Blockchain-empowered Keyword Searchable Provable Data Possession for Large Similar Data

Ying Miao, Keke Gai, *Senior Member, IEEE,* Jing Yu, *Member, IEEE,* Yu'an Tan, Liehuang Zhu, *Senior Member, IEEE,* and Weizhi Meng, *Senior Member, IEEE*

***Abstract*—*Provable Data Possession* (PDP) is an alternative technique that guarantees the integrity of remote data. However, most current PDP schemes are inapplicable to similarity-like data checking with the same attribute, i.e., when there are numerous similar files to be checked by *Data Owners* (DOs). Some traditional models cannot resist the corrupt auditors who always generate biased challenge information. Besides, a copy-summation attack exists in some schemes, which means the *Cloud Server* (CS) can bypass the verification by storing the median value instead of initial data via summation operation. To address the issues above, in this work, we propose a keyword searchable PDP scheme for large similar data checking. To achieve searchability, we introduce the notion of a keyword in PDP and design a specific index structure to match the authenticator. The scheme enables all matched files to be auditable and verifiable, while guaranteeing privacy protections. Unlike existing methods, our *Third Party Auditor* (TPA) checks all similar data containing the same keyword simultaneously. We utilize unpredictable yet verifiable public information on the blockchain to generate challenge information, rather than relying on a centralized TPA. The proposed scheme can resist copy-summation attacks. Theoretical analysis demonstrates that the proposed scheme satisfies the security requirements, and our evaluations demonstrate its efficiency.**

***Index Terms*—Blockchain, Keyword search, Provable data possession, Similar data.**

I. Introduction

Cloud-based services have become a desirable alternative for acquiring on-demand services, due to their multiple merits, e.g., pay-as-you-go and wide resource access. Along with the expansion of clouds, it is observable that security and privacy concerns have been impacting the implementation of cloud systems in various dimensions, such as cloud storage. One of the major concerns is the lack of control for *Data Owners* (DOs), which may cause hazards in data integrity, e.g., downtime events. Data losses caused by clouds derive from various aspects, e.g., hardware/software failure and improper operations [1], [2]. As a remote check technique without downloading the entire dataset, *Provable Data Possession* (PDP) has been developed to ensure efficient data integrity

Y. Miao, K. Gai, Y. Tan and L. Zhu are with School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing, 100081, China (e-mail: {yingmiao, gaikeke, tan2008, liehuangz}@bit.edu.cn).

J. Yu is with the School of Information Engineering, Minzu University of China, Beijing, China, jing.emy.yu01@gmail.com.

W. Meng is with School of Computing and Communications, Lancaster University, United Kingdom. Email: weizhi.meng@ieee.org.

This work is partially supported by the National Key Research and Development Program of China (Grant No. 2021YFB2701300), National Natural Science Foundation of China (Grant No. 62372044).

Corresponding author: Keke Gai (gaikeke@bit.edu.cn)

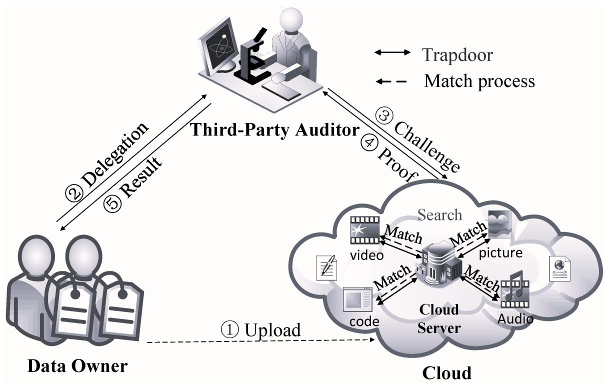


Fig. 1: Searchable provable data possession.

checks, considering reliable cloud service offerings [3]. In this paper, we mainly address two key issues in PDP.

On one hand, ensuring a fair challenge is a key issue that needs to be addressed. Prior studies have explored PDP schemes in multiple dimensions, e.g., privacy-preserving [4], dynamic PDP [5], multi-copy [6], and multi-cloud PDP [7]. In practice, a *Third Party Auditor* (TPA) is an alternative for assisting in the integrity check of outsourced data within an auditing model. A TPA generally executes **Challenge** and **Verify** algorithms in the model. To be specific, there are two major methods to generate challenge information. One method is that TPA selects two random seeds and sends them to the *Cloud Server* (CS) as the challenge; the other method is that TPA generates the challenge index set and sends it to the cloud server. We see that both challenge seeds and indices are selected by TPA, regardless of which method is applied. Unfortunately, the credibility of auditing cannot be guaranteed when TPA colludes with CS and generating improper challenge information, since PDP adopts a challenge-response and a randomly sampled pattern to ensure reliability. For example, when the corrupt data blocks ratio is 1%, it only needs to challenge 300 blocks to guarantee 99% credibility.

Another critical challenge in PDP schemes is ensuring integrity checks for data with similar attributes, particularly given the large volumes of data commonly generated in commercial and scientific contexts [13]. Users often prioritize the integrity of specific data categories, such as data related to particular diseases or specific time frames, since these focused datasets can yield more valuable insights. Therefore, robust integrity assurance for these data groups is essential. However, many existing schemes [17]–[19] do not fully address the

TABLE I: The Index Structure and a Partial Proof Information

Scheme	Index structure	A partial proof information
Gao et al. [8]	$\Omega_{w_k,j} = [(\sum_{i\in S_{w_k}} H_1(ID_i\|j)^{-1}) \times H_3(j) \times H_2(\pi(w_k)\|j)]^x$	$\mu = \sum_{i\in S_{w_k}} \sum_{j\in Q} c_{ij} \times v_j$
Xue et al. [9]	$\Omega_{w_k,j} = [H_2(Z)^{-1} \times H_3(\pi(w_k)\|j) \times \prod_{i\in S_{w_k}} (H_1(ID_i\|j)^{-1})]^{sk_u}$	$\mu = \sum_{i\in S_{w_k}} \sum_{j\in Q} c_{ij} \times v_j$
Xue et al. [10]	$\Omega_{w_k,j} = [H_2(Z)^{-1} \times H_3(\pi(w_k)\|j) \times \prod_{i\in S_{w_k}} (H_1(ID_i\|j)^{-1})]^{ssk}$	$\mu = \sum_{i\in S_{w_k}} \sum_{j\in Q} c_{ij} \times v_j$

w_k: keyword; S_{w_k}: the file set containing keyword w_k; $\pi(w_k)$: pseudo-random function; Z: the number of file updates; ID_i: unique filename. j: data block identifier; Q: challenge set; c_{ij}: the data blocks; v_j: the random number.

integrity verification needs for large datasets with similar attributes. Most current PDP schemes focus on verifying individual files rather than performing broad integrity checks across data sharing common characteristics [11], [12]. These schemes typically require users to inform the TPA of detailed file information before each integrity check, resulting in redundant communication and reduced efficiency. Additionally, when DO lacks precise knowledge of the total data volume in the database, it may lead to incomplete or ineffective audits. One promising solution to this issue is the use of keyword-based search mechanisms. In this approach, users can extract keywords from the stored files and associate them with tags (see Fig. 1), allowing for efficient integrity verification across files with shared attributes.

However, there are some challenges that need to overcome. (i) Keywords generally have a high likelihood of carrying sensitive information, so that preventing keywords from leaking to TPA and CS during the searching and auditing phase is a challenge. (ii) Finding out the method of maintaining the consistency of audit data between TPA and CS sides is a challenging issue. In general, a DO only transmits a trapdoor to TPA whose primary role is performing integrity audits. Unfortunately, TPA can hardly distinguish which files contain a particular keyword or the quantity of files with keywords. Thus, mistrustful CS may utilize incomplete search results to generate the proof while passing verification. (iii) Constructing a secure index structure is a tough task because the quantity of the required index is great. In order to achieve the searchable auditing, the index is used for matching files and participated in the auditing, and only matched files can be audited and verified. One keyword corresponds to one or a few file(s); therefore, each matched file needs an index to achieve all corresponding files to participate in the auditing.

In order to eliminate the workload of the index, existing works [8]–[10] tried adopting aggregated indexes, which easily susceptible to a summation attack. TABLE I lists the index structures in sample schemes. We describe potential attacks as well as attacking mechanisms here. In the proof generation phase, the proof information generated by CS is $P = \{\mu = \sum_{i\in S_{w_k}} \sum_{j\in Q} c_{ij} \times v_j\}$. We observe that for $k \in [1, s]$, the summation $\sum_{i\in S_{w_k}} \sum_{j\in Q} c_{ij} \times v_j$ can be handled by CS to reduce storage cost. CS can easily deceive the verifier in the following ways:

$$\sum_{i\in S_{w_k}} \sum_{j\in Q} c_{ij} \times v_j = \sum_{j\in Q} v_j (\sum_{i\in S_{w_k}} c_{ij}).$$

As shown in Fig. 2, the CS can store only the summation of the values $\{\sum_{i\in S_w} c_{ij}\}_{j\in[1,s]}$ rather than original data $\{c_{ij}\}_{i\in S_w, j\in[1,s]}$. In this way, CS can pass the verification as all summation files will be checked at each time; thus, CS not only saves storage overhead but also pass the verification without possessing the original data. In general, cloud storage services are available by the service payment [14]. For less frequently used data, CS adopts this method to reduce storage overhead so that summation attacks must be resisted.

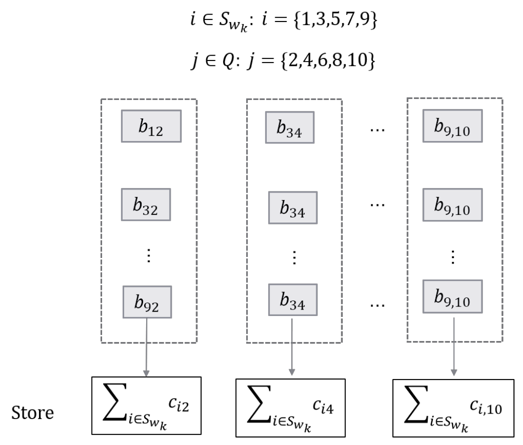

$i \in S_{w_k}$: $i = \{1,3,5,7,9\}$
$j \in Q$: $j = \{2,4,6,8,10\}$
b_{12}
b_{32}
b_{92}
b_{34}
b_{34}
b_{34}
$b_{9,10}$
$b_{9,10}$
$b_{9,10}$
Store
$\sum_{i\in S_{w_k}} c_{i2}$
$\sum_{i\in S_{w_k}} c_{i4}$
$\sum_{i\in S_{w_k}} c_{i,10}$

Fig. 2: Copy-summation attack from the same keyword files.

In order to address challenges mentioned above, we explore to find out the solution to achieving decentralized challenge information generation and keyword searchable provable data possession. The main contributions of this paper are listed as follows.

Our Contributions:

- In order to improve the auditing dimension and achieve similar data integrity checking, we propose a new keyword searchable provable data possession, which can effectively verify the data integrity based on keywords. In contrast to earlier approaches, this scheme empowers the DO and CS to scrutinize all data linked to a particular keyword. CS's proof is considered valid by TPA only if it successfully demonstrates that all files associated with the same keyword have been accurately stored, setting it apart from previous methods.
- To achieve similar data integrity checking, we introduce the notion of keyword and design an index structure to match the authenticator. All matched files can be audited and verified. Privacy can be guaranteed at the same time. Besides, a copy-summation attack was identified. We give the detailed attack process and provide the solution to resist this attack.
- To resist the corrupt auditor from generating biased chal-

lenge information, we utilize unpredictable but verifiable public information on the blockchain to generate challenge information instead of relying on the centralized TPA, which can achieve decentralized auditing.

- Security analysis demonstrates that the proposed scheme can achieve auditing soundness. Experiments show that it is efficient in authenticator generation and index generation. Experiments also show that the verification time does not increase much with the number of files.

In the rest of this paper, we organize it as follows. We start by reviewing related work in Section II, providing context for research status. Section III covers the basic knowledge and definitions to understand our work. Our core contributions are discussed in Section IV, where we detail our proposed scheme. Section V assesses its security aspects, and Section VI evaluates its performance. Finally, in Section VII, we make a conclusion and list the future explore direction of the proposed scheme.

II. Related Work

Provable Data Possession (PDP) [3] and *Proof of Retrievability* (PoR) [15] are two fundamental techniques for verifying the integrity of data stored remotely in cloud services. These technologies aim to ensure data integrity without obtaining the entire dataset, thus enhancing efficiency and reducing bandwidth usage. Specifically, PoR recovers complete data from partial data; while PDP detects data corruptions by random sampling. For the convenience of the data owner, Wang *et al.* [16] developed a TPA in the PoR model to assist the auditing check, leading to the proposal of many PDP/PoR schemes in subsequent work.

