AF-Dedup: Secure Data Deduplication Based on Adaptive Dynamic Merkle Hash Forest PoW for Cloud Storage

***Abstract*— For encrypted data deduplication, Proof of ownership (PoW) verifies a client's ownership of an entire file, preventing malicious users from exploiting a single segment of information to gain access to the file. By establishing the identity of two users who possess the same file, CSP can maintain a single copy for the file, enabling deduplication. However, existing PoW schemes based on Merkle hash tree (MHT) cannot guarantee the security of small files. Therefore, we propose a novel data structure called adaptive dynamic Merkle hash forest (ADMHF) for PoW, and present an encrypted data deduplication scheme called AF-Dedup. It reduces the risks of data content exposure resulting from multiple ownership verification attempts in traditional schemes. Specifically, we first construct the file tag as a unique identifier of the file. Second, different encryption schemes are employed depending on the popularity of the data. Then, the corresponding ADMHF is generated for subsequent ownership verifications.**

After security analysis and simulation experiments, our scheme is proven to significantly enhance the security of small files. In a given situation for files with only 2 blocks, our scheme achieves the same level of security as the existing scheme for a file with 91 blocks.

***Index Terms*— ADMHF, bilinear mapping, encrypted data deduplication, proof of ownership**

I. Introduction

CURRENTLY, an increasing number of users prefer to upload their local files to cloud storage platforms in order to free up storage space and access their data anytime and anywhere. However, this trend results in a significant amount of duplicated data stored in the cloud, leading to waste of the storage space. To address this issue, the mainstream methods include data compression and data deduplication, with this paper focused on the latter. Encrypted data deduplication is a technique where the cloud server stores a single copy of a file and generates an access link for other authorized users. Studies indicate that encrypted data deduplication ratios can range from 1:10 to 1:500, allowing for over 90% savings in storage space for backup file systems [1].

The practice of outsourcing private data to the cloud service providers (CSPs) necessitates that users place unconditional trust in the security of their data. In reality, users are often concerned about data confidentiality. One viable solution involves encrypting the data before uploading it to the CSP. However, traditional encryption schemes use different keys for different users, resulting in the same plaintext being encrypted into different ciphertexts. This makes deduplication a complex and challenging problem [2].

In order to address the aforementioned challenge, Convergent Encryption (CE) [3] is proposed. However, CE cannot resist dictionary attacks. There are also variants [4] [5] of CE, but they still cannot address the core issue. To enhance the security of CE, several schemes introduce the use of a Trusted Third Party (TTP) [6]. However, TTPs are considered to be fully trusted in these schemes, posing implementation challenges in real-world applications.

Proof of ownership (PoW) plays a vital role in encrypted data deduplication, as it can efficiently prove that a user indeed possesses a certain file [7]. The current method for PoW involves using file hash to differentiate between identical files, commonly referred to as hash-as-proof. However, if ans attacker can enumerate the hash value offline, he may gain the ownership of the file even he does not own the file. In response to this, Halevi *et al.* suggest the Merkle hash tree (MHT) [8] as a means of validating data [9]. Compared to the hash-as-proof scheme, this scheme enables users to efficiently prove ownership of an entire file to the CSP, rather than just a fragment of the file. For prover that passes MHT-PoW, it can be inferred with high probability that the prover possesses most of the leaves associated with the file.

However, the security of MHT-PoW scheme is compromised, especially for small files. As demonstrated by our analysis in Section V-C, considering side-channel attacks, each verification attempt poses a potential risk of exposing relevant nodes. With an increasing number of verifications N_{cr}, the percentage of exposed nodes γ also increases. Once N_{cr} exceeds a certain value, the MHT becomes fully exposed, rendering the PoW mechanism ineffective. We then perform experiments in Section VI-A to assess the correlation between N_{cr} and γ.

Building on the aforementioned concerns, we propose a secure encrypted data deduplication scheme, namely AF-Dedup. Our contributions are as follows:

- We propose an encrypted data deduplication scheme that does not rely on any TTPs and guarantees semantic security of ciphertexts.
- We propose a PoW method based on ADMHF, thereby preventing the exposure of MHT node information via multiple verifications. For example, when the file is divided into 32 blocks, the percentage of exposed nodes in a single PoW round decreased by a factor of 7.199 compared to the traditional MHT-PoW scheme.
- We utilize blind signature to generate file tag and enhance system security by employing bilinear mapping for sig-

nature verification.

II. Related Work

The majority of existing deduplication schemes are based on Convergent Encryption (CE), which address the issue of incongruity between conventional encryption algorithms and deduplication systems. Douceur *et al.* [3] is the first to introduce the concept of CE. This method entails encrypting a file by utilizing its hash value as the key, thereby allowing for the same plaintext to be encrypted into identical ciphertext. Despite its efficiency and straightforwardness, CE remains susceptible to dictionary attacks as its data entropy is low [4]. Moreover, it lacks the ability to achieve semantic security. To formalize the concept of CE, Bellare *et al.* [5] propose a new cryptographic primitive–Message Lock Encryption (MLE), which complicates the key computation and encryption methods but has no change in its core idea compared to CE, thus also fails to achieve semantic security [10] [11]. In the follow-up works, Chen *et al.* [12] develop a novel scheme named Block-Level Message-Locked Encryption (BL-MLE), which enables both file-level and block-level deduplication to be performed simultaneously with a smaller set of metadata [13]. Subsequently, Xu *et al.* [14] propose a multi-client crossover deduplication scheme named Xu-CDE as a natural extension of CE. However, the PoW credentials used in this scheme lack real-time protection, causing it vulnerable to replay attacks [15].

There are some works based on Trusted Third Party (TTP) in encrypted data deduplication. Stanek *et al.* [18] put forward a concept of popularity to classify data into popular data and unpopular data. In this scheme, the TTP consists of an identity provider and an indexing service. The former is employed to mitigate sybil attacks, while the latter serves to prevent the leakage of unpopular data information to the CSP. Other schemes also employ TTP [16] [17]. Nevertheless, the practicality of finding a suitable TTP remains a significant challenge. To get rid of TTP, Fan *et al.* [19] present a secure deduplication scheme that leverages the Trusted Execution Environment (TEE) [20]. Due to the distinctiveness of TEE, sensitive operations such as key management are restricted to TEE, thereby eliminating the need for a TTP.

There is another line of works based on the proof of ownership (PoW) technique. Early deduplication schemes based on the hash-as-proof mechanism are vulnerable against brute-force attacks as the hash is easy to get. Then, Halevi *et al.* [9] propose a scheme to thwart counterfeit attacks utilizing Merkle hash tree (MHT). In this approach, the client and server encode the file into a buffer, which is segmented into fixed-size blocks, with the hash of each data block serving as a leaf node. The parent node is obtained by hashing two child nodes, and ultimately the root node of the entire tree is computed. During the MHT-PoW verification process, the server randomly selects k leaf nodes as challenges, and the client returns the path information from leaf nodes to the root node for response. Subsequently, the server can calculate the value of the root node based on the client response. By comparing the two root node values, the server can determine whether the user possesses the file. However, the scheme does not take into consideration the insecurity of the communication channel [21] [22]. If an adversary can eavesdrop on the content between the server and the client, the adversary can recover the whole MHT after a few PoW verifications especially for small files.

