

Article

SDAP-K: A Kerberos-Assisted Secure Data Auditing Protocol for Cloud Storage

Thangavel Murugan ^{1,*}, Nasurudeen Ahamed Noor Mohamed Badusha ¹, Priyan Malarvizhi Kumar ¹ and Varalakshmi Perumal ²

¹ College of Information Technology, United Arab Emirates University, Al Ain P.O. Box 15551, United Arab Emirates; nasurudeenahamed@uaeu.ac.ae (N.A.N.M.B.); priyanit085@gmail.com (P.M.K.)

² Department of Information Science and Technology, College of Engineering, Anna University, Chennai 60025, Tamilnadu, India; varanip@gmail.com

* Correspondence: thangavelm@uaeu.ac.ae; Tel.: +971-505753257

Abstract

Cloud storage services have become a fundamental component of modern computing infrastructures, enabling scalable and cost-effective data management. However, outsourcing data to remote cloud servers introduces significant security challenges, particularly in ensuring data integrity, secure access control, and efficient auditing of stored information. Existing cloud auditing schemes primarily focus on integrity verification and often rely on trusted third-party auditors, leading to additional trust assumptions, communication overhead, and metadata management complexity. To address these limitations, this research presents a Kerberos-Assisted Secure Data Auditing Protocol (SDAP-K) that integrates authenticated service exchange with lightweight integrity verification for outsourced cloud storage. The proposed framework employs Kerberos-based mutual authentication and ticket-driven access control to establish secure communication among the Data Owner, Authentication Server, Metadata Server, and Cloud Data Server. To verify storage correctness, an N-ary hash tree with the Modified Murmur hash algorithm is used to enable efficient file- and block-level auditing without requiring a trusted third-party auditor. The framework further incorporates metadata-assisted auditing, dynamic data operations, and an error localization and recovery mechanism that identifies and restores corrupted data blocks. Security analysis demonstrates that the proposed protocol mitigates unauthorized access, replay attacks, impersonation attempts, and malicious data modification. Experimental results indicate that SDAP-K reduces storage execution time by 18.6%, retrieval time by 24.3%, update time by 21.8%, file-level auditing overhead by 31.5%, and block-level auditing latency by 36.2% compared with state-of-the-art research, while eliminating the need for a trusted third-party auditor. The results indicate that the proposed framework offers a practical, lightweight, and reliable solution for secure cloud data auditing in enterprise cloud storage environments.

Keywords: cloud computing security; secure data auditing; Kerberos authentication; N-ary hash tree; integrity verification; error localization; data recovery

Academic Editor: Paolo Bellavista

Received: 17 June 2026

Revised: 23 July 2026

Accepted: 29 July 2026

Published: 3 August 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Recent trends in information technology and Internet-enabled devices have made cloud computing a primary concern for data storage and computation in various domains

such as finance and banking, weather monitoring, e-learning, e-governance, and e-health care services. Cloud computing provides access to a shared pool of configurable resources, such as network elements, storage, applications, and services, on demand. Risks such as data leakage, unauthorized access, illegitimate verification requests, delayed data access, malicious attacks or modifications to data, and improper data updates pose data security threats in the cloud. These risks lead to cloud data security breaches of real-time applications.

The widespread adoption of cloud storage services has enabled organizations and individuals to outsource data storage and management to Cloud Service Providers (CSPs), benefiting from scalability, flexibility, and reduced infrastructure costs. However, outsourcing data to remote cloud servers introduces significant security challenges, particularly in ensuring that stored data remains intact, accessible only to authorized entities, and resistant to unauthorized modification [1,2]. As cloud users no longer maintain direct physical control over outsourced data, efficient mechanisms for verifying storage correctness and maintaining secure access to cloud resources have become essential requirements in cloud security research [3,4].

Authentication and integrity verification serve distinct but complementary security objectives in cloud environments. Authentication mechanisms ensure that only authorized users can access cloud services and initiate storage, retrieval, or auditing requests. In contrast, integrity auditing mechanisms verify the correctness and consistency of outsourced data and detect unauthorized modifications. Kerberos is a well-established ticket-based authentication framework that provides mutual authentication, secure session establishment, and controlled service access without repeatedly transmitting user credentials across the network [5,6]. However, Kerberos alone does not provide storage correctness verification or integrity auditing capabilities. Consequently, there is a need for an integrated framework that combines authenticated service access with efficient integrity verification while maintaining low communication and computation overhead.

To address data integrity concerns, numerous cloud auditing schemes have been proposed, including provable data possession, remote data integrity verification, authenticated data structures, and Merkle hash tree-based auditing techniques [7,8]. These approaches enable users to verify the correctness of outsourced data without retrieving the entire dataset. Despite significant advances, several challenges remain. Many existing solutions rely on trusted third-party auditors (TPAs), which introduce additional trust assumptions and operational complexity. Other approaches incur substantial metadata management overhead, communication costs, or computational burdens during auditing operations. Furthermore, while secure authentication is critical for protecting cloud services against unauthorized access, authentication mechanisms are often treated separately from integrity verification processes, resulting in limited integration between secure service access and cloud auditing operations [9–11].

Recent research trends have explored advanced cloud auditing paradigms, including privacy-preserving auditing, certificateless auditing, decentralized trust models, blockchain-assisted verification, and scalable cloud-native security architectures [11,12]. While these approaches improve specific aspects of cloud security, many introduce additional deployment complexity or computational overhead. In practical enterprise cloud environments, lightweight and centrally managed architectures remain attractive because they simplify deployment, support scalable metadata management, and facilitate secure operational control. Motivated by these requirements, this work focuses on developing a practical authenticated cloud auditing framework that integrates secure service exchange and lightweight integrity verification within a unified architecture.

In 2017, UpGuard reported that nearly 200 million voters (about 1.1 TB of data) were exposed after a server hosted by the Republican National Committee on AWS was

breached. Later that year, ESEA suffered a data breach that exposed the names, login credentials, and payment information of hundreds of thousands of gamers. In 2016, there was a similar leak of approximately 167 million LinkedIn users' credentials, and the US Treasury reported that approximately 700,000 individuals [13] had compromised tax records and other personal information, including Seagate employees' records leaked through phishing attacks. In 2015, the LastPass password management service was the target of a cyberattack that exposed users' emails, passwords, salts, and hashes. These attacks illustrate how security researchers identify serious risks in cloud computing through large-scale data breaches arising from cloud operations [14]. Several major data breaches occurred between 2016 and 2017, during which large amounts of user information [15] were leaked.

Various technical solutions have been proposed by researchers to address those issues, including authenticated data structures, signature schemes, challenge-response protocols, and error-correcting codes; however, an efficient and secure auditing and data storage solution for cloud computing remains elusive. In cloud auditing, the Data Owner (DO) can verify the integrity of data stored on the Cloud Storage Server (CSS) by issuing challenges that do not require the DO to access the data itself. If the DO identifies any integrity violations, the DO will replace the corrupted blocks of the Data Owner's data. Data blocks that do not require verification of their integrity have been verified using data block signatures, which greatly reduce both communication requirements and bandwidth usage. The scheme introduces challenges for maintaining metadata and authentication in a cloud environment.

Therefore, this research proposes a new protocol that uses the Kerberos authentication protocol to audit data securely and is designed to overcome those challenges. In this approach, the data owners and users connect to the Cloud Service over the Internet. The Cloud Service does not permit unauthorized access to its data. It therefore issues service requests based on the authenticated identity of the user in question (e.g., storage, retrieval, or auditing). User machines alone are not sufficiently trusted to verify the identity of the user attempting to access a cloud service; therefore, Kerberos provides a mechanism for authenticating a data owner or user, as applicable, thereby allowing the Cloud Service to provide services to an authenticated user. If it so, the following threats exist: (i) any user may gain access to client machines and try to pretend to be another user from the same client machine, (ii) user can alter the network address of the client machine and impersonate like another user, and (iii) user may eavesdrop the communications and perform replay attacks against cloud servers to disrupt the service or operations. In all three threats, the user may gain unauthorized access to data or services. To address these authentication issues in the cloud, a centralized authentication server has been deployed in SDAP-K to authenticate users/owners to the cloud server and vice versa. While performing an audit, information about pre-stored cloud data is necessary. But handling metadata by data owners seems to be overhead. To manage and accumulate metadata for cloud services used by users/owners, a centralized metadata server has been deployed in SDAP-K. So, cloud server services like storage, retrieval, and auditing have been implemented using the Kerberos authentication protocol in the background.

To address the identified challenges, this paper presents a Secure Data Auditing Protocol (SDAP-K) that combines Kerberos-based authenticated service exchange with N-ary hash tree and Modified Murmur hashing algorithm-based integrity auditing. In the proposed architecture, Kerberos is employed to provide mutual authentication and secure access control among the Data Owner (DO), Authentication Server (AS), Metadata Server (MS), and Cloud Data Server (CDS). Separately, the N-ary hash tree structure with the Modified Murmur hashing algorithm is utilized to support lightweight file-level and block-level integrity auditing. A dedicated metadata management mechanism is

incorporated to maintain auditing information. At the same time, error localization and recovery procedures enable the identification and correction of corrupted data blocks during storage, retrieval, and auditing operations. By integrating these components, the proposed framework aims to reduce communication and computation overhead while supporting secure and efficient cloud data auditing.

The novelty of SDAP-K lies in its protocol-level integration into a unified cloud data auditing framework. SDAP-K integrates operation-bound Kerberos service tickets, metadata-assisted integrity verification, N-ary hash-tree-based auditing, the Modified Murmur hashing algorithm, and integrated error localization and recovery into a single authenticated workflow. Unlike existing cloud auditing schemes that primarily focus on either authentication or integrity verification, SDAP-K handles authentication, metadata management, auditing, and recovery, enabling secure, efficient, and scalable cloud data auditing without a trusted third-party auditor.

Although many cloud auditing and authentication schemes have been proposed, some existing methods treat data integrity verification and secure authentication as two independent functions, leading to high communication overhead, increased metadata management complexity, or reliance on trusted third-party auditors. These limitations underscore the need for an integrated framework that combines authenticated service access with efficient integrity auditing. Section 2 presents a detailed review of related cloud auditing and authentication techniques. The main contributions of this work are summarized as follows:

- (i) A Kerberos-assisted authenticated cloud auditing architecture is proposed to secure service interactions among data owners, metadata servers, and cloud storage servers through mutual authentication and ticket-based access control.
- (ii) An N-ary hash tree-based integrity verification mechanism with the Modified Murmur hashing algorithm is introduced to support lightweight file-level and block-level auditing of outsourced cloud data without requiring a trusted third-party auditor.
- (iii) A metadata-assisted auditing framework is developed to facilitate efficient management of auditing information and reduce the burden on data owners during integrity verification operations.
- (iv) An error localization and data recovery mechanism is incorporated to identify corrupted blocks and support data correctness validation during storage, retrieval, and auditing processes.
- (v) A reproducible experimental evaluation framework to measure the computation, communication, and auditing overhead of SDAP-K over different file sizes, block sizes, and N-ary tree branching factors.

The remainder of this paper is organized as follows. Section 2 reviews related work on cloud auditing and authentication mechanisms. Section 3 presents the system model, threat assumptions, and design goals. Section 4 introduces the preliminaries of the proposed framework. Section 5 describes the proposed SDAP-K architecture and operational protocols. Section 6 presents the security analysis, while Section 7 evaluates the performance of the proposed scheme. Finally, Section 8 concludes the paper and outlines future research directions.

2. Literature Review

2.1. Cloud Data Auditing and Integrity Verification

Auditing of cloud data is an important research area for ensuring the correctness of outsourced data without downloading the entire data. Existing approaches include provable data possession (PDP) and proof-of-retrievability (POR) [7,16]. The homomorphic property-based approach relies on proof-based block authentication using Dynamic

Provable Data Possession (DPDP) schemes, which incur high communication and computational costs. It has been defined using a Merkle Hash Tree [17], in which the leaf nodes are the hash values of the blocks to verify data integrity. Merkle hash tree-based verification, identity-based auditing, and certificateless auditing schemes [8,18]. Computation cost increases with each block update, since the tag is updated each time [19]. These techniques help data owners verify data integrity at lower communication costs between cloud users and cloud service providers [11,12,20].