To prevent data from leakin to the honest-but-curious TPA, some prior explorations have tried a variety of auditing methods, such as combining homomorphic linear authenticators with masking techniques to prevent the TPA from deducing the actual data content [17]–[19]. Blinding is an optional solution for conditional anonymity to protect privacy [20], [21]. Using IBC technology has been examined for achieving data integrity checking in order to optimize the certificate management of *Public Key Infrastructure* (PKI) [22]. For example, Li *et al.* [23] applied this method in the context of maritime transportation systems via adopting identity-based aggregate signatures. Another type of scheme aims to utilize certificate-less aggregate signatures to data integrity verification. For instance, [24] used certificate-less signatures to design an integrity checking method tailored for multi-copy data, thereby reducing the risk of data loss. [25] tried a certificate-less signature-based method to enable batch auditing with a designated verifier. In response to the emerging threat of quantum attacks, Tian *et al.* [26] turned to *Learning With Errors* (LWE)-based ring signatures as means to ensure secure data auditing. Similarly, [27] combined blind signature with code-based proofs of retrievability for data auditing. We see that most existing PDP and PoR schemes rely on the centralized TPA, meaning that semi-honest TPAs may collude with CS and generate biased challenge information or lead to data loss.

Some attempts have focused on using blockchain technology to address aforementioned issues [28], [29]. To prevent TPA from generating biased challenge information, prior studies utilized some features of blockchain networks, such as using block hashes as seeds for challenge information creation. The drawback of this type of scheme was that the TPA still controlled the challenge information, as the timestamp is selected by TPA to obtain random and public information. To resist collusion, [30] was a two-party approach that combined public information from blockchain with the random challenge seed from a TPA for generating the challenge information. Some prior explorations utilized blockchain smart contracts to record the auditing results in improve the credibility of TPAs [31]–[33]. For example, [34] merged smart contracts with binary search to ensure faults locations in multiple copies and heterogeneous clouds. Recent studies also had examined various perspectives, such as using smart contract for guaranteeing fairness in privacy auditing [35], establishing punishment mechanism for avoiding malicious parties [36], and using decentralized ledger to achieve data deduplication in information sharing [37], [38].

Maintaining the integrity of similar data is a challenging task, as auditing each piece of data incurs a significant overhead. Gao *et al.* [8] presented a keyword-based auditing approach to extend the coverage of the audited data. Subsequently, [10] incorporated a privacy-preserving keyword-file index table to safeguard file privacy. [9] adopted a bloom filter for fuzzy matching and maintaining an index table with update timestamps for a broader range of file auditing. Unfortunately, relying heavily on PKI requires the management of digital certificates, resulting in higher complexities and overhead in the system. We improved it according to certificateless cryptography.

III. Preliminaries and Definitions

1) Bilinear map: A bilinear $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$ satisfies three properties:
   - Computability: Given $\forall u, v \in \mathbb{G}_1$, it is easy to compute $e(u, v)$.
   - Non-degeneracy: $\exists u, v \in \mathbb{G}_1$, $e(u, v) \neq 1_{\mathbb{G}_T}$.
   - Bilinearity: For any $\forall a, b \in \mathbb{Z}_q^*$ and $\forall u, v \in \mathbb{G}_1$, the equation $e(u^a, v^b) = e(u, v)^{ab}$ holds.
2) Computational Diffie-Hellman (CDH) Assumption: Given $(g, g^a, g^b) \in \mathbb{G}_T$, $a, b \in \mathbb{Z}_q^*$ and $g \in \mathbb{G}_1$, and for any probabilistic polynomial time (PPT) algorithm $\mathcal{A}$, the advantage of $\mathcal{A}$ to solve the CDH problem is negligible, defined as

$$Adv_{\mathcal{A}}^{CDH} = Pr[\mathcal{A}(g, g^a, g^b) \to g^{ab}] \leqslant \varepsilon.$$

A. Basic notations

Some main notations and its meaning utilized in this paper are presented in TABLE II.

B. System Model

As depicted in Fig. 3, the system comprises four key entities: the DO, CS, TPA, and blockchain.

1) A DO intends to outsource multiple files to CS for saving storage space. To guarantee the integrity of similar files,

TABLE II: Main Notations and Its Meanings

Notation	The meaning
DO	Data owner
TPA	Third party auditor
CS	Cloud server
$e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$	A bilinear map
(α, P_0)	The master secret key and public key
sk_{ID}	The private key
H_1, H_2, H_3, H_4	The hash function
$f_{k_1}(\cdot), \psi_{k_2}(\cdot), \pi_{k_3}(\cdot)$	The pseudo random permutation and the pseudo random function
$\{a_k\}_{k=1}^{s}, \{A_k\}_{k=1}^{s}$	The secret value and its public public value
$W = \{w_1, w_2, \cdots, w_m\}$	The keyword set
$V = \{v_{w_1}, v_{w_2}, \cdots, \cdots, v_{w_m}\}$	The index matrix
$c_w = [n_{w_1}, n_{w_2}, \cdots, n_{w_m}]$	The keyword number vector
$F = \{F_1, F_2, \cdots, F_v$	The file set
σ_{ij}	The authenticator for block b_{ij}
$T_w' = \{\pi_o(w'), f_l(\pi_o(w'))\}$	The trapdoor
$t_{start}, t_{end}, \triangle t$	The timestamp of start and end and its interval
$\mathcal{T}$	The interval of successive blocks
$B(t)$	The blockhash within the timestamp t
$\mathcal{B}_i = \{B(t)\}$, $t \in [t_{start} + (i - 1) \triangle t, t_{start} + i \times \triangle t]$	The blockhash set
k_1, k_2	The random challenge seed
$\{C = \{(j, v_{ij})\}\}_{j \in [1,c], i \in [1,v]}\}$	The challenge information
$Proof = \{T, \{\mu_k\}_{1 \leqslant k \leqslant s}\}$	The proof information

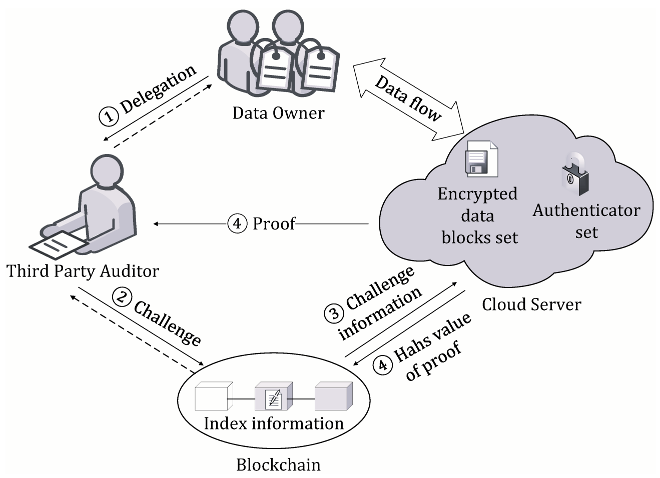


Fig. 3: System model.

DO extracts the keywords and generates a secure index matrix. DO outputs the secure index matrix, encrypted blocks set and the authenticator set to CS.

2) CS: CS provides enough storage service. CS is obliged to respond to challenges from the TPA to demonstrate the data integrity. However, it's important to note that CS, while typically trusted, may have an incentive to conceal data loss to maintain its reputation.
3) TPA: To alleviate the computational burden on DO, it's common practice for DO to delegate an independent entity to assist with integrity checking. TPA is semi-trust, it may collude with CS and always generate bias challenge information.
4) Blockchain: The blockchain is a decentralized entity. It provides public, decentralized and unforgeable information.

DO stores its data information, index information and authenticator in the cloud. Then, DO make a delegation for TPA to conduct data integrity verification through blockchain. Upon receiving the authorization, TPA initiates a challenge via smart contract. Then, CS generates the corresponding proof, sends the specific proof information to TPA, and hashes the proof information to blockchain. Finally, TPA retrieves challenge information and index information from blockchain, performs verification and informs the DO of the verification result.

C. Definition

Definition 1. *A blockchain-based and keyword searchable provable data possession scheme contains nine algorithms:*

1) $(pp, \alpha, P_0) \leftarrow SysIni(\lambda)$: When provided with the security parameter λ as input, this process generates and produces the following outputs: the system parameters pp, the master secret key α and the master public key P_0.
2) $(x, R, \{a_k\}_{k=1}^{s}, \{A_k\}_{k=1}^{s}, W, V) \leftarrow Setup(F)$: When provided with the file F as input, this process generates the following outputs: secret values x and public values R and $\{A_k\}_{k=1}^{s}$, along with the keyword set W and the index vector set V.
3) $sk_{ID} \leftarrow KeyGen(ID, \alpha)$: When provided with both the identity ID and the master secret key α as inputs, this process yields the private key sk_{ID} as its output.
4) $(I, \Omega) \leftarrow IndexGen(x, W, V)$: When provided with a random value x, the keyword set W and the index matrix V as input, this process produces the encrypted index matrix I and the matched index authenticator set Ω as its output.
5) $\Phi \leftarrow AuthGen(F, x, sk_{ID})$: When provided with the file F, a random value x, and the private key sk_{ID} as input, this process generates the authenticator set Φ as its output.
6) $T_{w'} \leftarrow TrapdoorGen(w')$: When provided with the keyword w' as input, this process generates the search trapdoor T_w' as its outputs.
7) $Chal \leftarrow ChalGen(Chal, I, \Omega, F, \Phi)$: When provided with the search trapdoor $T_{w'}$, the encrypted index matrix I, the matched index authenticator set Ω, file set F and the authenticator set Φ as input, this process produces the challenge information $Chal$ as output.
8) $Proof \leftarrow ProofGen(Chal, I, \Phi)$: When provided with the challenge $Chal$, the encrypted index matrix I, and the authenticator set Φ as input, this process generates the auditing proof $Proof$ as its outputs.
9) $(0, 1) \leftarrow ProofVerify(Proof)$: When provided with the auditing proof $Proof$ as input, this process generates the auditing result as its output.

D. Security Model

The security game is established as an interaction between two parties: a simulator $\mathcal{S}$ and the adversary $\mathcal{A}$.

E. Unforgeability of Authenticators

SysIni, Setup: The simulator $\mathcal{S}$ initiates the process by executing the $SysIni$ and $Setup$ algorithms, thereby obtaining public parameters $(pp, P_0 = g^a, R = g^x, \{A_k\}_{k=1}^s, W, v)$. These parameters are then transmitted to the adversary $\mathcal{A}$, while $\mathcal{S}$ maintains the confidentiality of secret values $(a, x, \{a_k\}_{k=1}^s)$.

User Creation: $\mathcal{S}$ selects the challenge set by randomly tossing a coin randomly.

Queries: $\mathcal{A}$ poses queries to $\mathcal{S}$, and $\mathcal{S}$ responds to these queries.

1) $Hash\,Query$: $\mathcal{A}$ makes hash queries to $\mathcal{S}$. $\mathcal{S}$ responds to the corresponding hash results.
2) $Key\,Query$: When $\mathcal{A}$ requests the private key for a specific ID from $\mathcal{S}$, $\mathcal{S}$ employs the $KeyGen$ algorithm to produce the private key sk_{ID} and provides it in response to $\mathcal{A}$.
3) $Authenticator\,Query$: When $\mathcal{A}$ adaptively chooses any block and requests the authenticator tag for the identity ID from $\mathcal{S}$, $\mathcal{S}$ utilizes the $AuthGen$ algorithm to acquire the authenticator tag, which is then provided in response to $\mathcal{A}$.

Forge: $\mathcal{A}$ can produce a forged authenticator σ'_{ij} for block b'_{ij} associated with user identity ID'. $\mathcal{A}$ is declared the winner of the game, when the forged authenticator successfully passes the verification with the following conditions holding:

1) $(ID', \sigma'_{ij}, b'_{ij})$ passes the verification.
2) $\mathcal{A}$ has never conducted an authenticator query on $(ID', \sigma'_{ij}, b'_{ij})$.

Definition 2. *The proposed scheme ensures the unforgeability of authenticators. For any adversary $\mathcal{A}$ that operates within probabilistic polynomial time (PPT), the probability of $\mathcal{A}$ winning the game is negligible.*

F. Unforgeability of Proof

The **SysIni, Setup, Queries** phases are the same as in definition III-E, we omit it here.