TABLE I: Description of symbols

Method	Semantic security	Without TTP	Secure communication	PoW
CE [3]	×	✓	×	×
MLE [5]	×	✓	×	×
Xu-CDE [14]	×	✓	×	×
Key-sharing [17]	✓	×	×	✓
TEE [19]	✓	✓	×	✓
MHT [9]	✓	✓	×	✓
Our scheme	✓	✓	✓	✓

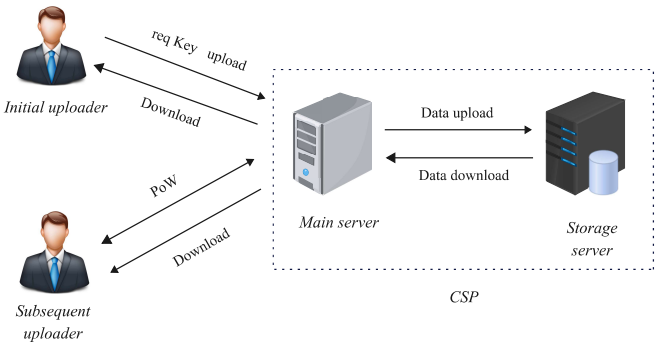

Fig. 1: System Model

In summary, existing schemes have multiple shortcomings in terms of their resistance to brute force attacks, and some of the prerequisites are unfeasible for real-world appliaction. So this study aims to address the prevailing weaknesses in current encrypted data deduplication scheme. In order to illustrate the distinctions between our scheme and the existing works, we conduct a comparative analysis, and the results are presented in Table I.

III. System and Adversary Model

A. System Model

In the proposed scheme, the system model primarily includes two entities: User and CSP. These entities are illustrated in Fig. 1.

User: User is an entity that outsources the data storage to the CSP and can subsequently access the downloaded data from the cloud. In a deduplicated cloud storage system, users can minimize network transfers by uploading data files only once, instead of uploading files that already exist in the cloud storage. Users can be further classified into two categories: initial uploaders and subsequent uploaders.

CSP: CSP is an entity that consists of a main server and a storage server. The main server of CSP is tasked with conducting cross-user file duplicity detection, verifying data ownership, and generating tags. The storage server, on the other hand, is responsible for securely storing the encrypted

file data blocks and facilitating user access requests by communicating with the main server.

B. Adversary Model

In the proposed scheme, two possible adversaries are considered: internal adversary and external adversary.

Internal adversary: The internal adversary refers to someone with authorized access to the CSP and may attempt to retrieve users' encrypted data without permission. This adversary, may be a cloud service provider or one of its employees, and can access ciphertext without the knowledge or consent of the respective user.

External adversary: In the context of cloud computing, external adversaries are typically defined as malicious users who engage in unauthorized access of private data belonging to other users. These adversaries can employ various tactics to achieve their objectives, such as attacking legitimate clients to obtain partial information about specific copies of data.

C. Design Goals

The design goal of this scheme is that CSP can securely accomplish the deduplication of encrypted data. Therefore, our scheme should satisfy the following properties:

- **Security**: 1) The uploaded ciphertext is semantically secure. 2) The security of file tags, including its unforgeability and distinguishability. 3) The security of the ADMHF-PoW process.
- **Efficiency**: The proposed solution should be efficient in terms of time and storage.

IV. Proposed Deduplication Scheme

A. Preliminary

*1) **Bilinear mapping**:* Let $(G, +)$ and $(G_T, \times)$ be addictive and multiplicative groups with the same prime order p respectively, g is the generator of G. Let $e : G \times G \to G_T$ be a bilinear map which satisfies the following properties.

- Bilinear: $\forall a, b \in Z_p^*$, $\forall P, Q \in G$, $e(aP, bQ) = e(P, Q)^{ab}$.
- Non-degenerate: $\exists P, Q \in G$, such that $e(P, Q) \neq 1$.
- Computable: There is an efficient algorithm to compute $e(P, Q)$, for $P, Q \in G$.

B. Implementation

Our scheme consists of four processes, including *SystemSet*, *CheckTagGen*, *FileUpload*, and *FileDownload*. The overview is shown in Fig.2.

1) SystemSet:

- ***Step 1.*** Two cyclic groups G_1 and G_T of prime order p are chosen and g is the generating element of G_1. Define bilinear mapping $e : G_1 \times G_1 \to G_T$.
- ***Step 2.*** CSP randomly choose $g', h, h' \xleftarrow{R} G$, $r \xleftarrow{R} Z_p^*$, and calculates $g_1 = g^r$, $Z = e(g^r, g')$. Set the system main public key as $mpk = g'^r$, the public parameter is $(G_1, G_2, p, e, g, g_1, g', h, h', Z, H, SH)$.
- ***Step 3.*** Each user in the system picks a random number $s \xleftarrow{R} Z_p^*$, calculates the user's public key as $pk = g^s$, and private key as $sk = (d_1, d_2, d_3) = (g'^r, (g_1^{pk} h)^s, g^s)$.
- ***Step 4.*** Input security parameters λ to update the symmetric encryption algorithm.

2) CheckTagGen: When a user U intends to upload a file F to CSP, s/he first executes the remote attestation to establish a secure communication channel. Then U tries to acquire a blind signature of F from CSP. The process is shown in Alg.1.

Algorithm 1 Check Tag Generation

User:
  Randomly chooses $q \xleftarrow{R} Z_p^*$.
  Computes the short hash value of the file $h_F = SH(F)$.
  Calculates $F' = h_F^q$.
User $\to$ CSP: Sends F' for blind signature.
CSP: Computes $\alpha' = F'^r$.
CSP $\to$ User: Send α' to U.
User: Calculates $\alpha = \alpha'^{q^{-1}}$.
if $e(\alpha, g) = e(h_F, g_1)$ holds **then**
  // Signature α' from CSP is right.
  User:
    Computes $K_1 = H(\alpha)$.
    Computes the ciphertext of F, $C_F = E(K_1, F)$.
    Computes the file tag $Tag_F = H(C_F)$.
    Stores Tag_F to verify data integrity.
  User $\to$ CSP: Send Tag_F as the file tag.
  if Tag_F already existed in CSP **then**
    CSP $\to$ User: U is a subsequent uploader.
    Execute Algorithm 2.2.
  else
    CSP $\to$ User: U is an initial uploader.
    Execute Algorithm 2.1.
  end if
else
  Upload termination.
end if

3) FileUpload: Based on the result of Alg. 1, U is divided into initial uploader or subsequent uploader.

Case 1: For initial uploaders, to achieve a balance between the security and the efficiency of data encryption, we adopt a strategy that classify the data into two categories: popular data and unpopular data. For popular data with low level of privacy, it is sufficient to upload C_F as ciphertext. For unpopular data, a two-layered encryption is required. When the number of authorized users reaches the popularity threshold t, that is $Cout_F = t$, the outer layer encryption of unpopular data is decrypted. Notably, t is a constant determined by the application scenario and security requirements. As t increases, the security of the system improves, but it increases the overhead for users to decrypt during the file download phase. Regarding the specific value of t in real-world setting, readers can refer to [18].