Approaches based on Merkle hash trees remain among the most widely adopted solutions for cloud auditing because they enable efficient integrity verification through hierarchical hash structures. A public auditing framework for data dynamics based on Merkle hash trees was proposed by Wang et al. [7]. Still, frequent data modifications might increase computational overhead for metadata updates and tag regeneration. Similarly, Li et al. [8] proposed a privacy-preserving public auditing protocol for resource-constrained devices that reduced the computational complexity. Still, it required a third-party auditor to perform the verification. To reduce the overhead of certificate [21] management while supporting secure integrity verification in a cloud environment, identity-based and certificateless [22,23] auditing schemes have subsequently been proposed [24,25].

Recently, several works have further enhanced integrity auditing [26,27] by supporting provable updates, data deletion verification, and dynamic cloud storage operations [16]. While these approaches have enhanced the validation of storage correctness, many still face issues with metadata synchronization, communication overhead, and the effective handling of dynamic data operations. In addition, techniques for error localization and data recovery are often treated as secondary goals rather than as integral components of the auditing process [11,28].

2.2. Privacy-Preserving and Lightweight Auditing Approaches

Privacy preservation and computational efficiency are the main design goals in cloud auditing research. Several works have proposed lightweight auditing mechanisms to reduce communication and storage costs while preserving user privacy [8,29]. Zhang et al. [30] improved secure auditing in fog-to-cloud computing environments by tackling integrity verification issues in distributed infrastructures. Wang et al. [7] proposed an auditing framework based on identity-based signatures and provable data possession to enable privacy-preserving verification and reduce bandwidth consumption. Similarly, secure user revocation, dynamic data support, and efficient auditing operations have been integrated into privacy-aware auditing frameworks for cloud-based Internet of Things (IoT) environments [31].

Although these approaches reduce auditing overhead and improve privacy protection, many rely on additional verification entities or complex cryptographic operations, which may affect scalability in large-scale cloud deployments. Moreover, metadata management and authenticated access control are often considered independently from the auditing workflow.

2.3. Authentication-Based Cloud Security Approaches

Integrity verification and authentication are cloud computing security requirements. Integrity auditing ensures data is stored properly, and authentication mechanisms ensure users are who they say they are and can manage their access to cloud services [5,6].

Researchers' contributions to data auditing [6,32] and to authentication issues in the cloud focus primarily on public auditing and also recommend private auditing. Based on the analysis of related works, the findings for enhancing the auditing protocol are as follows: (i) Authentication and Metadata management have been outsourced to a semi-trusted cloud service provider, which may lead to overhead and data breaches. (ii) In

private data auditing, the data owner needs to store, retrieve, and audit the data with a cloud server, which may increase computation and communication costs. (iii) Storage and auditing have been performed with file blocks, which may provide directions for the attacker to perform attacks on a cloud server. (iv) Error localization and data correctness operations will ensure the data integrity property of the cloud server, which is not addressed in many research works. The findings from related work indicate the need to enhance secure auditing protocols by incorporating the Kerberos authentication scheme, which will be discussed in the following sections. Very few studies have proposed Kerberos-based authentication schemes to improve secure communications and access control in cloud environments [10,15].

Kerberos offers mutual authentication, ticket-based access control, and secure establishment of sessions between users and cloud services without the repeated transmission of credentials over the network. Tiwari and Neogi [5] showed the application of Kerberos in hybrid cloud environments for secure resource access and multi-tenant operations. For secure communication and authentication, other studies [6,10] used Kerberos. Similarly, Shajina and Varalakshmi [15] proposed a dual authentication framework for multi-owner cloud environments. These techniques are helpful for authentication, but they do not provide direct methods for verifying data integrity or validating storage correctness. Thus, authentication-centric frameworks and auditing-centric frameworks have evolved independently. This creates a gap for integrated frameworks that ensure authenticated access to services while enabling efficient auditing mechanisms.

2.4. Metadata Management, Dynamic Operations, and Error Localization

In cloud auditing, metadata is important because integrity verification typically relies on auxiliary information such as block identifiers, signatures, hash values, and audit parameters. In several works [16,33], mechanisms are proposed to support dynamic updates, provable deletion, and efficient metadata maintenance [7]. These approaches enable more flexible auditing systems, but they may incur additional storage and synchronization overheads [11].

Another important challenge is error localization and data recovery. While many auditing schemes can detect integrity violations, relatively few provide mechanisms to identify corrupted blocks and restore data correctness. Existing approaches generally focus on integrity verification results rather than supporting practical recovery operations following audit failures. Consequently, integrating metadata management with error localization remains an important research direction for practical cloud auditing deployments.

2.5. Recent Trends in Cloud Auditing and Security

Recently, cloud security research has progressed from traditional auditing models to privacy-preserving verification, decentralized trust models, certificateless auditing [34,35], multi-copy auditing, and geographically distributed cloud verification [11,12,36]. These approaches aim to improve scalability, preserve privacy, and support dynamic cloud environments.

In addition, cloud security architectures are increasingly constructed on zero-trust principles, continuous authentication, distributed identity management, and secure cloud-native deployments. Decentralized and blockchain-assisted approaches offer promising directions for trust management and audit transparency; they may also introduce additional deployment complexity and computational overhead. But the enterprise cloud environment often requires lightweight, centrally managed solutions that can be deployed with minimal operational complexity while still providing secure access control and efficient integrity verification.

2.6. Research Gap and Motivation

A review of the existing literature reveals some key observations. First, many cloud auditing schemes [7,8] focus more on integrity verification and rely on separate authentication mechanisms or trusted third-party auditors [9,13,14]. Second, authentication frameworks, such as Kerberos-based solutions, guarantee secure access to services but do not address the verification of storage correctness [10,15]. Third, many current auditing architectures do not adequately address metadata management, communication overhead, and error localization. Finally, practical frameworks that integrate authenticated service exchange, lightweight integrity verification, centralized metadata coordination, and error recovery capabilities remain limited. Table 1 presents a summary of related work.

Table 1. Comparison of existing cloud data auditing schemes.

Ref.	Authenticat- tion	Integrity Auditing	TPA Required	Dynamic Updates	Error Localiza- tion	Metadata Manage- ment	Public Auditing	Privacy Preserva- tion	Formal Verification
[5]	✓	×	×	N/A	×	×	×	×	×
[6]	✓	×	×	N/A	×	×	×	×	×
[7]	×	✓	✓	✓	×	Partial	✓	✓	×
[8]	×	✓	✓	Limited	×	Partial	✓	✓	×
[9]	✓	×	×	N/A	×	×	×	×	×
[10]	✓	×	×	N/A	×	×	×	×	×
[11]	×	✓	✓	✓	×	Partial	✓	×	×
[12]	×	✓	×	✓	×	Partial	✓	✓	×
[15]	✓	×	×	N/A	×	×	×	×	×
[16]	×	✓	×	✓	×	Partial	✓	×	×
[17]	×	✓	✓	✓	×	Partial	✓	×	×
[24]	×	✓	×	✓	×	Partial	✓	✓	×
[25]	×	✓	×	✓	×	Partial	✓	✓	×
[29]	×	✓	✓	Limited	×	Partial	✓	✓	×
[30]	×	✓	×	✓	×	Partial	✓	✓	×
[31]	Partial	✓	×	✓	×	Partial	✓	×	×
[33]	✓	×	×	N/A	×	×	×	×	×
[35]	×	✓	✓	✓	×	Partial	✓	×	×
SDAP- K	✓	✓	×	✓	✓	✓	×	×	×

✓ Available, × Not Available, N/A Not Appear.

Motivated by these challenges, this work proposes an enhanced Secure Data Auditing Protocol (SDAP-K) that integrates Kerberos-based authentication with N-ary hash tree-based integrity auditing. The proposed framework combines authenticated service access, metadata-assisted auditing, file-level and block-level verification, and error localization within a unified architecture designed to reduce communication and computation overhead in cloud environments.

3. Problem Statement

The system model, threats, and goals for designing the proposed system are discussed in this section.

3.1. System Model

The proposed Secure Data Auditing Protocol (SDAP-K) is designed for outsourced cloud storage environments in which data owners store and manage data through cloud service providers. The system consists of four entities: Data Owner (DO), Authentication Server (AS), Metadata Server (MS), and Cloud Data Server (CDS), as shown in Figure 1.

- Data Owner (DO): The person who owns the data and performs storage, retrieval, update, and auditing operations on outsourced cloud data.
- Authentication Server (AS): A trusted authentication entity that authenticates identities, issues tickets, and establishes sessions between authorized users and cloud services.
- Metadata Server (MS): A metadata management entity that oversees keeping the auditing information, such as block identifiers, storage block ordering, integrity verification parameters, and audit-related metadata.
- Cloud Data Server (CDS): The cloud storage provider that stores the outsourced data and processes the storage, retrieval, update, and auditing requests.

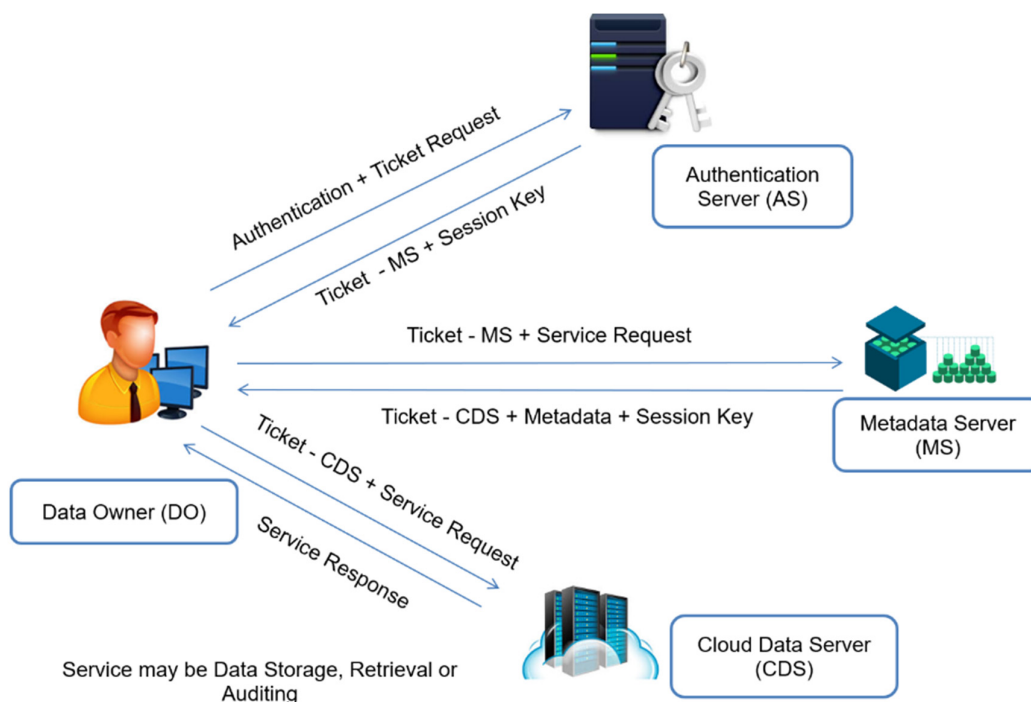


Figure 1. System architecture of the proposed SDAP-K framework.

The proposed architecture separates authentication from integrity verification. Kerberos-based authentication provides mutual authentication, secure ticket exchange, and controlled access to the services of the protocol entities. N-ary hash tree-based auditing with Modified Murmur Hashing mechanisms is used to independently audit the integrity and verify the correctness of outsourced data stored in the CDS.

Trust Assumptions

The following assumptions are considered throughout this work:

- AS and MS are assumed to be semi-trusted entities that correctly execute authentication and metadata management operations but may become targets of compromise attempts.