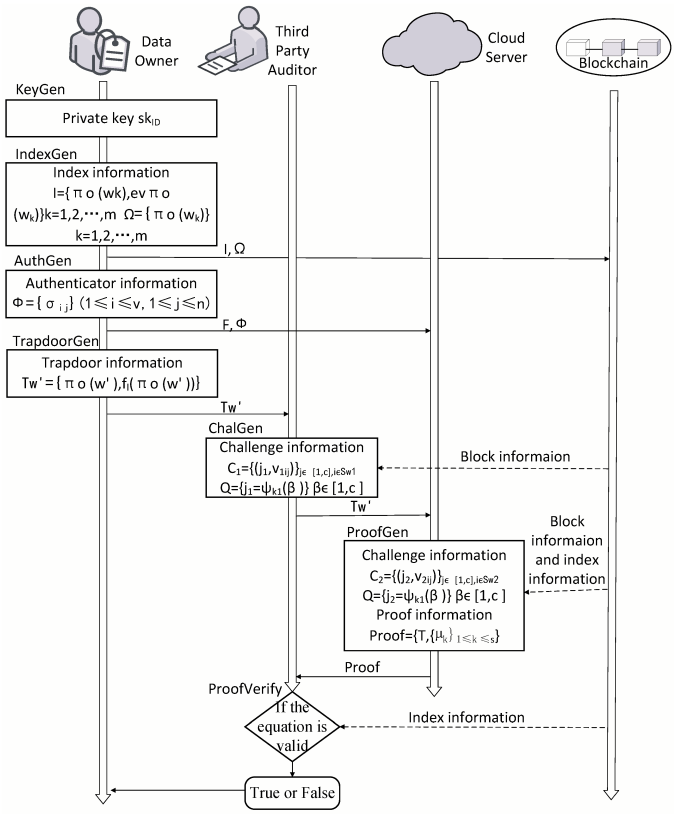
TrapdoorGen: $\mathcal{S}$ generates a search trapdoor $T_{w'} = \{\pi_o(w'), f_l(\pi_o(w'))\}$ and sends it to $\mathcal{A}$.

ChalGen: $\mathcal{A}$ generates and sends the challenge information $(T_{w'}, \{j, v_j\}_{j \in \tilde{Q}})$ to $\mathcal{S}$.

ProofGen: $\mathcal{S}$ generates the proof information based on the challenge information.

Forge: $\mathcal{A}$ produces a forged proof $Proof' = \{T', \{\mu_k\}\}$. $\mathcal{A}$ is declared the winner of the game, if the forged proof passes the verification with the following conditions holding: the proof information is based on the challenge information $(T_{w'}, \{j, v_j\}_{j \in \tilde{Q}})$, which is not queried before.

Definition 3. *The proposed scheme ensures security against semi-trusted CS. For any adversary $\mathcal{A}$ that operates within probabilistic polynomial time (PPT), the probability of $\mathcal{A}$ winning the game is negligible.*



Data Owner
Third Party Auditor
Cloud Server
Blockchain
KeyGen
Private key sk_{ID}
IndexGen
Index information
$I=\{\pi o(wk), ev\ \pi o(w_k)\}k=1,2,\cdots,m$ $\Omega=\{\pi o(w_k)\}$ $k=1,2,\cdots,m$
I, Ω
AuthGen
Authenticator information
$\Phi=\{\sigma_{ij}\}$ $(1\leqslant i\leqslant v, 1\leqslant j\leqslant n)$
F, Φ
TrapdoorGen
Trapdoor information
$Tw'=\{\pi o(w'), f_l(\pi o(w'))\}$
Tw'
ChalGen
Challenge information
$C_1=\{(j_1, v_{1ij})\}_{j\in[1,c], i\in Sw1}$
$Q=\{j_1=\psi_{k1}(\beta)\}$ $\beta\in[1,c]$
Block informaion
Tw'
ProofGen
Challenge information
$C_2=\{(j_2, v_{2ij})\}_{j\in[1,c], i\in Sw2}$
$Q=\{j_2=\psi_{k1}(\beta)\}$ $\beta\in[1,c]$
Proof information
$Proof=\{T, \{\mu_k\}_{1\leqslant k\leqslant s}\}$
Block informaion and index information
Proof
ProofVerify
If the equation is valid
Index information
True or False

Fig. 4: Data flow of the scheme.

G. Design Goals

The proposed scheme needs to satisfy the following security goals.

1) **Keyword searchable auditing**: to make sure that all files containing the same keyword are audited while DO needs not to provide these file information to TPA and CS, and the auditing correctness can be guaranteed at the same time.
2) **Decentralized challenge information generation**: to generate challenging information in a decentralized way instead of relying on the centralized TPA who may collude with CS to generate biased challenge information.
3) **Privacy preservation**: to prevent TPA and CS from guessing the content of the files according to the keyword and the trapdoor information.
4) **Resist the same keyword files summation attack**: to avoid the same keyword file summation attack performed by CS. The attack is that CS only stores the file summation of the same keyword instead of the original file to save storage but can still pass the auditing verification.

IV. Proposed Construction

A. High-level Overview

The proposed scheme operates through four main phases: data preparation, challenge, proof generation, and verification. In the data preparation phase, authenticators are generated for each data block of the stored files, while keywords are extracted from the data to create index information. The DO then uploads the data along with the authenticator information to the cloud, and stores the index information on the blockchain. In the challenge phase, both the DO and TPA participate; the DO first generates trapdoor information and sends it to the TPA, which then creates specific challenge information and forwards it to the CS. During the proof generation phase, the CS processes the challenge information to retrieve relevant data, constructs the proof information, and sends it back to the TPA. Finally, in the verification phase, the TPA verifies the proof information provided by the CS.

To better understand the whole scheme, the data interaction of the whole scheme is shown in Fig. 4. This flowchat illustrates the workflow of data integrity auditing process. The process begins at $KeyGen$ and through a series of steps to reach auditing result. Initially, step A involves $KeyGen$, $IndexGen$, $AuthGen$, $TrapdoorGen$, the resulting intermediate values are used for storage on cloud, blockchain and TPA. Followed by step A, TPA and CS conducts the auditing process through step B $ChalGen$, step C $ProofGen$ and step D $ProofVerify$. The arrows indicate data transmition during each step. The dashed line is the data to get from other entity.

Algorithm 1 IndexGen

```
Require: The secret key $x$, the keyword set $W$, the index vector set $V$
Ensure: The index $I$ matrix
 1: for Each $w_k \in W (1 \leqslant k \leqslant m)$ do
 2:     Extract $v_{w_k}$ from $V$
 3:     Compute $\pi_o(w_k)$
 4:     Compute $ev_{\pi_o(w_k)} = v_{w_k} \oplus f_l(\pi_o(w_k))$
 5:     Initiate an empty set $S_{w_k} = \emptyset$
 6:     for Each $i \in [1, v]$ do
 7:         if $v_{w_k}[i] == 1$ then
 8:             Insert $i$ to set $S_{w_k}$
 9:         end if
10:     end for
11:     for Each $i \in S_{w_k}$ do
12:         for Each $1 \leqslant j \leqslant n$ do
13:             Compute $\Omega_{w_k,ij} = [H_3(ID_i||j)^{-1} \times H_4(\pi_o(w_k)\|j)]^x$
14:         end for
15:     end for
16: end for
17: Set $\Omega_{\pi_o(w_k)} = \{\Omega_{w_k,i1}, \Omega_{w_k,i2}, \cdots, \Omega_{w_k,in}\}_{i, v_{w_k}[i]=1}$
18: return $I = \{\pi_o(w_k), ev_{\pi_o(w_k)}\}_{k\in[1,m]}$
19: and $\Omega = \{\Omega_{\pi_o(w_k)}\}_{k\in[1,m]}$
```

B. Our Scheme

1) $(pp, \alpha, P_0) \leftarrow SysIni(\lambda)$
This algorithm is conducted by KGC and aimes to generate the system public parameters $pp = (e, H_1, H_2, H_3, H_4, f(\cdot), \pi(\cdot))$, master secret/public key pair (α, P_0), and some secret keys (k_1, k_2, k_3).
   a) *The Key Generation Center* (KGC) defines the system parameters pp as follows: it involves two multiplicative cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_T$ with q-order, establishes a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$, incorporates four hash functions $H_1, H_2, H_3, H_4 : \{0,1\}^* \rightarrow \mathbb{G}_1$ for various purposes, employs a *Pseudo-Random Permutation* (PRP) denoted as $f_{k_1}(\cdot) : \{0,1\}^* \rightarrow [1, v]$ with a key $k_1 \in \mathbb{Z}_q^*$, and a PRP denoted as $\psi_{k_2}(\cdot) : \{0,1\}^* \rightarrow [1, n]$ with a key $k_2 \in \mathbb{Z}_q^*$. Additionally, it utilizes a *Pseudo-Random Function* (PRF) denoted as $\pi_{k_3}(\cdot) : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$ with a key $k_3 \in \mathbb{Z}_q^*$. Note that keys K_1, k_2 and k_3 are generated in a random way in the $ChalGen$ and $ProofGen$ algorithms.
   b) KGC randomly chooses the master secret key α from the set $\mathbb{Z}_q^*$ and computes the master public key as $P_0 = g^\alpha$. The secret key α is utilized to generate identity key for ID.

2) $(x, R, \{a_k\}_{k=1}^s, \{A_k\}_{k=1}^s, W, V) \leftarrow Setup(F)$
This algorithm is conducted by DO and produces a secret value x for generating matched index authenticators and data block authenticators, public values $R, \{A_k\}_{k=1}^s$ for data block authenticator verification, a keyword set W for file classification, and an index matrix V for fast search.
   a) DO splits each file into n blocks and distributes them across s sectors.
   b) DO randomly selects a value x from the set $\mathbb{Z}_q^*$ and generates $R = g^x$. While keeping x secret, the DO publicly shares the value R.
   c) DO randomly selects s values $a_k \in \mathbb{Z}_q^*$ and keeps them secretly. Then, the DO generates s public values $\{A_k = g^{a_k}\}_{k=1}^s$ and publishes them. The public values are utilized to generate authenticators, and these values can be precomputed in advance.
   d) DO identifies and extracts all keywords from the content, subsequently forming the keyword set $W = \{w_1, w_2, \cdots, w_m\}$.
   e) For each keyword w_k, DO initiates a binary index vector v_{w_k} of length v with all elements set to 0. For each file F_i containing the keyword w_k, DO writes the *i-th* bit of the index vector to 1: $v_{w_k}[i] = 1$.
   f) All the individual index vectors v_{w_k} are collectively assembled to create the index matrix as $V = \{v_{w_1}, v_{w_2}, \cdots, v_{w_m}\}$. The index matrix is utilized as auxilary information to achieve searchability and the matched index content.

3) $sk_{ID} \leftarrow KeyGen(ID, \alpha)$
This algorithm is conducted by DO and produces a private key for index generation and authenticator generation.
KGC computes the private key for the user with identity ID as $sk_{ID} = H_1(ID)^\alpha$. The identity key is utilized to generate authenticators.

4) $(I, \Omega) \leftarrow IndexGen(x, W, V)$
This algorithm is conducted by DO and produces the encrypted index matrix $I = \{\pi_o(w_k), ev_{\pi_o(w_k)}\}_{k=1,2,\cdots,m}$ and the matched index authenticator set $\Omega = \{\Omega_{\pi_o(w_k)}\}_{k=1,2,\cdots,m}$ as its output. $\pi_o(w_k)$ represents the blind row in I. To protect the privacy of index matrix, each row in the index matrix is encrypted as $ev_{\pi_o(w_k)}$. Each row in the index matrix represents a file that contains a certain keyword. Each column in the index matrix represents all keywords contained in a file. The matched index content is involved in the matching process. The index matrix is utilized to assist TPA and CS in locating

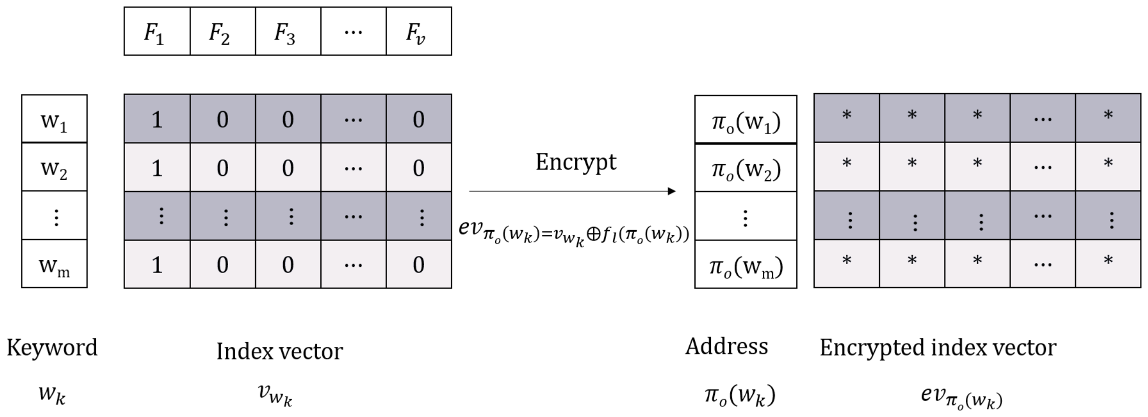

F_1 F_2 F_3 ⋯ F_v
w_1 w_2 ⋮ w_m
1 0 0 ⋯ 0
Encrypt
$ev_{\pi_o(w_k)}=v_{w_k}\oplus f_l(\pi_o(w_k))$
$\pi_o(w_1)$ $\pi_o(w_2)$ ⋮ $\pi_o(w_m)$
Keyword w_k
Index vector v_{w_k}
Address $\pi_o(w_k)$
Encrypted index vector $ev_{\pi_o(w_k)}$

Fig. 5: Index structure.

files that need to be checked.