After obtaining the ciphertext, the corresponding ADHMF is generated for subsequent verifications. To mitigate the potential vulnerabilities associated with the MHT, we propose using a new verification data structure called the adaptive dynamic Merkle hash forest (ADMHF). It is a collection of m mutually independent MHTs. Moreover, each ADMHF corresponding

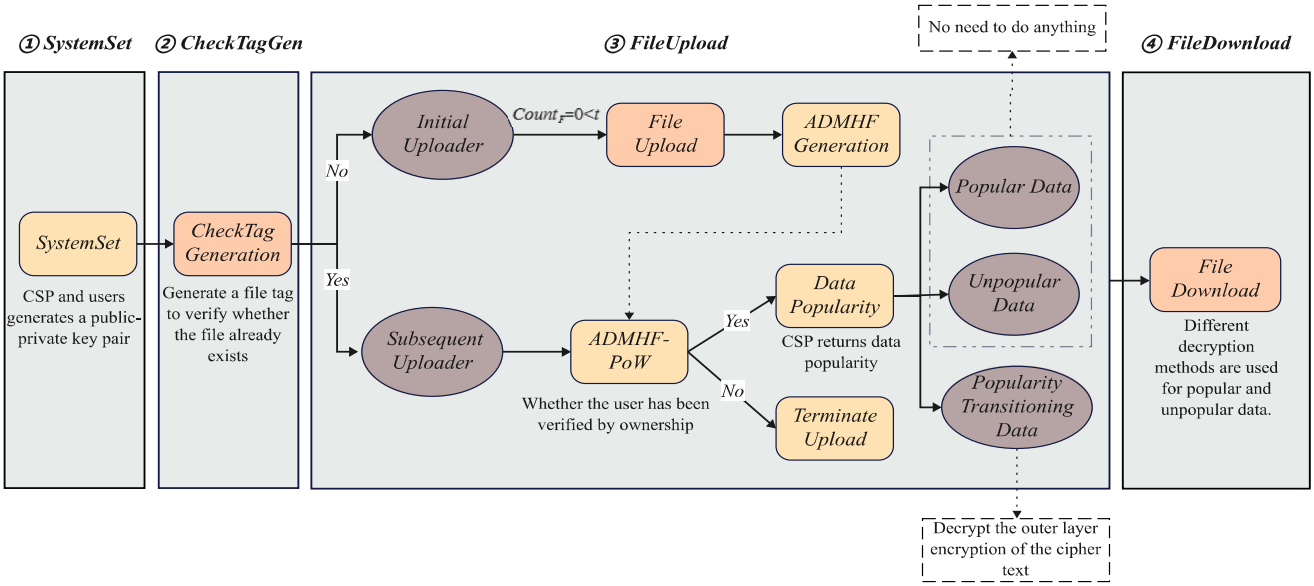

Fig. 2: The Overview of AF-Dedup

to a particular file. The number of trees contained within the ADMHF is determined by the file size, according to a specified formula.

$$y = F(x) = \frac{a}{1 + e^{b - \frac{c}{x}}} \tag{1}$$

The equation (1) illustrates the dynamic relationship between y, which denotes the number of trees in ADMHF, and x, representing the number of data blocks. The graph of the function appears as an inverted S-curve. Parameter a determines the upper bound of the function value. Parameter b and c affect the critical point and change tendency of the curve.

To achieve the diversity of MHTs in ADMHF, we introduce salt to the hash function in the MHT generation process. $Salt = \{s_1, s_2, \cdots, s_m\}$ is a set of randomly generated strings, in which each element $s_i, i \in \{1, \cdots, m\}$ corresponds to a unique MHT in the forest. The value of the parent node is obtained by concatenating the values of its two child nodes with a salt value in $Salt$, and then performing a hash operation on the resulting string. The detailed process is shown in Alg. 2.1.

Algorithm 2.1 File Upload: Initial Uploader

// $Count_F = 0 < t$, which implies the data is unpopular data now.
User:
  $L \xleftarrow{R} Z_p^*$ for two-layered encryption.
  $C_1 = g^L$, $C_2 = C_F \oplus Z^L$,
  $C_3 = (g_1^{pk} h)^L$, $C_4 = (g_1^{h_F} h')^L$.
  Computes the ciphertext $C'_F = C_1 || C_2 || C_3 || C_4$.
  Computes $K_2 = H(F)$, $\tau_F = K_2 \oplus L$.
User $\rightarrow$ CSP: Send $< C'_F, \tau_F, Tag_{pop} = 1 >$ to CSP.
CSP:
  Divides C_F into blocks $\{B_i\}$, $i \in \{1, \cdots, Num_t\}$.
  Calculates m according to (1).
  Generates random salt values $Salt = \{s_1, s_2, \cdots, s_m\}$.
for $i = 1$ to m **do**
  Generates MHT_i and add s_i to MHT_i when calculate the hash value of each node.
  Signs the root node n_{Root_i}, and obtains $Sig_{F_i} = n^r_{Root_i}$ to verify the integrity of MHT_i.
  Stores s_i and the root node n_{Root_i}.
end for

Among them, Tag_{pop} is a variable that indicates the current data popularity. When $Count_F < t$, F is unpopular data now, that is $Tag_{pop} = 1$. When $Count_F > t$, F is popular data, then $Tag_{pop} = 0$. When $Count_F = t$, then $Tag_{pop} = 2$, which indicates that the file is undergoing a transition from unpopular data to popular data, and it is considered popularity transitioning data.

Case 2: For subsequent uploaders, CSP tends to perform PoW on U. The ADMHF-PoW process consists of **Challenge**, **Response** and **Verification**. The detailed process is shown in Alg. 2.2.

Algorithm 2.2 File Upload: Subsequent Uploader

Challenge:
// CSP generates a challenge set for PoW.
Randomly select a MHT_i, $i \in \{1, 2, \cdots, m\}$ from ADMHF.
Computes the hash value of current root node.
if $e(Sig_{F_i}, g) = e(n_{Root_i}, g_1)$ **then**
  Generates a challenge set $ch = \{ch_1, ch_2, \cdots, ch_k\}$, $ch_j \in \{1, 2, \cdots, Num_t\}$.
  CSP $\rightarrow$ User: Sends $< s_i, ch, \tau_F >$ to U.
else
  The data integrity of MHT_i has been corrupted.
end if

Response:
// U generates a response to prove his ownership.
Computes $K_2 = H(F)$, $L = \tau_F \oplus K_2$.
Encrypts C_F with L to obtain C'_F.
Divides C'_F into blocks $\{B_i\}$, $i \in \{1, \cdots, Num_t\}$.
Adds salt s_i to each of the nodes to obtain MHT'_i.
Computes the corresponding response $res = \{leaf_i, Bro(leaf_i)\}$, $i \in ch$ for each node indexed by ch.
User $\rightarrow$ CSP: Send res to CSP.