- CDS is considered honest-but-curious and may attempt to hide data corruption, delete infrequently accessed data, or provide incomplete responses during auditing.
- Data Owners securely maintain their private credentials and cryptographic keys.
- Standard cryptographic primitives are assumed to provide their intended security properties.
- Secure channels are available for initial key distribution among trusted entities.

The proposed architecture can be deployed in centralized enterprise environments. At the same time, metadata replication and distributed deployment strategies can be adopted to improve scalability and fault tolerance in large-scale cloud infrastructures.

The Metadata Server (MS) is treated as a semi-trusted entity responsible for maintaining auditing metadata, but it cannot access the plaintext contents of the outsourced data. The metadata record comprises the file identifier, version number, number of data blocks, N-ary tree branching factor, storage block ordering, set of block hash values, root hash, timestamp, and the digital signature (or message authentication code) of the Data Owner. Before storing the metadata record, the Data Owner signs it to guarantee its integrity and authenticity. Rollback protection is provided by version numbers and timestamps that prevent the use of stale metadata during auditing. After an insert, delete, or modify operation, the metadata record is updated and re-signed before being stored on the Metadata Server. If a Metadata Server is compromised, metadata integrity can be verified by the owner's signature and metadata replication; audit logging, version control, periodic consistency checks, and failover mechanisms can mitigate risks to availability and reliability. Therefore, while the Metadata Server simplifies metadata management, it introduces a trust dependency that is best suited for enterprise cloud environments with centrally managed security infrastructure.

$$MD = \{FID, \text{Version}, n, N, SBO, H, R, TS, \text{Sig}_{DO}(MD)\}$$

where:

- FID: File identifier;
- Version: Metadata version number;
- n: Number of data blocks;
- N: N-ary tree branching factor;
- SBO: Storage Block Ordering;
- H: Set of block hash values;
- R: Root hash;
- TS: Timestamp;
- $\text{Sig}_{DO}(MD)$: Data Owner's digital signature (or MAC) over the metadata.

3.2. Threat Model

The threat model considers three categories of adversaries.

3.2.1. External Adversary

An external adversary can intercept, eavesdrop on, replay, or modify messages exchanged between protocol entities. The adversary may also attempt impersonation attacks to gain unauthorized access to cloud services. However, the adversary is assumed to be unable to break standard cryptographic primitives such as AES and RSA in polynomial time.

3.2.2. Malicious Cloud Server

The CDS may attempt to:

- Modify stored data blocks;

- Delete infrequently accessed files;
- Conceal storage failures;
- Return incomplete audit responses.

The CDS is not assumed to possess the decryption keys for outsourced data and therefore cannot recover the plaintext contents.

3.2.3. Insider Adversary

Authorized users with legitimate credentials may attempt to misuse access privileges or manipulate service requests. Such adversaries may attempt to gain unauthorized access to metadata or to audit operations beyond their authorized scope.

3.2.4. Attack Surfaces

Based on the above adversarial capabilities, the proposed framework addresses the following threats:

- Unauthorized service access;
- Replay attacks;
- Man-in-the-middle attacks;
- Forgery attempts;
- Data modification attacks;
- Data deletion attacks;
- Metadata misuse attacks.

3.3. Research Problem and Design Challenges

There are many proposed cloud auditing schemes, but some challenges remain unresolved.

- Initially, many integrity verification frameworks depend on trusted third-party auditors, adding additional trust assumptions and management complexity.
- Second, metadata maintenance could be a cumbersome task for data owners, especially with large datasets and dynamic updates.
- Third, authentication mechanisms and integrity auditing mechanisms are usually designed independently, which leads to fragmented security architectures.
- Finally, most of the existing auditing schemes focus on integrity verification but provide limited support for error localization and data recovery.

Hence, the research problem formulated in this work is as follows:

To design a lightweight authenticated cloud auditing framework for secure service access, efficient integrity verification, metadata-assisted auditing, and error localization with minimal communication and computational overhead in an outsourced cloud storage environment.

To solve this problem, the proposed SDAP-K integrates Kerberos-based authenticated service exchange, N-ary hash tree-based integrity verification with the Modified Murmur Hashing algorithm, and centralized metadata management.

3.4. Security and Functional Goals

The SDAP-K design objectives are divided into 3 categories: security, functional, and performance.

3.4.1. Security Goals

- Authentication Service Access: To enable both parties to authenticate each other and to enable authenticated authorization for services to the cloud storage service.

- Confidentiality: Guard communications channels and outsourced data against unauthorized disclosure.
- Integrity checking: Enable the data owners to check the correctness of the outsourced data without downloading the whole data set.
- Replay Resistance: Protect against an attacker re-sending previously sent authentication or service messages.
- Prevent unauthorized access: Restrict cloud operations to authenticated and authorized entities.

3.4.2. Operational Goals

- File-Level Auditing: Support integrity checks of the complete outsourced files.
- Block-Level Auditing: Allows you to verify only the data blocks that are in dispute.
- Dynamic Data Operations: Support for outsourced data storage, retrieval, and update operations.
- Metadata Management: Centralized storage and distribution of auditing metadata.
- Error Localization and Recovery: Detect corrupted data blocks and enable recovery of data correctness after audit failures.

3.4.3. Performance Objectives

- Low Communication Overhead: Reduce communication costs for storage, retrieval, and auditing operations.
- Lightweight Computation: Alleviate the computational load of data owners and cloud servers.
- Scalable Metadata Management: Provide efficient metadata management for large-scale cloud storage environments.
- Practical Deployment: Make deployment possible in enterprise and cloud service environments, with centralized authentication and metadata coordination.

4. Preliminaries

In this section, we discuss the cryptographic and system foundations of the proposed Secure Data Auditing Protocol (SDAP-K). The framework integrates service exchange and integrity verification through Kerberos-based authentication and N-ary hash tree-based auditing. Authentication and integrity checking are considered separate security functions. Kerberos enables authenticated communication and secure access to services. Hash-tree auditing mechanisms ensure integrity.

4.1. Cryptographic Foundations

The proposed framework employs standard cryptographic primitives to support secure communication, authenticated access, and integrity verification.

4.1.1. Symmetric Encryption

The session keys generated during the Kerberos authentication process are used to establish authenticated communication between protocol entities. Exchanged tickets, authenticators, and service responses are secure using symmetric encryption.

4.1.2. Public Key Cryptography:

Public-key cryptography is used for identity verification and secure key management. Let (PBi) denote the public key of entity (i) , and (PRi) denote the corresponding private key. Public-key operations are employed during registration and identity verification procedures, while symmetric session keys are used for subsequent communication.

4.1.3. Hash Function

In 2018, Austin Appleby [23] proposed Murmur hash for general hash-based computation. The two basic operations used in Murmur hash are multiply and rotate. Presently, Murmur Hash 3 produces a 32-bit hash value for variable inputs. To increase randomness and unbreakability, Modified Murmur hash has been proposed by altering the rotation operation with a bitwise OR operation. Algorithm 1 shows the Modified Murmur hashing.

Algorithm 1: Modified Murmur Hashing Technique

```

1.  procedure ModifiedMurmur_32(k, l, s)
2.    Input: k-key, l-length of the input, s-seed
3.    Output: h-hash
4.    x1, x2, y1, y2, a, b—random variables
5.    fourByteChunk—input value in four bytes in size
6.    rB—remaining bytes in the input
7.    x1 ← 0xcc9e2d51 & x2 ← 0x1b873593
8.    y1 ← 15 & y2 ← 13
9.    a ← 5, b ← 0xe6546b64, h ← s
10.   for each fourByteChunk of k
11.     m ← fourByteChunk
12.     m ← m × x1
13.     m ← (m << y1)OR(m >> (32 − y1))
14.     m ← m × x2
15.     h ← h XOR m
16.     h ← (m << y2)OR(m >> (32 − y2))
17.     h ← h × a + b
18.   end for
19.   with any remainingBytesInKey
20.     rB ← SwapToLittleEndian(remainingBytesInKey)
21.     rB ← rB × x1
22.     rB ← (rB << y1)OR(rB >> (16 − y1))
23.     rB ← rB × x2
24.     h ← h XOR rB
25.   end with
26.   h ← h XOR l
27.   h ← h XOR (h >> 16)
28.   h ← h × 0x85ebca6b
29.   h ← h XOR (h >> 13)
30.   h ← h × 0xc2b2ae35
31.   h ← h XOR (h >> 16)
32. end procedure

```

Adam [23] proved that the Murmur hashing technique computes faster than MD5, SHA1, and SHA256. Since the OR operation simplifies the rotation operation, the computation of the Modified Murmur hashing technique will be faster. In SDAP-K, DO computes hash values for every block and N-ary hash tree leaf nodes; intermediate nodes are hashed using the Modified Murmur hashing technique.

The proposed SDAP-K framework uses the Modified Murmur cryptographic hash function to generate fixed-length message digests for integrity verification. We selected the 32-bit Modified Murmur hash function (MMH-32) because it offers good collision resistance, preimage resistance, and second-preimage resistance, making it suitable for

secure cloud auditing. We assume in this work that the hash function is computationally collision-resistant, i.e., it is computationally infeasible to find two different inputs that hash to the same value.

$$\text{Let, } h: \{0,1\}^* \rightarrow \{0,1\}^{32}$$

where:

- $h(\cdot)$ is a cryptographic function;
- $\{0,1\}^*$ denotes an arbitrary-length input;
- $\{0,1\}^{32}$ denotes a fixed-length output (32 bits for MMH-32).

For every data block m_i , the hash value is computed as

$$H_i = h(m_i)$$

where:

- m_i is the i th data block;
- H_i is the corresponding block digest.

The set of block hashes is $H = \{H_1, H_2, \dots, H_n\}$.

The root hash is $R = \text{NHT}(H)$.

4.2. N-Ary Hash Tree Construction

The N-ary hash tree is the main data structure for integrity verification in SDAP-K. It enables efficient verification of outsourced data without requiring the entire file.

For a data file (DF) divided into n blocks

$$\text{DF} = \{m_1, m_2, \dots, m_n\}.$$

Each block is hashed independently using MMH-32

$$H_i = h(m_i), 1 \leq i \leq n.$$

The resulting hash values form the leaf nodes of the N-ary hash tree.

Intermediate nodes are generated by hashing the concatenation of child node values.

For a parent node (P_i),

$$P_i = h(H_{j1} \parallel H_{j2} \parallel \dots \parallel H_{jk})$$

where:

- $\parallel$ denotes concatenation;
- $k \leq N$ is the number of child nodes.

The root node is computed recursively.

The root hash value is

$$R = h(P_1 \parallel P_2 \parallel \dots \parallel P_t)$$

where:

- R is the root digest;
- t is the number of parent nodes at the previous level.

The audit verifies integrity by comparing the calculated root value with the original value stored in the metadata. Any difference indicates that the data has been altered or corrupted without authorization. Figure 2 provides an overview of the N-ary hash tree.

The data file has initially been divided into N blocks. Let us assume $N = 4$. Using a hashing technique, the contents of each block will be hashed, yielding hash values such as $H(m_1)$, $H(m_2)$, and so on. These hash values will be stored in leaf nodes of the N-ary hash tree. The parent nodes are derived by computing the hash values of the concatenated hash values of leaf nodes. In Figure 2, $H(I)$ holds the hash value of the concatenated hash

values of $H(m_1)$, $H(m_2)$, $H(m_3)$, and $H(m_4)$. Correspondingly, the root value $H(R)$ of the tree has been computed. Suppose DO wants to audit the data file and places the challenge to CDS for the blocks $\{m_3, m_{14}\}$. CDS responds with the hash value of $\{m_3, m_{14}\}$. DO uses metadata to regain authentic information. $\{H(m_1), H(m_2), H(m_4), H(m_{13}), H(m_{15}), H(m_{16}), H(II), H(III)\}$. DO verifies by computing $H(I) = \{H(m_1) || H(m_2) || H(m_3) || H(m_4)\}$, $H(IV) = \{H(m_{13}) || H(m_{14}) || H(m_{15}) || H(m_{16})\}$, and $H(R) = \{H(I) || H(II) || H(III) || H(IV)\}$. The resulting $H(R)$ needs to match the metadata.

