Enabling Verifiable Search and Integrity Auditing in Encrypted Decentralized Storage Using One Proof

Mingyang Song, Zhongyun Hua, *Senior Member, IEEE*, Yifeng Zheng, *Member, IEEE*, Qing Liao, and Xiaohua Jia, *Fellow, IEEE*

***Abstract*—Due to the properties of autonomy and scalability, decentralized storage networks (DSNs) leveraging blockchain technology have attracted growing attention. Integrity auditing and verifiable searchable encryption are two essential functions for DSNs. The former ensures reliable and fair storage services, while the latter enables users to conduct keyword searches over encrypted data and guarantees the public verifiability of search results. However, all existing research in DSN has focused either on integrity auditing or on verifiable searchable encryption separately. In this paper, we propose a novel scheme for encrypted decentralized storage that simultaneously supports verifiable search and integrity auditing. It employs a unified proof and supports one-time proof verification to validate both the correctness of the returned file identifiers and the integrity of the files associated with these identifiers. As a result, compared to previous schemes supporting only integrity auditing, our scheme maintains a similar proof size and the support for search result verification does not significantly increase the on-chain storage overhead. Additionally, our scheme allows users to dynamically update their outsourced files while ensuring forward security during the file insertion process. We formally analyze the correctness and security of our scheme, and implement a system prototype to evaluate its performance. The experimental results demonstrate that it achieves verifiable searchable encryption and integrity auditing with practically affordable overhead.**

***Index Terms*—Decentralized storage networks, integrity auditing, verifiable searchable encryption, forward security.**

Received 1 January 2025; revised 14 April 2025; accepted 3 May 2025. Date of publication 9 May 2025; date of current version 11 July 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62071142, in part by the Guangdong Basic and Applied Basic Research Foundation under Grant 2024A1515012299, and in part by the Research Grants Council of Hong Kong CityU under Grant 11213920 and Grant R1012-21. Recommended for acceptance by Y. Chen. *(Corresponding author: Zhongyun Hua.)*

Mingyang Song, Zhongyun Hua, and Qing Liao are with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen Guangdong 518055, China (e-mail: songmingyang2022@gmail.com; huazhongyun@hit.edu.cn; liaoqing@hit.edu.cn).

Yifeng Zheng is with the Department of Electrical and Electronic Engineering, The Hong Kong Polytechnic University, Hong Kong, SAR 518057, China (e-mail: yifeng.zheng@polyu.edu.hk).

Xiaohua Jia is with the Department of Computer Science, City University of Hong Kong, Hong Kong, SAR 518057, China (e-mail: csjia@cityu.edu.hk).

Digital Object Identifier 10.1109/TC.2025.3569182

I. Introduction

DECENTRALIZED storage networks (DSNs), such as Sia [1], Filecoin [2], and FileDAG [3], have the properties of autonomy and scalability. These attributes render them a promising alternative to traditional cloud storage [4]. A DSN typically utilizes blockchain technology to establish fair and transparent storage platform, allowing individual devices (i.e., storage nodes) to rent out their available storage space to earn profits [5], [6]. Integrity auditing is a fundamental function of DSN to achieve Proofs-of-Storage (PoS) [7], [8]. Storage nodes are required to execute PoS for their stored files, convincing their clients that their files are correctly stored and remain intact. Consequently, storage nodes charge clients for these storage services [5]. This paradigm results in a growing number of individual nodes actively participating in the DSN to rent out their unused storage space, while an increasing number of users store their files on the DSN [7].

With concerns on data privacy, users are strongly motivated to encrypt their data before outsourcing. This renders keyword search over encrypted data an essential function for storage outsourcing systems [9]. Without keyword search capability, clients can retrieve files only through file identifiers, adversely affecting the user experience. Searchable encryption enables clients to outsource encrypted data to a server while retaining the ability to search files that match specific keywords at a latter time [10]. During the search process, stringent privacy mechanisms can protect both the file contents and the search keywords against potential attacks from the server [11], [12]. As a result, searchable encryption provides clients significant convenience for retrieving files while protecting the privacy of their data and search keywords. However, in the DSN, a large portion of storage nodes are individual devices, so they may be more prone to violate protocols compared to commercial cloud service providers. These individual nodes may return fewer files to clients to save communication resources or even reuse previous search results without conducting new search operations to save computational resources. Consequently, it is essential for the DSN to support the verification of search results, ensuring that all files containing the search keyword are returned while these do not contain the search keyword are excluded from the search results.

To date, some research has focused either on integrity auditing or on verifiable searchable encryption separately in the

emerging DSN scenarios. Due to the limitations of on-chain storage space within the DSN, several integrity auditing schemes [13], [14], [15], [16] have been developed to optimize the on-chain storage cost. Regarding verifiable searchable encryption for DSN, Cai et al. [12] proposed a scheme that involves an arbiter (i.e., a group of non-colluding servers). It uses the arbiter to store clients' index databases and fairly judge the correctness of returned file identifiers by re-executing search operations. However, the introduction of an arbiter in the DSN fundamentally violates the decentralized principle. Furthermore, a large portion of storage nodes are resource-constrained individual devices and on-chain storage space is limited within the DSN [5], [17]. As the verification of both search results and data integrity causes storage burden on the blockchain and computational burden on storage nodes, a custom design is required to support these two functions in the DSN with minimal overhead.

Simultaneously achieving integrity auditing and verifiable searchable encryption within the DSN is a challenging task, and it involves overcoming two key issues to ensure practical usability. (1) Any third-party server cannot be introduced into the DSN as this is incompatible with its fundamental decentralized feature. (2) Since all data storage and deletion events must be permanently recorded on the blockchain to maintain the DSN ecosystem, it is crucial to minimize on-chain storage space usage for supporting both verifiable searchable encryption and integrity auditing.

In this paper, we propose a novel scheme that simultaneously supports verifiable searchable encryption and integrity auditing within the DSN. It can support both operations using a single proof with one-time verification, minimizing on-chain storage space usage. To achieve this, we design the keyword-associated tags to link all files that share the same keyword. When a client initiates a keyword search, the storage node uses these keyword-associated tags and authentication tags of the searched files to generate a keyword-associated proof. This proof can be used to simultaneously verify the correctness of searched file identifiers and integrity of searched files. By linking the file identifiers through keyword-associated tags, the loss of any file will result in an incorrect proof that fails to pass the verification. The integrity of other unsearched files is checked through the general auditing process within the DSN. This integrated approach ensures that the verification of search results does not significantly increase the on-chain storage overhead.

Since clients may update files stored on storage nodes [18], a DSN should support file dynamic operations while ensuring forward security [19] during the file insertion process. Forward security ensures that the addition of a new file does not reveal its association with previous searches [20]. To ensure forward security, we design a state-associated token involving both a unique state and encrypted keyword to search index database. During the file insertion process, the client selects different states for different files that share the same keyword. Thus, the storage node cannot use previous state-associated tokens of the same keyword to associate with a newly inserted file, ensuring the forward security. To address the situation that one state-associated token cannot match all files sharing the same keyword, we include an encrypted pointer in the index database. When adding a new keyword-file identifier pair, the client computes a pointer by encrypting the previous state using the newly selected state and sends it to the storage node. As a result, different states of the same keyword are linked chronologically based on file insertion, using these pointers. When conducting a keyword search, the client can provide only the current state and the encrypted keyword to the storage node. The storage node can then recover all previous keyword states from these pointers and generate all state-associated tokens of the keyword. This process can retrieve all files sharing the same keyword while preserving forward security. When deleting a file, the storage node can directly delete the ciphertext and its authentication tag. To maintain the search for other files sharing the same keyword as the deleted file, the storage node still retains the keyword-associated tag and pointer after the file deletion.

The contributions of this paper are summarized as follows.

- We propose a customized scheme for encrypted decentralized storage that simultaneously supports verifiable search and integrity auditing. Our scheme designs a unified proof and employs one-time proof verification to validate both the correctness of the returned file identifiers and the integrity of the searched files. Compared to previous schemes that support only integrity auditing in the DSN, our scheme maintains a similar proof size and the support for search result verification does not significantly increase the on-chain storage overhead.
- Our scheme is designed to support file dynamic operations while ensuring forward security, which indicates that storage nodes cannot establish associations between newly inserted files with previous searches.
- We formally verify the correctness of our scheme and prove its security. The experimental results show that our scheme can achieve practically affordable overhead.

The rest of the paper is organized as follows. Section II presents the problem formalization of our scheme and Section III presents its detailed design. Section IV provides the correctness and security analysis. We evaluate the performance of our scheme in Section V and review related works in Section VI. Finally, we conclude our study in Section VII.

II. Problem Statement

A. Preliminaries

1) Blockchain: A blockchain network is typically maintained by numerous nodes based on a consensus mechanism [21]. The blockchain consists of a linear set of data blocks, each contains a timestamp, a nonce, the hash value of the previous block, and other data. Given that malicious nodes hold less computation power than honest nodes, the verifiability and immutability of blockchain data can be ensured. The Ethereum blockchain [21] provides a decentralized and tamper-resistant platform for deploying and executing smart contracts, enabling a wide range of decentralized applications to be built on the Ethereum. Most existing DSNs, including Storj [22] and Swarm [8], utilize the Ethereum to establish fair and transparent storage platforms, allowing individual devices to rent out their available storage space to earn profits.

2) Bilinear Pairing of Composite Order: Suppose that κ is a security parameter, p and q are two κ-bit length primes. A bilinear pairing [23] $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ is an algorithm with the following three properties:

- *Bilinearity*: For $\forall \eta_1, \eta_2 \in \mathbb{G}_1$ and $\forall a, b \in \mathbb{Z}_N$, $e(\eta_1^a, \eta_2^b) = e(\eta_1, \eta_2)^{a \cdot b}$.
- *Non-degenerate*: If g is a generator of $\mathbb{G}_1$, then $e(g, g)$ is a generator of $\mathbb{G}_2$.
- *Computability*: For $\forall \eta_1, \eta_2 \in \mathbb{G}_1$, it is efficient to compute $e(\eta_1, \eta_2) \in \mathbb{G}_2$.

Definition 1 (The computational Diffie-Hellman (CDH) assumption [24]): Given g, g^x and $g_1 \in \mathbb{G}_1$, for unknown $x \in \mathbb{Z}_N$, there is no probabilistic polynomial-time algorithm that can compute g_1^x with non-negligible advantage.

B. System Model and Definitions

There are three types of entities in a DSN: client, storage node, and blockchain.

- **Client:** Clients are individuals or entities with the requirement to store files in the storage nodes of the DSN. To ensure the security of their data, they usually encrypt their files before uploading them for storage. They can retrieve their stored files from the DSN by employing encrypted keyword search.
- **Storage node:** Storage nodes rent out their hard disk space to clients and provide storage and retrieval services. During each auditing period, a storage node submits proofs to the blockchain, proving the integrity of its stored files and the correctness of the search results it has delivered to its clients. These proofs are verified by other storage nodes within the DSN, excluding the storage node that generates the proofs.
- **Blockchain:** The blockchain servers as a public ledger within the DSN. It records the search requests initiated by each client, the proofs generated by the storage node, and the verification results provided by other storage nodes.

We formally define our scheme over the aforementioned system model as follows.