Verification:
// CSP verifies the correctness of the response.
CSP uses the values in res to calculate the root node of MHT_i.
if calculated value is identical to the actual value **then**
  Adds U to the owner list, $Count_F+ = 1$.
  if $Count_F < t$ **then**
    CSP $\rightarrow$ User: Returns $Tag_{pop} = 1$.
  else if $Count_F = t$ **then**
    CSP $\rightarrow$ User: Returns $Tag_{pop} = 2$.
  else
    CSP $\rightarrow$ User: Returns $Tag_{pop} = 0$.
  end if
  Execute Algorithm 3.
else
  The verification fails.
end if

Among them, in the process of **Response**, $Route(leaf_i)$ denotes the set of nodes located on the path from the leaf node $leaf_i$ to the root node. And $Bro(leaf_i)$ denotes the set of sibling nodes for each node within $Route(leaf_i)$.

The ADMHF-PoW process is illustrated with the example shown in Fig. 3. In this case, the file is split into 8 blocks. During the challenge process, CSP randomly selects a MHT_i and generates a challenge set $ch = \{3, 7\}$, then sends $< s_i, ch, \tau_F >$ to U. U calculates the response $res = \{n_{31}, n_{43}, n_{44}, n_{33}, n_{47}, n_{48}\}$ with s_i. Then CSP uses the node values in res to calculate the root node to verify the response.

If U passes the PoW, it can be assumed that U owns the file. Then data deduplication needs to be performed, the detailed process is shown in Alg. 3.

Algorithm 3 Data Deduplication

if $Tag_{pop} = 2$ **then**
  // It is considered as popularity transitioning data.
  User:
    Decrypts C'_F using L.
    Then obtains $C_F = (C_2 \cdot e(C_3, d_3)) \oplus (e(d_1, C_1) \cdot e(d_2, C_1))$.
  User $\rightarrow$ CSP: Uploads $< C_F, Tag_{pop} = 0 >$ to replace the original tuple $< C'_F, \tau_F, Tag_{pop} = 1 >$.
else
  // Data Deduplication.
  U does not need to do anything.
end if

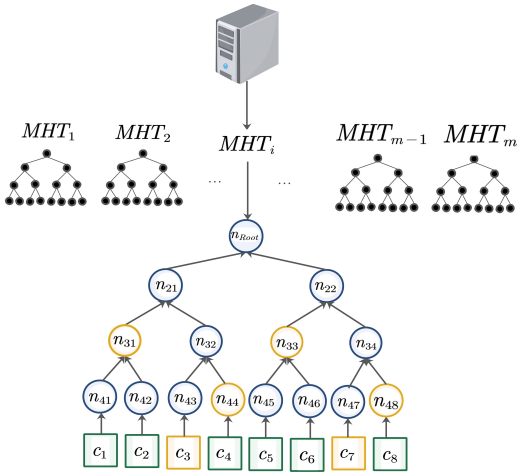

Fig. 3: ADMHF-PoW Process

Here is the correctness proof for the outer layer decryption:

$$
\begin{aligned}
&(C_2 \cdot e(C_3, d_3)) \oplus (e(d_1, C_1) \cdot e(d_2, C_1)) \\
&= ((C_F \oplus e(g^r, g')^L) \cdot e(C_3, d_3)) \oplus (e(d_1, C_1) \cdot e(d_2, C_1)) \\
&= C_F \oplus (e(g^r, g')^L \cdot e((g_1^{pk} h)^L, g^s)) \oplus (e(g'^r, g^L) \cdot e((g_1^{pk} h)^s), g^L) \\
&= C_F \oplus (e(g^r, g')^L \cdot e((g_1^{pk} h)^L, g^s)) \oplus ((e(g^r, g')^L \cdot e((g_1^{pk} h)^L, g^s))) \\
&= C_F \oplus 0 = C_F.
\end{aligned}
$$

4) FileDownload: When U intends to download file F, he sends a request to CSP. CSP authenticates U and searches the list of file owners to determine whether U has access. If found, CSP returns current data popularity to U. Otherwise, the request will be rejected. In addition, the user is informed if there is a change in data popularity. The encrypt and decrypt process is shown in Alg. 4.

Algorithm 4 File Download

CSP $\rightarrow$ User: Sends $< C'_F, \tau_F, Tag_{pop} = 1 >$ or $< C_F, Tag_{pop} = 0 >$.
if $Tag_{pop} = 1$ **then**
  User: Decrypts C'_F using L.
  User: Obtains $C_F = (C_2 \cdot e(C_3, d_3)) \oplus (e(d_1, C_1) \cdot e(d_2, C_1))$.
end if
Computes $Tag'_F = H(C_F)$.
if $Tag'_F = Tag_F$ **then**
  User: Computes $F = D(K_1, C_F)$ to get the plaintext.
else
  Data integrity is compromised.
end if

C. Computation Complexity Analysis

We analyze the computation complexity of users and CSP in different phases in Table II. Among them, $FSize$ represents the size of the file, Num_t is the number of leaf nodes of MHT, m represents the number of MHTs in ADMHF, and k is the number of nodes CSP challenges at each round.

TABLE II: Computation complexity analysis of different entities in different phases

	User	CSP
CheckTagGen	$O(FSize)+O(\log p)$	$O(\log p)$
OuterLayerEnc	$O(\log p)$	–
ADMHFGen	–	$O(mNum_t \log Num_t)$
Challenge	–	$O(k)$
Response	$O(Num_t)+O(k \log Num_t)$	–
Verify	–	$O(k \log Num_t)$

V. Security Analysis

As described in Section III-C, this paper focuses on achieving three key security goals: ensuring the security of file tags, maintaining semantic security for ciphertexts, and providing the security of the PoW process. In the following security proof, we will explain how AF-Dedup achieves these security goals.

A. Security of File Tag

First, we analyze the correctness of the file tag, which is ensured by the fact that the CSP returns the correct signature during the file tag generation phase.

Theorem 1. For the file F, if $e(\alpha, g) = e(h_F, g_1)$ holds, the CSP must return the correct signature α' during the generating process of file tag.

Proof. According to the feature of bilinear mapping:

$$\begin{aligned} e(\alpha, g) &= e(\alpha'^{q^{-1}}, g) \\ &= e(h_F^{qrq^{-1}}, g) = e(h_F^r, g) \\ &= e(h_F, g^r) = e(h_F, g_1). \end{aligned}$$

If the equation above holds, U can confirm that the signature is valid, thus ensuring the correctness of the file tag. □

The security of file tags includes their unforgeability and distinguishability. In the following text, Theorem 2 analyzes the distinguishability of file tags, while Theorem 3 demonstrates the unforgeability of file tags.