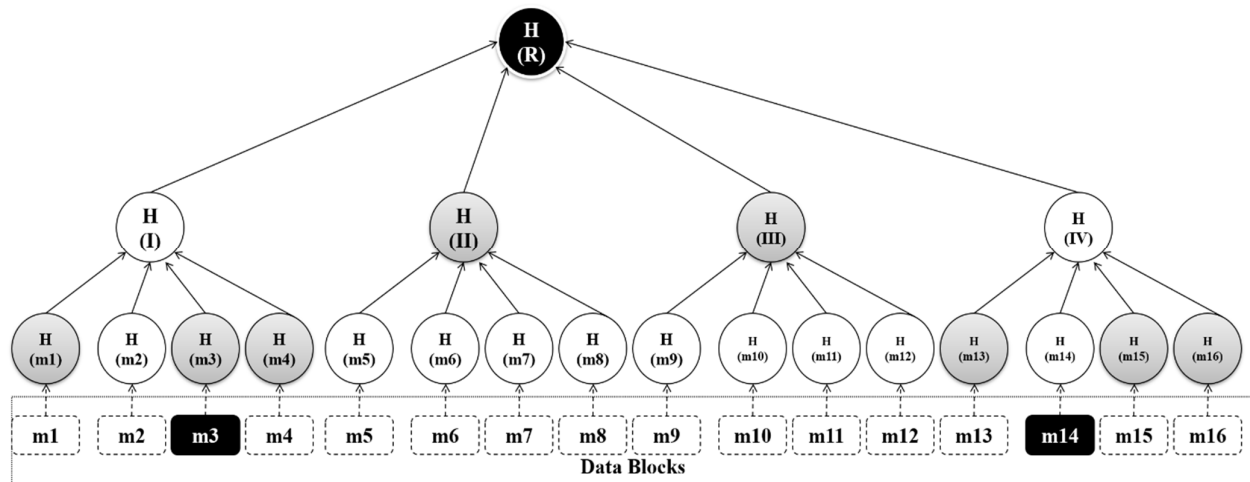


Figure 2. N-ary hash-tree structure for data integrity verification.

Complexity Analysis

The computational complexity of major N-ary hash tree operations is in Table 2.

Table 2. Computational complexity of N-ary hash-tree operations.

Operation	Complexity
Tree Construction	$O(n)$
File—Level Auditing	$O(n)$
Block—Level Auditing	$O(\log n)$
Root Verification	$O(1)$

An N-ary structure has shallower depth, reducing traversal overhead compared to traditional binary trees and thus improving scalability for large cloud datasets.

4.3. Kerberos-Based Authenticated Service Exchange

In SDAP-K, Kerberos is used to authenticate protocol entities to one another, authorize them with tickets, and establish secure sessions. The Kerberos framework is composed of:

- Authentication Server (AS);
- Metadata Server (MS);
- Cloud Data Server (CDS);
- Data Owner (DO);

The authentication is carried out through a series of ticket exchanges.

- DO authenticates with the AS;
- AS sends MS a service ticket;
- MS authenticates the ticket and provides a service ticket for CDS;
- CDS provides services such as storage, retrieval, updating, and auditing.

The generated session keys are used to provide confidentiality between protocol entities. It is important to note that Kerberos is used to secure authenticated communication and access to services between protocol entities, not to provide integrity verification itself. Integrity verification is achieved independently by the N-ary hash tree auditing mechanism.

Message Complexity

The communication overhead associated with authenticated service exchange is summarized in Table 3.

Table 3. Message complexity of Kerberos service exchanges.

Exchange	Messages
DO-AS	2
DO-MS	2
DO-CDS	2

The ticket-based design minimizes credential transmission and scales authenticated interactions across cloud operations.

4.4. Integrity Verification Model

The integrity verification process is based on a challenge-response auditing model. For block-level auditing, the challenge set is as follows:

Let

$$C = \{m_{i1}, m_{i2}, \dots, m_{ik}\}, C \subset DF.$$

The Cloud Data Server computes as follows:

$$\text{Let } Hc = \{h(m_{i1}), h(m_{i2}), \dots, h(m_{ik})\}.$$

Using the challenge response and metadata MD , the Data Owner computes as follows:

$$\text{Let } R' = \text{ComputeRoot}(Hc, MD).$$

The integrity verification decision is as follows:

Let

$$IV = \begin{cases} 1, & R' = R \\ 0, & R' \neq R \end{cases}$$

For file-level auditing, let

$$C = DF.$$

For block-level auditing, let

$$C \subset DF, |C| = k, k < n \text{ if } IV = 0.$$

Then, error localization is initiated.

4.5. Formal Protocol Definition

This section formalizes the SDAP-K protocol. It defines the participating entities, cryptographic keys, service tickets, metadata records, audit messages, and verification rules. The protocol consists of six phases: authenticated data storage, secure data retrieval, dynamic data update, file-level auditing, block-level auditing, and error localization with recovery. Each phase defines the required inputs and outputs, protocol operations, and

verification conditions for secure and efficient cloud data auditing. Table 4 shows the formal definition of the Proposed Framework.

Table 4. Formal definition of the SDAP-K protocol.

Component	Definition
Entities	Data Owner (DO), Authentication Server (AS), Metadata Server (MS), Cloud Data Server (CDS)
Keys	Long-term secret keys, Kerberos session keys, encryption keys
Service Tickets	Kerberos tickets authorizing specific cloud operations
Metadata Record	$(MD = \{FID, Version, n, N, SBO, H, R, TS, Sig_{\{DO\}}(MD)\})$
Audit Challenge	$(Chal_B = \{FID, I, Nonce, TS\})$
Audit Response	$(Proof_B = \{Tag(m_i), AuthPath_i, Nonce, TS\})$
Verification Rule	Accept only if ticket, nonce, timestamp, authentication path, and root hash are valid

4.6. Notations

The notations of the proposed SDAP-K scheme are as follows,

(PB, PR)	Public key and private key pairs
$ $	Concatenation
$\lceil \cdot \rceil$	Ceiling function
ACL	Access Control List
ADc	Prevents access from a workstation initiated to another workstation
AS	Authentication Server
ATc	Generated by DO to validate TK
$AuthPath_i$	N-ary authentication path
B	Set of block names corresponding to M
CDS	Cloud Data Server
D	Depth (height) of the N-ary hash tree
DF/F	Data File/Filename
DO	Data Owner.
E	Error block set
e	Error blocks hashed block name
EDF	Encrypted Data File
$EK(m)$	Symmetric Encryption
$E_K()$	Symmetric encryption function with K
$EK(\cdot)$	Symmetric encryption using key K
FA/BA	File-level/Block-level Auditing
FID	Unique file identifier.
fs/bs	File size/block size
$H = \{H1, H2, \dots, Hn\}$	Set of block hash values
$h(\cdot)$	Collision-resistant cryptographic hash function
$h(B')$	Set of hash block names used for update
$h(B)$	Set of the hashed block names
$h(F)$	Hashed file name
$h(m)$	MMH-32 hash function
$h(M')/h(M')$	Set of computed hash values of data blocks
$h(M)$	Set of hashed data blocks
$h(M_e)$	Hash values of M_e
$h(R)$	Hashed root value of N-ary hash tree
$h(R')/h(RN)/h(RN')$	Computed hash root value
H'	Updated block hash set
Hc	Hash values of challenged blocks
$H_i = h(mi)$	Hash value of the i th data block
I	Challenge Index Set
$ID_{DO}/ID_{AS}/ID_{MS}/ID_{CDS}$	Identity of DO/AS/MS/CDS
IHV'	Computed Intermediate Hash Values

IHV	Intermediate Hash Values
K/n	Key file/number of blocks
k	Number of challenged blocks
K_{DO-AS}	Secret key—to enable and access communications of DO and AS
K_{DO-CDS}	Session key—to enable and access communications of DO and CDS
K_{DO-MS}	Session key—to enable and access communications of DO and MS
LT	Ticket Lifetime
m/C	Set of blocks to be challenged
$M = \{m_1, m_2, \dots, m_n\}$	Set of data blocks obtained by dividing EDF
M	Actual data blocks of DF
$MACk(m)$	Message authentication code
$MD = \{FID, n, N, SBO, H, R\}$	Metadata maintained by the Metadata Server
M_e	Data blocks corresponding to e
mi	The i th encrypted data block.
m_{ij}	Data block corresponding to challenge index i_j
MS	Metadata Server
N	Branching factor/N-ary hash tree—the number of child nodes
$Nonce/N_1/N_2$	Nonce—Random Value
$Nonce$	Random Nonce
Op	Operation type
$R = NHT(H)$	Root hash computed from the set of block hash values using the N-ary hash tree/authentic root hash stored in Metadata Server
R	Root hash generated by the N-ary hash tree
$R(e)$	Replicas of error blocks e
R'	Updated root hash/reconstructed root hash from challenge response
$S/R/U$	Storage/Retrieval/Update
SBO	Storage Block Ordering
$Scope$	Authorization Scope
$Sig_{DO}(MD_{new})$	Data Owner Signature (or MAC)
$Sig_{FDO}(m)$	Digital signature
SK/K	Symmetric key used for secure communication
$Tag(mi)$	Authentication Tag
$Times$	Ticket time settings [from, till, renew]
TK_{MS}/TK_{CDS}	Ticket used by DO to access MS/CDS
TS/TS'	Time Stamp/Updated Time Stamp
$Version + 1$	Updated metadata version

5. Proposed Scheme

5.1. Overview of the SDAP-K Architecture

The proposed Secure Data Auditing Protocol (SDAP-K) is a practical authenticated cloud auditing scheme for outsourced cloud storage. The framework provides two distinct security features: authenticated service access and integrity checking. Kerberos is used for mutual authentication, ticket-based service access, and secure session establishment between the protocol entities. We independently verify the integrity with an N-ary hash tree-based auditing mechanism. The SDAP-K architecture comprises four entities: Data Owner (DO), Authentication Server (AS), Metadata Server (MS), and Cloud Data Server (CDS). The DO stores, retrieves, updates, and audits outsourced data. AS authenticates the DO and issues service tickets. The MS stores the auditing metadata, including hash values, storage block order, and root hash values. The CDS stores the outsourced data blocks and handles requests for storage, retrieval, update, and audit.

The proposed framework supports data storage, data retrieval, dynamic updates, file-level and block-level auditing, metadata management, and error localization. Authentication and integrity verification are treated as distinct: Kerberos protects the communication and access process. The N-ary hash tree checks the correctness of outsourced data.

5.2. Authenticated Service Exchange

To avoid repeating the authentication description in each protocol phase (Figure 3), SDAP-K first establishes authenticated service sessions via Kerberos-based ticket exchange. After authentication, the DO can safely communicate with MS and CDS.

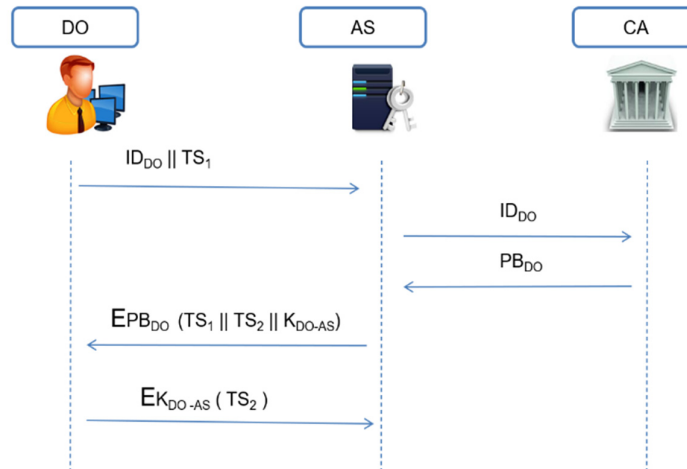


Figure 3. Kerberos-based user authentication workflow in SDAP-K.