Definition 2: The proposed scheme consists of an eight-tuple of polynomial-time algorithms (**Setup, KeyGen, Insert, Delete, Search, Proof, Verify.Search, Verify.Audit**). These algorithms are defined as follows:

- **Setup**$(\kappa) \rightarrow (PP)$. It takes a security parameter κ as input, and outputs the public parameter collection PP.
- **KeyGen**$(PP) \rightarrow (SK, pk)$. The algorithm accepts the public parameters PP as its input, and outputs the collection of secret keys SK and the public key pk.
- **Insert**$(F, W, SK, PP, SS, \mathrm{DB}) \rightarrow (TS_F, \mathrm{DB}', C, SS')$. The algorithm is designed to operate on various input parameters, including the plaintext file F, the keyword collection W of the file, the secret key collection SK, the public parameter collection PP, the keyword state collection SS, and the encrypted index database DB. It generates multiple outputs, including the ciphertext file C, the collection of authentication tags collection TS_F, the updated keyword state collection SS', and the updated index database DB'.
- **Delete**$(ID_F, SK, \mathrm{DB}) \rightarrow (\mathrm{DB}')$. It takes the file identifier ID_F of the file to be deleted, the collection of secret keys SK, and the encrypted index database DB as its inputs, and it outputs the updated index database DB'.
- **Search**$(w, SS(w), SK, TS, \mathrm{DB}) \rightarrow (T, Res, PS)$. The algorithm takes several inputs, including the keyword w, the state $SS(w)$ of the keyword, the collection of secret keys SK, the authentication tag collection TS, and index database DB. It then outputs the search token T, the search result Res, and a collection PS of keyword-associated proof.
- **Proof**$(ID_F, C, TS_F) \rightarrow (Proof)$. The algorithm takes the file identifier ID_F, the ciphertext C associated with ID_F, and the authentication tag collection TS_F corresponding to ID_F as its inputs. It generates the integrity proof as output.
- **Verify.Search**$(T, SS(w), Res, PS, PP, pk) \rightarrow (1/0)$. The algorithm accepts several inputs, including the search token T, the state $SS(w)$ of the search keyword, the search result Res, the collection PS of keyword-associated proof, the public parameters PP, and client's public key pk. It then produces an output of 1 to indicate that the search result Res is valid, and all the files in Res are intact. Conversely, the algorithm outputs 0 to signify that the search result Res is invalid.
- **Verify.Audit**$(ID_F, Proof, PP, pk) \rightarrow (1/0)$. The algorithm takes the file identifier ID_F, the integrity proof, the public parameters PP, and the client's public key pk as its inputs, and produces an output representing the integrity checking result.

C. Threat Model and Security Definitions

The threats of the system originate from the storage nodes, which involve three types of dishonest behaviors as follows.

- A storage node may attempt to learn the private contents of the files its stored or the keywords included in those files using its background knowledge.
- For practical reasons such as hardware failures, a storage node may experience data loss. When a file is lost, it may deceive the client by forging an integrity proof and claiming illegal storage fees.
- When a keyword search request involves some lost files, a storage node may provide an incorrect search result by returning incorrect files with forged proof.

Our scheme aims to achieve privacy against chosen keyword attacks (i.e., IND-CKA) [25], forward security during the file insertion process, verifiability of data integrity, and verifiability of search result correctness. We provide formal security definition for each security goal in the rest of this section. We will demonstrate the security of our scheme in Section IV-B by proving that our scheme satisfies these definitions.

Definition 3: The IND-CKA security of our scheme is modeled by a game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

- Setup: The challenger $\mathcal{C}$ runs the **KeyGen** algorithm to obtain the collection of its secret keys SK and the public key pk, and sends pk to the adversary $\mathcal{A}$.

- Phase 1: The adversary $\mathcal{A}$ initializes an empty index database DB and the challenger $\mathcal{C}$ initializes an empty keyword list $\mathcal{L}$. Then the adversary $\mathcal{A}$ and the challenger $\mathcal{C}$ perform the following oracles.
  - $\mathcal{O}_{Query}$: The adversary $\mathcal{A}$ sends a keyword w and its associated file F to the challenger $\mathcal{C}$.
  - $\mathcal{O}_{IndBuild}$: The challenger $\mathcal{C}$ and the adversary $\mathcal{A}$ perform the **Insert** algorithm to add the index associated with the keyword w to the index database DB. The challenger $\mathcal{C}$ also adds the keyword w to $\mathcal{L}$.
- Challenge: The adversary $\mathcal{A}$ selects two keywords $w_0, w_1 \notin \mathcal{L}$ and sends them and their associated files to the challenger $\mathcal{C}$. The challenger $\mathcal{C}$ randomly picks a bit $b \in \{0, 1\}$. Then the challenger $\mathcal{C}$ and the adversary $\mathcal{A}$ execute the **Insert** algorithm to add the index associated with the keyword w_b to the index database DB.
- Phase 2: The adversary $\mathcal{A}$ repeats Phase 1, with the limitation that w_0, w_1 cannot be reissued again in the two oracles mentioned above.
- Guess: The adversary $\mathcal{A}$ outputs a guess $\hat{b}$ of b.

The advantage of the adversary $\mathcal{A}$ can be defined as (1), and The keyword is secure if $\mathcal{A}$ has a negligible advantage.

$$\mathrm{Adv}_{\mathcal{A}}^{\mathrm{IND-CKA}}(\kappa) = 2 \cdot \Pr[\mathcal{A}(\hat{\mathrm{b}} = \mathrm{b})] - 1 \tag{1}$$

Definition 4: Our scheme exhibits forward security if no adversary can exploit the association between the newly inserted files and searches conducted in the past.

Definition 5: Our scheme guarantees verifiability of data integrity if no adversary can successfully pass the verification in the **Verify.Audit** algorithm except by computing the proof using the correct file.

Definition 6: The correctness of search result (i.e., returned file identifiers) in our scheme is verifiable if the storage node has a negligible probability of winning the following game.

- The client inserts the indices of files $\{F_i\}_{i=1}^{\delta}$ that contain the same keyword w into the index database DB stored in the storage node.
- The client searches for the keyword w by executing the **Search** algorithm with the storage node. The storage node outputs an incorrect collection Res of file identifiers and proof PS.

The adversary $\mathcal{A}$ wins the game if $Res \neq \{ID_{F_i}\}_{i=1}^{\delta}$ and the proof PS pass the verification of **Verify.Search** algorithm.

D. Design Goals

In this study, we aim to achieve the verification of search results and data integrity in encrypted decentralized storage, with the following specific goals.

- **Functionality:** Our scheme has two fundamental functions: **verifiable keyword search** and **integrity auditing** over encrypted data. A client can store his/her encrypted files on a storage node and conduct encrypted search on these files. The storage node can provide proofs to demonstrate the correctness of search results and the integrity of stored files. Both search result correctness and data integrity are verified concurrently. Additionally, clients can insert new files into encrypted databases or delete files from them. The two fundamental functions remain effective when file dynamic updates occur.
- **Security:** Our scheme offers security in terms of **keyword privacy**, **forward security** during the file insertion process, **verifiability of search result correctness** and **verifiability of data integrity**. These security features are described as follows:
  1) A storage node cannot deduce plaintext keywords from its known information.
  2) A storage node cannot discern the relationship between a newly inserted file and any past search request.
  3) The correctness of search results delivered by a storage node can be confirmed by other storage nodes within the DSN.
  4) The integrity of the files stored in a storage node can be verified by other storage nodes within the DSN.

III. The Proposed Scheme

This section begins with a high-level design overview of our scheme, followed by detailed algorithm design (i.e., detailed descriptions of functions defined in Definition 2). Finally, it presents the system workflow to illustrate the execution sequence of these functions.

A. Overview of Our Scheme

To simultaneously verify data integrity and search result correctness, we design a keyword-associated tag *kt*, including the file identifier *ID*, search token *T*, and keyword states *st*. The client signs these contexts using the same secret key that is used for generating general authentication tags. Then keyword-associated tags and authentication tags are combined to generate keyword-associated proof, proving both data integrity and search result correctness simultaneously. The combination of file identifier and search token in the keyword-associated tag prevents the storage node from substituting search result with files associated with different keywords. Additionally, each file with the same keyword has a unique state. During the file insertion process, the states in the keyword-associated tag include the state of the currently inserted file and the state of the last inserted file with the same keyword. Then all the files with the same keyword are linked in the chronological order of file insertion through keyword-associated tags. Thus, the loss of any file results in an incorrect keyword-associated proof that cannot pass the verification.

To ensure forward security, we introduce a state-associated token $\overline{T}$, derived from search token *T* and a state *st*, for indexing. For each item insertion, the client selects a new hidden state, keeps it hidden from the storage node, and computes a unique state-associated token. Even if the storage node obtains *T* and all previous states, it cannot link them to the newly inserted state-associated token, thereby preserving forward security. To preserve the search function, we introduce a state-associated pointer in the index database. When inserting an item, the client computes a pointer by encrypting the previous keyword state with the newly selected state, linking keyword states chronologically. During a search, the client sends the latest state and search token

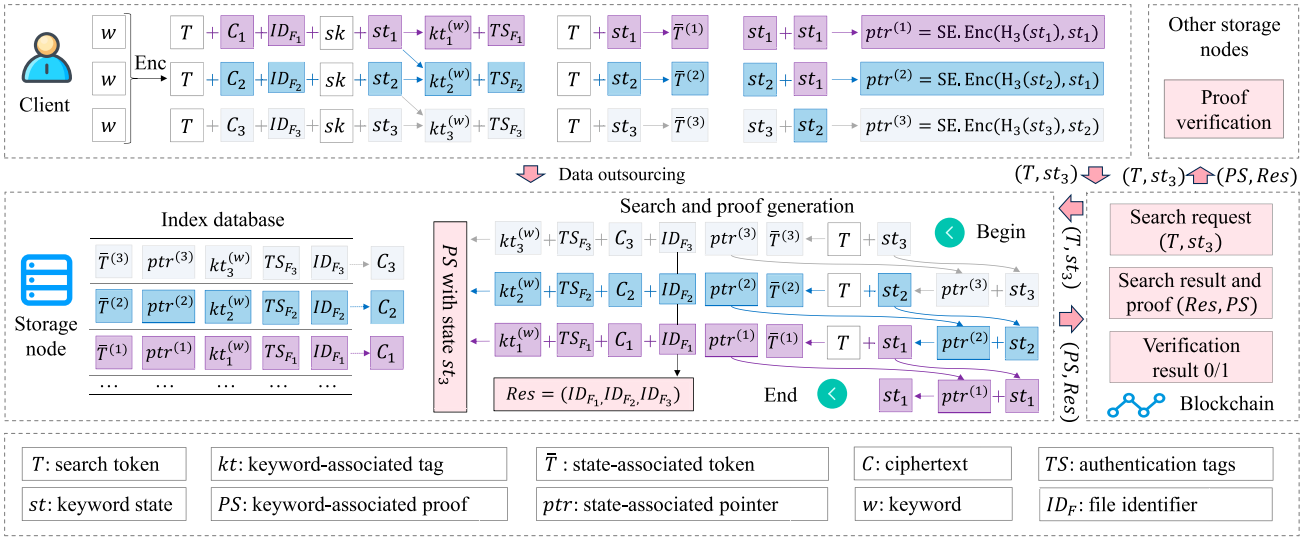


Fig. 1. Example of our scheme on three files associated with the same keyword.

to the storage node, which derives the state-associated token of the last inserted file and retrieves its pointer. The pointer reveals the previous state, enabling the retrieval of the next file. By repeating this process, all files associated with the keyword can be accessed.

Fig. 1 illustrates an example of our scheme involving three files $\{F_i\}_{(1\le i\le 3)}$ associated with the same keyword w. Before data outsourcing, the client performs the following operations. (1) Generate authentication and keyword-associated tags. The authentication tag is generated using the secret key sk, file identifier ID_F, and ciphertext C. The keyword-associated tag additionally incorporates the encrypted keyword T and keyword state st. For the first file F_1, the keyword-associated tag includes the unique state st_1. For subsequent files F_i, it includes both the current state st_i and previous state st_{i-1}. (2) Compute a state-associated token $\overline{T}^{(i)}$ for each file F_i using the encrypted keyword T and state st_i. (3) Generate the state-associated pointer $ptr^{(1)}$ of the first file F_1 by encrypting st_1 using the key $\mathrm{H}_3(st_1)$. For subsequent files F_i, generate the pointer $ptr^{(i)}$ by encrypting st_{i-1} using the key $\mathrm{H}_3(st_i)$. After data outsourcing, the storage node constructs the index database as shown in Fig. 1.

