SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

March 2026, Vol. 69, Iss. 3, 132101:1–132101:18
https://doi.org/10.1007/s11432-024-4409-6

Blockchain-assisted multi-keyword searchable provable data possession for cloud storage

Ying MIAO, Keke GAI* & Liehuang ZHU

School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing 100081, China

Received 20 December 2024/Revised 19 February 2025/Accepted 23 April 2025/Published online 4 January 2026

Abstract Keyword searchable provable data possession (PDP) is a type of schemes that aim to achieve a higher level capability in auditing, as most existing PDP schemes fail in having satisfied ability in auditing integrity of data containing the same types of properties. However, these schemes encounter a few challenges. One challenge is that files that users need to auditing will be lost since using single keywords may fail in achieving an accurate search. The other challenge is the integrity of the outsourced data is threatened by those attacks targeting at the Cloud Server (CS) side, such as the integrity loss of search results, pre-matching attacks, and pre-computation attacks. In this paper, we propose a scheme entitled multi-keyword searchable PDP (MKPDP) that aims to improve the accuracy of searchable files auditing and guarantee the integrity of data containing a specific kind of properties that participate in the auditing process. Specifically, we have developed an index matrix combining with a tag index to achieve multi-keyword files searchability. A blockchain-based data storage is constructed to store index information, which ensures the integrity of the searching files and prevents CS from using incomplete searching results to generate proof information. Our security analysis and evaluations have demonstrated that the proposed MKPDP is provably secure and efficient.

Keywords multi-keyword searchability, provable data possession, blockchain, cloud storage

Citation Miao Y, Gai K K, Zhu L H. Blockchain-assisted multi-keyword searchable provable data possession for cloud storage. Sci China Inf Sci, 2026, 69(3): 132101, https://doi.org/10.1007/s11432-024-4409-6

1 Introduction

Cloud services enable data owners and users to access data remotely without geographical restrictions, thereby facilitating data-driven collaboration and application development [1,2]. Key advantages of cloud computing include cost-effective storage and scalable capacity, which have made it a widely adopted solution for both individuals and organizations [3]. However, despite these benefits, security concerns remain, particularly regarding the integrity and completeness of data stored remotely. Data loss can occur due to unexpected events such as hardware failures or malicious attacks, and such incidents have become increasingly common in recent years. To address the challenge of verifying the integrity of remotely stored data, the concept of provable data possession (PDP) was introduced [4,5].

PDP provides an efficient mechanism that allows users to verify the integrity of their cloud-stored data without downloading the entire dataset [6,7]. In a typical PDP protocol, a data owner (DO) divides a file into multiple blocks and generates an authenticator for each block. The file and its authenticators are then uploaded to the cloud. During verification, the DO (or a designated third party auditor, TPA) randomly selects a subset of blocks, issues a challenge to the cloud server (CS), and verifies the server's response to determine whether the data remains intact. Over time, various PDP schemes have been developed to suit different contexts-supporting single [8] or multiple users [9–11], handling single copy data [12], and multiple copy data [13], privacy protection in the auditing process [14–16], auditing for dynamic data [17,18], and auditing from centralized system to distributed storage system [19,20].

As organizations increasingly migrate to the cloud, ensuring the integrity of data with specific properties becomes more critical [21]. For example, in the medical field, the integrity of research-relevant datasets is essential for disease studies. Medical databases are typically large and dynamic, containing diverse data from numerous patients. Similarly, biological databases store extensive information such as genome sequences, protein structures, and gene expression profiles, all systematically organized for convenient access by researchers. In many cases, researchers are only interested in the integrity of data related to a particular disease or condition—not the entire dataset.

* Corresponding author (email: gaikeke@bit.edu.cn)

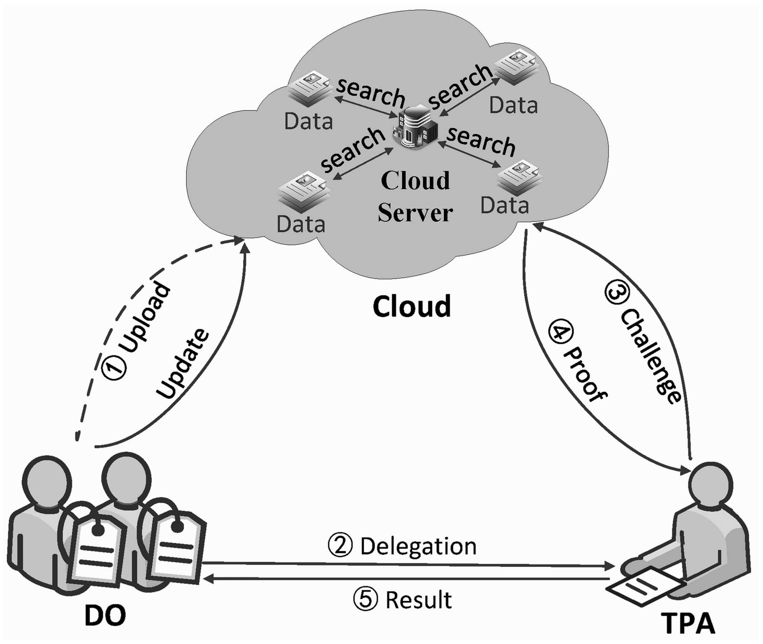

Figure 1 Sketch of keyword searchable PDP model.

However, they may not know the exact size or structure of the relevant subset within the evolving database. Keyword-searchable PDP has emerged as a promising approach to address this need.

In recent years, researchers have proposed keyword searchable PDP schemes [22–24]. As shown in Figure 1, these schemes have keywords to construct indexes that match authenticators. Instead of requiring explicit file identifiers, the CS retrieves matching files via a trapdoor mechanism, allowing the TPA to verify the integrity of all files associated with a given keyword. However, existing schemes often suffer from two major limitations: (1) they do not support dynamic file operations, meaning index structures become invalid when files are updated, and (2) they support only single-keyword searches, which reduces audit precision and efficiency, and may cause relevant files to be overlooked. To address these shortcomings, there is a need for multi-keyword searchable PDP schemes that enable more accurate and flexible integrity auditing. However, designing such schemes poses several significant security challenges.

(1) Integrity of search results. The TPA and users are unaware of the exact number of relevant files in the database. How can the TPA verify the integrity of returned results and detect any omitted files?

(2) Preventing pre-matching attacks. In some schemes, the CS can match authenticators to indexes in advance. In scheme [22, 23], one part of the proof information is $T = \prod_{i\in S_{w_k}} \prod_{j\in Q} \sigma_{i,j}^{v_j} \cdot \prod_{j\in Q} \Omega_{w_k,j}^{v_j}$, where S_{w_k} is the set searched by the CS to be audited, Q is the set of challenge data blocks, v_j is a random value chosen by the TPA, $\sigma_{i,j}$ is the authenticator, and $\Omega_{w_k,j}^{v_j}$ is the index. However, we find this method is not safe. The equation also holds $T = \prod_{i\in S_{w_k}} \prod_{j\in Q} \sigma_{i,j}^{v_j} \cdot \prod_{j\in Q} \Omega_{w_k,j}^{v_j} = \sum_{j\in Q}((\prod_{i\in S_{w_k}}(\sigma_{i,j}) \cdot \Omega_{w_k,j}))^{v_j}$, which means that given the set S_{w_k}, the CS can match the index and authenticator ahead of time. This undermines full verification and allows the CS to reuse stored intermediate values, even if the indexes are privacy-preserving but structurally static. With just one audit, the CS can get the specific content index of S_{w_k}. Then, the CS matches the index and authenticator based on the set S_{w_k} ahead of time, which can be represented as $\{(\prod_{i\in S_{w_k}} \sigma_{i,j}) \cdot \Omega_{w_k,j}\}_{w_k\in W, j\in[1,n]}$, and discards the index $\{\Omega_{w_k,j}\}_{w_k\in W, j\in[1,n]}$ and the authenticator $\{\sigma_{i,j}\}_{i\in[1,v], j\in[1,n]}$, where W is the keyword set, n is the total number of separated data blocks, and v is the number of total files. In this way, the CS saves storage overhead and can pass verification.

(3) Preventing proof pre-aggregation. Some schemes allow CS to pre-aggregate proof components across files or blocks. In schemes [22–24], another part of the proof information is $\mu = \sum_{i\in S_{w_k}} \sum_{j\in Q} c_{ij} \cdot v_j$, where $c_{i,j}$ is the challenged data blocks. However, the method is not safe as well. The equation can be derived as $\mu = \sum_{i\in S_{w_k}} \sum_{j\in Q} c_{ij} \cdot v_j = \sum_{j\in Q} v_j(\sum_{i\in S_{w_k}} c_{ij})$, which means that given the set $\omega_{w_k,j}$, the CS can pre-aggregating stored files by $\sum_{i\in S_{w_k}} c_{ij}$. The CS only needs to store the pre-aggregating value, which can be represented as $\{\sum_{i\in S_{w_k}} c_{ij}\}_{w_k\in W, j\in[1,n]}$, to pass the verification.

To overcome the limitations of prior work, we leverage the inherent advantages of blockchain, namely openness, transparency, immutability, and traceability [1, 20, 25, 26], and make the following key contributions.

(1) To improve the accuracy of searchable file auditing and guarantee the integrity of data containing specific properties, we design a multi-keyword searchable PDP. Specifically, we analyze the security issues in existing keyword-searchable PDP schemes, focusing on search result integrity and the risks of pre-matching and pre-computation attacks by the CS. We identify specific vulnerabilities in these schemes and find that they cannot resist pre-matching and pre-computation attacks.

(2) To improve the integrity of the searching files, we employed blockchain to store index information and guarantee the integrity of searching files. Both the CS and the TPA participate in searching and matching. Once the CS does not utilize the complete files to generate the proof information, the results cannot pass verification from the TPA. To overcome the problems of matching and calculation in advance by the CS, the TPA participates in matching and conducts verification. To prevent the CS from pre-aggregating stored files, we ensure that every challenged file and data blocks are involved in generating proof by including random numbers.

(3) We provide concrete security analysis for correctness, multi-keyword searchability, data privacy, keyword privacy, and file dynamics. Theoretical and performance analysis demonstrated that the proposed multi-keyword searchable PDP does not incur much cost in the authenticator and index generation phase while maintaining safety.

The remainder of the paper is organized as follows. Sections 2 and 3 present related work and preliminaries, respectively. Section 4 explains the system model and security model. We provide the specific construction of MKPDP in Section 5. We conduct security analysis, theoretical analysis, and performance evaluation in Sections 6–8, respectively. Finally, the discussion and conclusion are presented in Sections 9 and 10.

2 Related work

2.1 Privacy protection in provable data possession

Data privacy protection. To protect data privacy, many schemes employ masking to blind the proof information, thereby preventing the TPA from retrieving data block information during the auditing process [6]. Recently, Huang et al. [8] employed random masking technology to obscure the auditing proof. Additionally, they used permutations to mask the file index, ensuring audit frequency and data privacy. Wang et al. [9] adopted ciphertext-policy attribute-based encryption (CP-ABE) for privacy-preserving in smart health applications. Their approach encompassed a policy-hiding method, effectively ensuring privacy security during cloud auditing for smart health. Liu et al. [19] used a method for distributed and privacy-preserving data integrity auditing in 5G-enabled software-defined edge computing. Their approach involved blinding the proof information to improve privacy.