As shown in Fig. 5 and Algorithm 1, DO generates the index structure as follows:

a) For every keyword w_k in the keyword set W, DO calculates $\pi_o(w_k)$ as the address assigned to each row within the secure index. Here, $o \in \mathbb{Z}_q^*$ is a pseudo-random permutation key selected by DO.
b) For each keyword w_k within the set W, DO performs encryption on the index vector, resulting in $ev_{\pi_o(w_k)} = v_{w_k} \oplus f_l(\pi_o(w_k))$. In this expression, $l \in \mathbb{Z}_q^*$ represents a pseudo-random function key selected by DO.
c) For each keyword $w_k \in W$, DO initializes an empty set $S_{w_k} = \emptyset$. If $v_{w_k}[i] = 1$ for any $i \in [1, v]$, DO adds the corresponding file index i to the set S_{w_k}. The total number of elements satisfying $v_{w_k}[i] = 1$ is denoted by ξ.
d) For each keyword $w_k \in W$, DO generates the matched index authenticators $\Omega_{\pi_o(w_k)} = \{\Omega_{w_k,i1}, \Omega_{w_k,i2}, \cdots, \Omega_{w_k,in}\}_{i,v_{w_k}[i]=1}$, where $\Omega_{w_k,ij} = [H_3(ID_i\|j)^{-1} \times H_4(\pi_o(w_k)\|j)]^x$, $j \in [1, n]$.
e) DO sets the encrypted index matrix as $I = \{\pi_o(w_k), ev_{\pi_o(w_k)}\}_{k=1,2,\cdots,m}$ and the matched index authenticator set as $\Omega = \{\Omega_{\pi_o(w_k)}\}_{k=1,2,\cdots,m}$. The encrypted index matrix I is utilized to for searchiability. The matched index authenticator set Ω is utilized to match data block authenticators.

5) $\Phi \leftarrow AuthGen(F, x, sk_{ID})$
This algorithm is conducted by DO and produces authenticators for all data blocks.
As shown in Algorithm 2, the authenticator is generated as follows:

a) Suppose the file set $F = \{F_1, F_2, \cdots, F_v\}$ and each file $F_i \in F$ has a unique identity ID_i. DO splits each file $F_i, i \in [1, v]$ into n blocks, i.e., $F_i = \{m_{i1}, m_{i2}, \cdots, m_{in}\}$.
b) DO creates a blinded version of each block, denoted as b_{ij}, using the formula $b_{ij} = m_{ij} + H_2(sk_{ID}\|ID_i\|i\|j)$, where $i \in [1, v]$ and $j \in [1, n]$. DO can subsequently recover the original plaintext m_{ij} for each block as $m_{ij} = b_{ij} - H_2(sk_{ID}\|ID_i\|i\|j)$.
c) Each block b_{ij} undergoes further division into s sectors, resulting in a set $\{b_{ijk}\}_{k\in[1,s]}$.
d) For every encrypted data block b_{ij}, DO calculates the authenticator σ_{ij} using the formula $\sigma_{ij} = sk_{ID} \times [H_3(ID_i\|j) \times g^{\sum_{k=1}^{s} a_k b_{ijk}}]^x$.
e) The DO sets the data block authenticator set as $\Phi = \{\sigma_{ij}\}, (1 \leqslant i \leqslant v, 1 \leqslant j \leqslant n)$. The authenticators are utilized to conduct integrity auditing verification.

DO uploads the data $F = \{F_1, F_2, \cdots, F_v\}$, the data block authenticator set $\Phi = \{\sigma_{ij}\}, (1 \leqslant i \leqslant v, 1 \leqslant j \leqslant n)$ to CS. Assume that entities (DO, TPA, CS) have registered an address on blockchain. DO makes a data integrity checking delegation for the TPA. DO initiates a smart contract (as shown in Algorithm 3), DO defines the auditing task, including the file name, the number of challenge blocks, the address of TPA and CS in the blockchain. DO uploads the encrypted index matrix $I = \{\pi_o(w_k), ev_{\pi_o(w_k)}\}_{k=1,2,\cdots,m}$ and the matched index authenticator set $\Omega = \{\Omega_{\pi_o(w_k)}\}_{k=1,2,\cdots,m}$ to the blockchain.

Algorithm 2 AuthGen

```
Require: The file set $F = \{F_1, F_2, \cdots, F_v\}$
Ensure: The authenticator set $\Phi$
 1: for Each $F_i \in F (1 \leqslant i \leqslant v)$ do
 2:     Split $F_i$ into $n$ blocks $m_{i1}, m_{i2}, \cdots, m_{in}$
 3:     for Each $1 \leqslant j \leqslant n$ do
 4:         $\triangle$ : Blind each data block
 5:         Compute $b_{ij} = m_{ij} + H_2(sk_{ID}\|ID_i\|i\|j)$
 6:         Split $b_{ij}$ into $s$ sectors $\{b_{ijk}\}_{k\in[1,s]}$
 7:     end for
 8: end for
 9: Randomly selects $x \in \mathbb{Z}_q^*$
10: Computes $R = g^x$
11: for Each $1 \leqslant i \leqslant v$ do
12:     for Each $1 \leqslant j \leqslant n$ do
13:         $\triangle$ : The authenticator
14:         Compute $\sigma_{ij} = sk_{ID} \times [H_3(ID_i\|j) \times g^{\sum_{k=1}^{s} a_k b_{ijk}}]^x$
15:     end for
16: end for
17: return $\Phi = \{\sigma_{ij}\}, (1 \leqslant i \leqslant v, 1 \leqslant j \leqslant n)$
```

Algorithm 3 Smart Contract

```
Require: $Function\,name,\, invoked\,parameters$
Ensure: $Setting\,up\,functions$
1: Structure: $Task$
2: $\triangle$ : Define a structure of the auditing task
3: $taskID; fileName; numChallenges; startTime;$
4: $endTime; chalStatus; proofStatus; recordList;$
5: $\triangle$ : The participant
6: $addressTPA; addressCS;$
7: function: $newTask(user, name, n, TPA, CS,$
   $\Omega = \{\Omega_{\pi_o(w_k)}\}_{k=1,2,\cdots,m})$
8: $task = tasks[taskID]$
9: $task.fileName = name$
10: $task.n = n$
11: $task.index = \Omega$
12: $task.addressTPA = TPA$
13: $task.addressCS = CS$
14: $task.chalStatus = true$
15: $task.proofStatus = true$
16: function: $ChalGen(taskID, t_{start}, t_{end}, \triangle t, c)$
17: $require(tasks[taskID].addressTPA == msg.sender)$
18: $require((t_{end} - t_{start})/\triangle t > 12)$
19: $require(t_{end} > block.timestamp)$
20: $require(tasks[taskID].chalStatus == true)$
21: $task.startTime = t_{start}$
22: $task.endTime = t_{end}$
23: $task.numChallenges = c$
24: $compute\,\mathcal{T} = \lfloor \frac{t_{end}-t_{start}}{\triangle t} \rfloor$
25: for $i \in [1, \mathcal{T}]$ do
26:     $recordList \leftarrow [t_{start} + (i-1) \times \triangle t, t_{start} + i \times \triangle t]$
27: end for
28: $\triangle$ : The retrieved blocks are $\mathcal{B}_i = \{B(t)\}$
29: $task.chalStatus = false$
30: function: $addProof(taskID, proofhash)$
31: $require(tasks[taskID].addressCS == msg.sender)$
32: $require(tasks[taskID].proofStatus == true)$
33: $task.proof.push(proofhash)$
34: $tasks[taskID].proofStatus = false$
```

6) $T_{w'} \leftarrow TrapdoorGen(w')$
This algorithm is conducted by DO and produces the search trapdoor.
DO generates the search trapdoor as $T_{w'} = \{\pi_o(w'), f_l(\pi_o(w'))\}$ based on keyword w'. DO sends the trapdoor $T_{w'}$ to TPA. The trapdoor information is utilized to seach files that require integrity auditing.

7) $Chal \leftarrow ChalGen(T_{w'}, I, \Omega)$
To generate challenge information, the algorithm first produces challenge seeds based on the blockchain. The core concept behind generating random seeds is to utilize a series of public, unforgeable, and verifiable blocks in the blockchain. Then, the challenge information is generated based on these random seeds, including the indices of the challenge data blocks and their corresponding random values.
As shown in Smart Contract 3. The smart contract is posted by DO. DO first defines the structure of the auditing task, writes the participants' address to the smart contract and sets the access control for participants to use smart contracts. TPA makes a challenge according to the smart contract. We define an agreement denoted as Ag, which comprises five elements: $Ag = \{t_{start}, t_{end}, \triangle t, c\}$. Here, t_{start} represents the starting timestamp chosen by TPA for when the auditing begins, while t_{end} marks the end of the auditing period. $\triangle t$ signifies the time interval between two timestamps. We use $\mathcal{T}$ to denote the interval of successive blocks, where $\mathcal{T}$ is defined as $\mathcal{T} = \lfloor \frac{t_{end}-t_{start}}{\triangle t} \rfloor > 12$. This requirement ensures security, as more than 12 successive blocks is considered secure. Additionally, we define $B(t)$ as the function that returns the blockhash within the timestamp t.

a) The challenge information will be retrieved from a succession of blocks. Specifically, for $i \in [1, \mathcal{T}]$, $\mathcal{B}_i = \{B(t)\}$, $t \in [t_{start} + (i-1) \triangle t, t_{start} + i \times \triangle t]$. Here, c denotes the number of challenge blocks.
b) TPA parses the challenge information from the blockchain as follows: TPA generates $k_{11} = H_2(1\|\mathcal{B}_i)$ and $k_{12} = H_2(2\|\mathcal{B}_i)$. The value k_{11} serves as a key for generating challenge data blocks, while k_{12} is used to generate random values.
c) Using the address $\pi_o(w_k) = \pi_o(w')$ for reference, TPA locates the row $ev_{\pi_o(w_k)}$ from blockchain. Subsequently, TPA obtains the plaintext vector using the equation $v_{w_k} = ev_{\pi_o(w_k)} \oplus f_l(\pi_o(w_k))$.
d) TPA begins by initializing an empty set $S_{w1} = \emptyset$. For each $i \in [1, n]$, if $v_{w'}[i] = 1$, TPA includes i in the set S_{w1}.
e) Then, TPA generates the challenge information $\{C_1 = \{(j_1, v_{1ij})\}\}_{j_1 \in [1,c], i \in S_{w1}}$, where $j_1 = \psi_{k_{11}}(\beta)$ for $\beta \in [1, c]$ and $v_{1ij} = \pi_{x_{1i}}(j_1)$ for $x_{1i} = \pi_{k_{12}(i)}$, $i \in S_{w1}$, $j_1 \in [1, c]$. Let the index of challenge data blocks set on TPA side be defined as $Q_1 = \{j_1 = \psi_{k_{11}}(\beta)\}_{\beta \in [1,c]}$. TPA sends the challenge information $Chal = (Ag, c, T_{w'})$ to CS.