During the search process, the client publishes the search token T and latest state st_3 on the blockchain. The storage node uses the tuple (T, st_3) to perform search as follows. (1) Compute $\overline{T}^{(3)}$ using (T, st_3) and utilize $\overline{T}^{(3)}$ to retrieve $(ptr^{(3)}, kt_3^{(w)}, TS_{F_3}, ID_{F_3}, C_3)$. (2) Decrypt $ptr^{(3)}$ using $\mathrm{H}_3(st_3)$ to obtain st_2, then compute $\overline{T}^{(2)}$ using (T, st_2) to retrieve $(ptr^{(2)}, kt_2^{(w)}, TS_{F_2}, ID_{F_2}, C_2)$. (3) Decrypt $ptr^{(2)}$ using $\mathrm{H}_2(st_2)$ to obtain st_1, and compute $\overline{T}^{(1)}$ using (T, st_1) to retrieve $(ptr^{(1)}, kt_1^{(w)}, TS_{F_1}, ID_{F_1}, C_1)$. (4) Decrypt $ptr^{(1)}$ using $\mathrm{H}_2(st_1)$ to reveal st_1 again, indicating the termination of search chain. For proof generation, the storage node uses the retrieved set $\{C_i, TS_{F_i}, kt_i^{(w)}\}_{(1\le i\le 3)}$ to generate a keyword-associated proof PS, which is then published along with the search result Res on the blockchain. Other storage nodes can verify the proof. Since the files are linked through states embedded in keyword-associated tags, any file loss or tampering will result in the inclusion of invalid states (i.e., states other than st_3) in the proof, thereby causing the verification to fail.

B. Detailed Design

1) System Setup: In this process, a storage node runs the **Setup**$(\kappa) \to (PP)$ algorithm to generate the collection of public parameters PP.

- The storage node runs the composite bilinear parameter generator $\mathcal{G}en(\kappa)$ to obtain $(p, q, N, \mathbb{G}_1, \mathbb{G}_2, \mathrm{e})$, where κ is a randomly selected security parameter.
- The storage node selects a generator $g \in \mathbb{G}_1$ and a random element $\mu \in \mathbb{G}_1$. It then publishes the public parameters $PP = \{N, g, \mu\}$ on the blockchain.
- The storage node chooses three hash functions $\mathrm{H}_1 : \{0,1\}^* \to \mathbb{Z}_N$, $\mathrm{H}_2 : \{0,1\}^* \to \mathbb{G}_1$, $\mathrm{H}_3 : \{0,1\}^* \to \{0,1\}^\iota$, and a pseudo-random function $\pi : \{0,1\}^* \times \mathbb{Z}_N \to \mathbb{Z}_N$, where ι is the bit length of encryption key. These functions, $\mathrm{H}_1, \mathrm{H}_2, \mathrm{H}_3, \pi, \mathrm{e}$, are public functions.

2) Key Generation: A client executes the **KeyGen**$(PP) \to (SK, pk)$ algorithm to generate his/her secret keys SK and public key pk before outsourcing files to the storage node.

- The client selects an integer $sk \in \mathbb{Z}_N$ and two binary strings $mk, ek \in \{0,1\}^\iota$, where ι represents the bit length of encryption key.
- The client computes the public key as $pk = g^{sk}$, and publishes the public key on the blockchain. The secret keys $SK = \{mk, sk, ek\}$ are securely kept by the client.

3) File Insertion: A client outsources a new file $F = \{m_i\}_{i=1}^n$, along with its corresponding keywords $W = \{w_i\}_{i=1}^\ell$ to a storage node. The client interacts with the storage node to execute the **Insert**$(F, W, SK, PP, SS, \mathrm{DB}) \to (TS_F, \mathrm{DB}', C, SS')$ algorithm, completing the outsourcing process, where SS is a state collection maintained by the client, recording the latest state of each keyword. Note that the keyword state collection is initially empty before the first execution of this algorithm.

- The client encrypts each block m_i of the file F using an existing symmetric encryption algorithm (e.g., AES-256)

to obtain ciphertext $c_i = \text{SE.Enc}(ek, m_i)$, $(1 \le i \le n)$. Additionally, the client computes the file identifier as $ID_F = \text{H}_1(C)$, where $C = \{c_i\}_{i=1}^n$.

- For each ciphertext block $c_i \in C$, the client computes the authentication tag using Eq. (2), where a ciphertext block c_i consists of s sectors $\{c_{i,j}\}_{j=1}^s$. Finally, the client obtains the authentication tag collection $TS_F = \{\sigma_i\}_{i=1}^n$.

$$\sigma_i = \left[\text{H}_2(ID_F \parallel i) \cdot \prod_{j=1}^{s} \mu^{c_{i,j}}\right]^{sk} \tag{2}$$

- For each keyword w_i, the client generates a search token T_i by encrypting the keyword w_i using the secret key mk, namely $T_i = \text{SE.Enc}(mk, w_i)$. The client then selects a state $st_d \in \mathbb{Z}_N$, and computes state-associated token $\overline{T}_i = \text{H}_2(T_i \parallel st_d)$. The client also extracts the previous state of the keyword w_i as $st_{d-1} = SS(w_i)$. If $st_{d-1} = \perp$, indicating that there is no previous state associated with the keyword, the client adds (w_i, st_d) to the state collection SS. It then computes the state-associated pointer $ptr^{(d-1)} = \text{SE.Enc}(\text{H}_3(st_d), st_d)$ and the keyword-associated tag $kt^{(w_i)}$ using Eq. (3).

$$kt^{(w_i)} = [\text{H}_2(ID_F) \cdot \text{H}_2(st_d \parallel T_i)]^{sk} \tag{3}$$

Otherwise (i.e., when $st_{d-1} \neq \perp$), the client updates the entry (w_i, st_{d-1}) with (w_i, st_d) in the keyword state collection SS. It then computes the state-associated pointer as $ptr^{(d-1)} = \text{SE.Enc}(\text{H}_3(st_d), st_{d-1})$ and the keyword-associated tag $kt^{(w_i)}$ using Eq. (4).

$$kt^{(w_i)} = \left[\text{H}_2(ID_F) \cdot \frac{\text{H}_2(st_d \parallel T_i)}{\text{H}_2(st_{d-1} \parallel T_i)}\right]^{sk} \tag{4}$$

- The client sends $\{\{ID_F, \overline{T}_i, ptr^{(d-1)}, kt^{(w_i)}\}_{i=1}^{\ell}, TS_F, C\}$ to the storage node. Then, the storage node inserts $\{\overline{T}_i, ptr^{(d-1)}, valid, kt^{(w_i)}, ID_F\}_{i=1}^{\ell}$ into its index database DB and directly stores TS_F and C.

4) File Deletion: A client interacts with the storage node to delete a file F using the **Delete**$(ID_F, SK, \text{DB}) \rightarrow (\text{DB}')$.

- The client computes $del = \text{H}_2(ID_F)^{sk}$ and then sends (ID_F, del) to the storage node.
- The storage node deletes the ciphertext and authentication tags associated with ID_F. For each item $(\overline{T}, ptr^{(d-1)}, valid, kt^{(w_i)}) \in \text{DB}[ID_F]$, the storage node updates $kt^{(w_i)}$ with $\frac{kt^{(w_i)}}{del}$. Additionally, the storage node marks the file state as *invalid*.

5) Keyword Search: A client can perform searches within the database DB by executing **Search**$(w, SS(w), SK, TS, \text{DB}) \rightarrow (T, Res, PS)$ in collaboration with the storage node. Suppose d files containing the same keyword w (i.e., the keyword w associates with d states), the keyword w corresponds to a pointer chain, and all files containing the keyword w can be sequentially retrieved using the latest state st_d associated with the keyword and the search token T. Additionally, the search process generates a keyword-associated proof. To enhance the readability of algorithm, we directly use the seed θ from periodic auditing in the DSN to generate challenges.

- The client retrieves the latest state $st_d = SS(w)$ of the keyword w and encrypts the keyword w to generate the search token $T = \text{SE.Enc}(mk, w)$. Then, the client publishes (T, st_d) on the blockchain.
- The storage node obtains (T, st_d) from the blockchain and initializes an empty set $AS = \emptyset$ to assist in keyword-associated proof generation. Starting from the state st_α $(\alpha = d)$, the storage node performs the search as follows.
  1) The storage node computes the state-associated token $\overline{T}^{(\alpha)} = \text{H}_2(T \parallel st_\alpha)$ and retrieves the item $(ptr^{(\alpha-1)}, op, kt_\alpha^{(w)}, ID_{F_\alpha}) = \text{DB}[\overline{T}^{(\alpha)}]$ corresponding to the $\overline{T}^{(\alpha)}$, where ID_{F_α} is the identifier of the α^{th} file containing the keyword and $kt_\alpha^{(w)}$ is the keyword-associated tag corresponding to the file. It then decrypts and obtains $st_{\alpha-1} = \text{SE.Dec}(\text{H}_3(st_\alpha), ptr^{(\alpha-1)})$ while adding ID_{F_α} to the set AS.
  2) Only when $op = valid$, the storage node includes ID_{F_α} in the result set Res and generates $(\psi_\alpha, \varphi_\alpha)$ as specified in Eq. (5), where $\{c_i^{(\alpha)}\}_{i=1}^n$ and $\{\sigma_i^{(\alpha)}\}_{i=1}^n$ are the ciphertext and authentication tags associated with ID_{F_α}, $c_i^{(\alpha)}$ consists of s sectors $\{c_{i,j}^{(\alpha)}\}_{j=1}^s$, and θ is the seed for generating random challenges.

$$\begin{cases} \psi_\alpha = \sum_{i=1}^{n}\sum_{j=1}^{s} \pi(\theta \parallel ID_{F_\alpha}, i) \cdot c_{i,j}^{(\alpha)} \\ \varphi_\alpha = \prod_{i=1}^{n} \left(\sigma_i^{(\alpha)}\right)^{\pi(\theta \parallel ID_{F_\alpha}, i)} \end{cases} \tag{5}$$

  Then the storage node adds $(ID_{F_\alpha}, \psi_\alpha, \varphi_\alpha)$ to the proof set PS.
  3) If $st_{\alpha-1} \neq st_\alpha$, the storage node decrements α by one $(\alpha = \alpha - 1)$ and repeats steps 1) and 2).
- The storage node computes the proof φ using Eq. (6) and includes φ to the keyword-associated proof set PS. Finally, it posts (T, st_d, Res, PS) on the blockchain.

$$\varphi = \prod_{ID_{F_\alpha} \in AS} kt_\alpha^{(w)} \tag{6}$$

6) Proof Generation: In the DSN, the storage node must periodically provide integrity proofs for its stored files to demonstrate the continuous data availability. Within a single auditing period, the integrity of files that have been searched can be verified using the keyword-associated proof generated by the **Search** algorithm. The storage node providing storage service for clients executes the **Proof**$(ID_F, C, TS_F) \rightarrow (Proof)$ algorithm to prove the integrity of its stored each file F that has not been searched.