Data owner's identity privacy protection. To safeguard the identity of the DO, numerous schemes use anonymous identity techniques. Li et al. [12] adopted blind technology and anonymous identity to guarantee the privacy of IoT data. Their method incorporated an algebraic signature and homomorphic hash functions to enhance auditing efficiency. Gai et al. [14] used random masking technology to obscure the auditing proof and safeguarded DO identity privacy through the private key generator. Zhang et al. [10] offered diverse forms of identity anonymity while permitting de-anonymity in cases of malicious behavior. They employed vector commitment to enable data integrity verification and privacy protection. Moreover, some schemes safeguard DO privacy through the distinctive properties of signatures. Ring signatures enable users to maintain anonymity during signature operations; the signer's identity is not directly disclosed. Tian et al. [15] used a ring signature based on the learning with errors problem to achieve post-quantum security. Qi et al. [27] designed a linkable homomorphic authenticable ring signature to achieve anonymity of group data owners. Zero-sum set (ZSS) signature protects the signer's identity and the confidentiality of signature content. Goswami et al. [16] employed ZSS signatures as authenticators to safeguard data privacy. Additionally, it used faked data recovery techniques to restore original block elements at the block level.

2.2 Data dynamic technique in provable data possession

To achieve data dynamic operations, various methods were employed at the block level, including deletion, insertion, and modification [28]. Shen et al. [17] adopted a double link table and a location array to achieve dynamic management. Guo et al. [18] employed an implicitly-indexed balanced Merkle tree to attain dynamic proof of data possession. A single authenticated tree was shared among multiple replicas, enabling batch auditing of multiple replicas within a single time auditing. Xu et al. [29] used an extended double-linked list information table to enable data dynamic management. Li et al. [11] utilized dynamic group cooperation files, where any user within the group could monitor the identity of the signer and timestamps based on the file's state information. Peng et al. [30]

embedded multi-set hash functions into data tags to enable dynamic auditing. They designed a multi-functional data tag aimed at achieving data deduplication. Guo et al. [13] employed a modified Merkle hash tree, consolidating all copies of a raw block into the same leaf node. This approach allowed multi-copy data auditing and dynamic data management. Moreover, they designed a key generation and update strategy to enable group user revocation. Liu et al. [31] employed a quad Merkle hash tree to allow dynamic management and enhance computing and storage efficiency. In addition, they employed blockchain technology to ensure auditing reliability.

2.3 Blockchain technique in provable data possession

To improve transparency and decentralization, blockchain technology played a pivotal role in data integrity auditing [7]. Zhang et al. [32] used message-locked encryption to achieve data deduplication. Their approach utilized smart contracts to establish user ownership and facilitate automatic off-chain data integrity auditing. Song et al. [33] attained simultaneous low-entropy data privacy, deduplication, and integrity. Their approach utilized blockchain technology to facilitate key recoverability and limit local key storage costs. Zhang et al. [34] used polynomial commitment to enable multiple copies auditing for a single file. They enhanced a batch auditing scheme across multiple copies of multiple files, leveraging blockchain to assist the auditing work. Liu et al. [35] introduced the concept of auditing and file frequency prediction in group-sharing data. Their approach used blockchain to ensure the traceability and tamper-resistance of auditing information. Li et al. [20] employed vector commitment technology to achieve lightweight auditing. Their approach utilized smart contracts to limit privacy leakage and performance risks associated with the TPA. Li et al. [25] utilized hash operations to lower costs and achieved auditing in an edge computing environment with potentially malicious nodes. Their approach leveraged blockchain technology to replace the function of the TPA. Liu et al. [26] utilized an Nary tag commitment tree to integrate all information into the authenticator, allowing block-level deduplication. Their approach achieved a blockchain-enabled compact auditing and deduplication protocol.

2.4 Keyword-based provable data possession

Recently, keyword-based auditing to improve auditing efficiency has emerged as a method. Gao et al. [22] proposed a keyword-based data auditing scheme to preserve sensitive information privacy. In their approach, they designed an index and trapdoor to achieve searchability. Xue et al. [23] introduced auditing frequency and designed a keyword-based remote auditing scheme. In their approach, they used a bloom filter to achieve fuzzy matching. However, despite its advantages, the scheme [22, 23] did not support data dynamic management. Shen et al. [24] achieved a keyword-based remote data integrity auditing system that promotes full data dynamics. However, the scheme encountered a security vulnerability in which malicious users could pre-treat files containing the same keyword to pass verification. Moreover, these schemes primarily cater to single keyword searches and do not allow for the diverse search requirements of users. Our solution improves the accuracy of keyword searches while ensuring the integrity of the search results in auditing.

3 Preliminaries

Bilinear pairing. In the multiplicative cyclic group $\mathbb{G}_1$ and $\mathbb{G}_T$, the bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ satisfies the following.

(1) Bilinearity: for any $\alpha, \beta \in \mathbb{Z}_q^*$, $u, v \in \mathbb{G}_1$, $e(u^\alpha, v^\beta) = e(u, v)^{\alpha\beta}$.

(2) Non-degeneracy: $e(g, g) \neq 1$, where g is a generator of $\mathbb{G}_1$.

(3) Computability: for any $u, v \in \mathbb{G}_1$, it is easy to retrieve $e(u, v)$.

Discrete logarithm (DL) problem. Given $(g, g^x) \in \mathbb{G}_1$, in which $x \in \mathbb{Z}_q^*$ is private. The DL assumption is that it is computationally infeasible to retrieve x.

Computational Diffie-Hellman (CDH) problem. Given (g, g^x, g^y), in which $x, y \in \mathbb{Z}_q^*$ are private. The CDH assumption is that it is computationally infeasible to retrieve g^{xy}.

4 System model and security model

4.1 Design goals

In order to design a multi-keyword searchable provable data possession, the scheme should satisfy some goals.

(1) Audiring soundness. Only when all files which contain the specific keyword are stored intactly and participated in the proof generation, the proof can pass the auditing.

(2) Multi-keyword searchability. The DO can audit the data through several keywords. The CS searches the files containing the keywords and generates the corresponding proof from the searched files. The TPA can correctly conduct the auditing of files containing the specific keywords. The integrity of the search result can be guaranteed.

(3) Resist pre-matching and pre-computation attacks. The CS cannot match the authenticator and the index in advance. Even if the CS performs a pre-matching of the authenticator and index, it does not threaten the integrity of the data. The CS cannot pre-aggregate the stored files and only utilizes the aggregated middle value to generate proof information and pass verification.

(4) The integrity of the searching results. If the CS uses incomplete search results to generate proof information, it will not pass verification.

4.2 System model

The system architecture of MKPDP, as shown in Figure 2, includes four entities: DO, CS, TPA, and blockchain.

(1) CS. The CS possesses large storage spaces and has strong computational capabilities. The CS provides storage and management services for the DO. The CS also provides data integrity checking service for the DO. When receiving a challenge request based on the trapdoor information, the CS decrypts the index matrix and searches the files which contain the keyword. Then, the CS generates the proof based on all matched files and returns the proof to the TPA.

(2) DO. The DO uploads large amounts of files to the CS for storage. The DO first retrieves some related keywords for the file. Based on the keywords, the DO generates an index matrix and computes the keyword information to match the authenticators. Then the DO retrieves the index switch and generates the authenticator for the data block. The DO uploads the file name, the index switch and the encrypted index matrix to the blockchain and outsources the data and the authenticator to the CS. When the DO wants to check the file with some keywords, the DO generates the corresponding trapdoor and sends the trapdoor to the TPA.

(3) TPA. The TPA accepts the delegation request, replaces the DO, and conducts the checking task on time. The TPA generates the challenge information and sends the trapdoor information and the challenge information to the CS. After receiving the proof information from the CS, the TPA can conduct the audit of files which contain the keyword and inform the result to the DO.

(4) Blockchain. Blockchain, as an intermediate connecting entity, provides a public, transparent and unforgeable platform for the system model. Blockchain stores the basic file information, index switch, and encrypted matrix. The update information of the file, including add file and delete file, is synchronized on the blockchain.

The DO uploads the data and authenticators to the cloud, and uploads the file name, index, and encryption matrix to the blockchain. Then the DO entrusts a TPA to conduct an audit and send the trapdoor information to the TPA. The TPA initiates a challenge to the CS, and the CS generates proof information and returns it to the TPA. The TPA obtains the relevant information of the file from the blockchain, verifies the proof information, and sends the verification result information to the DO.

4.3 Definition

Definition 1. MKPDP consists of eight algorithms.

(1) $\boldsymbol{(W,\mathcal{I}_{a\times a})\leftarrow}$ **SysIni**$\boldsymbol{(F):}$ With the files $F=\{F_i\}_{i\in[1,v]}$ to be stored, the algorithm retrieves a keyword set W for F and establishes an index matrix $\mathcal{I}_{a\times a}$ for the file.

(2) $\boldsymbol{(param,(pk,sk))\leftarrow}$ **Setup**$\boldsymbol{(1^{\kappa}):}$ With a security parameter λ, the algorithm outputs the system parameters $param$ and the secret key (x,l,o,sk).

(3) $\boldsymbol{(\{\Omega_i\}_{i\in[1,v]},Addr,E_I)\leftarrow}$ **IndexGen**$\boldsymbol{(F,\mathcal{I}_{a\times a}):}$ With the secret key x, the file set F, the keyword set W, and the index matrix $\mathcal{I}_{a\times a}$, the algorithm updates the index matrix $\mathcal{I}_{a\times a}$, encrypts the matrix to E_I and generates the blind address *Addr* for the index matrix. The algorithm generates the match index $\{\Omega_i\}_{i\in[1,v]}$ for each file.

(4) $\boldsymbol{(\{FID_i,S_i,\phi_i\}_{i\in[1,v]})\leftarrow}$ **AuthGen**$\boldsymbol{(F):}$ With the secret key x and the file set F, the algorithm generates the index switch set $S_i(i\in[1,v])$ and the authenticator $\phi_i(i\in[1,v])$.

(5) $\boldsymbol{(T_w)\leftarrow}$ **TrapdoorGen :** With the key word set W, the algorithm generates the trapdoor information T_w.

(6) $\boldsymbol{(Chal)\leftarrow}$ **ChalGen**$\boldsymbol{(T_w):}$ With the trapdoor information T_w, the algorithm generates the challenge information $Chal$.

(7) $\boldsymbol{(Proof)\leftarrow}$ **ProofGen**$\boldsymbol{(Chal,Addr,E_I\{\phi_i,\{m_{i,j}\}_{j\in[1,n]}\}_{i\in[1,v]}):}$ With the challenge information $Chal$, the encrypted matrix E_v, the index switch set S_i, and the file set F, the algorithm generates the proof information $Proof$.

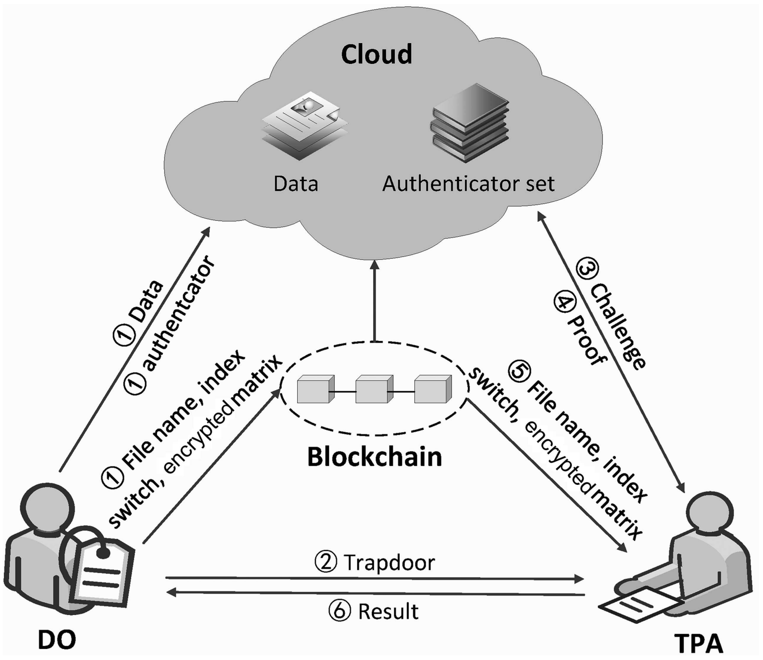

Figure 2 System model of MKPDP.