8) $Proof \leftarrow ProofGen(Chal, I, \Omega, F, \Phi)$
Upon receiving the challenge information $Chal$, as outlined in Algorithm 4, CS generates the proof information as follows:

a) Using the address $\pi_o(w_k) = \pi_o(w')$ for reference from blockchain, CS locates the row $ev_{\pi_o(w_k)}$. Subsequently, CS obtains the plaintext vector using the equation $v_{w_k} = ev_{\pi_o(w_k)} \oplus f_l(\pi_o(w_k))$.
b) CS begins by initializing an empty set $S_{w2} = \emptyset$. For each $i \in [1, n]$, if $v_{w'}[i] = 1$, CS includes i in the set S_{w2}.
c) CS parses the challenge information from the blockchain as follows: CS generates $k_{21} = H_2(1\|\mathcal{B}_i)$ and $k_{22} = H_2(2\|\mathcal{B}_i)$. The value k_{21} serves as a key for generating challenge data blocks, while k_{22} is used to generate random values. Then, CS generates the challenge set $\{C_2 = \{(j_2, v_{2ij})\}\}_{j_2 \in [1,c], i \in S_{w2}}$, where $j_2 = \psi_{k_{21}}(\beta)$ for $\beta \in [1, c]$ and $v_{2ij} = \pi_{x_{2i}}(j_2)$ for $x_{2i} = \pi_{k_{22}}(i)$, $i \in S_{w2}$, $j_2 \in [1, c]$. Let the index of challenge data blocks set on TPA side be defined as $Q_2 = \{j_2 = \psi_{k_{21}}(\beta)\}_{\beta \in [1,c]}$.
d) CS computes aggregated sigmature as $T = \prod_{i \in S_{w2}} \prod_{j \in Q_2} (\sigma_{ij})^{v_{2ij}}$, and aggregated data information as $\mu_k = \sum_{i \in S_{w2}} \sum_{j \in Q_2} (v_{2ij} \times b_{ijk})$. CS

Algorithm 4 ProofGen

```
Require: The challenge $Chal$, the index vector $I$, the matched index content $\Omega$, the file set $F$, the authenticator set $\Phi$
Ensure: The auditing proof $Proof$
1: if $\pi_o(w_k) = \pi_o(w')$ then
2:     Obtain plaintext vector $v_{w_k} = ev_{\pi_o(w_k)} \oplus f_l(\pi_o(w_k))$
3: end if
4: Initialize an empty set $S_{w2} = \emptyset$
5: for $i \in [1, n]$ do
6:     if $v_{w'}[i] = 1$ then
7:         Include $i$ into set $S_{w2}$
8:     end if
9: end for
10: $\triangle$ : Parse the challenge information
11: for $\beta \in [1, c]$ do
12:     Generate challenge set $\{C_2 = \{(j_2, v_{2ij})\}\}_{j_2 \in [1,c], i \in S_{w2}}$, where $j_2 = \psi_{k_{21}}(\beta)$
13: end for
14: for $i \in S_{w2}$ do
15:     Generate $x_{2i} = \pi_{k_{22}}(i)$
16:     for $x_{2i} = \pi_{k_{22}}(i)$ do
17:         $v_{2ij} = \pi_{x_{2i}}(j_2)$
18:     end for
19: end for
20: Search the corresponding row in the index matrix, where $\pi_o(w_k) = \pi_o(w')$
21: Compute $v_{w_k} = ev_{\pi_o(w_k)} \oplus f(\pi_o(w_k))$ $\triangle$ : Encrypt each row
22: Initiate an empty set: $W_{w_k} = \emptyset$
23: for Each $i \in [1, v]$ do
24:     if $v_{w_k}[i] == 1$ then
25:         Add $i$ to $S_{w_k}$
26:     end if
27: end for
28: Compute $T = \prod_{i \in S_{w2}} \prod_{j \in Q_2} (\sigma_{ij})^{v_{2ij}}$
29: for $k \in [1, s]$ do
30:     Compute $\mu_k = \sum_{i \in S_{w2}} \sum_{j \in Q_2} (v_{2ij} \times b_{ijk})$
31: end for
32: return $Proof = \{T, \{\mu_k\}_{1 \leqslant k \leqslant s}\}$
```

sets the auditing proof $Proof = \{T, \{\mu_k\}_{1 \leqslant k \leqslant s}\}$.
CS sends the proof information $Proof = \{T, \{\mu_k\}_{1 \leqslant k \leqslant s}\}$ to TPA. CS computes the hash value for the proof and then submits this hash value to the blockchain.

9) $(0, 1) \leftarrow ProofVerify(Proof)$

Upon receiving the challenge information $Proof$, TPA checks the validity of the following equation:

$$\begin{aligned} &e(T \times \prod_{i \in S_{w1}} \prod_{j \in Q_1} (\Omega_{w_k, ij})^{v_{1ij}}, g) \\ &= e((\prod_{j \in Q_1} (H_4(\pi_o(w_k) \| j))^{\sum_{i \in S_{w1}} v_{1ij}}) \times \prod_{k=1}^{s} A_k^{\mu_k}, R) \\ &\times e((H_1(ID))^{\sum_{i \in S_{w1}} \sum_{j \in Q_1} v_{1ij}}, P_0). \end{aligned} \quad (1)$$

If the equation is satisfied, TPA outputs 1, signifying that similar files containing the queried keyword are securely stored in the cloud. Conversely, if the equation is not satisfied, TPA outputs 0, indicating that some files may have been tampered with. The data interaction of the whole scheme is shown in Fig. 4.

V. Security Analysis

This part includes storage correctness, the unforgeability of authenticators, the unforgeability of proof, decentralized challenge information generation, privacy preservation and resistance to the same keyword files summation attack.

Theorem 1. ***(Correctness)****. The proof information generated from CS can pass the verification.*

Proof. Since the block information on the blockchain is unforgeable, the information $i \in [1, \mathcal{T}]$, corresponding to $\mathcal{B}_i = \{B(t)\}$, where $t \in [t_{start} + (i-1) \triangle t, t_{start} + i \times \triangle t]$, is also unforgeable. Using the same seeds, the challenge information $\{\{C_1 = \{(j_1, v_{1ij})\}\}_{j_1 \in [1,c], i \in S_{w1}}, Q_1 = \{j_1 = \psi_{k_{11}}(\beta)\}_{\beta \in [1,c]}\}$ from TPA side is identical to $\{\{C_2 = \{(j_2, v_{2ij})\}\}_{j_2 \in [1,c], i \in S_{w2}}, Q_2 = \{j_2 = \psi_{k_{21}}(\beta)\}_{\beta \in [1,c]}\}$ on the CS side. Therefore, the following equation holds:

$$\begin{aligned} &e(T \times \prod_{i \in S_{w1}} \prod_{j \in Q_1} (\Omega_{w_k, ij})^{v_{1ij}}, g) \\ &= e(\prod_{i \in S_{w2}} \prod_{j \in Q_2} (sk_{ID} \times [H_3(ID_i \| j) \times g^{\sum_{k=1}^{s} a_k b_{ijk}}]^x)^{v_{2ij}} \times \\ &\prod_{i \in S_{w1}} \prod_{j \in Q_1} ([H_3(ID_i \| j)^{-1} \times H_4(\pi_o(w_k) \| j)]^x)^{v_{1ij}}, g) \\ &= e(\prod_{i \in S_{w2}} \prod_{j \in Q_2} (sk_{ID} \times (g^{\sum_{k=1}^{s} a_k b_{ijk}})^{x \cdot v_{2ij}} \times \\ &\prod_{i \in S_{w1}} \prod_{j \in Q_1} H_4(\pi_o(w_k) \| j)^{x \cdot v_{1ij}}, g) \\ &= e(\prod_{i \in S_{w2}} \prod_{j \in Q_2} sk_{ID}, g) \times \\ &\quad e(\prod_{i \in S_{w1}} \prod_{j \in Q_1} (g^{\sum_{k=1}^{s} a_k b_{ijk}} \times H_4(\pi_o(w_k) \| j))^{x \cdot v_{1ij}}, g) \\ &= e((H_1(ID))^{\sum_{i \in S_{w2}} \sum_{j \in Q_2} v_{2ij}}, P_0) \times \\ &e((\prod_{j \in Q_1} (H_4(\pi_o(w_k) \| j))^{\sum_{i \in S_{w1}} v_{1ij}}) \times \prod_{k=1}^{s} A_k^{\mu_k}, R). \end{aligned}$$

□

Theorem 2. ***(Unforgeability of Authenticators)****. If there exists a PPT adversary $\mathcal{A}$ making queries to H_1, KeyGen, H_3, and authenticator queries q_{H_1}, q_{key}, q_{H_3}, q_{auth} times, respectively, and winning the game defined in the Section III with advantage ε within time t, then a simulator $\mathcal{S}$ can break the CDH assumption with a probability ε', where $\varepsilon' \geq (1-\gamma)^{q_{key}+q_{auth}} \frac{\zeta}{q_u} \varepsilon \geq \varepsilon / (q_u(q_{key} + q_{auth}) \times 2e)$ within time $t' \leqslant t + \mathcal{O}(q_{H_1} + q_{key} + q_{H_3} + q_{auth})$.*

Proof. If the adversary $\mathcal{A}$ manages to win the security game in the Section III with a non-negligible probability, the simulator $\mathcal{S}$ is capable of leveraging $\mathcal{A}$ to compute g^{ab} with a non-negligible probability. The interaction between $\mathcal{S}$ and $\mathcal{A}$ unfolds as follows:

SysIni, Setup: $\mathcal{S}$ first generates the public parameters (pp, W, v) and sets $P_0 = g^a$. Then, it randomly chooses values $x \in \mathbb{Z}_q^*$ and s random values $(a_1, a_2, \cdots, a_s) \in \mathbb{Z}_q^*$. Afterward, $\mathcal{S}$ computes $R = g^x$ and $A_k = g^{a_k}$

for $1 \leqslant k \leqslant s$. Finally, $\mathcal{S}$ shares the public parameters $(pp, P_0, R, \{A_k\}_{k=1}^{s}, W, v)$ R with $\mathcal{A}$.

User Creation: $\mathcal{S}$ sets the challenge user identity set $\{\widetilde{ID}_i, \tau\}$ as follows: $\mathcal{S}$ tosses a coin $\tau \in \{0,1\}$ for each $\widetilde{ID}_i$, when $\tau = 1$, the probability is ζ, when $\tau = 0$, the probability is $1-\zeta$. Let the set be $\mathcal{I} = \{\widetilde{ID}_t\}_{t\in\{1,\cdots,q_u\}}$ when $\tau = 1$.

H_1 **Query**: During the interaction, $\mathcal{A}$ adaptively makes H_1-$Query$ requests for identity $\widetilde{ID}_i^*$. $\mathcal{S}$ keeps track of a list L_1 consisting of tuples in the form $L_1 = \{(\widetilde{ID}_i, h_1, Q_1)\}$. If L_1 contains the entry for $\widetilde{ID}_i^*$, $\mathcal{S}$ retrieves the tuple $(\widetilde{ID}_i^*, h_1^*, Q_1^*)$ and provides Q_1^* in response to $\mathcal{A}$. If L_1 does not include $\widetilde{ID}_i^*$, $\mathcal{S}$ checks whether $\widetilde{ID}_i^* \in \mathcal{I}$. Then $\mathcal{S}$ randomly selects $h_1^*, b \in \mathbb{Z}_q^*$ and generates the value as

$$Q_1^* = \begin{cases} g^{h_1^*} & i \notin \{1,\cdots,q_u\} \\ (g^b)^{h_1^*} & i \in \{1,\cdots,q_u\} \end{cases}$$

Then, $\mathcal{S}$ responds with Q_1^* to $\mathcal{A}$ and includes a new tuple (ID^*, h_1^*, Q_1^*) in L_1.

Key Query: $\mathcal{A}$ adaptively initiates a $Key, Query$ for identity $\widetilde{ID}_i^*$. $\mathcal{S}$ examines whether the tuple (ID^*, h_1^*, Q_1^*) exists in L_1. If it doesn't, $\mathcal{S}$ itself executes the H_1-$Query$ for $\widetilde{ID}^*$. Once $\mathcal{S}$ obtains the corresponding tuple $(\widetilde{ID}_i^*, h_1^*, Q_1^*)$ from L_1, it proceeds to check the value of $\widetilde{ID}_i^*$. If $\widetilde{ID}_i^* \notin \{\widetilde{ID}_t\}_{t\in\{1,\cdots,q_u\}}$, $\mathcal{S}$ computes $(Q_1^*)^a = (g^{h_1})^a = (g^a)^{h_1}$ and delivers this result to $\mathcal{A}$. However, if $\widetilde{ID}_i^* \in \{\widetilde{ID}_t\}_{t\in\{1,\cdots,q_u\}}$, $\mathcal{S}$ terminates the operation.

H_3 **Query**: $\mathcal{A}$ adaptively initiates a $H_3, Query$ for the pair (ID_i^*, j^*). To manage these queries, $\mathcal{S}$ maintains a list denoted as L_2 with entries in the form of $L_2 = \{(ID_i, j, Q_2)\}$. If (ID_i^*, j^*, Q_2^*) exists in L_2, $\mathcal{S}$ retrieves the corresponding tuple and forwards Q_2^* to $\mathcal{A}$. In cases where (ID_i^*, j^*, Q_2^*) is not present in L_2, $\mathcal{S}$ randomly selects $Q_2^* \in \mathbb{G}_1$ and transmits it to $\mathcal{A}$. Subsequently, $\mathcal{S}$ inserts a new tuple (ID_i^*, j^*, Q_2^*) into the L_2 list.

Authenticator Query: $\mathcal{A}$ forwards the tuple $(\widetilde{ID}_i^*, ID_i^*, j^*)$ to $\mathcal{S}$ to inquire about the tag for b_{ij}^*. Upon receiving the tag query, $\mathcal{S}$ initially checks for the existence of the tuple $(\widetilde{ID}_i^*, h_1^*, Q_1^*)$ in L_1 and (ID_i^*, j^*, Q_2^*) in L_2. If these tuples are not found, $\mathcal{S}$ proceeds to execute $H_1\, Query$ and $H_3\, Query$ to obtain them. In case when $\widetilde{ID}_i^* \in \{\widetilde{ID}_t\}_{t\in\{1,\cdots,q_u\}}$, $\mathcal{S}$ calculates the tag as $\sigma_{ij}^* = (g^a)^{h_1^*} \times (Q_2^* \times g^{\sum_{k=1}^{s} a_k b_{ijk}^*})^x$. However, if $\widetilde{ID}_i^* \in \{\widetilde{ID}_t\}_{t\in\{1,\cdots,q_u\}}$, $\mathcal{S}$ terminates the game.