- The storage node uses ID_F to retrieves the ciphertext $C = \{c_i\}_{i=1}^n$ and authentication tags $TS_F = \{\sigma_i\}_{i=1}^n$.
- The storage node computes the integrity proof $Proof = (\psi, \varphi)$ using Eq. (7), where θ represents the seed for generating challenges in the current auditing period.

$$\begin{cases} \psi = \sum_{i=1}^{n}\sum_{j=1}^{s} \pi(\theta \parallel ID_F, i) \cdot c_{i,j} \\ \varphi = \prod_{i=1}^{n} \sigma_i^{\pi(\theta \parallel ID_F, i)} \end{cases} \tag{7}$$

- $(ID_F, Proof)$ are published on the blockchain.

7) Proof Verification: During an auditing period, other storage nodes in the DSN can verify the correctness of search result and the integrity of these searched files by verifying the keyword-associated proof through the **Verify.Search**$(T, SS(w), Res, PS, PP, pk) \rightarrow (1/0)$ algorithm.

- A storage node in the DSN computes $(\zeta_1, \zeta_2, \zeta_3, \rho)$ as Eq. (8).

$$\begin{cases} \zeta_1 = \prod\limits_{ID_{F_\alpha} \in Res} \prod\limits_{i=1}^{n} \mathrm{H}_2(ID_{F_\alpha} \parallel i)^{\pi(\theta \| ID_{F_\alpha}, i)} \\ \zeta_2 = \prod\limits_{ID_{F_\alpha} \in Res} \mathrm{H}_2(ID_{F_\alpha}) \\ \zeta_3 = \prod\limits_{ID_{F_\alpha} \in Res} \varphi_\alpha \cdot \varphi \\ \rho = \sum\limits_{ID_{F_\alpha} \in Res} \psi_\alpha \end{cases} \tag{8}$$

- The storage node verifies the keyword-associated proof using the following equation.

$$\mathrm{e}(\zeta_3, g) \stackrel{?}{=} \mathrm{e}(\zeta_1 \cdot \zeta_2 \cdot \mathrm{H}_2(st_d \parallel T) \cdot \mu^\rho, pk) \tag{9}$$

  If the equation holds, it indicates that the search result *Res* is correct and the searched files remain intact.

For each file ID_F that has not been searched during the current auditing period, other storage nodes in the DSN verify its integrity using the **Verify.Audit**$(ID_F, Proof, PP, pk) \rightarrow (1/0)$ algorithm.

- A storage node computes

$$\zeta = \prod_{i=1}^{n} \mathrm{H}_2(ID_F \parallel i)^{\pi(\theta \| ID_F, i)}.$$

- The storage node verifies the $Proof = (\psi, \varphi)$ by checking whether Eq. (10) holds.

$$\mathrm{e}(\varphi, g) \stackrel{?}{=} \mathrm{e}(\zeta \cdot \mu^\psi, pk) \tag{10}$$

  If the equation holds, it indicates that the file corresponding to the identifier ID_F remains intact.

C. System Workflow

We present the system workflow of our scheme, i.e., the execution sequence of functions outlined in Definition 2.

- The storage node generates its public parameters by executing the **Setup** algorithm.
- The client executes the **KeyGen** algorithm to generate his/her secret keys and public key, before outsourcing files to the storage node. By executing the **Insert** algorithm, keyword indices of each file can be inserted into the storage node's index database.
- The client can interact with the storage node to delete a file by executing the **Delete** algorithm.
- The client can search for a keyword by executing the **Search** algorithm with the storage node. During this process, the storage node generates keyword-associated proof for both the correctness of the search result and the integrity of the searched files. The storage node executes the **Proof** algorithm to generate integrity proofs for unsearched files in the current auditing period.
- During each auditing period, other storage nodes within the DSN, excluding the storage node that generates the proofs, are responsible for checking the correctness of the search result and the integrity of the searched files by performing the **Verify.Search** algorithm. Additionally, they verify the integrity of other unsearched files using the **Verify.Audit** algorithm.

Discussion. The deployment of our scheme in practical DSNs faces two challenges: node heterogeneity and incentive compatibility. (1) Storage nodes are typically personal computers with varying computational and storage capabilities. With the introduction of encrypted data search functionality, some nodes with limited resources may be unable to process all search requests and proof generation tasks within the designated auditing period. Therefore, it may be necessary for DSNs to extend the auditing period to provide resource-constrained nodes with more time to complete these operations. (2) In practical DSNs, storage nodes are rewarded only based on the storage space they contribute, without incentive for performing encrypted data search. This lack of incentives may cause some nodes to decline these tasks, even when our scheme is deployed. Therefore, underlying incentive mechanisms are required in practical DSNs to encourage and reward nodes for conducting encrypted data searches and verifying results.

IV. Correctness and Security Analysis

A. Correctness

1) Keyword Search: Assume that the files $\{F_\alpha\}_{\alpha=1}^{d}$ contain the same keyword w, and after inserting their indices into the index database DB, $\{\overline{T}^{(\alpha)}, ptr^{(\alpha-1)}, ID_{F_\alpha}\}_{\alpha=1}^{d}$ are stored in the storage node, where $ptr^{(0)} = \mathrm{SE.Enc}(\mathrm{H}_3(st_1), st_1)$, $ptr^{(\alpha-1)} = \mathrm{SE.Enc}(\mathrm{H}_3(st_\alpha), st_{\alpha-1}), (2 \le \alpha \le d)$ and $\overline{T}^{(\alpha)} = \mathrm{H}_2(T \parallel st_\alpha)$, $(1 \le \alpha \le d)$. We now demonstrate how the storage node can retrieve the files that contain the same keyword w using the search token $T = \mathrm{SE.Enc}(mk, w)$ and the latest state st_d provided by the client.

It is obvious that the storage node can retrieve the correct file identifiers $\{ID_{F_\alpha}\}_{\alpha=1}^{d}$ if it can recover $\{\overline{T}^{(\alpha)}\}_{\alpha=1}^{d}$ from the provided (T, st_d). The storage node can first obtain the state-associated token $\overline{T}^{(d)} = \mathrm{H}_2(T \parallel st_d)$ and then derive the state-associated tokens $\{\overline{T}^{(\alpha)}\}_{\alpha=1}^{d-1}$ using the following steps.

1) Retrieve $ptr^{(\alpha-1)}$ from DB using the $\overline{T}^{(\alpha)}$.
2) Obtain $st_{\alpha-1} = \mathrm{SE.Dec}(\mathrm{H}_3(st_\alpha), ptr^{(\alpha-1)})$ and $\overline{T}^{(\alpha-1)} = \mathrm{H}_2(T \parallel st_{\alpha-1})$.
3) Repeat steps 1) and 2) until $st_{\alpha-1} = st_\alpha$.

It is evident that the storage node can recover $\{\overline{T}^{(\alpha)}\}_{\alpha=1}^{d}$ from (T, st_d). Thus, the storage node can accurately retrieve the file identifiers $\{ID_{F_\alpha}\}_{\alpha=1}^{d}$ from DB using $\{\overline{T}^{(\alpha)}\}_{\alpha=1}^{d}$, demonstrating the correctness of keyword search process.

2) Integrity Proof Verification: A storage node within DSN checks the integrity proof (ψ, φ) by verifying whether Eq. (10) holds, where $\psi = \sum_{i=1}^{n} \sum_{j=1}^{s} \pi(\theta \parallel ID_F, i) \cdot c_{i,j}$, $\varphi = \prod_{i=1}^{n} \sigma_i^{\pi(\theta \| ID_F, i)}$ and $\zeta = \prod_{i=1}^{n} \mathrm{H}_2(ID_F \parallel i)^{\pi(\theta \| ID_F, i)}$. The storage

node computes the two sides of Eq. (10) as follows.

$$\begin{aligned}&\mathrm{e}(\varphi, g)\\&=\mathrm{e}\left(\prod_{i=1}^{n}\sigma_i^{\pi(\theta\|ID_F,i)}, g\right)\\&=\mathrm{e}\left(\prod_{i=1}^{n}\left(\mathrm{H}_2(ID_F\parallel i)\cdot\mu^{\sum_{j=1}^{s}c_{i,j}}\right)^{\pi(\theta\|ID_F,i)}, g^{sk}\right)\\&\mathrm{e}(\zeta\cdot\mu^{\psi}, pk)\\&=\mathrm{e}\left(\prod_{i=1}^{n}\left(\mathrm{H}_2(ID_F\parallel i)\cdot\mu^{\sum_{j=1}^{s}c_{i,j}}\right)^{\pi(\theta\|ID_F,i)}, g^{sk}\right)\end{aligned}$$

It is obvious that $\mathrm{e}(\varphi, g)=\mathrm{e}(\zeta\cdot\mu^{\psi}, pk)$, demonstrating the correctness of the integrity proof verification.

3) Keyword-Associated Proof Verification: Assume that the files $\{F_\alpha\}_{\alpha=1}^{d}$ contain the same keyword w, and after inserting their indices into the index database DB, both the keyword-associated tags $\{kt_\alpha^{(w)}\}_{\alpha=1}^{d}$ and the authentication tags $\{\sigma_i^{(\alpha)}\}_{i=1}^{n}, (1\le\alpha\le d)$ are stored in the storage node, where $kt_1^{(w)}=[\mathrm{H}_2(ID_{F_1})\cdot\mathrm{H}_2(st_1\parallel T)]^{sk}$, $kt_\alpha^{(w)}=[\mathrm{H}_2(ID_{F_\alpha})\cdot(\mathrm{H}_2(st_\alpha\parallel T)/\mathrm{H}_2(st_{\alpha-1}\parallel T))]^{sk}$ $(2\le\alpha\le d)$, $\sigma_i^{(\alpha)}=[\mathrm{H}_2(ID_{F_\alpha}\parallel i)\cdot\prod_{j=1}^{s}\mu^{c_{i,j}^{(\alpha)}}]^{sk}$, and $c_i^{(\alpha)}=\{c_{i,j}^{(\alpha)}\}_{j=1}^{s}$ is the i^{th} ciphertext block of F_α.

A storage node within DSN checks the keyword-associated proof (i.e., $\{ID_{F_\alpha},\psi_\alpha,\varphi_\alpha\}_{\alpha=1}^{d}$ and φ) by verifying whether Eq. (9) holds, where $\psi_\alpha=\sum_{i=1}^{n}\sum_{j=1}^{s}\pi(\theta\parallel ID_{F_\alpha},i)\cdot c_{i,j}^{(\alpha)}$, $\varphi_\alpha=\prod_{i=1}^{n}(\sigma_i^{(\alpha)})^{\pi(\theta\|ID_{F_\alpha},i)}$ and $\varphi=\prod_{\alpha=1}^{d}kt_\alpha^{(w)}$. The storage node computes the two sides of Eq. (9) as follows, where $(\zeta_1,\zeta_2,\zeta_3,\rho)$ are calculated using Eq. (8) and $Res=\{ID_{F_\alpha}\}_{\alpha=1}^{d}$.

$$\begin{aligned}&\mathrm{e}(\zeta_3, g)\\&=\mathrm{e}\left(\prod_{\alpha=1}^{d}\left(kt_\alpha^{(w)}\cdot\prod_{i=1}^{n}(\sigma_i^{(\alpha)})^{\pi(\theta\|ID_{F_\alpha},i)}\right), g\right)\\&=\mathrm{e}\left(\prod_{\alpha=1}^{d}\left(\prod_{i=1}^{n}\left(\mathrm{H}_2(ID_{F_\alpha}\parallel i)\cdot\mu^{\sum_{j=1}^{s}c_{i,j}^{(\alpha)}}\right)^{\pi(\theta\|ID_{F_\alpha},i)}\right), g^{sk}\right)\cdot\\&\quad\mathrm{e}\left(\prod_{\alpha=1}^{d}\mathrm{H}_2(ID_{F_\alpha})\cdot\mathrm{H}_2(st_d\parallel T), g^{sk}\right)\\&\mathrm{e}(\zeta_1\cdot\zeta_2\cdot\mathrm{H}_2(st_d\parallel T)\cdot\mu^{\rho}, pk)\\&=\mathrm{e}\left(\zeta_1\cdot\mu^{\sum_{\alpha=1}^{d}\psi_\alpha}, g^{sk}\right)\cdot\mathrm{e}\left(\zeta_2\cdot\mathrm{H}_2(st_d\parallel T), g^{sk}\right)\\&=\mathrm{e}\left(\prod_{\alpha=1}^{d}\left(\prod_{i=1}^{n}\left(\mathrm{H}_2(ID_{F_\alpha}\parallel i)\cdot\mu^{\sum_{j=1}^{s}c_{i,j}^{(\alpha)}}\right)^{\pi(\theta\|ID_{F_\alpha},i)}\right), g^{sk}\right)\cdot\\&\quad\mathrm{e}\left(\prod_{\alpha=1}^{d}\mathrm{H}_2(ID_{F_\alpha})\cdot\mathrm{H}_2(st_d\parallel T), g^{sk}\right)\end{aligned}$$

$\mathrm{e}(\zeta_3, g)=\mathrm{e}(\zeta_1\cdot\zeta_2\cdot\mathrm{H}_2(st_d\parallel T)\cdot\mu^{\rho}, pk)$ holds, verifying the correctness of keyword-associated proof verification.