(8) $(true/false) \leftarrow$ **ProofVerify**$(Proof, Chal)$ **:** With the index switch set S_i, the trapdoor information T_w, the encrypted matrix E_v and the proof information $Proof$, the algorithm verifies the proof information.

4.4 Threat model

In MKPDP, we consider the security in keyword searchability and data integrity auditing. In the threat model, the DO is honest and the CS is semi-honest. The CS stores the data from the DO and conducts the auditing requests from the TPA. The CS tries to probe the contents of the data through the search trapdoor. The CS may use incomplete search files to generate the proof information but pass the checking. The TPA is semi-honest. The TPA executes the auditing honestly and does not collude with the CS in the challenge information generation. The TPA tries to probe the contents of the data through the proof information.

(1) The integrity loss of search results. The CS tries to utilize partial matched files to generate the audit proof but passes the TPA's check.

(2) The pre-computation attacks. The CS tries to match the index and authenticator ahead of time and only stores the intermediate values instead of the original index and authenticator values to pass the verification. The CS tries to pre-aggregate stored files and stores the pre-aggregating values instead of the original files to pass the verification.

4.5 Security model

The security model formalizes the game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. The game content is described as follows.

(1) SysIni and Setup phase. $\mathcal{C}$ conducts SysIni and Setup algorithm to conduct system initialization retrieve system public parameters. Then $\mathcal{S}$ returns $param$ to $\mathcal{A}$.

(2) Query phase. $\mathcal{A}$ makes queries for index information, the authenticator information and the trapdoor information, and the challenge information and the proof information. $\mathcal{S}$ conducts IndexGen, AuthGen, TrapdoorGen, and ProofGen algorithms to retrieve the corresponding information and sends the information to $\mathcal{A}$.

(3) Challenge phase. $\mathcal{C}$ conducts the challenge, and transmits $Chal = (T_w, c, k_1, k_2)$ to $\mathcal{A}$ for requesting the proof information.

(4) Forgery phase. $\mathcal{A}$ produces the corresponding auditing proof which contains all files with keyword w_k. If the proof passes verification, $\mathcal{A}$ wins in the game.

In the security model, we assume that the challenge information has not been queried before.

The main notations utilized in MKPDP scheme are summarized in Table 1.

Table 1 Notations in MKPDP.

Notation	The meaning	Notation	The meaning
e	The bilinear map satisfies: $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$	SE_{w_k}	The file index set of keyword w_k
$\mathbb{G}_1, \mathbb{G}_T$	Two multiplicative cyclic groups with the same prime order p	FID_i	The unique name of file F_i
g, u	Two different generators in group $\mathbb{G}_1$	W_{F_i}	The file F_i's keyword set
H_1, H_2, H_3, h_1	Four hash functions in different functionality	V, E_v	The index vector and it is encrypt version
sk, x, l, o	The DO's secret key with different usage	$Addr$	The address vector for the index vector
pk	The public key of the DO	m_{ij}	The file F_i's j-th block
$ssg_{sk}(\cdot)$	The signature	S_i	The file F_i's index switch set
$f_{\text{key}}(\cdot)$, $\psi_{\text{key}}(\cdot)$, $\pi_{\text{key}}(\cdot)$	The pseudo random permutation and pseudo random function	$\sigma_{i,j}$	The data block m_{ij}'s authenticator
v	The number of all files	T_w	The trapdoor
n	The split number of files	ϕ_i	The file F_i's authenticator set
$F = \{F_i\}_{i\in[1,v]}$	The file set	$Chal$	The challenge
$W = \{w_1, w_2, \ldots, w_m\}$	The keyword set	$Proof$	The proof

5 The proposed MKPDP scheme

5.1 Overview

The multi-keyword searchable provable data possession permits the TPA to check multiple files which contain the keywords in one audit task. The user can conduct "add" file and "delete" file operations. The data integrity checking can be conducted when the file is updated. To achieve multi-keyword searchable provable data possession, one of the challenges is how to guarantee that all files that meets the condition are utilized to generate the proof information. That is, the integrity of the search result should be guaranteed. In previous schemes, there is no method to guarantee the integrity of the search results. When CS utilizes partial files to generate the proof information, CS can also pass the verification.

5.2 Construction of MKPDP

(1) $(W, \mathcal{I}_{a\times a}) \leftarrow$ SysIni(F). Suppose that there are v files stored in CS. All files $\{F_i\}_{1\leqslant i\leqslant v}$ contain K keywords. The keyword set can be represented as $W = \{w_1, w_2, \ldots, w_K\}$. In order to prevent the number of files and keywords from leaking to the CS, the DO initiates an index matrix $\mathcal{I}_{a\times a}$, where $a \gg v, a \gg K$, as shown in Figure 3. The DO initiates all elements in $\mathcal{I}_{a\times a}$ to 0. Each row in $\mathcal{I}_{a\times a}$ represents the keyword information. Each column in $\mathcal{I}_{a\times a}$ represents the file information. Note that $\mathcal{I}_{a\times a} = \{v_{w_1}, v_{w_2}, \ldots, v_{w_K}, v_{K+1}, \ldots, v_a\}$, where $v_i (i \in [1, a])$ represents row vectors.

(2) $(param, (pk, sk)) \leftarrow$ Setup(1^κ).

• The DO chooses two different multiplicative cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_T$ with the same prime order p, two generators $g, u\mathbb{G}_1$, and a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$.

• The DO picks four different hash functions: H_1–$H_3 : \{0,1\}^* \rightarrow \mathbb{G}_1$, $h_1 : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$.

• The DO randomly picks $x \in \mathbb{Z}_q^*$ as his secret key. Then, the DO generates $y = g^x$ as his public key. The DO picks a key $l \in \mathbb{Z}_q^*$ as a pseudo random permutation key and $o \in \mathbb{Z}_q^*$ as a pseudo random function key. The DO picks a secret key and public key pair (sk, pk) and utilizes sk to generate the signature for the information. The signature algorithm can be represented as $ssg_{sk}(\cdot)$, where '$\cdot$' denotes the message. The DO randomly selects $g, u \in \mathbb{G}_1$ as public values.

• The DO selects a pseudo random permutation $f_{\text{key}}(\cdot) : \{0,1\}^* \rightarrow \{0,1\}^a$ with a key *key*. The DO selects a pseudo random permutation $\psi_{\text{key}}(\cdot) : \{0,1\}^* \rightarrow [1, n]$ with a key *key*. The DO selects a pseudo random function $\pi_{\text{key}} : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$ with key.

• The DO publishes the system parameters $param = \{\mathbb{G}_1, \mathbb{G}_T, p, g, u, y, pk, e, H_1, H_2, H_3, h_1, f_{\text{key}}(\cdot), \psi_{\text{key}}(\cdot), \pi_{\text{key}}(\cdot), ssg_{sk}(\cdot)\}$.

(3) $(\{\Omega_i\}_{i\in[1,v]}, Addr, E_I) \leftarrow$ IndexGen$(F, \mathcal{I}_{a\times a})$.

• For each file $F_i \in \{F_i\}_{1\leqslant i\leqslant v}$ containing the keyword w_k, the DO writes corresponding i-th bit of the index vector to 1: $v_{w_k}[i] = 1$.

• For each keyword $w_k \in W$, the DO initializes an empty set $SE_{w_k} = \emptyset$. If $v_{w_k}[i] = 1$ for any $i \in [1, v]$, the DO adds the corresponding file index i into set SE_{w_k}.

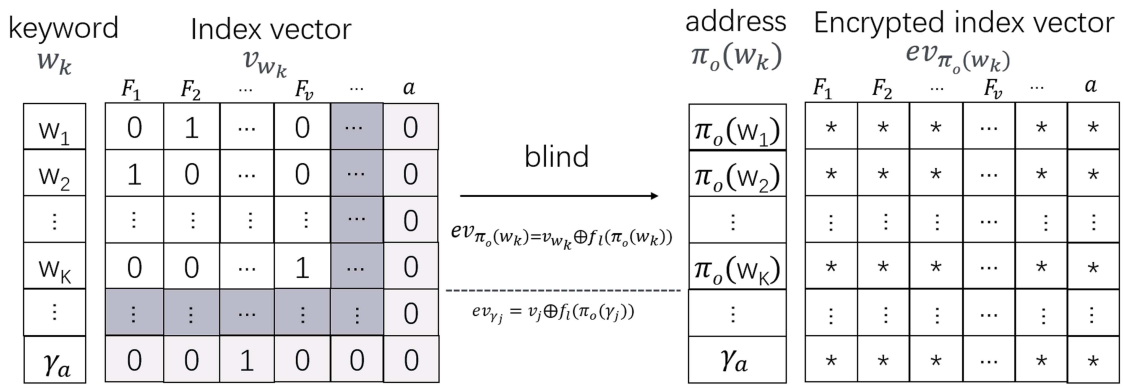

Figure 3 (Color online) Index matrix.

- For each file $F_i \in \{F_i\}_{1\leqslant i\leqslant v}$, the DO initializes an empty set $W_{F_i}=\emptyset$. Then the DO parses keywords of each file $F_i \in \{F_i\}_{1\leqslant i\leqslant v}$ into keyword set $W_{F_i}=\{w_{b_1},\ldots,w_{b_l},\ldots,w_{b_t}\}$. Suppose t_i is the total number of keywords in file F_i. Note that the number of keywords in each file can be different.
- For each file $F_i\{F_i\}_{1\leqslant i\leqslant v}$, the DO generates the keyword information as $HF_i=\prod_{l=1}^{t_i}H_1(w_{b_l})$, $w_{b_l}\in W_{F_i}$. Then, the DO generates the match index $\Omega_i=(HF_i\cdot H_2(FID_i))^{-x}$ for each file $F_i\{F_i\}_{1\leqslant i\leqslant v}$, where FID_i is the identifier of file F_i. The match index of all files can be represented as $\Omega=\{\Omega_i\}_{i\in[1,v]}$.
- For each keyword $w_k\in W$, the DO computes $\pi_o(w_k)$ as the blind address assigned to each row within the index matrix. For $j\in[K+1,a]$, the DO randomly selects $\gamma_k\in\mathbb{Z}_q^*$ as blind address for the vector v_k. All blind addresses can be represented as a set $Addr=\{\pi_o(w_1),\pi_o(w_2),\ldots,\pi_o(w_K),\gamma_{K+1},\ldots,\gamma_a\}$.
- For each keyword $w_k\in W$, the DO performs encryption on the index vector, resulting in $ev_{\pi_o(w_k)}=v_{w_k}\oplus f_l(\pi_o(w_k))$. For $j\in[K+1,a]$, the DO performs encryption $ev_{\gamma_j}=v_j\oplus f_l(\pi_o(\gamma_j))$. The encrypted version of $\mathcal{I}_{a\times a}$ can be represented as a matrix $E_I=\{ev_{\pi_o(w_1)},ev_{\pi_o(w_2)},\ldots,ev_{\pi_o(w_K)},ev_{\gamma_{K+1}},\ldots,ev_{\gamma_a}\}$. A better understanding of the index matrix and index information can be seen in Figure 3.

For instance, suppose that file F_1 contains keyword w_2 and w_4, file F_2 contains keyword w_1, w_3 and w_4, and file F_3 contains keyword w_2 and w_5. The keyword set of F_1 is $W_{F_1}=\{w_2,w_4\}$. The keyword set of F_2 is $W_{F_2}=\{w_1,w_3,w_4\}$. The keyword set of F_3 is $W_{F_3}=\{w_2,w_5\}$. The key information of F_1 is $HF_1=\prod_{l=1}^{2}H_1(w_{b_l})=H_1(w_2)\cdot H_1(w_4)$ and the match index is $\omega_1=(HF_1\cdot H_2(FID_1))^{-x}$. The key information of F_2 is $HF_2=\prod_{l=1}^{3}H_1(w_{b_l})=H_1(w_1)\cdot H_1(w_3)\cdot H_1(w_4)$ and the match index is $\omega_2=(HF_2\cdot H_2(FID_2))^{-x}$. The key information of F_3 is $HF_3=\prod_{l=1}^{2}H_1(w_{b_l})=H_1(w_2)\cdot H_1(w_5)$ and the match index is $\omega_3=(HF_3\cdot H_2(FID_3))^{-x}$.