Lemma 1. For a secure hash function $H : \{0,1\}^* \to G_1$, $\forall F_1, F_2 \in [0,1]^*$, $F_1 \neq F_2$, the probability that $H(F_1) = H(F_2)$ is negligible, that is:

$$Pr[H(F_1) = H(F_2) | F_1 \neq F_2] \leq \varepsilon. \tag{2}$$

Theorem 2. Let the file tag uploaded by user U_i be Tag_F and the file tag uploaded by U_j be $Tag_{F'}$. If $F \neq F'$, the probability that $Tag_F = Tag_{F'}$ is negligible. That is:

$$Pr[Tag_F = Tag_{F'} | F \neq F'] \leq \varepsilon. \tag{3}$$

Proof. The proof of this theorem can be done by proof by contradiction. Suppose there exists $F \neq F'$ such that $Tag_F = Tag_{F'}$. According to the scheme of this paper that:

$$\begin{aligned} &Tag_F = Tag_{F'} \\ &\Leftrightarrow H(C_F) = H(C_{F'}) \\ &\Leftrightarrow H(E(H(\alpha_F), F)) = H(E(H(\alpha_{F'}), F')) \\ &\Leftrightarrow H(E(H(h_F^r), F)) = H(E(H(h_{F'}^r), F')) \\ &\Leftrightarrow H(E(H(SH(F)^r), F)) = H(E(H(SH(F')^r), F')). \end{aligned}$$

If the equation above holds, according to **Lemma 1** and the determinism of the encrypted algorithm, there must be $F = F'$. However, it contradicts the assumption $F \neq F'$, therefore the assumption does not hold. □

Theorem 3. Let the file tag uploaded by user U_i be Tag_F and the file tag uploaded by U_j be $Tag_{F'}$. If $Tag_F = Tag_{F'}$, then the probability that $F \neq F'$ is negligible, that is:

$$Pr[F \neq F' | Tag_F = Tag_{F'}] \leq \varepsilon. \tag{4}$$

Proof. Without loss of generality, from the steps in the scheme, we can conclude:

$$Tag_F = H(E(H(SH(F)^r), F)). \tag{5}$$

If $F \neq F'$, we discuss attacks on the file tag Tag_F by the adversary $\mathcal{A}$ from the following two perspectives:

(1) Assuming $\mathcal{A}$ is a malicious user, the parameters F and r are unknown.

(2) Assuming $\mathcal{A}$ is a CSP, the parameter F is unknown.

In both of these cases, $\mathcal{A}$ cannot construct a file tag that satisfy $Tag_F = Tag_{F'}$. □

B. Data Privacy

We illustrate the semantic security of the ciphertext by analyzing its resistance to offline brute-force attacks. Based on our proofs, for an attacker with limited computational power, it is impossible to obtain any important information about the plaintext from the ciphertext.

*1) **Offline brute-force attack**:* If an internal adversary $\mathcal{A}$, such as CSP, acquires the file tag, ciphertext and data popularity, he performs an offline brute-force attack on F. In addition, for unpopular data, $\mathcal{A}$ also gets τ_F.

As for popular data:

- ***Step 1.*** $\mathcal{A}$ lists a possible data set $\{F_i\}$, $|F| = FSize$, where $i \in [1, 2^{FSize}]$. Then he computes $\{h_{F_i} = SH(F_i)\}$.
- ***Step 2.*** $\mathcal{A}$ lists all possible private key values of CSP $\{r_j\}$, $j \in [1, p-1]$, then calculates $\{\alpha_{ij} = h_{F_i}^{r_j}\}$.
- ***Step 3.*** Then $\mathcal{A}$ calculates the ciphertext set $\{C_{F_{ij}} = E(H(\alpha_{ij}), F_i)\}$.

As for unpopular data:

- ***Step 1.*** $\mathcal{A}$ lists a possible data set $\{F_i\}$, $|F| = FSize$, where $i \in [1, 2^{FSize}]$. Then $\mathcal{A}$ computes $\{K_{2_i} = H(F_i)\}$, $\{L_i = K_{2_i} \oplus \tau_F\}$.
- ***Step 2.*** $\mathcal{A}$ uses the above method of popular data to get the ciphertext $\{C_{F_i}\}$.
- ***Step 3.*** $\mathcal{A}$ Encrypts $\{C_{F_i}\}$ with $\{L_i\}$ to obtain $\{C'_{F_i}\}$.

If there exists $C_{F_i} = C_F$ or $C'_{F_i} = C'_F$, where $i \in [1, 2^{FSize}]$, $\mathcal{A}$ can obtain the plaintext data. The time complexity of this process is $O(p \cdot 2^{FSize})$, which is computationally infeasible.

Lemma 2. The Discrete Logarithm problem (DL problem): Given $g^a \in G$ where $a \in Z_p^*$, computing a is hard.

Theorem 4. The scheme is resistant to offline brute-force attack.

Proof. From **Lemma 2** in $\alpha = h_F^r$, guessing r is difficult. Therefore, $\mathcal{A}$ cannot compute the plaintext from the ciphertext even if it perform an offline brute-force attack. □

By analyzing the scheme's resistance to online attacks, it can be proved that an adversary cannot gain ownership of a certain file by interacting with the CSP in the case of unknown the plaintext.

*2) **Online brute-force attack**:* After eavesdropping the communication channel, the external adversary $\mathcal{A}$ obtains the file tag Tag_F. Using this information, $\mathcal{A}$ performs an online brute-force attack on F.

- ***Step 1.*** $\mathcal{A}$ uploads Tag_F to CSP. As Tag_F is previously stored, $\mathcal{A}$ is recognized as a subsequent uploader.
- ***Step 2.*** CSP generates a challenge ch to $\mathcal{A}$, assuming that $|ch| = 1$.
- ***Step 3.*** $\mathcal{A}$ generates response by enumerating the hash values in res. For one hash value, $\mathcal{A}$ lists all its possible values $\{h_i\}$, $|h| = HashLen$, $i \in [1, 2^{HashLen}]$. In one verification round, $\mathcal{A}$ needs to give a total of $\log(Num_t) + 1$ such hash values.
- ***Step 4.*** $\mathcal{A}$ sends res to CSP for verification.
- ***Step 5.*** CPS returns the verification result.

Theorem 5. If $\mathcal{A}$ knows Tag_F and unknows F, the probability that $\mathcal{A}$ wins the $Game$ to obtain the ownership of F is negligible. That is:

$$Pr[\mathcal{A}_{wins}] \leq \varepsilon. \tag{6}$$

Proof. The probability of $\mathcal{A}$ guessing one hash value is $(\frac{1}{2})^{HashLen}$. $\mathcal{A}$ must guess all $\log(Num_t) + 1$ hash values correctly at the same time to win the game. Hence, the probability that $\mathcal{A}$ wins is $(\frac{1}{2})^{HashLen(\log Num_t + 1)}$. For SHA-256, $HashLen$ is 256 bits. Furthermore, in practice, the value of $|ch|$ is usually greater than 1, and this probability decreases sharply as $|ch|$ increases. Therefore, it is computationally infeasible for $\mathcal{A}$ to win the $Game$. □

C. Security of PoW Process

1) Exposure of Merkle Hash Tree: Typically, PoW is designed to operate reliably, regardless of the number of times it occurs. However, in the MHT-PoW scheme [9], a portion of the MHT nodes becomes exposed with each ownership verification attempt. If the adversary can collect enough response sets res, all nodes of the entire MHT can be reconstructed. At this point, no matter what challenge CSP initiates, the adversary can return the correct response, thereby gaining ownership of the entire file. It should be noted that for smaller files, the adversary needs to collect fewer different response sets, making the MHT constructed on it more vulnerable.

Theorem 6. If the adversary collects $Num_t/2$ distinct response sets of the same file, where Num_t represents the number of leaf nodes, then the adversary can reconstruct this MHT.

Proof. Suppose for file F, the adversary collected Num_t distinct response sets. We represent this with a matrix, where each row of the matrix represents the response set collected by the adversary each round:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{c,1} & a_{c,2} & \cdots & a_{c,n} \end{bmatrix}. \tag{7}$$

For the sake of convenience, we use n to represent the length of res and c to denote $Num_t/2$. Here, we discuss the scenario where only one challenge node is initiated by CSP. In practical applications, there are typically multiple challenge nodes, resulting in the exposure of a greater number of nodes.