B. Security

1) Keyword Privacy: Since the outsourced files are encrypted using a standard symmetric encryption algorithm and the authentication tags are generated from the ciphertexts, neither the authentication tags nor the ciphertexts can reveal the private contents of files. Our primary consideration is the potential leakage of keywords from the index database stored in the storage node.

We state that the storage node cannot deduce plaintext keywords from its known information.

Proof: Let $\mathcal{C}$ represent a challenger, and $\mathcal{A}$ (i.e., the storage node) be an adversary that can obtain the private keyword with an advantage of $\epsilon(\kappa)$.

- Setup: The challenger $\mathcal{C}$ selects $sk\in\mathbb{Z}_N$ and $mk, ek\in\{0,1\}^{l}$, and then computes $pk=g^{sk}$. Subsequently, the challenger $\mathcal{C}$ sends pk to the adversary $\mathcal{A}$.
- Phase 1: The adversary $\mathcal{A}$ initializes an empty index database DB, and the challenger $\mathcal{C}$ initializes an empty keyword list $\mathcal{L}$. Then $\mathcal{A}$ and $\mathcal{C}$ engage in the following oracles.
  - $\mathcal{O}_{Query}$: The adversary $\mathcal{A}$ sends a keyword w and its corresponding file F to the challenger $\mathcal{C}$.
  - $\mathcal{O}_{IndBuild}$: The challenger $\mathcal{C}$ encrypts the file as $C=\mathrm{SE.Enc}(ek,F)$ and computes $ID_F=\mathrm{H}_1(C)$ and $T=\mathrm{SE.Enc}(mk,w)$. Then, the challenger $\mathcal{C}$ randomly selects $st\in\mathbb{Z}_N$ and generates $\overline{T}=\mathrm{H}_2(T\parallel st)$, $ptr=\mathrm{SE.Enc}(\mathrm{H}_3(st),st)$, and $kt=[\mathrm{H}_2(ID_F)\cdot\mathrm{H}_2(st\parallel T)]^{sk}$. Finally, the challenger $\mathcal{C}$ sends $(C, ID_F, \overline{T}, kt, ptr)$ to the adversary $\mathcal{A}$ and adds the keyword w to $\mathcal{L}$.
- Challenge: The adversary $\mathcal{A}$ selects two keywords w_0 and w_1 that have not been queried (i.e., $w_0, w_1\notin\mathcal{L}$), and then sends w_0 and w_1, along with their associated files F_0 and F_1, to the challenger $\mathcal{C}$. The challenger $\mathcal{C}$ randomly chooses a bit $b\in\{0,1\}$. It then encrypts the file F_b as $C_b=\mathrm{SE.Enc}(ek,F_b)$ and computes $ID_{F_b}=\mathrm{H}_1(C_b)$ and $T_b=\mathrm{SE.Enc}(mk,w_b)$. Then, the challenger $\mathcal{C}$ randomly selects $st\in\mathbb{Z}_N$ and generates $\overline{T}_b=\mathrm{H}_2(T_b\parallel st)$, $ptr_b=\mathrm{SE.Enc}(\mathrm{H}_3(st),st)$, and $kt_b=[\mathrm{H}_2(ID_{F_b})\cdot\mathrm{H}_2(st\parallel T_b)]^{sk}$. Finally, the challenger $\mathcal{C}$ sends $(C_b, ID_{F_b}, \overline{T}_b, kt_b, ptr_b)$ to the adversary $\mathcal{A}$.
- Phase 2: The adversary $\mathcal{A}$ repeats Phase 1, with the limitation that w_0 and w_1 cannot be queried again in the previous two oracles.
- Guess: The adversary $\mathcal{A}$ outputs a guess $\hat{b}$ of b.

Assume that the keyword space is $\mathcal{W}$ and the adversary $\mathcal{A}$ has queried each keyword $w\in\mathcal{W}'$, where $\mathcal{W}'$ comprises all the keywords in $\mathcal{W}$ except w_0 and w_1. Thus, we can calculate that the probability of $\mathcal{A}$ winning the game is

$$\Pr[\mathcal{A}(b=\hat{b})]=\frac{1}{|\mathcal{W}|-|\mathcal{W}'|}+\epsilon(\kappa)=\frac{1}{2}+\epsilon(\kappa).$$

Since the queried files are encrypted using a standard symmetric encryption algorithm and the file identifiers are hash values of ciphertexts, both the ciphertexts and file identifiers cannot leak the keyword information. Furthermore, since ptr_b does not contain keyword information, it cannot reveal any keyword-related information. Regarding $\overline{T}_b$, since mk is not accessible to $\mathcal{A}$, the adversary cannot compute $T_{\hat{b}}=\mathrm{SE.Enc}(mk,w_{\hat{b}})$ and $\overline{T}_{\hat{b}}=\mathrm{H}_2(T_{\hat{b}}\parallel st)$ to compare $\overline{T}_{\hat{b}}$ with $\overline{T}_b$. As a result, the advantage $\epsilon(\kappa)$ of $\mathcal{A}$ can be safely ignored in polynomial time, leading to

the conclusion that

$$\Pr[\mathcal{A}(b=\hat{b})]=\frac{1}{2}.$$

Based on the analysis above, we can obtain

$$\mathrm{Adv}_{\mathcal{A}}^{\mathrm{IND-CKA}}(\kappa)=2\cdot\Pr[\mathcal{A}(\mathrm{b}=\hat{\mathrm{b}})]-1=0.$$

As a result, the proposed scheme satisfies the security Definition 3 and achieves IND-CKA security for keywords. □

2) Forward Security: During the file insertion process, the adversary $\mathcal{A}$ (i.e., the storage node) cannot discern the relationship between a newly inserted file and any past search.

Proof: Assume that the client has previously conducted searches for keywords $\{w_i\}_{i=1}^{\delta}$ using search tokens $\{T_i=\mathrm{SE.Enc}(mk,w_i)\}_{i=1}^{\delta}$ and now inserts a new file F containing the keyword w_ℓ $(1\le\ell\le\delta)$. After the file insertion, the adversary $\mathcal{A}$ can obtain the state-associated token $\overline{T}=\mathrm{H}_2(T_\ell\parallel st_d)$, which can potentially be used to exploit the association between the new file and previous searches.

To exploit the association between the new file and previous search tokens, the adversary $\mathcal{A}$ needs to compute all possible state-associated tokens $\{\overline{T}_i=\mathrm{H}_2(T_i\parallel st_d)\}_{i=1}^{\delta}$ and compares them with $\overline{T}$ to determine the previous search token that is associated with the new file. However, our scheme requires the client to select a new state st_d during each insertion process, and the adversary $\mathcal{A}$ cannot know the latest state st_d. Thus, the adversary $\mathcal{A}$ cannot exploit the association between the new file and previous search tokens.

As a result, the proposed scheme satisfies the security Definition 4 and achieves the forward security during the file insertion process. □

3) Verifiability of Data Integrity: The integrity of the files stored in a storage node can be verified by other storage nodes within the DSN. We state that if the CDH assumption holds, the storage node is unable to compute a valid proof that can successfully pass the verification in the **Verify.Audit** algorithm when a file is lost or corrupted.

Proof: Suppose that the storage node attempts to pass the verification in Eq. (10) using a forged file $C'=\{c_i'\}_{i=1}^{n}$ when the original file is lost or corrupted. Let (ψ',φ') represent the forged proof that can successfully pass the verification in Eq. (10), we have

$$\begin{aligned}\mathrm{e}(\varphi',g)&=\mathrm{e}(\zeta\cdot\mu^{\psi'},pk)\\&=\mathrm{e}((\zeta\cdot\mu^{\psi'})^{sk},g).\end{aligned}$$

Thus, to pass the verification in Eq. (10), the storage node should compute the forged proof (φ',ψ') that satisfies $\varphi'=(\zeta\cdot\mu^{\psi'})^{sk}$. The storage node can easily compute ψ' and ζ as the following equation and $\eta=\zeta\cdot\mu^{\psi'}$.

$$\begin{cases}\psi'=\sum_{i=1}^{n}\sum_{j=1}^{s}\pi(\theta\parallel ID_F,i)\cdot c_{i,j}'\\ \zeta=\prod_{i=1}^{n}\mathrm{H}_2(ID_F\parallel i)^{\pi(\theta\parallel ID_F,i)}\end{cases}$$

Since the storage node is unable to obtain the secret key sk, it cannot directly compute $\varphi'=\eta^{sk}$. Then, without knowing sk, the complexity of computing φ' from η and $pk=g^{sk}$ is equivalent to the complexity of solving the CDH problem. According to Definition 1, there is no probabilistic polynomial-time algorithm to solve the CDH problem. Thus, the storage node is unable to compute (φ',ψ') and cannot successfully pass the verification in the **Verify.Audit** algorithm.

Based on the analysis above, the proposed scheme satisfies the security Definition 5 and guarantees the verifiability of data integrity. □

4) Verifiability of Search Result Correctness: In our scheme, the correctness of search results delivered by a storage node can be verified by other storage nodes within the DSN.

Proof: Assume that the client inserts the indices of files $\{F_\alpha\}_{\alpha=1}^{d}$, all of which include the same keyword w, into the index database DB stored in the storage node. During the search process, the client sends the search token T and latest state st_d of keyword w to the storage node, and there exist three types of incorrect search results: fewer files, more files, and replaced files.