(4) $(\{FID_i,S_i,\phi_i\}_{i\in[1,v]})\leftarrow$ AuthGen(F).

- The DO firstly splits each file $F=\{F_i\}_{i\in[1,v]}$ into n blocks denoted as $F_i=\{m_{i,j}\}_{j\in[1,n]}$. Then the DO creates the index switch set $S_i=\{\theta_{i,j}\}_{j\in[1,n]}(i\in[1,v])$, where $\theta_{i,j}$ denotes $\theta_{i,j}=(j,au_{i,j})$ and j denotes the block index of file F_i.
- The DO computes the authenticator $\sigma_{i,j}=(HF_i\cdot H_2(FID_i)\cdot H_3(au_{i,j})\cdot u^{m_{i,j}})^x$ for each data block $m_{i,j}(i\in[1,v],j\in[1,n])$ by utilizing the authenticator index $au_{i,j}$. Let $\phi_i=\{\sigma_{ij}\}_{j\in[1,n]}$ be the authenticator set of the file $F_i(i\in[1,v])$.

The DO sends $\{FID_i,S_i,Addr,E_I,\Omega\}_{i\in[1,v]}$ to the blockchain according to the smart contract as shown in Algorithm 1. The DO sends $\{\phi_i,\{m_{i,j}\}_{j\in[1,n]}\}_{i\in[1,v]}$ to the CS. The CS verifies the authenticator by $e(\prod_{i=1}^{v}\prod_{j=1}^{n}\sigma_{i,j},g)=e(\prod_{i=1}^{v}\prod_{j=1}^{n}(HF_i\cdot H_2(FID_i)\cdot H_3(au_{i,j})\cdot u^{m_{i,j}}),y)$. If it holds, the CS accepts the data and the authenticator.

(5) $(T_w)\leftarrow$ TrapdoorGen. When the DO wants to check files with h different file keywords and the target keyword set can be represented as $W'=\{w_1,w_2,\ldots,w_h\}$, the DO generates the search trapdoor as $T_w=\{(\pi_o(w_i),f_l(\pi_o(w_i)))\}_{i\in[1,h]}$. The DO sends the trapdoor information to the TPA.

(6) $(Chal)\leftarrow$ ChalGen(T_w). When the TPA receives the trapdoor information, it randomly selects $k_1,k_2\in\mathbb{Z}_q^*$, challenge block numbers c, and transmits the challenge information $Chal=(T_w,c,k_1,k_2)$ to the CS. Then the TPA computes the auditing information ahead of time.

- The TPA generates the challenge information $\{C=\{(j_1,v_{ij})\}_{j_1\in[1,n],i\in[1,v]}\}$, where $j_1=\psi_{k_1}(\beta)$ for $\beta\in[1,c]$ and $v_{ij}=\pi_{y_i}(j)$ for $y_i=\pi_{k_2}(i)$, $i\in[1,v]$, $j\in[1,c]$. Suppose $Q_1=\{j_1=\psi_{k_1}(\beta)\}_{\beta\in[1,c]}$.
- For each blind address $\{\pi_o(w_i)\}_{i\in[1,h]}$ from the trapdoor, the TPA checks whether $\pi_o(w_i)$ exists, if so, the TPA locates the row $ev_{\pi_o(w_i)}$. Suppose the TPA locates $\zeta(1\leqslant\zeta\leqslant h)$ rows, and the located row set to be $\mathcal{U}=\{ev_{\pi_o(w_{1i})}\}_{1\leqslant i\leqslant\zeta}$. For each $ev_{\pi_o(w_{1i})}\in\mathcal{U}$, the TPA obtains the plaintext vectors $\mathcal{V}=\{v_{w_{1i}}\}_{1\leqslant i\leqslant\zeta}$, where

Algorithm 1 Smart contract.

```
Require: Set the initial state $State := Init$; call SysIni, Setup, IndexGen algorithms to obtain public parameters $param$; secret key $(x, l, o, sk)$, keyword set $W$, index matrix $\mathcal{I}_{a\times a}$, encrypted matrix $E_I$, and the blind address $Addr$; match index $\{\Omega_i\}_{i\in[1,v]}$.
1: Define a structure of the auditing task $taskID, fileName, indexStatus$;
2: the participant address $addrDO, addrTPA, addrCS$;
3: Create: $Task$;
4: Assert $State := Init$;
5: Assert $balance[DO] \leqslant \$deposit + Create.gasLimit * gasPrice$;
6: Assert $balance[DO] := balance[DO] - deposit - Create.gas * gasPrice$;
7: $State := Created$;
8: Send $Create.gas * gasPrice$ to the miner;
9: function: $newTask(FID, addrDO, addrTPA, addrCS)$;
10: task=$tasks[taskID]$;
11: task.fileName=$FID$;
12: task.addressDO=$addrDO$;
13: task.addressTPA=$addrTPA$;
14: task.addressCS=$addrCS$;
15: task.indexStatus=$true$;
16: function: IndexGen$(taskID, Addr, E_I, \Omega = \{\Omega_{\pi_o(w_k)}\}_{k=1,2,\ldots,m})$;
17: Require $(tasks[taskID].addressTPA == msg.sender)$;
18: Require $(tasks[taskID].indexStatus == true)$;
19: task.Addr=$Addr$;
20: task.E_I=$E_I$;
21: task.index=$\Omega$;
22: task.indexStatus=$false$;
23: function: $addFile(taskID, FID_z, S_z, E_{I'}, \Omega_z)$;
24: Require $(tasks[taskID].addressDO == msg.sender)$;
25: task.updatefile.push($FID_z$);
26: task.updatefile.push($S_z$);
27: task.update.push($E_{I'}$);
28: task.updatefile.push($\Omega_z$);
29: function: $deleteFile(taskID, E_{I'})$;
30: Require $(tasks[taskID].addressDO == msg.sender)$;
31: task.updatefile.push($E_{I'}$).
```

$v_{w_{1i}} = ev_{\pi_o(w_{1i})} \oplus f_l(\pi_o(w_{1i}))$.

• For $1 \leqslant i \leqslant \zeta$, the TPA begins by initializing an empty set $\mathcal{T}_i = \emptyset$. For each $j \in [1, \zeta]$, if $v_{w_i}[j] = 1$, the TPA includes j in the set $\mathcal{T}_i$. For $1 \leqslant i \leqslant \zeta$, the TPA generates $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2 \cup \cdots \cup \mathcal{T}_\zeta$.

(7) $(Proof) \leftarrow$ ProofGen$(Chal, Addr, E_I, \{\phi_i, \{m_{i,j}\}_{j\in[1,n]}\}_{i\in[1,v]})$.

• The CS generates the challenge information $\{C = \{(j_2, e_{ij})\}_{j_2\in[1,n],i\in[1,v]}\}$, where $j_2 = \psi_{k_1}(\beta)$ for $\beta \in [1, c]$ and $e_{ij} = \pi_{x_i}(j)$ for $x_i = \pi_{k_2}(i)$, $i \in [1, v]$, $j \in [1, c]$. Suppose $Q_2 = \{j_2 = \psi_{k_1}(\beta)\}_{\beta\in[1,c]}$.

• For each blind address $\{\pi_o(w_i)\}_{i\in[1,h]}$ from the trapdoor, the CS checks whether $\pi_o(w_i)$ exists, if so, the CS locates the row $ev_{\pi_o(w_i)}$. Suppose the CS locates $\gamma(1 \leqslant \gamma \leqslant h)$ rows, and the located row set to be $\mathcal{W} = \{ev_{\pi_o(w_{2i})}\}_{1\leqslant i\leqslant\gamma}$. For each $ev_{\pi_o(w_{2i})} \in \mathcal{V}$, the CS obtains the plaintext vectors $\mathcal{X} = \{v_{w_{2i}}\}_{1\leqslant i\leqslant\gamma}$, where $v_{w_{2i}} = ev_{\pi_o(w_{2i})} \oplus f_l(\pi_o(w_{2i}))$.

• For $1 \leqslant i \leqslant \gamma$, the CS begins by initializing an empty set $\mathcal{R}_i = \emptyset$. For each $j \in [1, a]$, if $v_{w_i}[j] = 1$, the CS includes j in the set $\mathcal{R}_i$. For $1 \leqslant j \leqslant \gamma$, the CS generates $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2 \cup \cdots \cup \mathcal{R}_\gamma$.

• According to the challenge set Q and the retrieved index set S_w, the CS produces proof information $Proof = \{T, \mu\}$, where $\mu = \sum_{i\in\mathcal{R}} \sum_{j\in Q_2} e_{ij} \cdot m_{i,j}$ and $T = \prod_{i\in\mathcal{R}} \prod_{j\in Q_2} \sigma_{i,j}^{e_{ij}}$.

• The CS randomly selects $r \in \mathbb{Z}_q^*$ and generates $R = g^r$. Then the CS blinds μ by utilizing r and R as $\mu_1 = \mu - rh_1(T_w, R)$ for guaranteeing the data privacy.

The CS transmits the proof $Proof = \{T, \mu_1, R\}$ to the TPA.

(8) $(true/false) \leftarrow$ ProofVerify$(Proof, Chal)$.

• After obtaining $Proof$, the TPA requests the index switch set S_i of the file $F_i(i \in \mathcal{T})$, searches the match index $\Omega_i, i \in \mathcal{T}$ and finds the authenticator indexes $au_{i,j}(i \in \mathcal{T}, j \in Q_1)$ of challenged blocks $m_{i,j}(i \in \mathcal{T}, j \in Q_1)$.

• With the keyword tag tag_{w_k} and the authenticator indexes $au_{i,j}(i \in \mathcal{T}, j \in Q_1)$, the TPA checks $Proof$'s correctness by

$$e\left(T \cdot \prod_{i\in\mathcal{T}} \Omega_i^{\sum_{j\in Q_1} v_{ij}}, g\right) = e\left(\left(\prod_{i\in\mathcal{T}} \prod_{j\in Q_1} H_3(au_{i,j})^{v_{ij}}\right) \cdot u^{\mu_1} \cdot R^{h_1(T_w,R)}, y\right). \tag{1}$$

If the equation holds, the files which contain the keyword in set W' are intact; otherwise, the files are incomplete.

5.3 Extension

5.3.1 *Add file*

When the DO wants to add a new file F_z into the cloud, the DO first adds keywords for this file.

(1) The DO initiates a column vector c_{f_z} with length of a and sets all elements to 0. If the new file contains keyword $w_k(k \in [1, K])$, the DO sets $c_{f_z}[k] = 1$. The DO updates the $(v+1)$-th column of matrix $\mathcal{I}_{a\times a}$ to c_{f_z}. Suppose the latest matrix of $\mathcal{I}_{a\times a}$ be $\mathcal{I}'_{a\times a} = \{v'_{w_1}, v'_{w_2}, \dots, v'_{w_K}, v'_{K+1}, \dots, v'_a\}$.

(2) The DO retrieves all keywords of file F_z into its corresponding keyword set $W_{F_z} = \{w_{b_1}, \dots, w_{b_l}, \dots, w_{b_{t_z}}\}$. Suppose t_z is the total number of keywords of file F_z. DO computes the keyword information $HF_z = \prod_{l=1}^{t_z} H_1(w_{b_l})$, $w_{b_l} \in W_{F_z}$. Then the DO calculates the match index $\Omega_z = (HF_z \cdot H_2(FID_z))^{-x}$ for file F_z, where FID_z is the identifier of file F_z.

