 2020. Currently, she is working toward the MS degree with the School of Information, Central University of Finance and Economics. Her current research focuses on searchable encryption, privacy protection and blockchain.

Jianming Zhu received the PhD degree in computer application technology from Xidian University, Xi'an, China, in 2004. He is working as a professor with the School of Information, Central University of Finance and Economics. From September 2008 to March 2009, he was a research fellow with the University of Texas at Dallas, USA. He has published more than 100 research papers in refereed international journals and conferences. His research interests include wireless network security, data privacy and blockchain.

Zhiyuan Sui received the bachelor's degree in mathematics form Shandong Jianzhu University, Ji'nan, China in 2008, the master's degree in computer science from Xidian University, Xi'an, China in 2011, and the PhD degree from Passau University, Germany in 2018. He is currently working as a lecturer with the Central University of Finance and Economics, Beijing, China. His research interests include applied cryptography, privacy preservation, and security in critical infrastructure.

Rui Zhang received the PhD degree in information security from Beijing Jiaotong University, Beijing, in 2011. She is currently an associate researcher with the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences. Her research interests include cloud data security, privacy preservation, and security protocols.

Xindi Ma (Member, IEEE) received the BS degree from the school of computer science and technology, Xidian University in 2013 and the PhD degree in computer science from Xidian University in 2018. He is now an associate professor with the School of Cyber Engineering, Xidian University. His current research interests include privacy computing, recommender system and machine learning with focus on security and privacy issues.