Step 1: Data Owner Authentication

$M1: DO \rightarrow AS: (ID_{DO}, TS_1).$

Step 2: Ticket Generation for Metadata Server

$M2: AS \rightarrow DO: (TK_{MS}, K_{DO-MS})$

where

$TK_{MS} = E_{K_{AS-MS}} (ID_{DO}, K_{DO-MS}, TS_2).$

Step 3: Metadata Service for Request

$M3: DO \rightarrow MS: (TK_{MS}, Auth_{DO})$

where

$Auth_{DO} = E_{K_{DO-MS}} (ID_{DO}, TS_3).$

Step 4: Ticket Generation for Cloud Data Server

$M4: MS \rightarrow DO: (TK_{CDS}, K_{DO-CDS})$

where

$TK_{CDS} = E_{K_{MS-CDS}} (ID_{DO}, K_{DO-CDS}, TS_4).$

Step 5: Cloud Service Request

$M5: DO \rightarrow CDS: (TK_{CDS}, Auth'_{DO})$

where

$Auth'_{DO} = E_{K_{DO-CDS}} (ID_{DO}, TS_5).$

Step 6: Service Response

$M6: CDS \rightarrow DO: E_{K_{DO-CDS}} (Response, TS_6).$

Kerberos is used only for authenticating access to services and establishing secure sessions. It does not directly verify integrity. The integrity verification is performed based on a hash-tree auditing model described in the following subsections.

Kerberos Ticket Structure and Authorization

In SDAP-K, Kerberos service tickets for an operation are issued only after successful authentication by the Authentication Server (AS). The ticket is for a specific cloud operation (STORE, RETRIEVE, UPDATE, FILE_AUDIT, BLOCK_AUDIT, or RECOVERY) within a specific authorization scope. Each ticket contains the Data Owner identity, service identity, file identifier, operation type, authorization scope, session key, timestamp, nonce, and ticket lifetime. Before any service request can be processed, the ticket's validity is verified by checking timestamp freshness, clock-skew tolerance, ticket expiration, replay-cache validation, and operation-scope checking. The authenticator is rejected during verification if the unique nonce is not in the replay cache, or if the authenticator has been accepted and/or stored in the replay cache before. SDAP-K uses renewable service tickets with a short lifetime to reduce the damage of credential compromise. The Authentication Server manages key renewal and ticket revocation to provide secure authenticated communication. Table 5 shows the security purpose for Kerberos message exchange.

$$\text{Ticket}_{\text{MS}} = \text{Enck}_{\text{AS-MS}} (\text{ID}_{\text{DO}}, \text{ID}_{\text{MS}}, \text{FID}, \text{Op}, \text{Scope}, \text{K}_{\text{DO-MS}}, \text{TS}, \text{LT}, \text{Nonce})$$

Table 5. Kerberos message exchange and security functions.

Message	Description	Security Purpose
M1	DO → AS: Authentication request	User authentication
M2	AS → DO: Metadata Server ticket	Session key establishment and authorization
M3	DO → MS: Ticket + Authenticator	Mutual authentication and replay protection
M4	MS → DO: Cloud Data Server ticket	Authorization for cloud service access
M5	DO → CDS: Ticket + Authenticator	Secure service request (STORE, RETRIEVE, UPDATE, FILE_AUDIT, BLOCK_AUDIT, RECOVERY)
M6	CDS → DO: Encrypted service response	Secure communication and service completion

In SDAP-K, each Kerberos service ticket is explicitly bound to the Data Owner identity, service identity, file identifier, authorized operation, timestamp, nonce, and ticket lifetime. The Metadata Server (MS) and Cloud Data Server (CDS) will validate the ticket by checking the service identity, operation scope, timestamp freshness, ticket expiration, and nonce uniqueness before a service request can be processed. Both MS and CDS maintain a replay cache of previously accepted authenticators and nonces to reject duplicate requests. The service request is rejected if: the ticket has expired, the nonce has already been used, the timestamp is outside the allowed clock-skew tolerance, the requested operation is outside the authorized scope, or the service identity does not match the destination server.

5.3. Secure Data Storage Procedure

As the DO is authenticated, the data file (DF) to be stored in CDS will be encrypted as E(DF) and divided into blocks M. The keys used for data file encryption will be stored in a key file K. An Access Control List (ACL) is generated to identify users with read or write access. The data blocks are generated in such a way that it enables the construction of an N-ary hash tree, a tree where every parent node contains 'N' child nodes. DO chooses a random number for the variable 'n' and selects a random divisor N of 'n'. Given an even block size, the file is divided into data blocks as shown in Equation (1).

$$\text{Block size (bs)} = \frac{\text{File size (fs)}}{n} \quad (1)$$

After generating data blocks ($M=\{m_1, m_2, \dots, m_n\}$), DO sends the key file (K) and ACL to AS and also requests a ticket to communicate with MS. On receiving, DO stores the metadata (MD1) of the data file in MS. Initially, MD1 consists of file name (F), block names ($B = \{m_1, m_2, \dots, m_n\}$), hash of data blocks ($h(M) = \{h(m_1), h(m_2), \dots, h(m_n)\}$), number of blocks n , N , fs and bs . MS computes the hashed file name $h(F)$, hashed block names $h(B)$, generates Storage Block Ordering (SBO), constructs an N-ary hash tree to determine the root value $h(R)$ and intermediate hash values (IHV) with SBO. SBO is a random ordering of blocks used to store and construct the N-ary hash tree. The computed values are stored as Metadata (MD2) and sent to DO with the ticket to communicate with CDS. DO holds F as metadata. DO reorders the blocks to store in CDS with SBO. DO also replaces the block names B with hashed block names $h(B)$. After the mutual authentication exchange, DO sends the ordered data blocks to CDS with $h(F)$ and $h(B)$. CDS stores and replicates the data blocks M. CDS maintains metadata such as $h(F)$, n , and replica locations of M. Then, CDS computes the root value $h(R')$ with the data block hash values $h(M')$. The hashed file name $h(F)$ and root value $h(R')$ are sent to DO. DO checks that both root values $h(R)$ and $h(R')$ are equal, indicating that the data stored is intact. Otherwise, the error blocks need to be updated. As shown in Figure 4, authenticated storage requests are processed through the Metadata Server before the encrypted data blocks are securely stored on the Cloud Data Server.

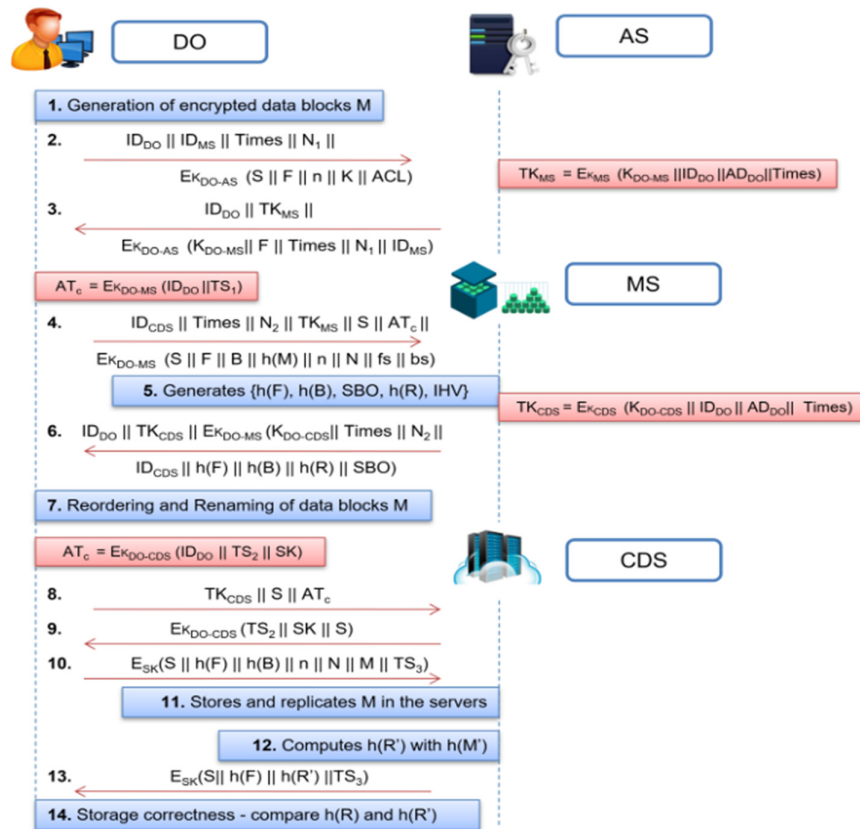


Figure 4. SDAP-K data storage workflow.

Let the data file be denoted as $DF = \{m_1, m_2, \dots, m_n\}$

where DF is the data file and m_i denotes the i th data block.

Each data block is hashed as $H_i = h(m_i)$, $1 \leq i \leq n$.

The complete set of block hash values is $H = \{H_1, H_2, \dots, H_n\}$.

Substituting $H_i = h(m_i)$, this can be written as $H = \{h(m_1), h(m_2), \dots, h(m_n)\}$.

The N-ary hash tree root is computed as $R = NHT(H)$.

The metadata stored in the Metadata Server is $MD = \{FID, n, N, SBO, H, R\}$.

For storage verification, the Cloud Data Server computes $R' = NHT(H')$

where $H' = \{h(m'_1), h(m'_2), \dots, h(m'_n)\}$.

The storage verification decision is

$$Verify_{Store} = \begin{cases} 1, & R' = R \\ 0, & R' \neq R \end{cases}$$

Algorithm 2 formally specifies the authenticated data storage procedure, including inputs, outputs, operational steps, and storage verification.

Algorithm 2: Data Storage

Input: Data file (DF), branching factor (N)

Output: Stored data blocks and metadata (MD)

Step 1: Divide (DF) into (n) blocks: $DF = \{m_1, m_2, \dots, m_n\}$

Step 2: Compute block hashes: $(H_i = h(m_i))$

Step 3: Construct an N-ary hash tree and compute the root (R).

Step 4: Generate metadata ($MD = \{FID, n, N, SBO, H, R\}$).

Step 5: Store metadata in MS.

Step 6: Send ordered encrypted data blocks to CDS.

Step 7: CDS stores blocks and computes (R').

Step 8: If ($R' = R$), storage is verified; otherwise, initiate error localization.

5.4. Secure Data Retrieval Procedure

The data retrieval protocol is used to retrieve the blocks of the requested data file, retrieve its actual contents, and verify its integrity. DO is authenticated to AS and requests a key file K and a ticket to MS. AS verifies the user access policies using the ACL for the requested data file and sends it to DO. It then requests SBO, n, the hash of the data blocks, $h(M)$, and the root value, $h(R)$, from MS using the ticket. MS verifies the DO and sends the requested metadata values with the ticket to communicate with CDS. After the mutual authentication exchange, DO sends the hashed filename $h(F)$ to the CDS, and CDS sends the data blocks M to DO. DO verifies n and computes the root value $h(R')$ for the tree constructed with SBO. It checks the root value $h(R)$ received from the MS to verify its integrity. If the comparison result is TRUE, the blocks are reordered and merged to obtain the encrypted file. The encrypted file is decrypted using the key file K received from AS.

If the comparison result is FALSE, error blocks need to be corrected. Figure 5 shows that the retrieved data blocks are validated by comparing the reconstructed root hash with the authentic root hash stored in the Metadata Server.