(3) For each keyword $w_k \in W_{F_z}$, the DO performs encryption on the index vector, resulting in $ev'_{\pi_o(w_k)} = v'_{w_k} \oplus f_l(\pi_o(w_k))$. For $j \in [K+1, a]$, DO performs encryption $ev_{\gamma_j} = v'_j \oplus f_l(\pi_o(\gamma_j))$. The encrypted version of $\mathcal{I}'_{a\times a}$ can be represented as a matrix $E_{I'} = \{ev'_{\pi_o(w_1)}, ev'_{\pi_o(w_2)}, \dots, ev'_{\pi_o(w_K)}, ev'_{\gamma_{K+1}}, \dots, ev'_{\gamma_a}\}$.

(4) The DO splits file F_z into n blocks denoted as $F_z = \{m_{zj}\}_{j\in[1,n]}$. Then the DO generates the index switch set $S_z = \{\theta_{z,j}\}_{j\in[1,n]}$, where $\theta_{z,j} = (j, au_{z,j})$, and j indicates the block index of file F_z. The DO computes the authenticator $\sigma_{z,j} = (HF_z \cdot H_2(FID_z) \cdot H_3(au_{z,j}) \cdot u^{m_{z,j}})^x$ for each data block $m_{z,j}(j \in [1, n])$ by utilizing the authenticator index $au_{z,j}$. Let $\phi_z = \{\sigma_{zj}\}_{j\in[1,n]}$ be the authenticator set of the file F_z.

The DO sends an add request $\{add, FID_z, S_z, E_{I'}, \Omega_z\}$ to the blockchain. The DO sends $\{\phi_z, \{m_{z,j}\}_{j\in[1,n]}\}$ to the CS. The CS verifies the authenticator by $e(\prod_{j=1}^n \sigma_{z,j}, g) = e(\prod_{j=1}^n (HF_z \cdot H_2(FID_z) \cdot H_3(au_{z,j}) \cdot u^{m_{z,j}}), y)$. If it holds, the CS accepts the data and the authenticator.

5.3.2 *Delete file*

Suppose the DO wants to delete file F_z.

(1) The DO retrieves the z-th column of the latest version V' and sets the column to 0.

(2) For each keyword $w_k \in W_{F_z}$, the DO performs encryption on the index vector, resulting in $ev'_{\pi_o(w_k)} = v'_{w_k} \oplus f_l(\pi_o(w_k))$. For $j \in [K+1, a]$, the DO performs encryption $ev_{\gamma_j} = v'_j \oplus f_l(\pi_o(\gamma_j))$. The encrypted version of $\mathcal{I}'_{a\times a}$ can be represented as a matrix $E_{I'} = \{ev'_{\pi_o(w_1)}, ev'_{\pi_o(w_2)}, \dots, ev'_{\pi_o(w_K)}, ev'_{\gamma_{K+1}}, \dots, ev'_{\gamma_a}\}$.

The DO sends $\{E_{I'}\}$ to the blockchain. The DO sends $\{delete, FID_z, ssg_{sk}(delete\|FID_z)\}$ to the CS. The CS verifies the $ssg_{sk}(delete\|FID_z)$ and deletes the corresponding data and authenticators if the signature is correct.

6 Security analysis

Theorem 1 (Correctness)**.** When all files which contain the specific keyword are stored intactly and participated in the proof generation, the proof can pass the auditing.

Proof. Eq. (1) can be derived as below:

$$\begin{aligned}
& e\left(T \cdot \prod_{i\in\mathcal{T}} \Omega_i^{\sum_{j\in Q_1} v_{ij}}, g\right) \\
&= e\left(\prod_{i\in\mathcal{T}}\prod_{j\in Q_1} \sigma_{i,j}^{v_{ij}} \cdot \prod_{i\in\mathcal{T}} (HF_i \cdot H_2(FID_i))^{-x\cdot\sum_{j\in Q_1} v_{ij}}, g\right) \\
&= e\left(\prod_{i\in\mathcal{T}}\prod_{j\in Q_1} ((HF_i \cdot H_2(FID_i) \cdot H_3(au_{i,j}) \cdot u^{m_{i,j}})^x)^{v_{ij}} \prod_{i\in\mathcal{T}} (HF_i \cdot H_2(FID_i))^{-x\cdot\sum_{j\in Q_1} v_{ij}}, g\right) \\
&= e\left(\left(\prod_{i\in\mathcal{T}}\prod_{j\in Q_1} H_3(au_{i,j})^{v_{ij}}\right) \cdot u^{\mu_1} \cdot R^{h_1(T_w, R)}, y\right).
\end{aligned}$$

Theorem 2 (Auditing soundness)**.** Assume that the CDH problem is hard. The CS can pass the auditing verification if and only if it possesses the intact data.

Proof. **Game 0**: Game 0 is defined the same as described in the security model.

Game 1: Game 1 is set almost the same as Game 0, except for a change.

Since $Proof = \{T, \mu_1, R\}$ is valid produced by the CS, the following equation:

$$e\left(T \cdot \prod_{i\in\mathcal{T}} \Omega_i^{\sum_{j\in Q_1} v_{ij}}, g\right) = e\left(\left(\prod_{i\in\mathcal{T}}\prod_{j\in Q_1} H_3(au_{i,j})^{v_{ij}}\right) \cdot u^{\mu_1} \cdot R^{h_1(T_w,R)}, y\right) \tag{2}$$

holds. Suppose a forged proof be $Proof^* = \{T^*, \mu_1^*, R\}$ and can pass the verification. The following equation:

$$e\left(T^* \cdot \prod_{i\in\mathcal{T}} \Omega_i^{\sum_{j\in Q_1} v_{ij}}, g\right) = e\left(\left(\prod_{i\in\mathcal{T}}\prod_{j\in Q_1} H_3(au_{i,j})^{v_{ij}}\right) \cdot u^{\mu_1^*} \cdot R^{h_1(T_w,R)}, y\right). \tag{3}$$

We can observe that $\mu_1 \neq \mu_1^*$, otherwise $T = T^*$. Let $\Delta\mu = \mu_1^* - \mu_1$. If $\mathcal{A}$ wins Game 1 with a non-negligible probability, $\mathcal{C}$ can leverage $\mathcal{A}$'s capability to solve the CDH problem. That is, given (g, g^a, g^b), $\mathcal{S}$ is able to retrieve g^{ab}. $\mathcal{C}$ randomly picks two values $a, d \in \mathbb{Z}_q^*$, and computes $u = g^a g^{bd}$, $y = g^a$. $\mathcal{S}$ randomly selects $x_{ij} \in \mathbb{G}_1$ for the challenge i and j.

Dividing (3) by (2), we obtain

$$e(T^*/T, g) = e(u^{\mu_1^* - \mu_1}, y).$$

Further, we can obtain

$$e(T^*/T, g) = e((g^a \cdot g^{bd})^{\mu_1^* - \mu_1}, g^a).$$

Hence, we can obtain $g^{ab} = ((\frac{T^*}{T})^{\frac{1}{\mu_1^* - \mu_1}} \cdot g^{-a^2})^{\frac{1}{a}}$.

The probability that $\triangle\mu = \mu_1^* - \mu_1 \neq 0$ is $1 - 1/q$, which is non-negligible. Thus, the CDH can be solved with the probability $1 - 1/q$. This contradicts with the assumption: CDH problem is hard.

Game 2: Game 1 is equivalent to Game 0, except for a change. Assume $Proof^* = \{T^*, \mu_1^*, R\}$ is a valid auditing proof produced by the CS. We get

$$e\left(T^* \cdot \prod_{i\in\mathcal{T}} \Omega_i^{\sum_{j\in Q_1} v_{ij}}, g\right) = e\left(\left(\prod_{i\in\mathcal{T}}\prod_{j\in Q_1} H_3(au_{i,j})^{v_{ij}}\right) \cdot u^{\mu_1^*} \cdot R^{h_1(T_w,R)}, y\right).$$

According to the valid proof $Proof = \{T, \mu_1, R\}$, we get

$$e\left(T \cdot \prod_{i\in\mathcal{T}} \Omega_i^{\sum_{j\in Q_1} v_{ij}}, g\right) = e\left(\left(\prod_{i\in\mathcal{T}}\prod_{j\in Q_1} H_3(au_{i,j})^{v_{ij}}\right) \cdot u^{\mu_1} \cdot R^{h_1(T_w,R)}, y\right).$$

From Game 1, we obtain that $T^* = T$. Let $\triangle\mu = \mu_1^* - \mu_1$. DL problem can be solved by $\mathcal{C}$ as follows: Given (g, g^a), $\mathcal{C}$ can obtain a. $\mathcal{S}$ chooses $a, \beta \in \mathbb{Z}_q^*$ and calculates $u = g^a g^{b\beta}$. Since $T = T^*$, we have

$$e\left(\left(\prod_{i\in\mathcal{T}}\prod_{j\in Q_1} H_3(au_{i,j})^{v_{ij}}\right) \cdot u^{\mu_1^*} \cdot R^{h_1(T_w,R)}, y\right). = e\left(T^* \cdot \prod_{i\in\mathcal{T}} \Omega_i^{\sum_{j\in Q_1} v_{ij}}, g\right),$$

$$e\left(T \cdot \prod_{i\in\mathcal{T}} \Omega_i^{\sum_{j\in Q_1} v_{ij}}, g\right) = e\left(\left(\prod_{i\in\mathcal{T}}\prod_{j\in Q_1} H_3(au_{i,j})^{v_{ij}}\right) \cdot u^{\mu_1} \cdot R^{h_1(T_w,R)}, y\right).$$

We can further derive that $u^{\mu_1^*} = u^{\mu_1}$ and $u^{\triangle\mu} = (g^a g^{b\beta})^{\triangle\mu} = g^{a\triangle\mu} g^{b\beta\triangle\mu} = 1$. Thus, the DL problem can be solved by $\mathcal{C}$ as $g^a = g^{-b\beta}$. The probability that $\triangle\mu \neq 0$ is $1 - 1/q$ and non-negligible. Thus, the DL problem can be solved with a non-negligible probability. This contradicts the assumption: the DL problem is hard.

Theorem 3 (Multi-keyword searchability)**.** The DO can audit the data through several keywords. The TPA can correctly conduct the auditing of files containing the specific keywords. The integrity of the search result can be guaranteed.

Table 2 Comparison of provable data possession schemes.

Scheme	Ref. [22]	Ref. [24]	Ref. [23]	Ours
Public auditing	✓	✓	✓	✓
Searchable auditing	✓	✓	✓	✓
Keyword privacy	✓	×	✓	✓
Multi-keyword	×	×	×	✓
Data privacy	×	×	×	✓
File dynamic	×	✓	×	✓

Proof. In MKPDP, the DO generates the search trapdoor $T_w = \{(\pi_o(w_i), f_l(\pi_o(w_i)))\}_{i\in[1,h]}$ from the keyword set $W = \{w_1, w_2, \ldots, w_h\}$. The CS conducts the search by utilizing the trapdoor. The CS first checks whether $\pi_o(w_i)$, $i \in [1, h]$ exists, if so, the CS locates the row $ev_{\pi_o(w_i)}$, $i \in [1, h]$. After locating the row, the CS decrypts $ev_{\pi_o(w_i)}$, $i \in [1, h]$ and obtains the file index. All file indexes are collected in set $\mathcal{R}$. All files that meet the matching conditions in $\mathcal{R}$ are used to generate the proof information. Assume that the set $\mathcal{R}$ contains the correct and intact file indexes. When the CS utilizes an incomplete set $\mathcal{R}^* \subset \mathcal{R}$, the proof is intact and cannot pass the verification. All index information is stored on the blockchain and cannot be modified or deleted. The TPA generates the set $\mathcal{T}$ as well and conducts the match of the proof information T. Normally, $\mathcal{R} = \mathcal{T}$. Once T does not contain all the information in $\mathcal{T}$, the checking would fail.