Forge: In the end, $\mathcal{A}$ generates a forged tag σ'_{ij} for block b'_{ij} associated with user identity ID'. $\mathcal{A}$ wins if the conditions hold:

1) $(ID', \sigma'_{ij}, b'_{ij})$ passes the verification.
2) $\mathcal{A}$ has never conducted a authenticator query on $(ID', \sigma'_{ij}, b'_{ij})$.

Analysis: If $ID' \notin \{ID_t\}_{t\in\{1,\cdots,q_u\}}$, $\mathcal{S}$ aborts. Since $(ID', \sigma'_{ij}, b'_{ij})$ passes the verification, the proof information satisfies:

$$e(\sigma'_{ij}, g) = e(H_1(ID'), P_0) \times e(H_3(ID_i \| j) \times g^{\sum_{k=1}^{s} a_k b'_{ijk}}, R).$$

$\mathcal{S}$ retrieves the (ID', h'_1, Q'_1) from L_1 and looks up the tuple (ID', j', Q'_2) from L_2. With the verification equation mentioned above, $\mathcal{S}$ obtains: $e(\sigma'_{ij}, g) = e(g^{bh'_1}, g^a) \times e(Q'_2 \times \prod_{k=1}^{s} u_k^{b'_{ijk}}, g^r)$. Consequently, the result for the given CDH instance is $g^{ab} = (\frac{\sigma'_{ij}}{(Q'_2 \prod_{k=1}^{s} u_k^{b'_{ijk}})^r})^{\frac{1}{h'_1}}$.

Probability Analysis: If the following situation holds, $\mathcal{S}$ can obtain a instance of CDH problem. (1) E_1: $\mathcal{S}$ does not terminates in the game. (2) E_2: $\mathcal{A}$ successfully forges a valid authenticator. (3) E_3: $ID' \in \{ID_t\}_{t\in\{1,\cdots,q_u\}}$. The probability that $\mathcal{A}$ wins in the game is $Pr = Pr[E_1 \wedge E_2 \wedge E_3] = Pr[E_1] \wedge Pr[E_2] \wedge Pr[E_3] = Pr[E_1] \cdot Pr[E_2|E_1] \cdot Pr[E_3|E_1 \wedge E_2]$.

1) If E_1 happens, there are two situations to consider:
   a) $\mathcal{S}$ does not abort in the **Key Query** phase. The probability is $(1 - \frac{1}{q_u}\zeta)^{q_{key}}$ in this situation.
   b) $\mathcal{S}$ does not abort in **Authenticator Query** phase. The probability is $(1 - \frac{1}{q_u}\zeta)^{q_{auth}}$ in this situation.

   Thus, it can retrieve $Pr[E_1] = (1 - \frac{1}{q_u}\zeta)^{q_{key}+q_{auth}} \geq (1-\zeta)^{q_{key}+q_{auth}}$.
2) If E_2 happens, it obtains $Pr[E_2|E_1] = \varepsilon$.
3) If E_3 happens, it obtains $Pr[E_3|E_1 \wedge E_2] = \frac{\zeta}{q_u}$.

Thus, the advantage that $\mathcal{A}$ wins is $\varepsilon' \geq (1-\zeta)^{q_{key}+q_{auth}} \frac{\zeta}{q_u} \varepsilon \geq \varepsilon / (q_u(q_{key} + q_{auth})2e)$. That means, $\mathcal{S}$ retrieves the value g^{ab} with a non-negligible probability ε' by using the capability of $\mathcal{A}$ under time $t' \leqslant t + \mathcal{O}(q_{H_1} + q_{key} + q_{H_3} + q_{auth})$, which violates the CDH assumption. □

Theorem 3. ***(Unforgeability of Proof).*** *The probability of the CS successfully fabricating a complete proof that can pass the verification without access to the actual data is negligible.*

Proof. Assuming that the forged aggregate proof is denoted as $Proof' = \{T', \{\mu'_k\}_{1\leqslant k\leqslant s}\}$, it is important to note that this forged proof can successfully pass the verification process, indicating that it fulfills the verification equation

$$\begin{aligned} &e(T' \times \prod_{i\in S_{w1}} \prod_{j\in Q_1} (\Omega_{w_k,ij})^{v_{1ij}}, g) \\ &= e((\prod_{j\in Q_1} (H_4(\pi_o(w_k)\|j))^{\sum_{i\in S_{w1}} v_{1ij}}) \times \prod_{k=1}^{s} A_k^{\mu'_k}, R) \\ &\times e((H_1(ID))^{\sum_{i\in S_{w1}} \sum_{j\in Q_1} v_{1ij}}, P_0). \end{aligned} \tag{2}$$

Assume the real proof is $Proof = \{T, \{\mu_k\}_{1\leqslant k\leqslant s}\}$, it also satisfies the verification

$$\begin{aligned} &e(T \times \prod_{i\in S_{w1}} \prod_{j\in Q_1} (\Omega_{w_k,ij})^{v_{1ij}}, g) \\ &= e((\prod_{j\in Q_1} (H_4(\pi_o(w_k)\|j))^{\sum_{i\in S_{w1}} v_{1ij}}) \times \prod_{k=1}^{s} A_k^{\mu_k}, R) \\ &\times e((H_1(ID))^{\sum_{i\in S_{w1}} \sum_{j\in Q_1} v_{1ij}}, P_0). \end{aligned} \tag{3}$$

When we compare the two equations mentioned above, we can deduce that $T' = T$ only if each $\mu_k = \mu'_k$, which contradicts the assumption. According to Theorem 1, it is impossible to forge a single tag. Therefore, there must be at least one $\mu_k \neq \mu'_k$ but $T' = T$, the adversary passes the verification. Hence, for both equations to hold, it is required that $T' = T$

TABLE III: Comparison of Provable Data Possession Schemes

Scheme	[8]	[39]	[40]	Ours
Type	PKI	Certificateless	IBC	IBC
Public auditing	✓	✓	✓	✓
Decentralization	×	✓	✓	✓
Keyword searchable auditing	✓	×	×	✓
Resist summation attack	×	—	—	✓

and at least one $\mu_k \neq \mu'_k$. Comparing the two equations presented above, we observe that $\prod_{k=1}^{s} u_k^{\mu_k} = \prod_{k=1}^{s} u_k^{\mu'_k}$. Thus, we have

$$g^{\sum_{i\in S_{w2}}\sum_{j\in Q_2} v_{2ij}\sum_{k=1}^{s} a_k b_{ijk}} = g^{\sum_{i\in S_{w2}}\sum_{j\in Q_2} v_{2ij}\sum_{k=1}^{s} a_k b'_{ijk}},$$

which implies

$$\sum_{i\in S_{w2}}\sum_{j\in Q_2} v_{2ij}\sum_{k=1}^{s} a_k b_{ijk} = \sum_{i\in S_{w2}}\sum_{j\in Q_2} v_{2ij}\sum_{k=1}^{s} a_k b'_{ijk}.$$

The equation can be derived as $\sum_{i\in S_{w2}}\sum_{j\in Q_2}\sum_{k=1}^{s} v_{2ij} a_k (b_{ijk} - b'_{ijk}) = 0$. The equation can also be represent as $\sum_{i\in S_{w2}}\sum_{j\in Q_2}\sum_{k=1}^{s} a_{ijk}\cdot b_{ijk} = 0$ where $a_{ijk} \in \mathbb{Z}_q^*$. Without loss of generality, we suppose there are $\beta(1 \leqslant \beta \leqslant |S_{w2}|\cdot c\cdot s)$ different data blocks that satisfies $b_{ijk} \neq b'_{ijk}$. Subsequently, the number of tuples $(a_{ij1},\cdots,a_{ijs})_{i\in S_{w2},j\in Q_2}$ that satisfy this property is bounded by $q^{\beta-1}$. Since the values $(a_{ij1},\cdots,a_{ijs})_{i\in S_{w2},j\in Q_2}$ are random and unknown to the CS, the probability of the equation $\sum_{i\in S_{w2}}\sum_{j\in Q_2}\sum_{k=1}^{s} a_k(b_{ijk} - b'_{ijk}) = 0$ being true is less than $\frac{q^{\beta-1}}{q^{|S_{w2}|\cdot c\cdot s}} \leqslant \frac{q^{\beta-1}}{q^{\beta}} = \frac{1}{q}$, which is negligible. □

From the above theorems, we can further analyse how the proposed scheme satisfies the design goals.

Decentralized challenge information generation. This property is guaranteed by the characteristics of blockchain and smart contract. The challenge information is generated from unpredictable current hashes. This is guaranteed by the timestamp, only a task start time greater than the current block timestamp is considered valid. At the same time, the interval between the task start time and end time is also limited by the security interval of block numbers. Thus, if the blockchain is secure, the method of generating challenge information is decentralized.

Privacy preservation. This property is guaranteed by the blind index structure. For the real keywords $(w_1,\cdots,w_m)$ and the corresponding index vector $(v_{w_1},\cdots,v_{w_m})$, the stored information on the cloud side is $(\pi_o(w_1),\cdots,\pi_o(w_m))$ and $(ev_{\pi_o(w_1)},\cdots,ev_{\pi_o(w_m)})$ respectively, where $\pi_o(w_i)$, $i \in [1,m]$ is the pseudo random permutation for the keyword w_i and $ev_{\pi_o(w_i)}$ is the encrypted value of v_{w_k}. According to the blind keyword, the CS cannot infer the search content. Since the value $ev_{\pi_o(w_i)}$ can be decrypted by the CS, the value serves only as an index vector and cannot leak the content of the data.

Resistance to the same keyword files summation attack. This property is guaranteed by two-dimensional challenge set $\{C_1 = \{(j_1, v_{1ij})\}, X = \{x_{1i}\}\}_{j_1\in[1,c],i\in S_{w1}}\}$, the set C_1 is utilized for data blocks and the set X is utilized for different files. This ensures that each block is fresh for each challenge round.

TABLE IV: Comparison of Provable Data Possession Schemes

Scheme	[8]	[39]	[40]	Ours
Type	PKI	Certificateless	IBC	IBC
Public auditing	✓	✓	✓	✓
Decentralization	×	✓	✓	✓
Keyword searchable auditing	✓	×	×	✓
Resist summation attack	×	—	—	✓

TABLE V: Communication Overhead

	[8]	Our scheme
DO → CS	$\|F\| + (v\times n + n\times m)\|\mathbb{G}\|$	$\|F\| + v\times\eta\|\mathbb{G}\|$
DO → TPA	$2\|\mathbb{Z}_q^*\|$	$2\|\mathbb{Z}_q^*\|$
TPA → CS	$(2+2c)\|\mathbb{Z}_q^*\|$	$2\|\mathbb{Z}_q^*\| + 4\|n\|$
CS → TPA	$\|\mathbb{G}\| + \|\mathbb{Z}_q^*\|$	$\|\mathbb{G}\| + s\|\mathbb{Z}_q^*\|$

VI. Performance Evaluation

A. Theoretical Analysis

We showed the function comparison with some different types of schemes [8], [39] and [40]. Scheme [8] utilized a PKI cryptosystem. Scheme [39] utilized a Certificateless cryptosystem. Both scheme [40] and ours were based on IBC cryptosystem. From Fig. IV, all schemes achieved public auditing. Only schemes other than [8] were capable of achieving decentralization. Both scheme [8] and our scheme can achieve keyword searchable auditing. However, scheme [8] was vulnerable to the summation attack.

The notion $H_{\mathbb{G}_1}$, $Mul_{\mathbb{G}_1}$, $Mul_{\mathbb{Z}_q^*}$, $Exp_{\mathbb{G}_1}$ denote hash operation to the group $\mathbb{G}_1$, multiplication in the group $\mathbb{G}_1$, multiplication in the group $\mathbb{Z}_q^*$ and exponentiation operation in the group $\mathbb{G}_1$. The notion $Pair$ denoted pairing operation. In a system with v files and m keywords, where each file F has a size of $n|\mathbb{Z}_q^*|$ and is divided into $\eta = \frac{n|\mathbb{Z}_q^*|}{s|\mathbb{Z}_q^*|}$ blocks, and each keyword w is associated with $|S_w|$ files, we can analyze the computation overhead in different phases. We excluding simple operations like PRF and XOR.