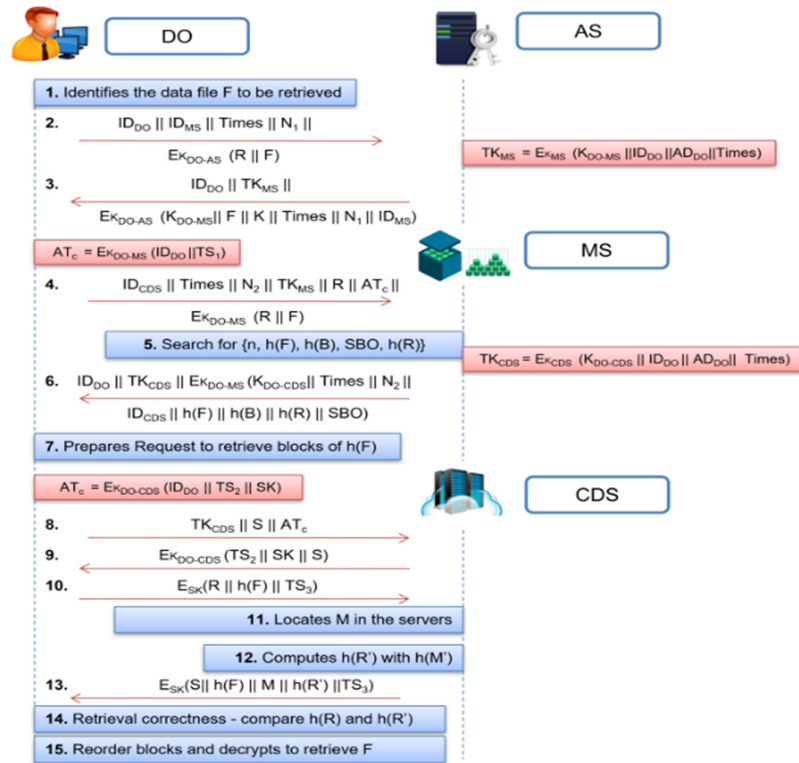


Figure 5. SDAP-K Data retrieval workflow.

During retrieval, the DO first authenticates through the service exchange described in Section 5.2. The DO then obtains the required metadata from MS and requests the stored data blocks from CDS.

The Cloud Data Server returns the stored data blocks

$$DF' = \{m'_1, m'_2, \dots, m'_n\}$$

where m'_i denotes the retrieved version of the i^{th} data block.

The Data Owner computes the hash value of each retrieved block as

$$H'_i = h(m'_i), 1 \leq i \leq n.$$

The set of retrieved block hash values is

$$H' = \{H'_1, H'_2, \dots, H'_n\}.$$

Using the retrieved block hashes, the Data Owner reconstructs the N-ary hash tree and computes the root hash value

$$R' = NHT(H')$$

where $NHT(\cdot)$ denotes the N-ary hash tree construction function.

The authentic root hash value R is obtained from the Metadata Server.

The integrity of the retrieved file is verified as

$$Verify_{retrieval} = \begin{cases} 1, R' = R \\ 0, R' \neq R \end{cases}$$

If $R' = R$, the retrieved file is considered intact, and the original file is reconstructed as

$$DF = Merge(SBO^{-1}(DF'))$$

where SBO^{-1} denotes the inverse storage block ordering operation used to restore the original block sequence.

Finally, the original file is recovered through decryption: $DF = DecK(E(DF))$

where K is the decryption key and $E(DF)$ represents the encrypted stored file.

If $R' \neq R$, the protocol initiates the error localization and recovery procedure to identify corrupted blocks and restore data correctness.

5.5. Dynamic Data Update Procedure

The data update protocol updates data blocks in the data file and verifies the correctness of the updated file. The update operations may insert, delete, and modify. DO receives the key file K and a ticket from MS in response to the authentication exchange with AS. DO updates and encrypts the data file $E(DF)$ and divides it into blocks M using Equation (1). Using the ticket, DO connects to MS and sends updated metadata $MD1'$ to MS. Metadata comprises $\{F, B, h(M), n, N, fs, bs\}$. MS compares the value of bs and the hash of the data blocks, $h(M)$, against the existing metadata ($MD1$) and determines whether the block's metadata needs to be updated. MS computes the hash of block names $h(B)$ and reconstructs the N -ary hash tree to determine the new root value $h(RN)$ and intermediate hash values (IHV') with SBO . MS updates metadata $MD2'$ with $h(RN)$, IHV' , and new hash values of modified block names ($h(M')$, $h(M)$). DO receives the hashed filename $h(F)$, hashed block names ($h(B') \subseteq h(B)$) to be updated in CDS, the new root value $h(RN)$, and a ticket to communicate with CDS. After mutual authentication exchange, DO sends blocks ($M' \subseteq M$) to be updated in CDS with hashed block names $h(B')$ and hashed file name $h(F)$. CDS updates the blocks M' and computes the root value $h(RN')$ with stored data block hash values. DO receives the hashed file name $h(F)$ and the computed root value $h(RN')$. DO verifies the CDS root value $h(RN)$ with the MS root value $h(RN')$ and ensures the correctness of the updated data file. Otherwise, error blocks need to be located and corrected. As shown in Figure 6, metadata and root hashes are updated to preserve data integrity after each modification.

SDAP-K supports the insertion, deletion, and modification of outsourced data blocks. Let the updated file be

$$DF = \{m_1, m_2, \dots, m_n\}.$$

After the update operation

$$DF^{new} = \{m_1^{new}, m_2^{new}, \dots, m_{n'}^{new}\}.$$

Each updated block is hashed as

$$H_i^{new} = h(m_i^{new}), 1 \leq i \leq n'.$$

The updated hash set is

$$H^{new} = \{H_1^{new}, H_2^{new}, \dots, H_{n'}^{new}\}.$$

The updated root hash is

$$R^{new} = NHT(H^{new}).$$

The Cloud Data Server computes

$$R'^{new} = NHT(H'^{new}).$$

The update verification condition is

$$Verify_{update} = \begin{cases} TRUE, & R'^{new} = R^{new} \\ FALSE, & R'^{new} \neq R^{new} \end{cases}$$

$$\text{where } n' = \delta = \begin{cases} > 0, \text{insertion} \\ < 0, \text{deletion} \\ 0, \text{modification} \end{cases}$$

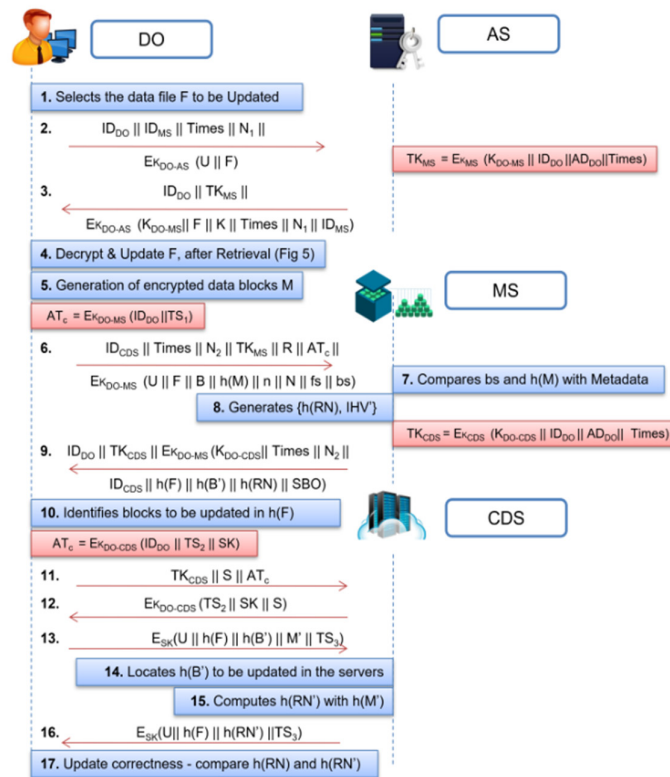


Figure 6. SDAP-K dynamic data update workflow.

Metadata Version Synchronization

SDAP-K ensures metadata consistency during dynamic data operations by storing a monotonically increasing version number for every metadata record. For each insert, delete, or modify operation, the Data Owner updates the corresponding block hash values, recomputes the root hash of the N-ary tree, increments the metadata version number, and signs (or computes a MAC over) the updated metadata before submitting it to the Metadata Server. An update request will be accepted only if the submitted metadata version is greater than the currently stored version. This mechanism prevents stale metadata from overwriting newer data and protects against rollback attacks. The Metadata Server uses version-based locking for concurrent update requests to ensure that only a single valid update of a metadata record occurs at any given time.

$$MD_{\text{new}} = \{FID, \text{Version} + 1, H', R', TS, \text{Sig}_{DO}(MD_{\text{new}})\}$$

$$\text{Version}_{\text{new}} > \text{Version}_{\text{stored}}$$

If true, Accept (MD_{new})

Otherwise, Reject (MD_{new})

If multiple update requests are issued simultaneously, the Metadata Server serializes updates using version-based locks. An update is committed only when the owner's signature and the metadata version number are successfully verified. Requests with stale version numbers are rejected, thus maintaining consistency among concurrent operations on metadata.

5.6. File—Level Auditing

File-level auditing verifies the integrity of the complete outsourced file. After authenticated service establishment, the DO obtains the authentic root hash (R) from MS and challenges CDS for the file root. Figure 7 shows that the integrity of the entire outsourced file is verified through root-hash comparison.

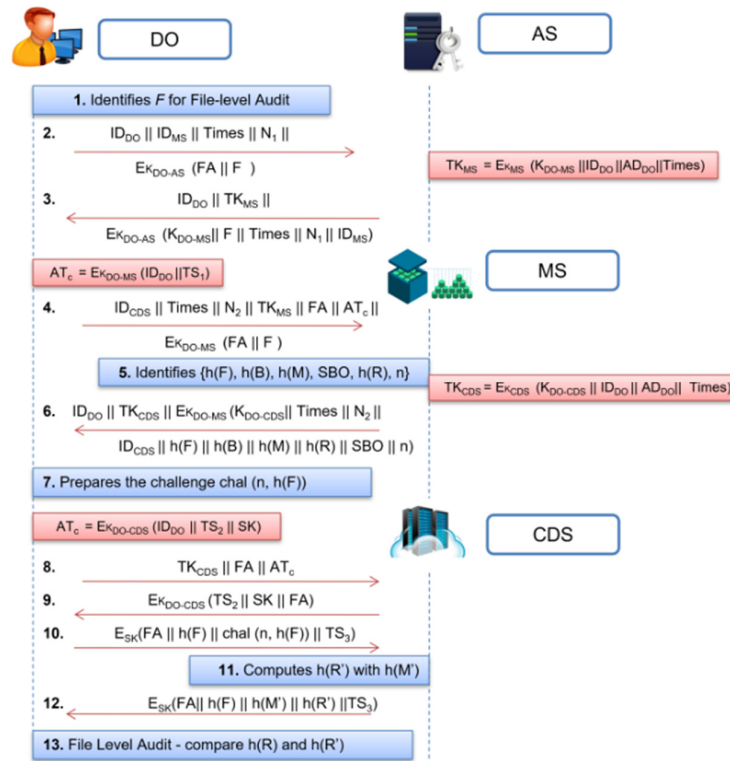


Figure 7. SDAP-K file-level auditing workflow.

Let the outsourced file be divided into n blocks $DF = \{m_1, m_2, \dots, m_n\}$.

The authentic block hash set stored in the Metadata Server is $H = \{H_1, H_2, \dots, H_n\}$

where $H_i = h(m_i)$, $1 \leq i \leq n$.

The authentic root hash value is $R = NHT(H)$.

During file-level auditing, the Data Owner sends the challenge

$$Chal_F = (FID, n).$$

The Cloud Data Server computes the hash values of the stored blocks

$$H'_i = h(m'_i), 1 \leq i \leq n.$$

The computed hash set is: $H' = \{H'_1, H'_2, \dots, H'_n\}$

The Cloud Data Server reconstructs the N-ary hash tree and computes

$$R' = NHT(H').$$

The Data Owner verifies file integrity as

$$Verify_{file} = \begin{cases} 1, R' = R \\ 0, R' \neq R \end{cases}$$

If $R' = R$, the outsourced file is considered intact. If $R' \neq R$, Algorithm 3 defines the secure data retrieval procedure and integrity verification process.