Case 1: Suppose that the search result is fewer files, specifically $Res=\{ID_{F_\alpha}\}_{\alpha=1}^{\delta-1}\cup\{ID_{F_\alpha}\}_{i=\delta+1}^{d}$. Let $\{ID_{F_\alpha},\psi_\alpha,\varphi_\alpha\}_{\alpha=1}^{\delta-1}$, $\{ID_{F_\alpha},\psi_\alpha,\varphi_\alpha\}_{\alpha=\delta+1}^{d}$, and φ represent the forged proof that can successfully pass the verification in Eq. (9), we can obtain

$$\begin{aligned}\mathrm{e}(\zeta_3,g)&=\mathrm{e}(\zeta_1\cdot\zeta_2\cdot\mathrm{H}_2(st_d\parallel T)\cdot\mu^{\rho},pk)\\&=\mathrm{e}((\zeta_1\cdot\zeta_2\cdot\mathrm{H}_2(st_d\parallel T)\cdot\mu^{\rho})^{sk},g),\end{aligned}\tag{11}$$

where $(\zeta_1,\zeta_2,\zeta_3,\rho)$ are calculated using the Eq. (8). Thus, to pass the verification in Eq. (9), the storage node should compute the forged proof satisfying the following equation.

$$\zeta_3=\left(\zeta_1\cdot\zeta_2\cdot\mathrm{H}_2(st_d\parallel T)\cdot\mu^{\sum_{1\le\alpha\le d,\alpha\ne\delta}\psi_\alpha}\right)^{sk}$$

The storage node can easily compute ζ_1, ζ_2, $\mathrm{H}_2(st_d\parallel T)$, and $\mu^{\sum_{1\le\alpha\le d,\alpha\ne\delta}\psi_\alpha}$. For simplicity, we use η to replace $(\zeta_1\cdot\zeta_2\cdot\mathrm{H}_2(st_d\parallel T)\cdot\mu^{\sum_{1\le\alpha\le d,\alpha\ne\delta}\psi_\alpha})$. However, since the storage node is unable to obtain the secret key sk, the complexity of computing $\zeta_3=\eta^{sk}$ from η and $pk=g^{sk}$ is equivalent to the complexity of solving the CDH problem.

Case 2: Assume that the search result is more files, specifically $Res=\{ID_{F_\alpha}\}_{\alpha=1}^{d+1}$. Let $\{ID_{F_\alpha},\psi_\alpha,\varphi_\alpha\}_{\alpha=1}^{d+1}$ and φ represent the forged proof that can successfully pass the verification in Eq. (9), we can obtain Eq. (11), where $(\zeta_1,\zeta_2,\zeta_3,\rho)$ are calculated using the Eq. (8). Thus, to pass the verification in Eq. (9), the storage node should compute the forged proof satisfying the equation as follows.

$$\zeta_3=\left(\zeta_1\cdot\zeta_2\cdot\mathrm{H}_2(st_d\parallel T)\cdot\mu^{\sum_{1\le\alpha\le d+1}\psi_\alpha}\right)^{sk}$$

The storage node can easily compute ζ_1, ζ_2, $\mathrm{H}_2(st_d\parallel T)$ and $\mu^{\sum_{1\le\alpha\le d+1}\psi_\alpha}$. For simplicity, we use η to replace $(\zeta_1\cdot\zeta_2\cdot\mathrm{H}_2(st_d\parallel T)\cdot\mu^{\sum_{1\le\alpha\le d+1}\psi_\alpha})$. However, since the storage node is unable to obtain the secret key sk, the complexity of computing $\zeta_3=\eta^{sk}$ from η and $pk=g^{sk}$ is equivalent to the complexity of solving the CDH problem.

Case 3: Assume that the search result is replaced files, specifically $Res=\{ID_{F_\alpha}\}_{\alpha=1}^{\delta-1}\cup\{ID_{F'_\delta}\}\cup\{ID_{F_\alpha}\}_{\alpha=\delta+1}^{d}$. Let $\{ID_{F_\alpha},\psi_\alpha,\varphi_\alpha\}_{\alpha=1}^{\delta-1}\cup\{ID_{F'_\delta},\psi'_\delta,\varphi'_\delta\}\cup\{ID_{F_\alpha},\psi_\alpha,\varphi_\alpha\}_{\alpha=\delta+1}^{d}$ and φ represent the forged proof that can successfully pass the verification in

Eq. (9), we can obtain Eq. (11), where $(\zeta_1, \zeta_2, \zeta_3, \rho)$ are calculated using the Eq. (8). Thus, to pass the verification in Eq. (9), the storage node should compute the forged proof satisfying the equation as follows.

$$\zeta_3 = \left(\zeta_1 \cdot \zeta_2 \cdot \mathrm{H}_2(st_d \parallel T) \cdot \mu^{\sum_{1\le\alpha\le d,\alpha\neq\delta} \psi_\alpha + \psi'_\delta}\right)^{sk}$$

The storage node can easily compute ζ_1, ζ_2, $\mathrm{H}_2(st_d \parallel T)$ and $\mu^{\sum_{1\le\alpha\le d,\alpha\neq\delta} \psi_\alpha + \psi'_\delta}$. For simplicity, we use η to replace $(\zeta_1 \cdot \zeta_2 \cdot \mathrm{H}_2(st_d \parallel T) \cdot \mu^{\sum_{1\le\alpha\le d,\alpha\neq\delta} \psi_\alpha + \psi'_\delta})$. However, since the storage node is unable to obtain the secret key sk, the complexity of computing $\zeta_3 = \eta^{sk}$ from η and $pk = g^{sk}$ is equivalent to the complexity of solving the CDH problem.

According to Definition 1, there is no probabilistic polynomial-time algorithm to solve the CDH problem. Thus, the storage node is unable to pass the verification in the **Verify.Search** algorithm using incorrect results and forged proofs.

Based on the analysis above, the proposed scheme satisfies the security Definition 6 and achieves the verifiability of search result correctness. □

5) Mitigation of Side-Channel Information Leakage: In the context of searchable encryption, if an attacker possesses background knowledge of the outsourced files, such as partial file contents or the distribution of keyword-file identifier pairs, it may leverage side-channel information, including access pattern, to deduce sensitive information about the outsourced files or keywords [26]. The core threat lies in the attacker's ability to collect auxiliary information (e.g., file sizes or the number of files matching a query) and correlate it with background knowledge to conduct frequency analysis attacks, thereby inferring keywords and file contents.

A common mitigation strategy is to obfuscate the original distribution of keyword-file identifier pairs in the outsourced files. Several existing solutions [27], [28] address the threat by inserting dummy keyword-file identifier pairs to mask the true distribution, thereby disrupting adversarial inference. Our scheme can incorporate these existing techniques as a preprocessing step before data outsourcing to mitigate the threat. Specifically, the client employs an existing method to obfuscate the original distribution of keyword-file identifier pairs prior to outsourcing. Then the client outsources the obfuscated index database to the storage node. However, due to the introduction of dummy pairs, the search results may include false positives. To ensure accurate search results, the client must locally store the dummy pairs and filter out the false positives.

V. Performance Evaluation

A. Experiment Settings

We implement the proposed scheme in C++. The implementation relies on the OpenSSL Library [29], Pairing-Based Cryptography (PBC) Library [30], and GNU Multiple Precision Arithmetic (GMP) Library [31]. To evaluate the efficiency of our scheme, we conduct experiments using three datasets (i.e., DB1-DB3) derived from the Enron emails dataset [32]. We extract three subsets with different number of emails and keyword-file identifier pairs from the original Enron emails dataset. The key attributes of these datasets are summarized in Table I.

TABLE I
Evaluation Datasets

Dataset	Number of Files	Keyword-File Identifier Pairs	Size (MB)
DB1	4,108	4,108	8.24
DB2	12,035	24,070	25.83
DB3	55,064	165,192	127.62

We utilize a Windows laptop with a 4-core Intel Core i7-7560U processor and 8GB of memory to run the storage node programs, while we employ a Windows laptop with a 4-core Intel Core i5-6200 processor and 8GB of memory to run the client programs. The programs of other nodes are implemented on a MacOS laptop with a 4-core Intel Core i5-1038NG7 processor and 16GB of memory. All the laptops are deployed within the same wireless LAN with a 20Mbps bandwidth. The on-chain data are stored on the Ethereum blockchain [21], and we use gas cost to estimate expenses associated with on-chain operations. The smart contract for on-chain data storage is implemented in Solidity, using a mapping data structure. It is deployed and tested in Remix Virtual Machine with a gas limit of 3000000. Besides, we set the bit length κ of security parameter to 512 and the bit length ι of encryption key to 256.

Existing schemes can only independently support either verifiable search or integrity auditing in encrypted DSN, whereas our scheme is the first to simultaneously achieve them through a unified proof verification. To comparatively evaluate the efficiency of our scheme, we design two baselines to match the functionalities of our scheme. In the DSN scenario, the scheme in [12] is the only prior work to achieve verifiable searchable encryption. For the integrity auditing, most practical DSNs employ the classical Merkle tree-based scheme [33] as their fundamental PoS modules due to its computational efficiency. Besides, the auditing scheme in [13] achieves the optimal on-chain gas efficiency in the DSN scenario. Therefore, we combine the verifiable searchable encryption in [12] with the Merkle tree-based auditing [33] as one baseline (i.e., baseline1). Meanwhile, we integrate the verifiable searchable encryption scheme in [12] with the auditing scheme in [13] as an additional baseline (i.e., baseline2) for comparison.

B. Experiment Results

1) Setup: We evaluate the time and gas cost in the setup process, and present the results in Fig. 2. In the setup process, each storage node needs to generate its public parameters and call a smart contract to store these parameters on the blockchain. The cost of our scheme falls between those of the two baseline schemes. A storage node publishes an integer and two elements from a multiplicative cyclic group on-chain in our scheme. In contrast, the Merkle tree-based auditing scheme [33] has less public parameters, while the scheme in [13] needs to publish more parameters on the blockchain. This process has a negligible impact on the overall efficiency of our scheme, as it is performed only once during the system's lifetime.

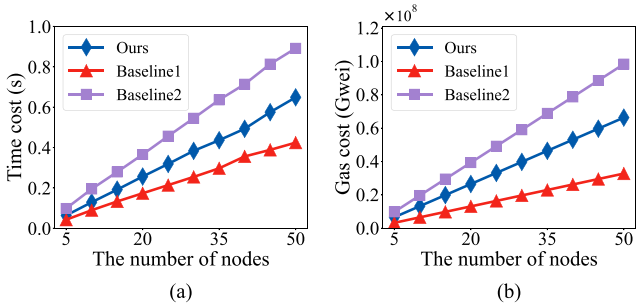


Fig. 2. Evaluation for the setup process. (a) The overall time cost. (b) The gas cost of on-chain data.

2) Data Outsourcing: We measure the computation and communication time costs, and present the results in Fig. 3(a) and 3(b). The client's computation time mainly comprises file reading and writing, file content encryption, and authentication tags generation. The client's communication time mainly comprises the transmission of the ciphertext, data indices, and authentication tags. As shown in Fig. 3(a), the client's communication cost grows with the number of uploaded files, while the client's computation cost increases with the number of keyword-file identifier pairs. The storage node's computation time mainly comprises building the index database. Its communication time mainly includes receiving the ciphertext, data indices and authentication tags. As shown in Fig. 3(b), the communication cost grows with the number of uploaded files and the computation cost grows with the number of keyword-file identifier pairs. Since DB3 contains more files and keyword-file identifier pairs compared to the other two datasets, both the client and storage node have higher communication and computation time costs when processing DB3. As can be seen in Fig. 3(a) and 3(b), our scheme incurs lower time overhead for both the client and the storage node compared to the two baseline schemes. This is because, in the two baseline schemes, the hash value of each item in index database needs to be published on the blockchain, resulting in numerous on-chain storage operations.

For the gas cost in the data outsourcing process, only the data storage events, including the information about the client, the storage node, and outsourced files, are stored on the blockchain. As shown in Fig. 3(c), our scheme achieves significantly lower gas costs than the two baseline schemes, as they require publishing the hash value of each item in the index database on the blockchain.

3) Verifiable Search and Integrity Auditing: For the verifiable search and integrity auditing process, we measure the efficiency of our scheme from the perspectives of client, storage node, verification node and on-chain gas cost, and present the results in Fig. 4. During the keyword search process, the client's time cost mainly comprises the encryption of keyword and the retrieval of keyword state. As shown in Fig. 4(a), the client's time cost grows with the number of searches. Our scheme incurs a slightly higher client time costs compared to the two baseline schemes, as the client generates the search token at query time to enhance keyword security. In contrast, in the two baseline schemes, the client stores all search tokens locally, allowing direct retrieval without additional computation.