Theorem 4 (Resist pre-matching and pre-computation attacks)**.** The proposed scheme can resist pre-matching and pre-computation attacks.

Proof. The CS cannot obtain valid information from pre-matching. In the proposed scheme, the authenticator is represented as $\sigma_{i,j} = (HF_i \cdot H_2(FID_i) \cdot H_3(au_{i,j}) \cdot u^{m_{i,j}})^x$, and the index is represented as $\Omega_i = (HF_i \cdot H_2(FID_i))^{-x}$. Since the CS does not know the one-to-one correspondence between the authenticator and the index, even if some tags match the index successfully, suppose the result after matching is expressed as $\Upsilon = (H_3(au_{i,j}) \cdot u^{m_{i,j}})$. The middle value can pass the verification $e(\Upsilon, g) = e(H_3(au_{i,j}) \cdot u^{m_{i,j}}, y)$. Once the matching is completed, the authenticator can be associated with the index. However, the CS cannot obtain any information from this information. The CS cannot infer the data content from the search process. To resist the data content leaking from the keyword, we design a blind index matrix. First, the length of row and column in $\mathcal{I}_{a\times a}$ is much larger than the number of files ($a \gg v$) and the keywords ($a \gg K$); thus the CS cannot guess the number of files and keywords. Second, for $j \in [1, K]$, the blind address assigned to each row within the index matrix is $\pi_o(w_k)$; the keyword w_k is encrypted by $\pi_o(\cdot)$. The CS cannot derive the content w_k from address $\pi_o(\cdot)$. For $j \in [K + 1, a]$, the blind address assigned to each row within the index matrix is $\gamma_j \in \mathbb{Z}_q^*$; the CS cannot derive any content from $\gamma_j \in \mathbb{Z}_q^*$. Thus, the CS cannot infer the data content from the search process from address γ_j. Furthermore, for $j \in [1, K]$, the index vector is encrypted by $ev_{\pi_o(w_j)} = v_{w_j} \oplus f_l(\pi_o(w_j))$. For $j \in [K + 1, a]$, the index vector is encrypted by $ev_{\gamma_j} = v_j \oplus f_l(\pi_o(\gamma_j))$. The CS cannot derive any content from $ev_{\pi_o(w_j)}, j \in [1, K]$ and $ev_{\gamma_j}, j \in [K + 1, a]$.

The CS cannot pre-aggregate the stored files and only utilizes the aggregated middle value to generate proof information and pass verification. In our scheme, the aggregate authenticator generated on the CS side is $T = \prod_{i\in\mathcal{R}} \prod_{j\in Q_2} \sigma_{i,j}^{e_{ij}}$, the information is related to authenticators, and the index is not participated in this process.

Theorem 5 (The integrity of the searching results)**.** The proposed scheme can guarantee the integrity of the searching results.

Proof. In the verification process, the TPA should generate the aggregate index information $\prod_{i\in\mathcal{T}} \Omega_i^{\sum_{j\in Q_1} v_{ij}}$; this information is retrieved from the blockchain according to the trapdoor. This search process is the same as the process of searching by the CS. This ensures that if the CS does not use all the information to generate the proof information, the verification will fail.

Theorem 6 (File dynamics)**.** The proposed scheme can achieve file dynamics, including add file and delete file operations.

Proof. In order to facilitate file dynamics, the column size of the file index matrix a is set significantly larger than the original storage file size v, denoted as $a \gg v$. The DO initiates all elements in $\mathcal{I}_{a\times a}$ to 0. When the DO intends to add a new file F_z to the cloud, the DO modifies the index matrix and inserts a new column vector cf_z with a length of a into $\mathcal{I}_{a\times a}$. Subsequently, the DO inserts cf_z with length a into $\mathcal{I}_{a\times a}$, resulting a new matrix $\mathcal{I}'_{a\times a}$. The DO then generates the index information for the new file. The DO retrieves the authenticator information and submits an add request to the blockchain. Subsequently, the DO sends both the authenticator information and the data to the cloud. When the DO decides to delete a file F_z, they locate the z-th column of the index matrix and reset the column to 0. Following this, the DO re-encrypts the index vector, resulting in a new matrix. The DO

Table 3 Computation overhead.

Phase	Ref. [22]	Ref. [24]	Ref. [23]	Our scheme
IndexGen	$K \cdot s \cdot [(\lvert SE_{w_k}\rvert + 2) \cdot H_{\mathbb{G}_1} + (\lvert SE_{w_k}\rvert + 1) \cdot Mul_{\mathbb{G}_1} + Exp_{\mathbb{G}_1}]$	$v \cdot (t_i \cdot H_{\mathbb{G}_1} + t_i \cdot Mul_{\mathbb{G}_1} + Exp_{\mathbb{G}_1}) + K \cdot \lvert SE_{w_k}\rvert Mul_{\mathbb{G}_1}$	$(\lvert SE_{w_k}\rvert + 1)(H_{\mathbb{G}_1} + Mul_{\mathbb{G}_1}) + Exp_{\mathbb{G}_1}$	$v \cdot (t_i \cdot H_{\mathbb{G}_1} + t_i \cdot Mul_{\mathbb{G}_1} + Exp_{\mathbb{G}_1})$
AuthGen	$v \cdot n \cdot (H_{\mathbb{G}_1} + Mul_{\mathbb{G}_1} + Exp_{\mathbb{G}_1})$	$v \cdot n \cdot (2H_{\mathbb{G}_1} + 2Exp_{\mathbb{G}_1} + 3Mul_{\mathbb{G}_1})$	$v \cdot n \cdot (H_{\mathbb{G}_1} + Mul_{\mathbb{G}_1} + 2 \cdot Exp_{\mathbb{G}_1})$	$v \cdot n \cdot (H_{\mathbb{G}_1} + Mul_{\mathbb{G}_1} + Exp_{\mathbb{G}_1})$
ProofGen	$(\lvert SE_{w_k}\rvert \cdot c + c)Mul_{\mathbb{G}_1} + \lvert SE_{w_k}\rvert \cdot cMul_{\mathbb{Z}_q^*} + 2cExp_{\mathbb{G}_1}$	$\lvert SE_{w_k}\rvert \cdot Mul_{\mathbb{G}_1} + (\lvert SE_{w_k}\rvert + c)Mul_{\mathbb{Z}_q^*} + c \cdot Exp_{\mathbb{G}_1}$	$(\lvert SE_{w_k}\rvert \cdot c + c)Mul_{\mathbb{G}_1} + \lvert SE_{w_k}\rvert \cdot c \cdot Mul_{\mathbb{Z}_q} + \lvert SE_{w_k}\rvert \cdot c \cdot Exp_{\mathbb{G}_1}$	$\lvert SE_{w_k}\rvert \cdot c \cdot Mul_{\mathbb{G}_1} + (\lvert SE_{w_k}\rvert \cdot c + 1)Mul_{\mathbb{Z}_q} + (\lvert SE_{w_k}\rvert \cdot c + 1)Exp_{\mathbb{G}_1}$
ProofVerify	$2 \cdot Pair + 2 \cdot c \cdot H_{\mathbb{G}_1} + (c+1) \cdot Mul_{\mathbb{G}_1} + (c+1) \cdot Exp_{\mathbb{G}_1}$	$2 \cdot Pair + (c + \lvert SE_{w_k}\rvert) \cdot Mul_{\mathbb{G}_1}) + (c+1) \cdot Exp_{\mathbb{G}_1}$	$2 \cdot Pair + 2 \cdot H_{\mathbb{G}_1} + Mul_{\mathbb{G}_1} + (c+1) \cdot Exp_{\mathbb{G}_1}$	$2 \cdot Pair + (\lvert SE_{w_k}\rvert + c \cdot \lvert SE_{w_k}\rvert)Mul_{\mathbb{G}_1} + (\lvert SE_{w_k}\rvert + c \cdot \lvert SE_{w_k}\rvert + 2)Exp_{\mathbb{G}_1}$

Table 4 Communication overhead.

Phase	Ref. [22]	Ref. [24]	Ref. [23]	Our scheme
DO $\rightarrow$ CS	$v \cdot \lvert F\rvert + (v \cdot n + K \cdot s)\lvert\mathbb{G}_1\rvert$	$v \cdot \lvert F\rvert + (v \cdot n + v)\lvert\mathbb{G}_1\rvert + (1 + v \cdot K)\lvert\mathbb{Z}_q^*\rvert$	$v \cdot \lvert F\rvert + (v \cdot n + K \cdot s)\lvert\mathbb{G}_1\rvert$	$v \cdot \lvert F\rvert + (v \cdot n + v)\lvert\mathbb{G}_1\rvert$
DO $\rightarrow$ TPA	$2\lvert\mathbb{Z}_q^*\rvert$	$(s+1)\mathbb{Z}_q^* + s\lvert n\rvert$	$2 \cdot v\lvert\mathbb{Z}_q^*\rvert$	$2 \cdot h\lvert\mathbb{Z}_q^*\rvert$
TPA $\rightarrow$ CS	$(2+c)\lvert\mathbb{Z}_q^*\rvert + c\lvert n\rvert$	$(c+1)\lvert\mathbb{Z}_q^*\rvert + c\lvert n\rvert$	$(2+c)\lvert\mathbb{Z}_q^*\rvert + (c+k)\lvert n\rvert + s\lvert\mathbb{G}_1\rvert + \lvert Enc\rvert$	$(2 \cdot h + 2)\lvert\mathbb{Z}_q^*\rvert + \lvert c\rvert$
CS $\rightarrow$ TPA	$\lvert\mathbb{Z}_q^*\rvert + \lvert\mathbb{G}_1\rvert$	$K\lvert n\rvert + \lvert\mathbb{Z}_q^*\rvert + \lvert\mathbb{G}_1\rvert$	$\lvert\mathbb{Z}_q^*\rvert + \lvert\mathbb{G}_1\rvert + \lvert Enc\rvert$	$\lvert\mathbb{Z}_q^*\rvert + 2\lvert\mathbb{G}_1\rvert$

then sends a deletion request to both the blockchain and the CS. In this manner, the proposed scheme enables file dynamics.

7 Theoretical analysis

We compared the functionality of the MKPDP with existing schemes [22–24] in Table 2. We mainly focused on the functionality in public auditing, searchable auditing, keyword privacy, multi-keyword, data privacy and file dynamics. All schemes satisfied the functionality of public auditing and searchable auditing. Except for scheme [24], other schemes satisfied the functionality of keyword privacy, in which a pseudorandom function was utilized to blind the keyword and the index matrix was blinded through encryption. Only our scheme supported the functionality of multi-keyword searchability, data privacy and file dynamics.

We compare the computation cost and communication cost in IndexGen, AuthGen, ProofGen, and ProofVerify algorithms among schemes [22–24].

7.1 Computation cost comparison

For convenient comparison, we adopted $H_{\mathbb{G}_1}$, $Mul_{\mathbb{Z}_q^*}$, $Mul_{\mathbb{G}_1}$, $Exp_{\mathbb{G}_1}$ to represent hash operation into group $\mathbb{G}_1$, multiplication operation into group $\mathbb{Z}_q^*$, multiplication operation into group $\mathbb{G}_1$ and exponentiation operation into group $\mathbb{G}_1$. We adopted $Pair$ to represent the pairing operation. Suppose that there are v files and each file contains n blocks. Suppose there were $|SE_{w_k}|$ files related to keyword w_k. Each file F_i is associated with t_i keywords. The total number of keywords was K. We overlooked some simple calculations like addtion, pseudo random permutation, and pseudo random function operations. The computation overhead in different algorithms was shown in Table 3.