Algorithm 3: File-Level Auditing

Input: File identifier (FID), metadata (MD)

Output: Integrity verification result

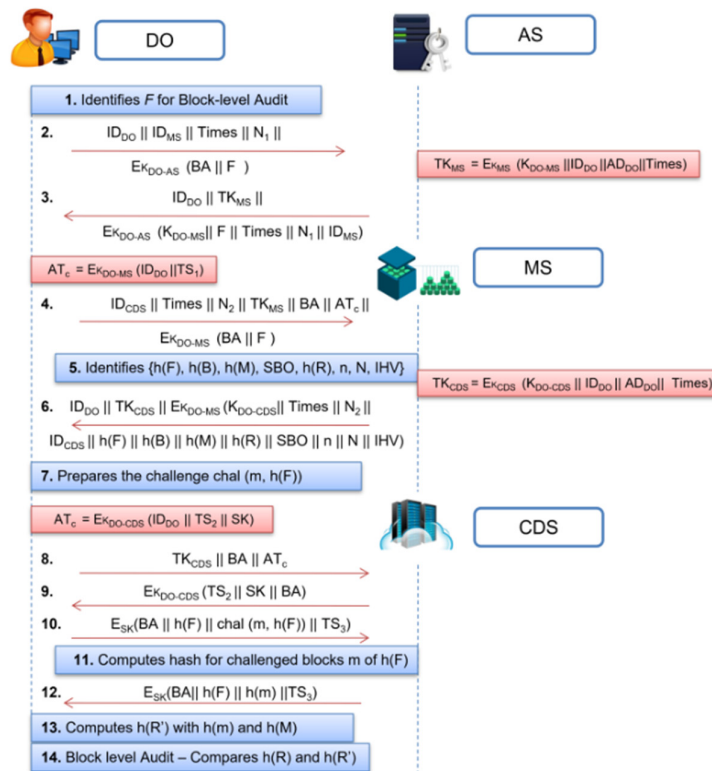
Step 1: DO obtains (R) from MS.

Step 2: DO sends file-level challenge ($\text{Chal}_F = (\text{FID}_n)$) to CDSStep 3: CDS computes root hash (R').Step 4: DO compares (R') with (R).Step 5: If ($R' = R$), return TRUE.

Step 6: Otherwise, return FALSE and initiate error localization.

5.7. Block-Level Auditing

To prevent a malicious Cloud Data Server from passing the audit with only the previously stored hash values, SDAP-K uses a nonce-bound challenge-response mechanism. The Data Owner creates a new nonce and timestamp for every audit request. The Cloud Data Server is required to return the requested data block (or its cryptographic authentication tag) together with the corresponding N-ary hash-tree authentication path. The fresh nonce ties the proof to the current audit session, thus preventing replay of previously generated audit responses and ensuring that the server has the challenged data blocks at the time of verification. As illustrated in Figure 8, only the challenged blocks and their authentication paths are verified, thereby reducing computation and communication overhead.

**Figure 8.** SDAP-K block-level auditing workflow.

Block-level auditing verifies selected data blocks rather than the entire file. Let the challenge index set be

$$I = \{i_1, i_2, \dots, i_k\},$$

denote the challenge index set, where each element i_j represents the index of a selected data block and

$$I \subseteq \{1, 2, \dots, n\}, 1 \leq k \leq n.$$

The corresponding challenged block set is defined as

$$C = \{m_{i1}, m_{i2}, \dots, m_{ik}\}.$$

Let the outsourced file be divided into (n) data blocks

$$DF = \{m_1, m_2, \dots, m_n\}.$$

Algorithm 4 specifies the block-level auditing protocol and the corresponding verification condition.

Algorithm 4: Block-Level Auditing

Input: File identifier (FID), challenge index set (I), metadata (MD)

Output: Integrity verification result

Step 1: DO selects challenge index set ($I = \{i_1, i_2, \dots, i_k\}$)

Step 2: DO generates (Nonce, TS) and sends $\text{ChalB} = \{\text{FID}, I, \text{Nonce}, \text{TS}\}$

Step 3: CDS returns $\text{ProofB} = \{\text{Tag}(m_i), \text{AuthPath}_i, \text{Nonce}, \text{TS}\}$

Step 4: DO verifies nonce freshness and the authentication path, and reconstructs the root hash.

Step 5: Accept only if Nonce, TS, AuthPath, and Root Hash are valid

Step 6: Otherwise, return FALSE and initiate error localization.

The Data Owner randomly selects (k) block indices for auditing

$$I = \{i_1, i_2, \dots, i_k\}, I \subseteq \{1, 2, \dots, n\}, 1 \leq k < n;$$

$$C = \{m_i | i \in I\}.$$

The challenge message is represented as

$$\text{ChalB} = \{\text{FID}, I, \text{Nonce}, \text{TS}\};$$

$$\text{ProofB} = \{\text{Tag}(m_i), \text{AuthPath}_i, \text{Nonce}, \text{TS}\};$$

$$H_C = \{h(m_{i_1}), h(m_{i_2}), \dots, h(m_{i_k})\};$$

$$R' = \text{ComputeRoot}(H_C, \text{MD});$$

$$\text{Verify}_{\text{block}} = \begin{cases} 1, & R' = R \\ 0, & R' \neq R \end{cases}$$

Audit Soundness: The proposed challenge-response protocol assures data possession, as the Cloud Data Server must generate a valid authentication proof using the challenged data block and the corresponding N-ary authentication path. Each challenge has a new nonce and timestamp, so previously generated proofs cannot be replayed. Thus, a malicious server that does not have the challenged data blocks can only succeed in the audit with negligible probability.

5.8. Error Localization and Recovery

The protocols are employed to verify the correctness of the data in the cloud. If the root hash comparison results are not TRUE, the error block must be identified and replaced with the original data block from the data file DF. If any verification step fails,

SDAP-K identifies corrupted blocks by comparing the stored block hash values with the recomputed ones.

5.8.1. Data Storage—Error Localization and Recovery

Figure 9 demonstrates how corrupted blocks are identified and replaced to restore data integrity. CDS computes hashes $h(M')$ of the stored data blocks and sends them to DO. DO compares $h(M)$ with $h(M')$ and identifies the error blocks e in $h(M')$. For example, with $N = 4$ and $n = 16$, comparing the hash values of the block values shows that $h(m_5)$ differs from the others. Then, the data correctness operation is performed, in which the m_5 data block, identified as the error block Me , is sent to CDS by DO. CDS replaces m_5 's data block with the received m_5 data block and constructs the N -ary hash tree. The resultant $h(R'')$ is sent to DO by CDS. DO compares $h(R)$ and $h(R'')$ and ensures data integrity if the result is TRUE. Else, the same process is repeated.

Let the authentic block hash set be

$$H' = \{H'_1, H'_2, \dots, H'_n\}$$

where

$$H_i = h(m_i)$$

and

$$H'_i = h(m'_i).$$

The set of corrupted blocks is identified as: $E = \{m_i \mid H_i \neq H'_i, 1 \leq i \leq n\}$

The cardinality of the error set is

corrupted ock.

For each corrupted block, ptei in $m_i \in E$, the corresponding replica m_i^r is verified by

$$h(m_i^r) = H_i.$$

The corrupted block is replaced as

$$m'_i \leftarrow m_i^r.$$

After recovery, the new hash set becomes

$$H'' = \{H''_1, H''_2, \dots, H''_n\}$$

where

$$H''_i = h(m''_i).$$

The recovered root hash is computed as

$$R'' = NHT(H'').$$

Recovery correctness is verified by

$$R'' = R.$$

The recovery decision is

$$Recover = \begin{cases} 1, & R'' = R \\ 0, & R'' \neq R \end{cases}$$

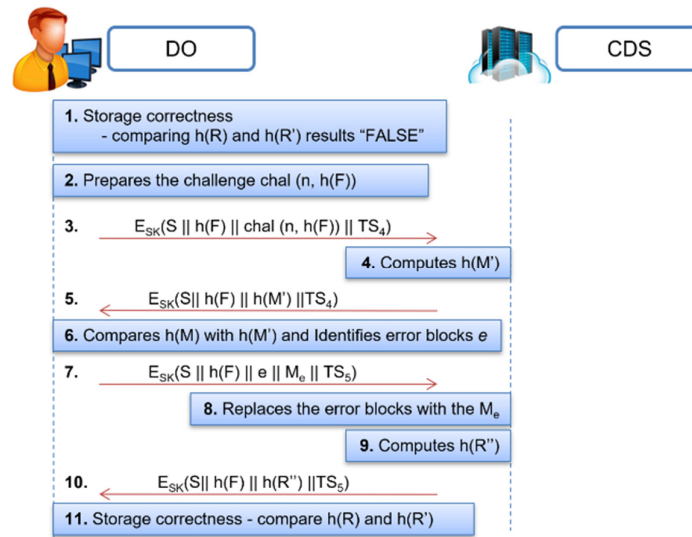


Figure 9. SDAP-K error localization and recovery during data storage.

5.8.2. Data Retrieval—Error Localization and Recovery

As shown in Figure 10, corrupted blocks are detected, verified against valid replicas, and recovered before the original data file is reconstructed. Then, DO compares $h(M)$ with $h(M')$ and localizes the error blocks. For example, with $N = 4$ and $n = 16$, it compares the two sets of hash values and identifies $h(m_5)$ as the error block e . Then, DO challenges the m_5 data block e to CDS. CDS identifies the replica block of m_5 and sends replicas $R(e)$ of the actual data block m_5 to DO. DO computes hashes of all replicas of the data block m_5 and compares them with $h(m_5)$ in $h(M)$. DO sends the data block m_5 , along with the hashed filename $h(F)$, to CDS. CDS replaces the existing m_5 data block of $h(F)$ in the storage server. Then, CDS computes $h(m_5)$ and sends it to DO. DO verifies the correctness of the data stored in the cloud. Otherwise, the retrieval auditing process repeats until the integrity is verified.

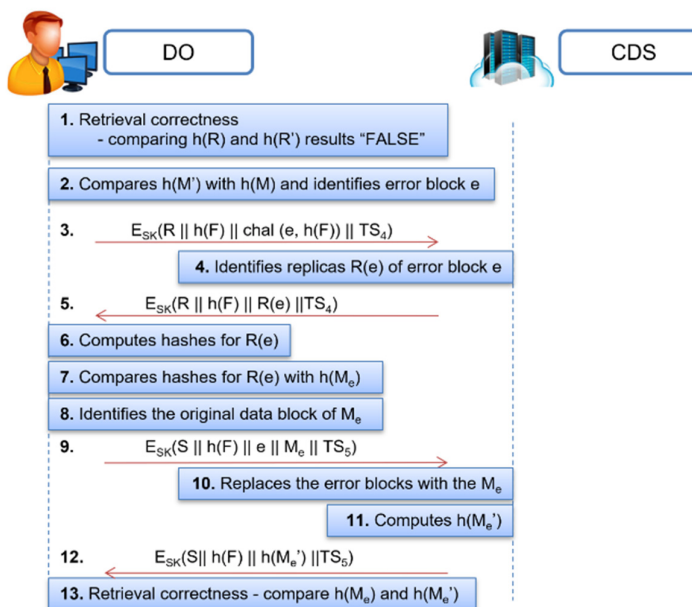


Figure 10. SDAP-K Error localization and recovery during data retrieval.

5.8.3. Replica Validation and Recovery

SDAP-K assumes that each outsourced data block is either replicated by the Cloud Data Server (CDS) or recoverable from a trusted owner's backup. During recovery, the CDS fetches the available replica of the corrupted block. The Data Owner checks its integrity by comparing the computed hash value with the original block hash stored in the Metadata Server. A copy is valid only if

$$h(m_i^r) = H_i$$

where m_i^r denotes the replica of the i -th data block and H_i is the authentic block hash. After successful validation, the corrupted block is replaced with the verified replica, and the root hash is recomputed to verify successful recovery. If no valid replica or owner backup is available, the recovery operation fails, and the Data Owner is notified of the integrity violation. Therefore, the success of the recovery depends on the existence of authentic replicas or backup data, the integrity of metadata records, and the correctness of replica storage. If the replicas are compromised or the metadata is corrupted, you might need administrative intervention to recover successfully.