The computation overhead comparisons were shown in TABLE VII. The computation overhead of index generation was $m\times s\times|S_w|\times 2\times H_{\mathbb{G}_1} + (|S_w|)\times Mul_{\mathbb{G}_1} + Pow_{\mathbb{G}_1} + m\cdot Enc_{XoR}$. The computation overhead of authenticator generation was $v\times\eta\times(H_{\mathbb{G}_1} + 2Mul_{\mathbb{G}_1} + 2Exp_{\mathbb{G}_1} + sMul_{\mathbb{Z}_q^*})$. The computation overhead of proof generation was $(|S_w|+c)Exp_{\mathbb{G}_1} + (|S_w|+c)Mul_{\mathbb{G}_1} + |S_w|\times cMul_{\mathbb{Z}_q^*} + Eec_{XoR}$. And the proof verification overhead was $3\times Pair + |S_w|\times c\times H_{\mathbb{G}_1} + (|S_w|\times c + s - 2)Mul_{\mathbb{Z}_q^*} + (|S_w|\times c + s + 1)Exp_{\mathbb{G}_1} + Mul_{\mathbb{Z}_q^*}$.

The communication and storage overhead comparisons were shown in TABLE V and VI. The communication overhead on the data upload phase in [8] was $|F| + (v\times n + n\times m)|\mathbb{G}|$, while it only needed $|F| + v\times\eta|\mathbb{G}|$ in our scheme. That was because we adopted the aggregate technique. The communication overhead on the trapdoor generation phase was same over the two schemes. In the challenge phase, the communication overhead

TABLE VI: Storage Overhead on Blockchain

	Our scheme
DO $\rightarrow$ blockchain	$v \times n \times m\lvert\mathbb{G}\rvert + m\lvert\mathbb{Z}_q^*\rvert + m \times v\lvert n\rvert$
TPA $\rightarrow$ blockchain	$5\lvert\mathbb{Z}_q^*\rvert$
CS $\rightarrow$ blockchain	$2\lvert\mathbb{Z}_q^*\rvert$

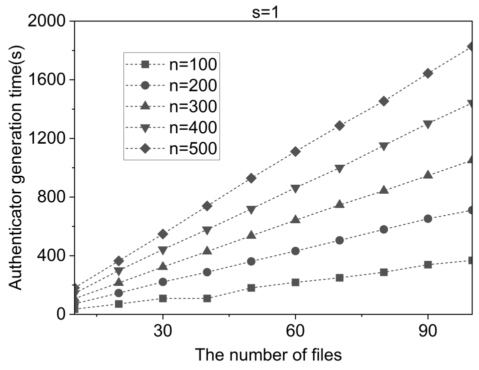


Fig. 6: The authenticator generation time with the number of files and the separated data block numbers in one file.

in [8] was $(2+2c)|\mathbb{Z}_q^*|$. In our scheme, we conducted the challenge according to the smart contract, the communiciation overhead was $2|\mathbb{Z}_q^*|+4|n|$. The communication overhead on the proof generathn phase was $|\mathbb{G}|+|\mathbb{Z}_q^*|$ in [8] and $|\mathbb{G}|+s|\mathbb{Z}_q^*|$ in our scheme. Our scheme incured higher communication costs since it needed to check every sector of the aggregate proof information. We also analysed the storage overhead on the blockchain in our scheme. In our scheme, the DO needed to publish a smart contract. The DO stored some basic information of the data and the index information on the blockchain. The storage overhead in this phase was $v \times n \times m|\mathbb{G}| + m|\mathbb{Z}_q^*| + m \cdot v|n|$. In the challenge phase, the TPA should define the auditing task including the start time, the end time, the time of duration and the challenge block numbers. The storage overhead in this phase was $5|\mathbb{Z}_q^*|$. In the proof generation phase, the CS should upload the hash value of the proof information on the blockchain. The storage overhead in this phase was $5|\mathbb{Z}_q^*|$.

B. Experiment Results

Experiments were conducted on a Windows 10 machine equipped with an Intel i7 2.5GHz CPU and 8GB of RAM. To evaluate the scheme's performance, key parameters were set. Each data sector had a size of 160 bits, and Type-A pairings with a group order of 160 bits were employed. Additionally, the length of elements in mathematical sets such as $\mathbb{Z}_q^*$ and $\mathbb{G}$ was standardized at 160 bits.

Parameter selection in the experiment. Assume that a file F consisted of n blocks and χ blocks were corrupted. Let the number of challenge blocks represent c. Let $P_c = \chi/n$ represent the corrupted data blocks ratio. Let P_d represent the probability of detecting corrupted blocks. Let $\overline{X}$ represent the number of corrupted blocks being challenged. P_d can be represented as

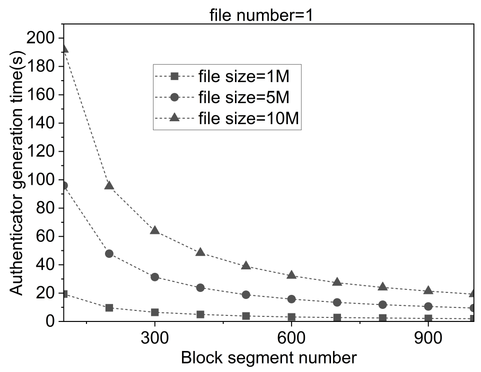


Fig. 7: The authenticator generation time with the block segment number and different file size.

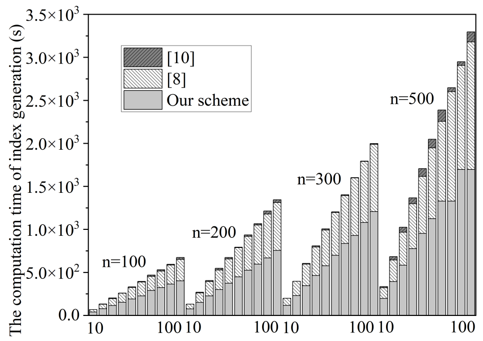


Fig. 8: The index generation time with the number of keyword and the separated data block numbers in one file. In the case of resisting the same keyword files summation attack, the number of files containing the same keyword in experiment was set to 1.

$$\begin{aligned} P_d &= P\{\overline{X} \geq 1\} = 1 - P\{\overline{X} = 0\} \\ &= 1 - \frac{n-\chi}{n} \times \frac{n-1-\chi}{n-1} \times \cdots \times \frac{n-c+1-\chi}{n-c+1}. \end{aligned}$$

Since $\frac{n-\chi}{n} > \frac{n-c+1-\chi}{n-c+1}$ holds, the following equation satisfies

$$1 - (\frac{n-\chi}{n})^c < P_d < 1 - (\frac{n-c+1-\chi}{n-c+1})^c.$$

From the above equation, when $P_d = 99\%$ and $P_c = 1\%$, the required number of challenge blocks was $c = 460$; when $P_d = 95\%$ and $P_c = 1\%$, c was set to $c = 300$. Therefore, we used $c = 300$ and $c = 460$, respectively, in the experiment.

Evaluation of authenticator generation. In order to evaluate the efficiency of authenticator generation, a series of tests were conducted with different numbers of files and blocks. The test scenarios included file numbers ranging from 10 to 100 in increments of 10 and block numbers ranging from 100 to 500 in increments of 100. As shown in Fig. 6, when 10 files were uploaded, each containing 100 blocks, the user required approximately 35.726 seconds to generate a total of 1000 authenticators. In the experiment, when 100 files and each containing 500 blocks were uploaded, the user took approximately

TABLE VII: Computation Overhead

Phase	[8]	Our scheme
IndexGen	$m \times s \times [(\lvert S_w\rvert + 2) \times H_{\mathbb{G}_1} + (\lvert S_w\rvert + 1) \times Mul_{\mathbb{G}_1} + Exp_{\mathbb{G}_1}]$	$m \times s \times \lvert S_w\rvert \times 2 \times H_{\mathbb{G}_1} + (\lvert S_w\rvert) \times Mul_{\mathbb{G}_1} + Pow_{\mathbb{G}_1} + m \cdot Enc_{XoR}$
AuthGen	$v \times n \times (H_{\mathbb{G}_1} + Mul_{\mathbb{G}_1} + Exp_{\mathbb{G}_1})$	$v \times \eta(H_{\mathbb{G}_1} + 2Mul_{\mathbb{G}_1} + 2Exp_{\mathbb{G}_1} + sMul_{\mathbb{Z}_q^*})$
ProofGen	$2cExp_{\mathbb{G}_1} + (\lvert S_w\rvert \times c + c)Mul_{\mathbb{G}_1} + \lvert S_w\rvert \times cMul_{\mathbb{Z}_q^*}$	$(\lvert S_w\rvert + c)Exp_{\mathbb{G}_1} + (\lvert S_w\rvert + c)Mul_{\mathbb{G}_1} + \lvert S_w\rvert \times cMul_{\mathbb{Z}_q^*} + 2H_{\mathbb{Z}_q^*} + Dec_{XoR}$
ProofVerify	$2 \times Pair + 2 \times c \times H_{\mathbb{G}_1} + (c+1) \times Mul_{\mathbb{G}_1} + (c+1) \times Exp_{\mathbb{G}_1}$	$3 \times Pair + \lvert S_w\rvert \times c \times H_{\mathbb{G}_1} + (\lvert S_w\rvert \times c + s - 2)Mul_{\mathbb{Z}_q^*} + (\lvert S_w\rvert \times c + s + 1)Exp_{\mathbb{G}_1} + Mul_{\mathbb{Z}_q^*}$

Enc_{XoR}: the encryption of XoR operation. Dec_{XoR}: the decryption of XoR operation.

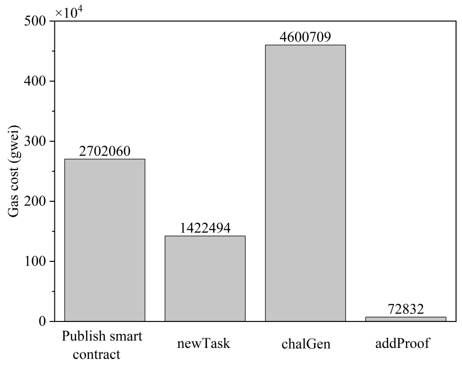


Fig. 9: Evaluation of gas cost on publishing smart contract and the transaction cost of newTask, chalGen and addProof.

1827.5 seconds to generate a total of $50,000$ authenticators. This translated to an average time of 0.037 seconds to generate a single authenticator. Notably, the authenticator generation time exhibited a linear relationship with both the number of files and the number of data blocks. Besides, we also tested the authenticator generation time with varying block segment numbers and file sizes. The segment numbers were set to range from 100 to 1000 with step size 100 and the file size to $1, 5, 10\,MB$ respectively. As shown in Fig. 7, we observed that the authenticator generation time decreased as the block segment number increased. The authenticator generation time increased with file size because the number of authenticators also increased.

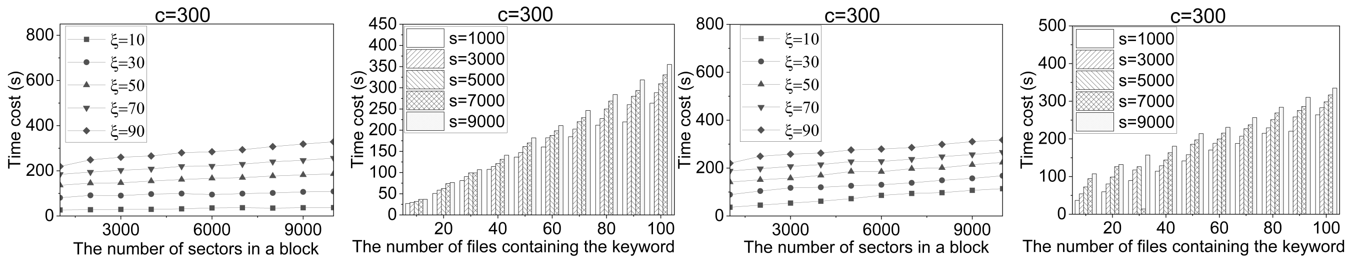
Evaluation of the index generation. The number of blocks in a file was set to 100, 200, 300 and 500, respectively, while the number of files containing the keyword ranged from 10 to 100 in increments of 10. As shown in Fig. 8, we compared the proposed scheme with [8] and [10]. When there were 100 files containing the keyword, and each file contained 200 blocks, the DO needed 755.399 seconds to generate the index structure, which was 44.9% less than that of scheme [10] and 42.4% less than that of scheme [8]. When there were 100 files containing the keyword, and each file contained 500 blocks, the DO required 1696.64 seconds to generate the index structure, which was 48.5% less than that of scheme [10] and 46.6% less than that of scheme [8]. From the figure, the index generation time increased as the number of keywords increased. For the same number of keywords, the index generation time was less than that of [8] and [10]. This was because the proposed scheme only needed to conduct the matching computation and incurred no extra computation cost.