For IndexGen algorithm, the computation overhead in [22] was the most biggest among other schemes, which was related to the number of separated sectors and the total number of keywords K. Similar to scheme [24], the computation overhead was related to the number of files and the total number of keywords K. The computation overhead in [23] was related to the size of $|SE_{w_k}|$ and our scheme was related to the total files v. For AuthGen algorithm, the computation overhead was all related to the number of total files v and the number of data blocks n. The computation cost was the same in scheme [22,23], which was $v \cdot n \cdot (H_{\mathbb{G}_1} + Mul_{\mathbb{G}_1} + Exp_{\mathbb{G}_1})$. The computation cost was the same in scheme [24] and ours, which was $v \cdot n \cdot (2H_{\mathbb{G}_1} + 2Exp_{\mathbb{G}_1} + 3Mul_{\mathbb{G}_1})$. For ProofGen algorithm, the computation cost was related to the size of $|SE_{w_k}|$ and the number of challenge blocks c. The computation cost was the least in scheme [24], which was $|SE_{w_k}| \cdot Mul_{\mathbb{G}_1} + (|SE_{w_k}| + c)Mul_{\mathbb{Z}_q^*} + c \cdot Exp_{\mathbb{G}_1}$. But their scheme suffered a potential security breach. The computation cost in our scheme was $|SE_{w_k}| \cdot c \cdot Mul_{\mathbb{G}_1} + (|SE_{w_k}| \cdot c + 1)Mul_{\mathbb{Z}_q} + (|SE_{w_k}| \cdot c + 1)Exp_{\mathbb{G}_1}$. Our scheme was a little bigger than scheme [24] but could resist a security breach. For ProofVerify algorithm, the computation cost was a little difference in scheme [22,23]. The computation cost in scheme [24] and our scheme was a little higher than scheme [22,23]. That was because the index matching process was conducted in the proof verification phase, which can guarantee the security during matching.

7.2 Communication cost comparison

We adopted $|F|$ to show the length of file to be uploaded, $|\mathbb{G}_1|$ to show the length of an element in $\mathbb{G}$, $|\mathbb{Z}_q^*|$ to show the length of an element in $\mathbb{Z}_q^*$, and $|n|$ to show the length of a constant. The computation overhead in different

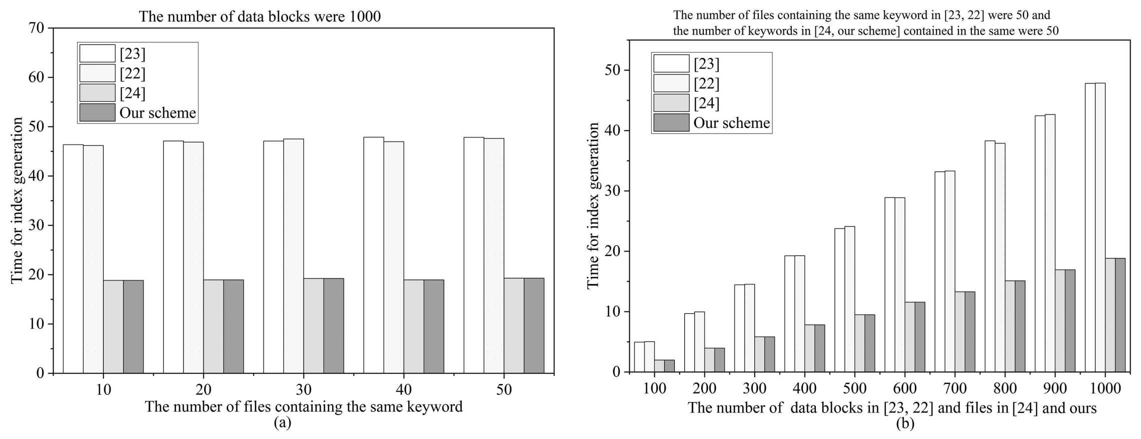

Figure 4 Time cost for index generation evaluation in (a) the number of files containing the same keyword and (b) the number of data blocks or files.

algorithms was shown in Table 4.

In the uploading phase, the DO transmitted the file data, the authenticator data or the index data to the CS. The communication cost was related to the total number of files v. Different from other schemes, the DO transmitted $\{\phi_i, \{m_{i,j}\}_{j\in[1,n]}\}_{i\in[1,v]}$ to the CS and the index information $\{FID_i, S_i, Addr, E_V,\ \Omega_i\}_{i\in[1,v]}$ to blockchain. The communication cost between the DO and the CS was $v\cdot|F|+(v\cdot n+v)|\mathbb{G}_1|$. In the delegation phase, the DO transmitted the auxiliary information or trapdoor to the TPA. In scheme [22], the DO transmitted the trapdoor $T_{w'}=\{\pi(w'), f(\pi(w'))\}$ to the TPA, and the communication cost was $2|\mathbb{Z}_q^*|$. In scheme [24], the DO transmitted $\{FID_i, S_i\}_{i\in[1,n]}$ to the TPA and the communication cost was $(s+1)\mathbb{Z}_q^*+s|n|$. In scheme [23], the DO transmitted the index table to the TPA, and the communication cost was $2\cdot v|\mathbb{Z}_q^*|$. The communication cost in our scheme was related to the number of queried keywords h and a little higher than other schemes, which was $2\cdot h|\mathbb{Z}_q^*|$. But our scheme improved the search flexibility. In the challenge phase, the TPA transmitted the challenge information to the CS. In scheme [22–24], the TPA transmitted the specific challenge index and the random value to the CS, so the communication cost was related to the challenge numbers c. In our scheme, the TPA transmitted the trapdoor and challenge seeds to the CS, so the communication cost was a constant value, which was $(2\cdot h+2)|\mathbb{Z}_q^*|+|c|$. In the proof generation phase, the CS transmitted the proof information to the TPA. The communication cost in these schemes was a little difference. Except that in scheme [24], the CS transmitted the index set to the CS. There was one more element communication cost than scheme [22, 23]. The element was utilized to protect the privacy of the auditing data.

8 Performance evaluation

We evaluated the computation cost through simulation the scheme. The simulation was conducted on Windows 10 equipped with an Intel i7 2.5 GHz CPU and 8 GB of memory, utilizing the JPBC library. We adopted Type-A pairings with a 512-bit length of each prime and a 160-bit group order. The cost of the scheme was mainly related to the number of blocks divided in the file, the file contained keywords, and the number of challenged data blocks. In the experiment, we compared our scheme in authenticator generation and index generation with schemes [22–24] since these schemes are the most related keyword-based provable data possession schemes. The specific test results were shown as follows.

The performance evaluation of index generation. In Figure 4(a), the number of files containing the same keyword was set from 10 to 50 with stepsize 10. The number of data blocks was set to 1000. The results showed that the time for index generation slowly increased with the number of files which contain the same keyword. For example, when the number of files containing the same keyword was 10 and 50, respectively, the time cost for our scheme was 18.831 and 19.287 s. It indicated that the file numbers containing the same keyword had little effect for the time cost for index generation. The results also showed that schemes [22, 23] had a similar computational cost. But the time cost in these two schemes was higher than scheme [24] and ours. Scheme [24] and ours had the same time cost. Since we adopted the same index structure in these two schemes. In Figure 4(b), the number of data blocks in [22, 23] was set from 100 to 1000 with stepsize 100, and the number of files which contain the same keyword was set to 50. The number of indices was related to that of files in scheme [22] and ours. Thus file numbers were set from 100 to 1000 with stepsize 100, and the number of keywords which contain in the same file

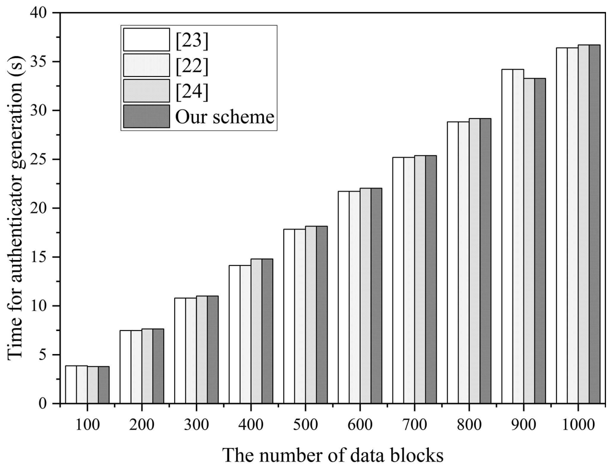

Figure 5 Computation cost of authenticator generation.

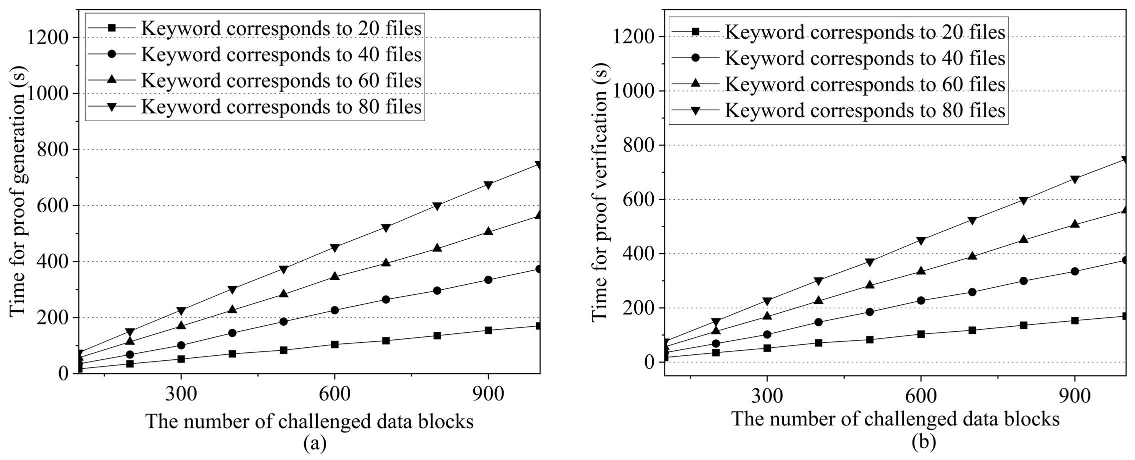

Figure 6 Time cost. (a) Proof generation; (b) proof verification.

was set to 50. The results showed that the computation cost was proportional to the number of data blocks or files. Schemes [22, 23] had a similar growing trend and there was no significant difference in the same file numbers. Our scheme performed well in terms of the computational overhead of index generation and did not bring additional overhead compared with existing schemes.

The performance evaluation of authenticator generation. In the experiment, the number of data blocks was set from 100 to 1000 with stepsize 100. Figure 5 showed that the computation cost of the authenticator grew linearly with the number of data blocks in each file. The results indicated that the computation cost was proportional to the number of data blocks in each file. For the same number of data blocks, the computation cost was a little difference. For example, when the number of data blocks was set to 500, the time cost of our scheme was 18.146 s. While it was 17.838 s in scheme [22, 24].