During the **Verification** process, to enable the CSP to correctly generate the root node based on res, it's necessary to label whether the node is a left child or a right child. However, adversaries can exploit this information along with the collected response sets to launch attacks. The last column in the matrix corresponds to the second level of the MHT. Since the second level has only two nodes, this column has only two distinct values. Based on this, attackers can reconstruct the second level. By repeating this process, attackers can use the same method to reconstruct all subsequent nodes and thereby reconstruct the entire MHT.

□

2) Security of ADMHF-PoW: The previous section demonstrated the insecurity of the MHT scheme. In the following section, we will demonstrate how the ADMHF-PoW scheme enhances the security significantly of the PoW process.

Theorem 7. Compared with MHT-PoW scheme, ADMHF-PoW scheme significantly reduces the node exposure rate in each round of verification.

Proof. Assuming MHT is a complete binary tree, its leaf nodes are 2^i. Therefore, the total number of nodes in the MHT is $Num_{sum} = 2^{i+1} - 1$. Assuming $|ch| = 1$ in each verification attempt, as the user needs to provide the values of all the sibling nodes on the path from the challenge node in ch to the root node, then $|res| = i + 1$. Since some of the other nodes can be computed by nodes in res as child node, the number of exposed nodes is $|Expose| = 2i + 1$.

We define γ as the percentage of the exposed content of MHT, which is shown in (8).

$$\gamma = \frac{|Expose|}{Num_{sum}} = \frac{2i+1}{2^{i+1}-1} \times 100\% \tag{8}$$

For ADMHF scheme, CSP generates an ADMHF with m MHTs. Among them, m can be calculated by (1). So the

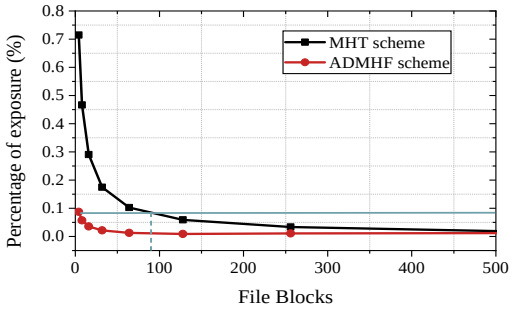

Fig. 4: Percentage of exposure rate in one challenge-response round

exposure rate of nodes in one verification attempt of the ADMHF scheme γ' is shown in (9).

$$\gamma' = \frac{|Expose|}{m \times Num_{sum}} = \frac{\gamma}{m} \times 100\% \tag{9}$$

Given $a = 8$, $b = 2.5$, $c = 488$ in (1), the relationship between exposure rate of nodes in one verification attempt and Num_t is shown in the Fig. 4. It can be observed that the ADMHF scheme for a file with 2 blocks achives the same level of security as MHT scheme for a file with 91 blocks. As a result, ADMHF-PoW significantly improves the security of small files. □

VI. Experiment and Evaluation

All the experiments are performed on multiple devices with Intel (R) Core (TM) i5-8500T CPU processor running at 2.1 GHz and 4GB RAM on Linux operation system. In which, 2 devices act as the main server and the storage server, and the rest of the devices act as users.

The proposed scheme is implemented with C++ language, using PBC [24], OpenSSL [25] function library to realize the cryptography algorithm. We use SHA-1 to compute the short hash value of the file and use SHA-256 as the hash function to calculate the hash value of the file, the encryption key, the file tag and generate ADMHF. Encryption and decryption are based on AES-256 symmetric encryption algorithm.

To simulate real-world scenarios, we store 3000 different files on a cloud server, with sizes ranging from 1KB to 256MB. The number of users owning each file is randomly assigned. We set the popularity threshold to $t = 7$, resulting in a rough ratio of 3:2 between unpopular data and popular data.

A. ADMHF Evaluation

Let N_{cr} denote the number of verification attempts, and γ represent the percentage of the exposed MHT content. We test the correlation between N_{cr} and γ experimentally with different numbers of challenge blocks. The results are shown in Fig. 5.

The data indicates that there is a positive correlation between N_{cr} and γ. Furthermore, as N_{cr} increases, the rate of increase in γ slows down. Simultaneously, as the number of challenge blocks increases, γ increases. Furthermore, it can be observed that the value of γ decreases as the number of file blocks increases. For example, in Fig. 5(a), γ reaches 99.327% when file block num=500. In Fig. 5(b), the value of γ is 92.156% with file block num=800 and other parameters kept the same.

Then, we conduct experiments to obtain the value of N_{cr} required to lead to full exposure. Let N_0 be the minimum number of N_{cr} that actually cause full exposure of the MHT content in our test. We evaluate the relation between N_0 and the number of file blocks by simulating multiple PoW processes. The results are shown in Fig. 6.

The results demonstrate that, in the case of small files with the same number of blocks, the value of N_0 derived from the ADMHF scheme is significantly larger than that derived from the MHT scheme. Furthermore, the value of N_0 based on ADMHF exhibits a increase as the file size increases, indicating a improvement in security. Upon reaching a certain file size, such as 600 blocks, it can be observed that ADMHF degrades into a single MHT, resulting in a negligible difference between the values of N_0 derived from ADMHF and MHT-based methods.

TABLE III: Description of symbols

Symbol	Description
S_C	Ciphertext size
S_T	File tag size
S_{tree}	MHT size, including the root and the salt
S_{sk}	Private key size
S_k	Encryption key size
S_{map}	Bilinear map size
S_h	Hash size
S_{ξ_F}	Authentication information size
S_s	Authentication parameter size
S_{A_m}	Verification initial value size
S_{A_α}	Privilege set size
S_{td}	Trapdoor size

TABLE IV: Communication overhead comparison

Scheme	Communication overhead
Our scheme	$2S_{map} + S_T + S_C + S_k$
Key-sharing	$S_{tree} + S_T + S_C + S_{map}$
TEE	$S_{\xi_F} + S_s + S_C + S_{A_m}$

B. Performance Analysis

We conduct a theoretical analysis of the scheme, examining its performance in terms of communication overhead and storage overhead. The measurement symbols used in our analysis are defined in Table III.

We first analyze the communication overhead required by the proposed scheme and compare it with key-sharing scheme [17] and TEE scheme [19]. The results are shown in Table IV. As the proposed scheme does not require any TTP, some communication costs are eliminated.

The storage overhead analysis includes overhead on the cloud server side, the client side and the additional TTP side. As the third party server is not used in the proposed scheme, no additional storage overhead is required. The results are shown in Table V.