Evaluation of gas cost. Ethereum was utilized to evaluate the cost of the smart contract in the proposed scheme. As shown in Fig. 9, the DO published the smart contract, which required about 2702060 gas. The DO added a new task transaction, which required about 1422494 gas. The TPA made a challenge transaction, which required about 4600709 gas. The CS added the proof information transaction, which required about 72832 gas.

Evaluation of proof generation and verification. The number of sectors in a block was set from 1000 to 10000 with step size 1000. The number of files containing the keyword ranged from 10 to 90 in increments of 10. The challenge numbers were set to 300 and 460, respectively. Fig. 10 showed the time cost when c=300. From Fig. 10 (a), we observed that the time cost did not increase significantly as the number of blocks increased but increased as the number of files containing the keyword. The results are also shown in Fig. 10 (b). Regarding the overhead on the TPA side, Fig. 10 (c) showed that the time cost increased as the number of files containing the keyword, while it did not increase significantly as the number of sectors in a block increased. The results are also shown in Fig. 10 (d). Fig. 11 showed the time cost when c=460. It exhibited a similar cost trend compared to c=300. From Fig. 11 (a) and Fig. 11 (b), we observed that it needed about more 0.5 times cost than c=300. From Fig. 11 (c) and Fig. 11 (d), we observed that it increased little and needed about more extra 0.5 times cost than when the challenge block numbers were set to c=300.

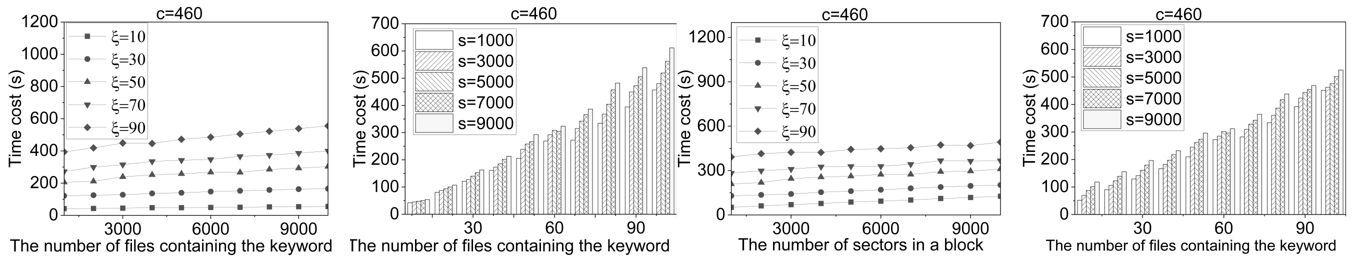
VII. Conclusion

In order to improve the auditing efficiency and reduce the burden on the DO side due to massive and similar data integrity checking, we focused on keyword searchable and provable data possession in this paper. We introduced the notion of the keyword and designed an index structure to match the authenticator. All matched files can be audited and verified, and privacy can be guaranteed at the same time. To resist the corrupt auditor to always generate biased challenge information, we utilized unpredictable but verifiable public information on the blockchain to generate challenging information instead of relying on the centralized TPA. Besides, a copy-summation attack was found in some schemes in this paper. We gave the detailed attack process and provided the



(a) (b) (c) (d)

Fig. 10: The time cost when c=300 (a): CSP side with different numbers of sectors in a block; (b): CSP side with different numbers of files containing the keyword; (c): TPA side with different sectors in a block; (d): TPA side with different numbers of files containing the keyword.



(a) (b) (c) (d)

Fig. 11: The time cost when c=460 (a): CSP side with different numbers of sectors in a block; (b): CSP side with different numbers of files containing the keyword; (c): TPA side with different sectors in a block; (d): TPA side with different numbers of files containing the keyword.

solution to resist this attack. Security analysis and performance evaluation both demonstrated that the proposed scheme was both secure and efficient. Experimental results shew that the index generation time was less than existing schemes in recent years.

REFERENCES

[1] H. Akbar, M. Zubair, and MS. Malik. The security issues and challenges in cloud computing. *International Journal for Electronic Crime Investigation*, 7(1):13–32, 2023.

[2] Y. Miao, K. Gai, L. Zhu, KKR Choo, and J. Vaidya. Blockchain-based shared data integrity auditing and deduplication. *IEEE Transactions on Dependable and Secure Computing*, PP(99):1, 2023.

[3] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 598–609, Alexandria, Virginia, USA, 2007. ACM.

[4] Y. Qi, Y. Luo, Y. Huang, and X. Li. Blockchain-based privacy-preserving public auditing for group shared data. *Intelligent Automation & Soft Computing*, 35(3), 2023.

[5] Y. Li, Y. Li, K. Zhang, and Y. Ding. Public integrity auditing for dynamic group cooperation files with efficient user revocation. *Computer Standards & Interfaces*, 83:103641, 2023.

[6] Z. Tu, X. Wang, W. Du, Z. Wang, and M. Lv. An improved multi-copy cloud data auditing scheme and its application. *Journal of King Saud University-Computer and Information Sciences*, 35(3):120–130, 2023.

[7] Z. Li, Y. Li, L. Lu, and Y. Ding. Blockchain-based auditing with data self-repair: From centralized system to distributed storage. *Journal of Systems Architecture*, page 102854, 2023.

[8] X. Gao, J. Yu, Y. Chang, H. Wang, and J. Fan. Checking only when it is necessary: Enabling integrity auditing based on the keyword with sensitive information privacy for encrypted cloud data. *IEEE Transactions on Dependable and Secure Computing*, 19(6):3774–3789, 2021.

[9] J. Xue, S. Luo, Q. Deng, L. Shi, X. Zhang, and H. Wang. Ka: Keyword-based auditing with frequency hiding and retrieval reliability for smart government. *Journal of Systems Architecture*, page 102856, 2023.

[10] J. Xue, S. Luo, L. Shi, X. Zhang, and C. Xu. Enabling hidden frequency keyword-based auditing on distributed architectures for a smart government. In *Frontiers in Cyber Security: 5th International Conference, FCS 2022, Kumasi, Ghana, December 13–15, 2022, Proceedings*, pages 48–68. Springer, 2022.

[11] H. Han, S. Fei, Z. Yan, and X. Zhou. A survey on blockchain-based integrity auditing for cloud data. *Digital Communications and Networks*, 8(5):591–603, 2022.

[12] A. Li, Y. Chen, Z. Yan, X. Zhou, and S. Shimizu. A survey on integrity auditing for data storage in the cloud: from single copy to multiple replicas. *IEEE Transactions on Big Data*, 8(5):1428–1442, 2020.

[13] M. Sadeeq, N. Abdulkareem, S. Zeebaree, D. Ahmed, A. Sami, and R. Zebari. Iot and cloud computing issues, challenges and opportunities: A review. *Qubahan Academic Journal*, 1(2):1–7, 2021.

[14] S. Duan, D. Wang, J. Ren, F. Lyu, Y. Zhang, H. Wu, and X. Shen. Distributed artificial intelligence empowered by end-edge-cloud computing: A survey. *IEEE Communications Surveys & Tutorials*, 2022.

[15] A. Juels, J. Kaliski, and S. Burton. Pors: Proofs of retrievability for large files. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 584–597, 2007.

[16] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou. Enabling public verifiability and data dynamics for storage security in cloud computing. In *Computer Security–ESORICS 2009: 14th European Symposium on Research in Computer Security, Saint-Malo, France, September 21-23, 2009. Proceedings 14*, pages 355–370. Springer, 2009.

[17] N. Dhakad and J. Kar. Eppdp: an efficient privacy-preserving data possession with provable security in cloud storage. *IEEE Systems Journal*, 16(4):6658–6668, 2022.

[18] H. Jin, R. Luo, Q. He, S. Wu, Z. Zeng, and X. Xia. Cost-effective data placement in edge storage systems with erasure code. *IEEE Transactions on Services Computing*, 2022.

[19] D. Kim and K. Kim. Privacy-preserving public auditing for shared cloud data with secure group management. *IEEE Access*, 10:44212–44223, 2022.

[20] X. Zhang, X. Wang, D. Gu, J. Xue, and W. Tang. Conditional anonymous certificateless public auditing scheme supporting data dynamics for cloud storage systems. *IEEE Transactions on Network and Service Management*, 2022.

[21] J. Zhao, Y. Zheng, H. Huang, J. Wang, X. Zhang, and D. He. Lightweight certificateless privacy-preserving integrity verification with conditional anonymity for cloud-assisted medical cyber–physical systems. *Journal of Systems Architecture*, 138:102860, 2023.

[22] W. Shen, J. Qin, J. Yu, R. Hao, and J. Hu. Enabling identity-based integrity auditing and data sharing with sensitive information hiding for secure cloud storage. *IEEE Transactions on Information Forensics and Security*, 14(2):331–346, 2018.

[23] X. Li, S. Shang, S. Liu, K. Gu, M. Jan, X. Zhang, and F. Khan. An identity-based data integrity auditing scheme for cloud-based maritime transportation systems. *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[24] L. Zhou, A. Fu, G. Yang, H. Wang, and Y. Zhang. Efficient certificateless multi-copy integrity auditing scheme supporting data dynamics. *IEEE Transactions on Dependable and Secure Computing*, 19(2):1118–1132, 2020.

[25] R. Li, X. Wang, H. Yang, K. Niu, D. Tang, and X. Yang. Efficient certificateless public integrity auditing of cloud data with designated verifier for batch audit. *Journal of King Saud University-Computer and Information Sciences*, 34(10):8079–8089, 2022.

[26] M. Tian, Y. Zhang, Y. Zhu, L. Wang, and Y. Xiang. DIVRS: data integrity verification based on ring signature in cloud storage. *Computers & Security*, 124:103002, 2023.

[27] S. Ji, W. Zhou, C. Ma, D. Li, K. Zhu, and L. Fang. Proofs of retrievability with tag outsourcing based on goppa codes. *Computer Standards & Interfaces*, 86:103719, 2023.

[28] K. Gai, J. Guo, L. Zhu, and S. Yu. Blockchain meets cloud computing: A survey. *IEEE Communications Surveys & Tutorials*, 22(3):2009–2030, 2020.

[29] L. Zhu, Y. Wu, K. Gai, and K. Kim-Kwang Raymond. Controllable and trustworthy blockchain-based cloud data management. *Future Generation Computer Systems*, 91:527–535, 2019.

[30] Y. Miao, Q. Huang, M. Xiao, and H. Li. Decentralized and privacy-preserving public auditing for cloud storage based on blockchain. *IEEE Access*, 8:139813–139826, 2020.

[31] Y. Yuan, J. Zhang, W. Xu, and Z. Li. Identity-based public data integrity verification scheme in cloud storage system via blockchain. *The Journal of Supercomputing*, 78(6):8509–8530, 2022.

[32] Z. Liu, L. Ren, Y. Feng, S. Wang, and J. Wei. Data integrity audit scheme based on quad merkle tree and blockchain. *IEEE Access*, 2023.

[33] C. Chen, L. Wang, Y. Long, Y. Luo, and K. Chen. A blockchain-based dynamic and traceable data integrity verification scheme for smart homes. *Journal of Systems Architecture*, 130:102677, 2022.

[34] Y. Miao, Q. Huang, M. Xiao, and W. W. Susilo. Blockchain assisted multi-copy provable data possession with faults localization in multi-cloud storage. *IEEE Transactions on Information Forensics and Security*, 17:3663–3676, 2022.

[35] Y. Huang, Y. Yu, H. Li, Y. Li, and A. Tian. Blockchain-based continuous data integrity checking protocol with zero-knowledge privacy protection. *Digital Communications and Networks*, 8(5):604–613, 2022.

[36] H. Yuan, X. Chen, J. Wang, J. Yuan, H. Yan, and W. Susilo. Blockchain-based public auditing and secure deduplication with fair arbitration. *Information Sciences*, 541:409–425, 2020.

[37] G. Tian, Y. Hu, J. Wei, Z. Liu, X. Huang, X. Chen, and W. Susilo. Blockchain-based secure deduplication and shared auditing in decentralized storage. *IEEE Transactions on Dependable and Secure Computing*, 19(6):3941–3954, 2021.

[38] S. Li, C. Xu, Y. Zhang, Y. Du, and K. Chen. Blockchain-based transparent integrity auditing and encrypted deduplication for cloud storage. *IEEE Transactions on Services Computing*, 2022.

[39] Y. Zhang, C. Xu, S. Yu, H. Li, and X. Zhang. Sclpv: Secure certificateless public verification for cloud-based cyber-physical-social systems against malicious auditors. *IEEE Transactions on Computational Social Systems*, 2(4):159–170, 2015.

[40] J. Xue, C. Xu, J. Zhao, and J. Ma. Identity-based public auditing for cloud storage systems against malicious auditors via blockchain. *Science China Information Sciences*, 62(3):1–16, 2019.