The performance evaluation of proof generation and verification. In the experiment, the number of the challenged data blocks was set from 100 to 1000 with stepsize 100. The number of keywords corresponding to files was set to 20, 40, 60, and 80, respectively. The computation cost of proof generation was shown in Figure 6(a). The results showed that the time for proof generation was proportionate to challenged data block numbers and also increased with the number of keywords corresponding to files. In Figure 6(b), the index generation of schemes [22, 23] was based on the number of file blocks, while that of scheme [24] and the scheme proposed in this paper was based on files. To enhance the comparability, we conducted settings according to the characteristics of different schemes. Specifically, we compared the index generation of scheme [22, 23] in terms of the number of file blocks, and compared that of scheme [24] and the scheme proposed in this paper in terms of the number of files. Similar results were conducted in the proof verification phase, as shown in Figure 6(b). The computation cost of proof generation had a similar increasing trend. Obviously, it is inevitable that both the proof generation time and verification time increase as the number of files contained in each keyword increases. The files contained

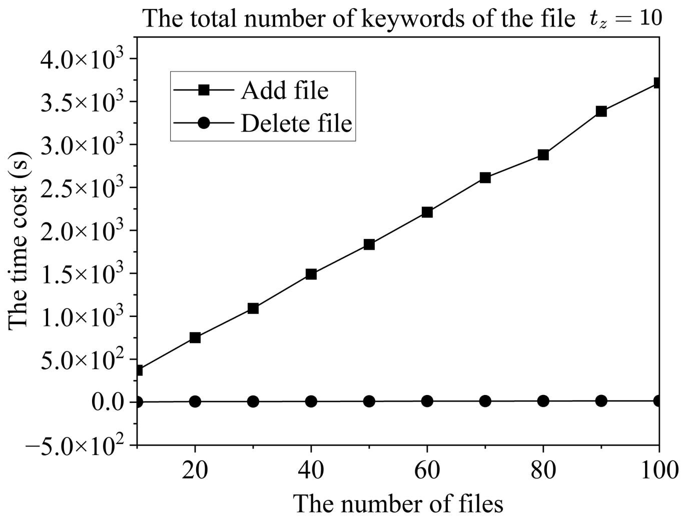

Figure 7 Computation cost of add and delete files.

Table 5 Gas estimation. In IndexGen, the DO sends $\{FID_i, S_i, Addr, E_V, \Omega\}_{i\in[1,v]}$ to the blockchain. In add file operation, the DO sends an add request $\{add, FID_z, S_z, E_{I'}, \Omega_z\}$ to the blockchain. In delete file operation, the DO sends $\{E_{I'}\}$ to the blockchain. Test date: May, 2024. Gas cost in posting the smart contract was about 981043. 1 Gwei = 10^{-9} Ether. 1 Ether = 2970.0234 USD.

Number of files	100		200		300		400		500		Average USD
Costs	Total gas	USD	Total gas	USD	Total gas	USD	Total gas	USD	Total gas	USD	
IndexGen	29006403	84.92	57059988	169.47	85120187	252.81	113186975	336.17	141260365	419.55	0.84
Add file	1095438	3.25	1958210	5.82	2821139	8.38	7092791	21.07	4547466	13.51	0.03
Delete file	863516	2.46	1601016	4.76	2338516	6.95	3076016	9.14	3813516	0.02	0.02

participated in the generation and verification of the proof.

The performance evaluation of add and delete files. In the experiment, the number of files was set from 10 to 100 with stepsize 10. The total number of keywords t_z was set to $t_z = 10$. As shown in Figure 7, the computation cost of add file grew linearly with the number of files. That was because updating a file requires updating the index matrix, re-encrypting the index matrix and generating the authenticator of the file, which was relatively expensive. While the computation cost of delete file required relatively little overhead and changed little as the number of files increases. That was because it only needed to update the index matrix and re-encrypt the index matrix.

The gas cost evaluation of add and delete files. In the experiment, we employed a test blockchain on Ethereum to assess the gas costs associated with add and delete file operations. The experiment was conducted on May, 2024, when the value of 1 Ether was approximately \$2970.0234. The cost results were shown in Table 5. To conduct the experiment, we developed a smart contract and deployed it on the test chain, incurring a deployment cost of approximately 981043 gas. We varied the number of files from 100 to 500 in increments of 100. Our analysis revealed a direct correlation between the number of files and the associated gas costs, with costs escalating as the number of files increased. Uploading the index information incurred the highest consumption of resources. While costs associated with adding and deleting files exhibited little difference. Because updating the index information of the file involved modifying matrix without the need for additional index tag information. Besides, we converted the gas into dollars to evaluate the economic feasibility. On average, the cost per file uploaded, added and deleted was approximately \$0.84, \$0.03, and \$0.02, respectively.

9 Future work

Our scheme achieves multi-keyword searchable provable data possession with file privacy protection and the integrity of the searching results. However, some aspects can be improved. (1) The proposed scheme supports file dynamics but cannot support block level dynamics. Since dynamic data at block level occurs frequently [36], we will explore how to achieve block level dynamics with searchable auditing. (2) The proposed scheme is not suitable for data integrity verification in group file sharing scenarios. Although many schemes have implemented group shared data auditing [37], these schemes are not directly used for searchable auditing. Future research will explore integrating group shared data auditing with searchable auditing.

10 Conclusion

This paper proposed a novel multi-keyword searchable provable data possession (MKPDP) scheme, which aimed to improve the accuracy and flexibility of the existing schemes and resist some attacks. MKPDP empowered DO to verify the integrity of files containing multiple keywords, thereby enhancing the accuracy of auditing files with the same attributes. Additionally, the integrity of the searched files was ensured with the support of blockchain technology. To resist pre-matching attacks, TPA instead of the CS itself involved in the matching process and subsequent verification procedures. This ensures that the CS cannot pass verification once it has used incomplete search information. Furthermore, to resist pre-aggregating attacks, random values were appended to all challenge files and their respective data blocks, ensuring the integrity of each data block involved in the auditing process. Security and efficiency were upheld through rigorous security analysis and comprehensive performance evaluation.

Acknowledgements This work was supported by National Key Research and Development Program of China (Grant No. 2021YFB2701300) and National Natural Science Foundation of China (Grant Nos. 62372044, U24B20146, 62232002).

References

1 Gai K, Guo J, Zhu L, et al. Blockchain meets cloud computing: a survey. IEEE Commun Surv Tutorials, 2020, 22: 2009–2030

2 Zhang Y, Gai K, Xiao J, et al. Blockchain-empowered efficient data sharing in Internet of Things settings. IEEE J Sel Areas Commun, 2022, 40: 3422–3436

3 Liu M, Pan L, Liu S. Cost optimization for cloud storage from user perspectives: recent advances, taxonomy, and survey. ACM Comput Surv, 2023, 55: 1–37

4 Ateniese G, Burns R, Curtmola R, et al. Provable data possession at untrusted stores. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, 2007. 598–609

5 Juels A, J. Kaliski S B. PORs: proofs of retrievability for large files. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, 2007. 584–597

6 Li A, Chen Y, Yan Z, et al. A survey on integrity auditing for data storage in the cloud: from single copy to multiple replicas. IEEE Trans Big Data, 2020, 8: 1428–1442

7 Han H, Fei S, Yan Z, et al. A survey on blockchain-based integrity auditing for cloud data. Digital Commun Netws, 2022, 8: 591–603

8 Huang Y, Shen W, Qin J, et al. Privacy-preserving certificateless public auditing supporting different auditing frequencies. Comput Secur, 2023, 128: 103181

9 Wang H, Liang J, Ding Y, et al. Ciphertext-policy attribute-based encryption supporting policy-hiding and cloud auditing in smart health. Comput Stand Inter, 2023, 84: 103696

10 Zhang Y, Bao Z, Wang Q, et al. OWL: a data sharing scheme with controllable anonymity and integrity for group users. Comput Commun, 2023, 209: 455–468

11 Li Y, Li Y, Zhang K, et al. Public integrity auditing for dynamic group cooperation files with efficient user revocation. Comput Stand Inter, 2023, 83: 103641

12 Li Y, Li Z, Yang B, et al. Algebraic signature-based public data integrity batch verification for cloud-IoT. IEEE Trans Cloud Comput, 2023, 11: 3184–3196

13 Guo Z, Zhang K, Wei L, et al. RDIMM: revocable and dynamic identity-based multi-copy data auditing for multi-cloud storage. J Syst Archit, 2023, 141: 102913

14 Gai C, Shen W, Yang M, et al. PPADT: privacy-preserving identity-based public auditing with efficient data transfer for cloud-based IoT data. IEEE Internet Things J, 2023, 10: 20065–20079

15 Tian M, Zhang Y, Zhu Y, et al. DIVRS: Data integrity verification based on ring signature in cloud storage. Comput Secur, 2023, 124: 103002

16 Goswami P, Faujdar N, Debnath S, et al. ZSS signature-based audit message verification process for cloud data integrity. IEEE Access, 2023, 11: 145485

17 Shen J, Shen J, Chen X, et al. An efficient public auditing protocol with novel dynamic structure for cloud data. IEEE Trans Inform Forensic Secur, 2017, 12: 2402–2415

18 Guo W, Qin S, Gao F, et al. Dynamic proof of data possession and replication with tree sharing and batch verification in the cloud. IEEE Trans Serv Comput, 2020, 15: 1813–1824

19 Liu D, Li Z, Jia D. Secure distributed data integrity auditing with high efficiency in 5G-enabled software-defined edge computing. Cyber Secur Appl, 2023, 1: 100004

20 Li Z R, Li Y, Lu L, et al. Blockchain-based auditing with data self-repair: from centralized system to distributed storage. J Syst Archit, 2023, 137: 102854

21 Bello S A, Oyedele L O, Akinade O O, et al. Cloud computing in construction industry: use cases, benefits and challenges. Autom Constr, 2021, 122: 103441

22 Gao X, Yu J, Chang Y, et al. Checking only when it is necessary: enabling integrity auditing based on the keyword with sensitive information privacy for encrypted cloud data. IEEE Trans Dependable Secure Comput, 2021, 19: 3774–3789

23 Xue J, Luo S, Deng Q, et al. KA: keyword-based auditing with frequency hiding and retrieval reliability for smart government. J Syst Archit, 2023, 138: 102856

24 Shen W, Gai C, Yu J, et al. Keyword-based remote data integrity auditing supporting full data dynamics. IEEE Trans Serv Comput, 2023, 17: 2516–2529

25 Li Y, Shen J, Ji S, et al. Blockchain-based data integrity verification scheme in AIoT cloud-edge computing environment. IEEE Trans Eng Manage, 2023, 71: 12556–12565

26 Liu S, Yao Y, Tian G, et al. A blockchain-based compact audit-enabled deduplication in decentralized storage. Comput Stand Inter, 2023, 85: 103718

27 Qi Y, Luo Y, Huang Y, et al. Blockchain-based privacy-preserving public auditing for group shared data. Intell Autom Soft Comput, 2023, 35: 2603–2618

28 Shakarami A, Ghobaei-Arani M, Shahidinejad A, et al. Data replication schemes in cloud computing: a survey. Cluster Comput, 2021, 24: 2545–2579

29 Xu Z, He D, Vijayakumar P, et al. Certificateless public auditing scheme with data privacy and dynamics in group user model of cloud-assisted medical WSNs. IEEE J Biomed Health Inform, 2021, 27: 2334–2344

30 Peng L, Yan Z, Liang X, et al. SecDedup: secure data deduplication with dynamic auditing in the cloud. Inf Sci, 2023, 644: 119279

31 Liu Z, Ren L, Feng Y, et al. Data integrity audit scheme based on quad Merkle tree and blockchain. IEEE Access, 2023, 11: 59263–59273

32 Zhang Q, Sui D, Cui J, et al. Efficient integrity auditing mechanism with secure deduplication for blockchain storage. IEEE Trans Comput, 2023, 72: 2365–2376

33 Song M, Hua Z, Zheng Y, et al. Blockchain-based deduplication and integrity auditing over encrypted cloud storage. IEEE Trans Dependable Secure Comput, 2023, 20: 4928–4945

34 Zhang Q, Zhang Z, Cui J, et al. Efficient blockchain-based data integrity auditing for multi-copy in decentralized storage. IEEE Trans Parallel Distrib Syst, 2023, 34: 3162–3173

35 Liu Z, Wang S, Liu Y. Blockchain-based integrity auditing for shared data in cloud storage with file prediction. Comput Netws, 2023, 236: 110040

36 Goswami P, Faujdar N, Singh G, et al. Stub signature-based efficient public data auditing system using dynamic procedures in cloud computing. IEEE Access, 2024, 12: 58502–58518

37 Miao Y, Gai K, Zhu L, et al. Blockchain-based shared data integrity auditing and deduplication. IEEE Trans Dependable Secure Comput, 2023, 21: 3688–3703