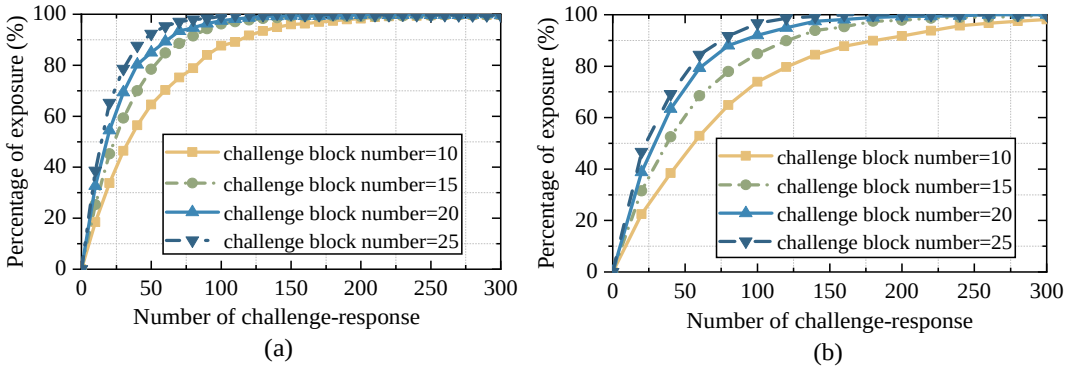

Fig. 5: Percentage of content exposure. (a)File block number=500, (b) File block number=800.

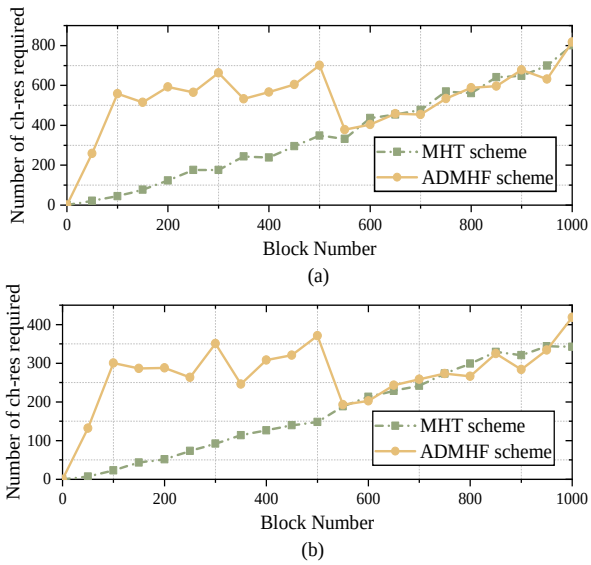

Fig. 6: Actual number of challenge-response resulting in full exposure. (a) Challenge block number=10, (b) Challenge block number=20.

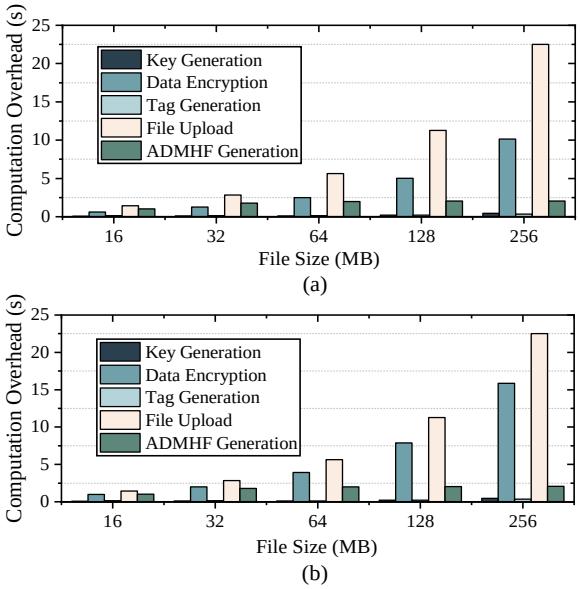

Fig. 7: Computation overhead for different file size. (a) Popular data Computation overhead, (b) Unpopular data computation overhead.

TABLE V: Storage overhead comparison

Scheme	Server side	Client side	TTP side
Our scheme	$S_C + NS_{tree} + S_{sk} + S_k$	$S_T + S_k$	-
Key-sharing	$S_k + S_T + S_C$	S_{sk}	$S_C + S_{sk}$
TEE	$S_C + S_s + S_{\xi_F} + S_{A_m}$	$S_h + bS_{A_\alpha} + S_{td}$	-

C. Computation Overhead

The computation overhead of the proposed scheme is evaluated by conducting experiments on five distinct file sets, ranging in size from 16MB to 256MB. Measurements are taken at various stages of the file upload process, including key generation, data encryption, file tag generation, file uploading, and ADMHF generation. The experimental results are presented in Fig. 7.

Fig. 7 highlights that the computation overhead associated with key and file tag generation is practically negligible. This is attributed to the fact that during the tag generation phase, we compute the short hash value for files of any size. The subsequent blind signature is applied to this hash value, resulting in higher efficiency. Besieds, the computation overhead for ADMHF generation demonstrates little difference between files of different sizes, which exhibits good scalability. For small files, more MHTs need to be generated, while for large files, although the number of MHTs is reduced, the computation overhead of generating a single MHT increases. Besides, the computation overhead required for both file encryption and upload process increases with file size.

We compare the latency of the PoW process with the MHT-PoW scheme [9] and the key-sharing scheme [17]. The results are illustrated in Fig. 8(a). In our experiment, We set the number of challenge nodes to be $10\%Num_t$, where Num_t represents the number of leaf nodes in the MHT. As shown in Fig. 8(a), our scheme has lower latency compared to [17], but slightly higher than [9]. Although the efficiency of [9] is higher, it comes at the cost of sacrificing security. Specifically, it can only prove security for a more restrictive set of input distributions and under an assumption about the linear code. On the other hand, [17] has a fixed number of leaf nodes in the construction of the MHT, resulting in

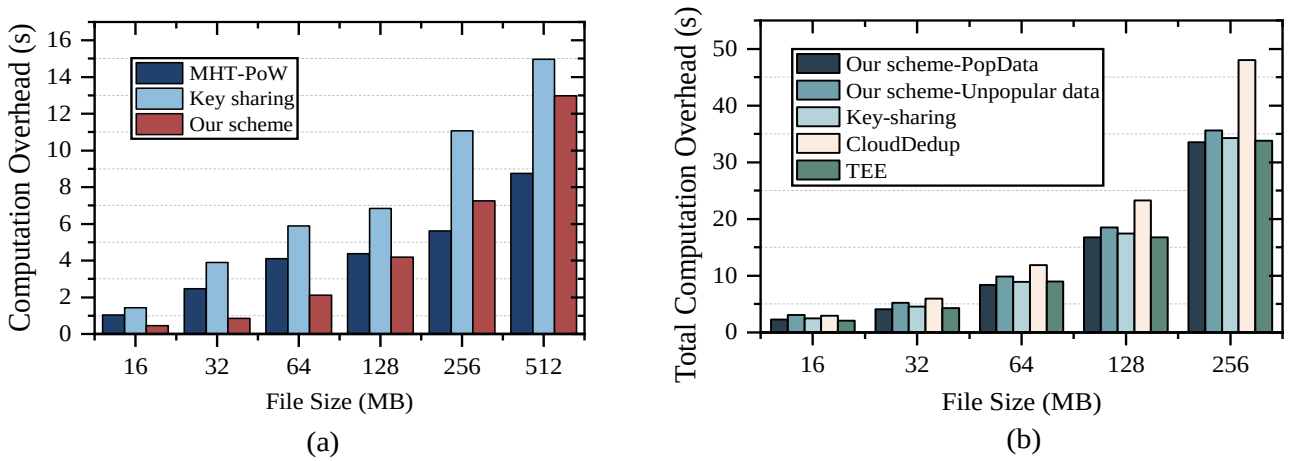

Fig. 8: (a) Comparison of computation overhead in the PoW process. (b) Total computation overhead comparison.

higher computation overhead for small files. Considering both security and efficiency, our scheme is superior.