The storage node's time cost mainly involves retrieving identifiers of files associated with the searched keyword from the index database and proving search result correctness and data integrity. The storage node has a higher time cost in DB3 compared to the other two datasets. This is due to the fact that keywords in DB3 are associated with more files, and the storage node needs to process more files during a single search process. Fig. 4(b) shows the comparison of storage node's time costs. Our scheme incurs significantly lower time costs than the two baseline schemes, as it generates a single proof for both the search result and data integrity. In contrast, the baseline schemes require the storage node to execute two separate proof generation processes. For the verification efficiency of other nodes depicted in Fig. 4(c), our scheme has significantly lower time costs than the two baseline schemes. In the two baseline schemes, separate verifications for search result correctness and data integrity are performed by other storage nodes, whereas our scheme simultaneously validates them through one-time proof verification, thereby resulting in lower time costs.

Fig. 4(d) illustrates the gas cost of on-chain data. The on-chain data mainly include the search request, file identifiers and proofs during a single search process. DB3 has a higher on-chain gas cost than the other two datasets. This is because keywords in DB3 are associated with more files, resulting in the storage of more file identifiers and proofs on the blockchain. In addition, our scheme incurs significantly lower gas costs than the two baseline schemes, demonstrating its suitability for DSN ecosystems with on-chain resource constraints.

VI. Related Work

A. Integrity Auditing

Data integrity auditing techniques have been widely developed in cloud storage services, and existing schemes can be roughly divided into two categories: Provable Data Possession (PDP) and Proofs of Retrievability (PoR). PDP paradigm [33] was first proposed by Ateniese et al. using homomorphic linear authentication tags, while Juels et al. [34] proposed the PoR paradigm by considering data recoverability. Inspired by these paradigms, numerous integrity auditing schemes [35], [36] have been proposed for traditional cloud storage scenarios. Yan et al. [35] proposed an efficient integrity auditing scheme based on homomorphic hash function. The scheme achieves data dynamic updates by tracking the operations on blocks in a record table. In the context of group data sharing, Li et al. [36] proposed an integrity auditing scheme utilizing certificateless signature, thereby avoiding the necessities of certificate verification and key escrow.

To achieve reliable and fair storage services, all the existing DSNs equip fundamental storage auditing functions [37]. By periodic integrity auditing, clients can know the status of their outsourced files and the honest storage nodes are rewarded with storage fees. Sia [1] is an early example of DSN project that first adopts Merkle tree-based auditing scheme to achieve reliable storage. In contrast, a concurrent DSN project called Storj [22] initially employed a similar auditing scheme in its early stage. However, it later shifted its approach by using third-party auditor to perform integrity checking. This deviates from the original

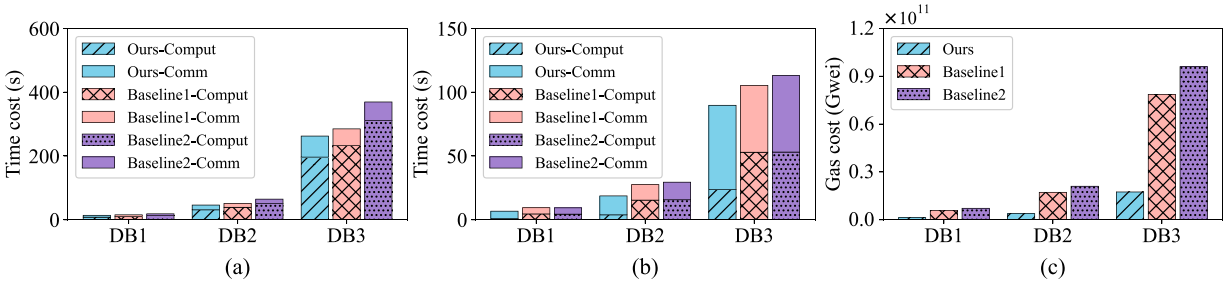


Fig. 3. Evaluation for the data outsourcing process. (a) The client's time cost. (b) The storage node's time cost. (c) The gas cost of on-chain data.

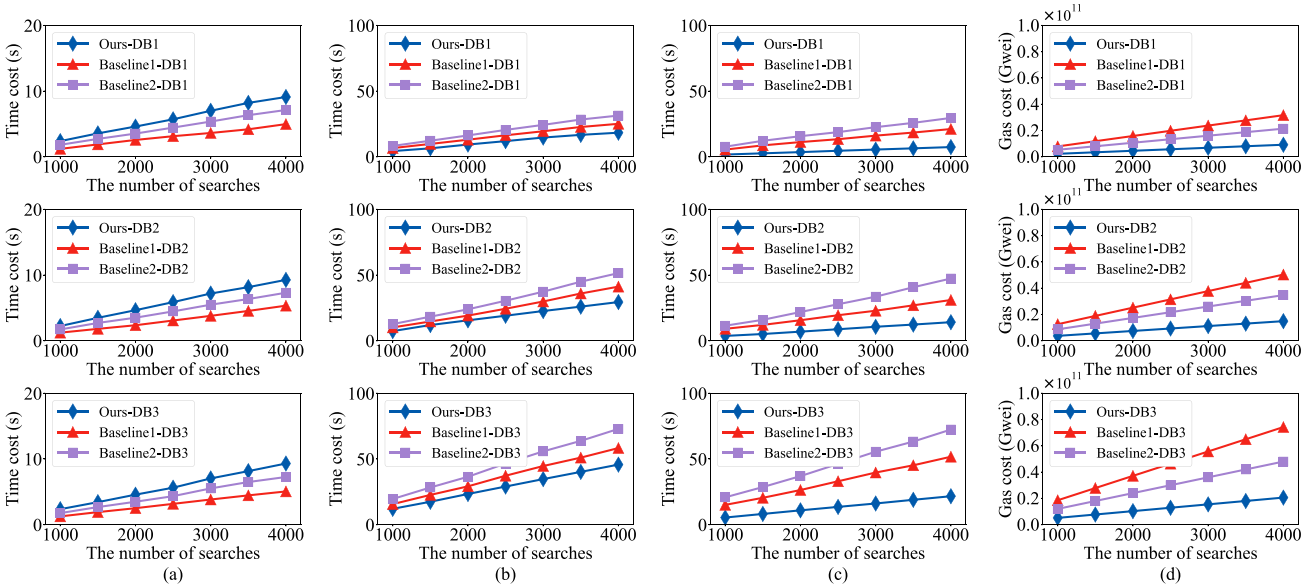


Fig. 4. Evaluation for the verifiable search and integrity auditing process. (a) The client's time cost. (b) The storage node's time cost. (c) The verification time cost of other nodes. (d) The gas cost of on-chain data.

principle of decentralization. Since Merkle tree-based auditing schemes inherently have limitations on the number of auditing rounds [13], Filecoin [2] uses SNARK auditing framework, which allows for the generation of unlimited auditing rounds. Additionally, Du et al. [13] proposed a storage auditing scheme for the DSN using the polynomial commitment technology, making the auditing process more efficient for nodes within the DSN.

All the aforementioned auditing schemes used in the DSN achieve static PoS, indicating that users cannot perform dynamic updates after the initial setup. Recently, Campanelli et al. [14] proposed an auditing scheme supporting dynamic data updates in the DSN using the subvector commitment technology [38]. Duan et al. [15] used succinct data structures and optimization techniques to manage index information and designed a dynamic auditing scheme for the DSN. This scheme can produce compact auditing logs similar to static PoS schemes.

B. Searchable Encryption

Searchable encryption schemes have been extensively developed in cloud storage services, with focuses on efficiency [39], privacy [40], [41], [42], and reliability [43], [44], [45], [46]. To improve reliability, numerous verifiable searchable encryption schemes [43], [44], [45], [46] have been developed over static data in traditional centralized cloud storage. The scheme in [43] studied verifiable searchable encryption, utilizing message authenticated code to achieve the verifiability of search results. Then several studies aimed to enhance the functionality of verifiable searchable encryption. The scheme in [44] focused on enabling verifiability for conjunctive keyword searches, while Wang et al. [45] proposed a verifiable fuzzy keyword search scheme using the symbol tree. Wu et al. [46] designed a verifiable searchable encryption scheme for the multi-user scenarios, where data owners can share data with multiple users. However, it is important to note that all the aforementioned schemes require the delegation of a trusted third-party for public verification.

To mitigate the reliance on the trusted third-party, some blockchain-based verifiable searchable encryption schemes [47], [48], [49] have been developed. The scheme in [47] stores index databases on the blockchain and users can search the index databases themselves, ensuring the correctness of search results. The scheme in [48] also publishes index databases on the blockchain and uses the smart contract to enforce the reliability of search process. However, directly publishing a common index database structure (i.e., keyword-file identifier pairs)

on the blockchain cannot ensure forward security during the file insertion process. The scheme in [49] addressed this issue through a delicate hybrid index design.

All the above schemes are not applicable to emerging DSN scenarios due to the limited on-chain storage. Recently, Cai et al. [12] proposed the first verifiable searchable encryption scheme tailored for the DSN. To reduce the on-chain storage overhead, the scheme involves a group of non-colluding servers. These servers act as the arbiter to store clients' index databases and fairly judge the correctness of search results by re-executing search operations. However, the introduction of an arbiter into the DSN violates the decentralized principle.

VII. Conclusion

To motivate more individual devices and users to join in and use the emerging DSN, the DSN should support ubiquitous functions like verifiable searchable encryption and integrity auditing. This paper proposed a novel verifiable search and integrity auditing scheme tailored for encrypted decentralized storage. By integrating the verification of search result correctness into the fundamental auditing process of the DSN, the proposed scheme maintains a similar proof size to general auditing and does not significantly increase the on-chain storage overhead, making it affordable for the DSN ecosystem. Besides, our scheme supports file dynamic updates and ensures the forward security during the file insertion process. We theoretically prove the correctness and security of our scheme. The evaluation results demonstrate its efficiency and practicability.

Limitations and future work. Our scheme currently focuses on the commonly used single-keyword search. However, it can be easily extended to support multi-keyword search. In this case, a user can submit multiple single-keyword search requests simultaneously, and the storage node computes the intersection or union of the individual search results based on the logical relationship specified by the user. The aggregated result is then returned to the user. To ensure verifiability, the storage node needs to publish a keyword-associated proof for each individual keyword, thereby proving the correctness of the aggregated search result. However, for more complex queries, such as similarity search, our scheme cannot be directly extended to support them. Our future work will explore enabling verifiability for complex queries in the DSN scenario.

References

[1] D. Vorick and L. Champine, "Sia: Simple decentralized storage," 2018. [Online]. Available: https://blockchainlab.com/pdf/whitepaper3.pdf

[2] J. Benet and N. Greco, "Filecoin: A decentralized storage network," 2017. [Online]. Available: https://filecoin.io/filecoin.pdf

[3] H. Guo et al., "FileDAG: A multi-version decentralized storage network built on dag-based blockchain," *IEEE Trans. Comput.*, vol. 72, no. 11, pp. 3191–3202, Nov. 2023.

[4] C. Gray, "Storj vs. Dropbox: Why decentralized storage is the future," 2014. [Online]. Available: https://bitcoinmagazine.com/articles/storjvs-dropboxdecentralized-storage-future

[5] N. Z. Benisi, M. Aminian, and B. Javadi, "Blockchain-based decentralized storage networks: A survey," *J. Netw. Comput. Appl.*, vol. 162, Jul. 2020, Art. no. 102656.

[6] H. Guo et al., "BFT-DSN: A Byzantine fault tolerant decentralized storage network," *IEEE Trans. Comput.*, vol. 73, no. 5, pp. 1300–1312, May 2024.