In order to assess the performance of our scheme, we conduct experiments to evaluate the total computation overhead and compared it with other schemes, including Key-sharing [17], ClouDedup [26], and TEE [19]. The results, as shown in Fig. 8(b), indicate that our scheme exhibits superior performance in terms of total computation overhead for popular data. However, for unpopular data, our total computation overhead is slightly higher due to the implementation of two-layered encryption, which enhances data security.

D. Extending to Large Files

To test the performance of our scheme with large files and its availability in real-world scenarios, we utilize a real-world dataset collected from Storage Lab at Stony Brook University [27]. For a file of size 1GB, we select a file chunk size of 1KB. At this configuration, the computation time for file tag generation is 0.473s, file encryption takes 69.754s, and the PoW process required 7.242s. Then we conduct tests with a file chunk size of 32KB. In this case, the times for the three steps are 0.475s, 69.241s, and 1.391s, respectively. In comparison with the PoW process and the scheme presented in [9], our scheme demonstrate a remarkable 95.089% reduction in computation overhead.

VII. Conclusion

This paper introduces the design of ADMHF, a novel data structure for PoW, which is applied to cloud storage deduplication. A file tag is constructed to check if the file already exists, and if not, the ciphertext is uploaded. The data popularity is categorized to optimize security and efficiency. In addtion, subsequent uploaders must execute ADMHF-PoW. After security analysis and simulation experiments, the percentage of exposure significantly decreases compared to the traditional MHT scheme. In addition, for small files, it leads to an increase in the value of N_0, which is the number of verification attempts required for full exposure of MHT. Both of these indicators prove that our scheme improves the security of files. At the same time, for large files, the computation overhead is relatively low.

References

[1] Jianwei Yin, Yan Tang, Shuiguang Deng, Bangpeng Zheng, and Albert Y. Zomaya. Muse: A multi-tierd and sla-driven deduplication framework for cloud storage systems. *IEEE Transactions on Computers*, 2021.

[2] W. You and B. Chen. Proofs of ownership on encrypted cloud data via intel sgx. In *The First ACNS Workshop on Secure Cryptographic Implementation (SCI 20)(in conjunction with ACNS '20)*, 2020.

[3] J. R. Douceur, A. Adya, W. J. Bolosky, P. Simon, and M. Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, 2002.

[4] Chen Zhang, Yinbin Miao, Qingyuan Xie, Yu Guo, Hongwei Du, and Xiaohua Jia. Privacy-preserving deduplication of sensor compressed data in distributed fog computing. *IEEE Transactions on Parallel and Distributed Systems*, 33(12), 2022.

[5] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. Message-locked encryption and secure deduplication. 2013.

[6] C. Guo, X. Jiang, Kkr Choo, and Y. Jie. R-dedup: Secure client-side deduplication for encrypted data without involving a third-party entity. *Journal of Network and Computer Applications*, 162:102664, 2020.

[7] M. Miao, G. Tian, and W. Susilo. New proofs of ownership for efficient data deduplication in the adversarial conspiracy model. *International Journal of Intelligent Systems*, (3), 2021.

[8] Ralph C. Merkle. A certified digital signature. 1989.

[9] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg. Proofs of ownership in remote storage systems. *ACM*, page 491, 2011.

[10] Xue Yang, Rongxing Lu, Jun Shao, Xiaohu Tang, and Ali A. Ghorbani. Achieving efficient secure deduplication with user-defined access control in cloud. *IEEE Transactions on Dependable and Secure Computing*, 19(1), 2022.

[11] Yuan Zhang, Chunxiang Xu, Nan Cheng, and Xuemin Shen. Secure password-protected encryption key for deduplicated cloud storage systems. *IEEE Transactions on Dependable and Secure Computing*, 19(4), 2022.

[12] R. Chen, Y. Mu, G. Yang, and F. Guo. Bl-mle: Block-level message-locked encryption for secure large file deduplication. *IEEE Transactions on Information Forensics & Security*, 10(12):2643–2652, 2015.

[13] Mohammed Gharib and MohammadAmin Fazli. Secure cloud storage with anonymous deduplication using id-based key management. *J. Supercomput.*, 79(2):2356–2382, 2023.

[14] X. Jia, E. C. Chang, and J. Zhou. Weak leakage-resilient client-side deduplication of encrypted data in cloud storage. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, 2013.

[15] Spark: Secure pseudorandom key-based encryption for deduplicated storage. *Computer Communications*, 154:148–159, 2020.

[16] S. Li, C. Xu, and Y. Zhang. Csed: Client-side encrypted deduplication scheme based on proofs of ownership for cloud storage. *Journal of Information Security and Applications*, 46:250–258, 2019.

[17] W. A. Liang, A Bw, S. A. Wei, and B Zz. A key-sharing based secure deduplication scheme in cloud storage - sciencedirect. *Information Sciences*, 504:48–60, 2019.

[18] J. Stanek, A. Sorniotti, E. Androulaki, and L. Kencl. A secure data deduplication scheme for cloud storage. In *Financial Cryptography*, 2014.

[19] Y. Fan, X. Lin, W. Liang, G. Tan, and P. Nanda. A secure privacy preserving deduplication scheme for cloud computing. *Future Generation Computer Systems*, 101, 2019.

[20] Garima Verma. Secure client-side deduplication scheme for cloud with dual trusted execution environment. *IETE Journal of Research*, 0(0):1–11, 2022.

[21] Yang Ming, Chenhao Wang, Hang Liu, Yi Zhao, Jie Feng, Ning Zhang, and Weisong Shi. Blockchain-enabled efficient dynamic cross-domain deduplication in edge computing. *IEEE Internet Things J.*, 9(17):15639–15656, 2022.

[22] Xiaoyu Zheng, Yuyang Zhou, Yalan Ye, and Fagen Li. A cloud data deduplication scheme based on certificateless proxy re-encryption. *J. Syst. Archit.*, 102, 2020.

[23] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. Association for Computing Machinery, 2007.

[24] Guangquan Xu, Longlong Rao, and Jiangang Shi. Novel android malware detection method based on multi-dimensional hybrid features extraction and analysis. *Intelligent Automation and Soft Computing*, 25:–1, 09 2019.

[25] Shuai Wang, Yuyan Bao, Xiao Liu, Pei Wang, Danfeng Zhang, and Dinghao Wu. Identifying cache-based side channels through secret-augmented abstract interpretation, 05 2019.

[26] P. Puzio, R. Molva, Melek Nen, and S. Loureiro. Cloudedup: Secure deduplication with encrypted data for cloud storage. In *IEEE International Conference on Cloud Computing Technology & Science*, 2013.

[27] Vasily Tarasov, Amar Mudrankit, Will Buik, Philip Shilane, Geoff Kuenning, and Erez Zadok. Generating realistic datasets for deduplication analysis. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USA, 2012. USENIX Association.