[7] H. Chen, Y. Lu, and Y. Cheng, "Fileinsurer: A scalable and reliable protocol for decentralized file storage in blockchain," in *Proc. 42nd IEEE Int. Conf. Distrib. Comput. Syst.*, Bologna, Italy, 2022, pp. 168–179.

[8] Swarm Fund. "Swarm Fund: The blockchain for private equity," 2019. [Online]. Available: https://swarm.fund/whitepaper

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

[10] Q. Song, Z. Liu, J. Cao, K. Sun, Q. Li, and C. Wang, "SAP-SSE: Protecting search patterns and access patterns in searchable symmetric encryption," *IEEE Trans. Inf. Forensics Secur.*, vol. 16, pp. 1795–1809, 2020.

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

[12] C. Cai, J. Weng, X. Yuan, and C. Wang, "Enabling reliable keyword search in encrypted decentralized storage with fairness," *IEEE Trans. Dependable Secur. Comput.*, vol. 18, no. 1, pp. 131–144, Jan./Feb. 2021.

[13] Y. Du, H. Duan, A. Zhou, C. Wang, M. H. Au, and Q. Wang, "Enabling secure and efficient decentralized storage auditing with blockchain," *IEEE Trans. Dependable Secur. Comput.*, vol. 19, no. 5, pp. 3038–3054, May 2022.

[14] M. Campanelli, D. Fiore, N. Greco, D. Kolonelos, and L. Nizzardo, "Incrementally aggregatable vector commitments and applications to verifiable decentralized storage," in *Proc. 26th Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, Daejeon, South Korea, 2020, pp. 3–35.

[15] H. Duan, Y. Du, L. Zheng, C. Wang, M. H. Au, and Q. Wang, "Towards practical auditing of dynamic data in decentralized storage," *IEEE Trans. Dependable Secur. Comput.*, vol. 20, no. 1, pp. 708–723, Jan. 2022.

[16] H. Yu, Y. Chen, Z. Yang, Y. Chen, and S. Yu, "EDCOMA: Enabling efficient double compressed auditing for blockchain-based decentralized storage," *IEEE Trans. Serv. Comput.*, vol. 17, no. 5, pp. 2273–2286, Jun. 2024.

[17] M. I. Khalid et al., "A comprehensive survey on blockchain-based decentralized storage networks," *IEEE Access*, vol. 11, pp. 10995–11015, 2023.

[18] K. He, J. Chen, Q. Zhou, R. Du, and Y. Xiang, "Secure dynamic searchable symmetric encryption with constant client storage cost," *IEEE Trans. Inf. Forensics Secur.*, vol. 16, pp. 1538–1549, 2020.

[19] Y. Zhang, J. Katz, and C. Papamanthou, "All your queries are belong to us: The power of file-injection attacks on searchable encryption," in *Proc. 25th USENIX Secur. Symp.*, Austin, USA, 2016, pp. 707–720.

[20] Z. Zhang, J. Wang, Y. Wang, Y. Su, and X. Chen, "Towards efficient verifiable forward secure searchable symmetric encryption," in *Proc. 24th Eur. Symp. Res. Comput. Secur.*, 2019, pp. 304–321.

[21] G. Wood et al., "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, no. 2014, pp. 1–32, Oct. 2014.

[22] S. Wilkinson, T. Boshevski, J. Brandoff, and V. Buterin, "Storj a peer-to-peer cloud storage network," 2014. [Online]. Available: https://storj.io/storj.pdf

[23] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proc. 4th Theory Cryptogr. Conf.*, Amsterdam, The Netherlands, 2007, pp. 535–554.

[24] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," *J. Cryptol.*, vol. 17, no. 4, pp. 297–319, 2004.

[25] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. 13th ACM SIGSAC Conf. Comput. Commun. Secur.*, Alexandria, VA, USA, 2006, pp. 79–88.

[26] L. Blackstone, S. Kamara, and T. Moataz, "Revisiting leakage abuse attacks," in *Proc. Annu. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 1–18.

[27] I. Demertzis, D. Papadopoulos, C. Papamanthou, and S. Shintre, "SEAL: Attack mitigation for encrypted databases via adjustable leakage," in *Proc. USENIX Secur. Symp.*, 2020, pp. 2433–2450.

[28] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2015, pp. 668–679.

[29] OpenSSL. "Open source socket layer," 2015. [Online]. Available: https://www.openssl.org/

[30] B. Lynn, "The pairing-based cryptography (PBC) library," 2010. [Online]. Available: https://crypto.stanford.edu/pbc/

[31] T. Granlund, "GNU multiple precision arithmetic library," 2010. [Online]. Available: http://gmplib.org/

[32] Enron emails dataset. 2015. [Online]. Available: https://www.cs.cmu.edu/~./enron

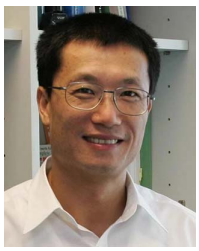
[33] G. Ateniese et al., "Provable data possession at untrusted stores," in *Proc. 14th ACM Conf. Comput. Commun. Secur.*, Alexandria, VA, USA, 2007, pp. 598–609.
[34] A. Juels and B. S. Kaliski, Jr., "PORs: Proofs of retrievability for large files," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2007, pp. 584–597.
[35] H. Yan, J. Li, J. Han, and Y. Zhang, "A novel efficient remote data possession checking protocol in cloud storage," *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 1, pp. 78–88, Jan. 2017.
[36] J. Li, H. Yan, and Y. Zhang, "Certificateless public integrity checking of group shared data on cloud storage," *IEEE Trans. Serv. Comput.*, vol. 14, no. 1, pp. 71–81, Jan./Feb. 2021.
[37] Y. Du, H. Duan, A. Zhou, C. Wang, M. H. Au, and Q. Wang, "Towards privacy-assured and lightweight on-chain auditing of decentralized storage," in *Proc. 40th Int. Conf. Distrib. Comput. Syst.*, Singapore, 2020, pp. 201–211.
[38] R. W. Lai and G. Malavolta, "Subvector commitments with application to succinct arguments," in *Proc. 39th Annu. Int. Cryptol. Conf.*, Santa Barbara, USA, 2019, pp. 530–560.
[39] Y. Lu and J. Li, "Lightweight public key authenticated encryption with keyword search against adaptively-chosen-targets adversaries for mobile devices," *IEEE Trans. Mob. Comput.*, vol. 21, no. 12, pp. 4397–4409, Dec. 2022.
[40] L. Chen, J. Li, and J. Li, "Towards forward and backward private dynamic searchable symmetric encryption supporting data deduplication and conjunctive queries," *IEEE Internet Things J.*, vol. 10, no. 19, pp. 17408–17423, Oct. 2023.
[41] S. Patranabis and D. Mukhopadhyay, "Forward and backward private conjunctive searchable symmetric encryption," in *Proc. 28th Annu. Netw. Distrib. Syst. Secur. Symp.*, 2021, pp. 1–16.
[42] H. Xie, Y. Guo, Y. Miao, and X. Jia, "Access-pattern hiding search over encrypted databases by using distributed point functions," *IEEE Trans. Comput.*, 2024. [Online]. Available: http://dx.doi.org/10.1109/TC.2024.3504288.
[43] K. Kurosawa and Y. Ohtaki, "UC-secure searchable symmetric encryption," in *Proc. 16th Int. Conf. Financial Cryptography Data Secur.*, Kralendijk, Netherlands, 2012, pp. 285–298.
[44] Z. Fu, X. Wu, C. Guan, X. Sun, and K. Ren, "Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement," *IEEE Trans. Inf. Forensics Secur.*, vol. 11, no. 12, pp. 2706–2716, Jul. 2016.
[45] J. Wang et al., "Efficient verifiable fuzzy keyword search over encrypted data in cloud computing," *Comput. Sci. Inf. Syst.*, vol. 10, no. 2, pp. 667–684, 2013.
[46] A. Wu, A. Yang, W. Luo, and J. Wen, "Enabling traceable and verifiable multi-user forward secure searchable encryption in hybrid cloud," *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 1886–1898, Apr. 2022.
[47] M. Poongodi, M. Hamdi, V. Varadarajan, B. S. Rawal, and M. Maode, "Building an authentic and ethical keyword search by applying decentralised (blockchain) verification," in *Proc. IEEE INFOCOM—IEEE Conf. Comput. Commun. Workshops*, Toronto, ON, Canada, 2020, pp. 746–753.
[48] S. Hu, C. Cai, Q. Wang, C. Wang, Z. Wang, and D. Ye, "Augmenting encrypted search: A decentralized service realization with enforced execution," *IEEE Trans. Dependable Secur. Comput.*, vol. 18, no. 6, pp. 2569–2581, Dec. 2019.
[49] Y. Guo, C. Zhang, C. Wang, and X. Jia, "Towards public verifiable and forward-privacy encrypted search by using blockchain," *IEEE Trans. Dependable Secur. Comput.*, vol. 20, no. 3, pp. 2111–2126, May 2022.

Mingyang Song received the B.E. and M.E. degrees in software engineering from Sun Yat-sen University, Guangzhou, China, in 2019 and 2021, respectively. He is currently working toward the Ph.D. degree with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. His works have appeared in prestigious venues such as USENIX Security, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, IEEE TRANSACTIONS ON COMPUTERS, and IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS. His research interests include security and privacy related to cloud computing, applied cryptography, and blockchain.

Zhongyun Hua (Senior Member, IEEE) received the B.S. degree in software engineering from Chongqing University, Chongqing, China, in 2011, and the M.S. and Ph.D. degrees in software engineering from the University of Macau, Macau, China, in 2013 and 2016, respectively. He is currently a Professor with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. His works have appeared in prestigious venues such as USENIX Security, ACM CCS, ICML, CVPR, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, and IEEE TRANSACTIONS ON SIGNAL PROCESSING. He has been recognized as a "Highly Cited Researcher 2024." He has published more than 120 papers on the subject, receiving more than 8500 citations. His research interests include applied cryptography, multimedia security, trustworthy artificial intelligence, and nonlinear systems and applications. He is an Associate Editor of IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING and IEEE SIGNAL PROCESSING LETTERS.

Yifeng Zheng (Member, IEEE) is an Assistant Professor with the Department of Electrical and Electronic Engineering, The Hong Kong Polytechnic University. His work has appeared in prestigious conferences (such as ESORICS, DSN, ACM AsiaCCS, and IEEE PERCOM), as well as journals (such as IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, and IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING). His research interests include the intersection of security, privacy, data mining, cloud computing, and machine learning.

Qing Liao received the Ph.D. degree in computer science and engineering supervised by Prof. Qian Zhang from the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, in 2016. Currently, she is a Professor with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. Her works have appeared in prestigious venues such as IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, IEEE TRANSACTIONS ON IMAGE PROCESSING, and IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS. She has published more than 100 papers on the subject. Her research interests include artificial intelligence, information security, and data mining.



Xiaohua Jia (Fellow, IEEE) received the B.Sc. and M.Eng. degrees from the University of Science and Technology of China, in 1984 and 1987, respectively, and the D.Sc. degree in information science from the University of Tokyo, in 1991. Currently, he is the Chair Professor with the Department of Computer Science, City University of Hong Kong. His research interests include cloud computing and distributed systems, computer networks, and mobile wireless networks. He is the General Chair of ACM MobiHoc 2008, the Area-Chair of IEEE INFOCOM 2010, a TPC Co-Chair of IEEE GlobeCom 2010-Ad Hoc and Sensor Networking Symposium, and a Panel Co-Chair of IEEE INFOCOM 2011. He is an Editor of *Journal of World Wide Web,* IEEE TRANSACTIONS ON COMPUTERS, etc.