Recovery is only possible if at least one replica or owner backup exists. Furthermore, the Metadata Server must contain valid metadata, including valid block and root hashes. If both the stored replicas and the metadata are compromised, SDAP-K cannot guarantee successful recovery, and the corrupted data must be restored from an external backup.

5.9. Complexity and Scalability Analysis

The computational complexity of the proposed SDAP-K operations is summarized in Table 6.

Table 6. Computational complexity of SDAP-K operations.

Operation	Complexity
Block hashing	$O(n)$
N-ary hash tree construction	$O(n)$
File-level auditing	$O(n)$
Block-level auditing	$O(k \log Nn)$
Metadata lookup	$O(\log n)$
Error localization	$O(n)$
Root verification	$O(1)$

The N-ary hash tree reduces tree depth compared with binary hash tree structures. The tree depth is approximately

$$D = \lceil \log_N(n) \rceil.$$

For large-scale deployment, the MS can be replicated or partitioned to avoid a single point of failure. Metadata can be indexed by file identifier (FID), and distributed metadata storage can support scalability in an enterprise cloud environment. Similarly, the AS can be deployed using a replicated Kerberos infrastructure to improve authentication availability and load balancing.

6. Security Analysis

6.1. Security Model and Assumptions

The proposed SDAP-K framework provides authenticated service access and integrity verification for outsourced cloud data. Authentication is achieved through Kerberos-based ticket exchange, while data integrity verification is achieved through N-ary hash

tree-based auditing. These two functions are treated as separate but complementary security mechanisms. Table 7 presents the attack-resistance mechanism.

Table 7. Attack Resistance Mechanisms in SDAP-K.

Attack	Protocol Fields	Defense Mechanism
Replay Attack	Timestamp, Nonce, Ticket Lifetime	Replay cache, timestamp freshness, ticket expiration
Impersonation Attack	User Identity, Session Key, Kerberos Ticket	Mutual authentication and secret session keys
Unauthorized Operation	Operation Type, Authorization Scope	Operation-scope verification
Wrong Service Access	Service Identity	Service identity verification at MS/CDS
Expired Ticket Attack	Ticket Lifetime	Ticket expiration check
Metadata Tampering	Owner Signature/MAC	Signature/MAC verification
Data Modification	Root Hash, Authentication Path	MMH-32 hash verification and N-ary authentication path

6.1.1. Adversary Model

The security analysis considers the following adversaries:

- **External Network Adversary:** An external adversary can intercept, modify, replay, or inject messages exchanged between the Data Owner (DO), Authentication Server (AS), Metadata Server (MS), and Cloud Data Server (CDS). However, the adversary is assumed to be unable to break standard cryptographic primitives in polynomial time.
- **Malicious Cloud Data Server:** The CDS can try to delete, modify, replace, or hide corrupted data blocks. It can also try to return incorrect auditing responses. But the CDS does not have the Data Owner's private keys or valid Kerberos session credentials.
- **Insider Adversary:** A legitimate but malicious user may attempt to abuse authorized access, request unauthorized metadata, or perform improper auditing operations. These are prevented through ticket-based authorization and access control.

The security analysis of SDAP-K is performed against four types of adversaries: an external network adversary, a malicious Cloud Data Server, a malicious authorized user, and a compromised or stale Metadata Server. Each security claim is associated with a specific protocol mechanism and cryptographic assumption. Timestamps, nonces, ticket lifetimes, and replay-cache validation provide replay resistance. Kerberos tickets and secret session keys are used to provide resistance to impersonation. Data Owner signatures or MACs over metadata records are used to protect metadata integrity. Nonce-bound block proofs and N-ary authentication paths guarantee audit soundness. Thus, the Cloud Data Server, which does not store the challenged data blocks, cannot produce a valid audit proof with non-negligible probability under the collision-resistance assumption of MMH-32. Table 8 presents the security analysis of SDAP-K.

Table 8. Security properties and cryptographic assumptions of SDAP-K.

Security Claim	Protocol Mechanism	Assumption
Replay resistance	Timestamp, nonce, ticket lifetime, replay cache	Bounded clock skew
Impersonation resistance	Kerberos tickets and session keys	Secret long-term keys
Unauthorized access prevention	Operation-bound service tickets	Correct access-control policy
Metadata integrity	Owner signature/MAC over metadata	Signature/MAC unforgeability
Metadata freshness	Version number and timestamp	Monotonic version control
Audit soundness	Nonce-bound block proof and authentication path	MMH-32 collision resistance
Data modification detection	N-ary hash-tree root verification	Collision-resistant hash function

Recovery correctness

Replica/block verification before replacement

At least one valid replica exists

6.1.2. Cryptographic Assumptions

The following assumptions are used in the analysis:

- AES-256 provides secure encryption for authenticated communication.
- RSA provides secure key generation and identity verification. $h(m_1) \neq h(m_2), \forall m_1 \neq m_2$, where $h(\cdot)$ is assumed to be a collision-resistant hash function.

$$K_{DO-MS} \neq \emptyset$$

$$K_{DO-CDS} \neq \emptyset$$

where K_{DO-MS} and K_{DO-CDS} denote valid session keys established through Kerberos authentication.

The integrity verification process assumes that R is the authentic root hash maintained by the Metadata Server, and

$$R' = NHT(H')$$

is the root hash reconstructed during auditing.

Integrity is verified as

$$Verify(R, R') = \begin{cases} TRUE, & R = R' \\ FALSE, & R \neq R' \end{cases}$$

6.2. Authentication Security

Kerberos is used in SDAP-K to provide authenticated service exchange among protocol entities. It ensures that only authorized DOs can communicate with MS and CDS.

Let $(Ticket_{MS})$ denote the service ticket issued for MS

$$Ticket_{MS} = E_{K_{AS-MS}}(ID_{DO}, K_{DO-MS}, TS_1)$$

where $Ticket_{MS}$ is the service ticket issued by AS for accessing MS.

Similarly, the service ticket issued for CDS is

$$Ticket_{CDS} = E_{K_{MS-CDS}}(ID_{DO}, K_{DO-CDS}, TS_2)$$

where $Ticket_{CDS}$ is the service ticket issued by MS for accessing CDS.

The authenticator generated by DO for MS is

$$Auth_{DO-MS} = E_{K_{DO-MS}}(ID_{DO}, TS_3).$$

The authenticator generated by DO for CDS is

$$Auth_{DO-CDS} = E_{K_{DO-CDS}}(ID_{DO}, TS_4).$$

Authentication is successful only if

$$Verify(Ticket_{MS}, Auth_{DO-MS}) = TRUE$$

and $Verify(Ticket_{CDS}, Auth_{DO-CDS}) = TRUE$.

Therefore, the authentication decision is

$$Auth_{SDAP} = \begin{cases} TRUE, & Ticket_{MS} \text{ and } Ticket_{CDS} \text{ are valid} \\ FALSE, & \text{otherwise} \end{cases}$$

Thus, SDAP-K provides authenticated service access, assuming that Kerberos tickets and session keys remain confidential.

Theorem 1. *Authentication Correctness. If the shared and session keys remain secret, only an authenticated Data Owner can obtain valid access to the MS and CDS services.*

Proof. Argument. Given the assumptions stated above, to violate the claimed property, the adversary must either forge a Kerberos ticket, forge the Data Owner's signature/MAC, replay a fresh nonce, or find an MMH-32 collision. The events are computationally infeasible; thus, SDAP-K satisfies the stated security property except with negligible probability. $\square$

The MS accepts a service request only if $(\text{Ticket}_{\text{MS}})$ is valid and encrypted under $(K_{\text{AS-MS}})$. Similarly, CDS accepts a request only if $(\text{Ticket}_{\text{CDS}})$ is valid and encrypted under $(K_{\text{MS-CDS}})$. Since an adversary cannot derive these keys under the stated cryptographic assumptions, forging valid tickets is computationally infeasible. Therefore, unauthorized access attempts are rejected.

6.3. Integrity Verification Security

Integrity verification in SDAP-K is provided by the N-ary hash tree, not by Kerberos. Kerberos protects authenticated communication, while the hash-tree mechanism verifies the correctness of storage.

The authentic block hash set maintained by the Metadata Server is

$$H = \{H_1, H_2, \dots, H_n\}$$

where

$$H_i = h(m_i), 1 \leq i \leq n.$$

The authentic root hash value is computed as

$$R = \text{NHT}(H).$$

During auditing, the Cloud Data Server returns the stored block hash set

$$H' = \{H'_1, H'_2, \dots, H'_n\}$$

where

$$H'_i = h(m'_i), 1 \leq i \leq n.$$

The Data Owner reconstructs the root hash value as

$$R' = \text{NHT}(H').$$

Integrity verification is performed as

$$\text{Verify}_{\text{Integrity}} = \begin{cases} \text{TRUE}, & R' = R \\ \text{FALSE}, & R' \neq R \end{cases}$$

A data integrity violation exists if

$$\exists i \in \{1, 2, \dots, n\} \text{ such that } H_i \neq H'_i$$

which implies:

$$R' \neq R.$$

Therefore, any modification, deletion, or replacement of a data block is detected through root hash mismatch.

Theorem 2. *Integrity Detection.* If an outsourced data block is modified, deleted, or replaced, SDAP-K detects the integrity violation with high probability due to the hash function's collision resistance.

Proof. Sketch. Any change in a block (m_i) changes its hash value from (H_i) to (H'_i) . Since the N-ary hash tree root is computed from all block hashes, a changed leaf hash

propagates to the root. Therefore, the recomputed root (R') differs from the authentic root (R), and the verification function returns FALSE. $\square$

6.4. Resistance to Replay Attacks

SDAP-K uses timestamps, nonces, and ticket expiration to prevent replay attacks. A valid authentication request must satisfy freshness constraints.

6.4.1. Timestamp Freshness Verification

$$|TS_{current} - TS_{received}| \leq \Delta T$$

where:

- $TS_{current}$ = current system time;
- $TS_{received}$ = timestamp received in the request;
- ΔT = maximum allowable time difference.

6.4.2. Ticket Validity Condition

A ticket is considered valid if

$$TS_{issue} \leq TS_{current} \leq TS_{expire}$$

where:

- TS_{issue} = ticket issuance time;
- TS_{expire} = ticket expiration time.

6.4.3. Replay Detection Condition

A replay attempt is detected if

$$TS_{current} > TS_{expire}.$$

6.4.4. Authentication Acceptance Decision

$$Verify_{Replay} = \begin{cases} ACCEPT, & |TS_{current} - TS_{received}| \leq \Delta T \\ & \text{and } TS_{issue} \leq TS_{current} \leq TS_{expire} \\ REJECT, & \text{otherwise} \end{cases}$$

6.4.5. Optional Nonce-Based Freshness

If you use nonces

$$N_{received} \notin N_{used}$$

where:

- $N_{received}$ = received nonce;
- N_{used} = set of previously accepted nonces;

the final replay-resistance condition becomes

$$Verify_{Replay} = \begin{cases} TRUE, & TS \text{ valid} \wedge N_{received} \notin N_{used} \\ FALSE, & \text{otherwise} \end{cases}$$

Theorem 3. *Replay Resistance. If timestamps, nonces, and ticket expiration are correctly enforced, replayed authentication or service messages are rejected.*

Proof. Sketch. A replayed message contains an outdated timestamp, a reused nonce, or an expired ticket. Since SDAP-K validates freshness before accepting service requests, stale or repeated messages fail verification and are rejected. $\square$

6.5. Resistance to Man-in-the-Middle Attacks

In SDAP-K, session keys are distributed via Kerberos tickets encrypted with shared server keys. An adversary intercepting the communication cannot derive the session keys:

$$K_{DO-MS}$$

without access to the corresponding long-term shared keys. Service messages are encrypted or authenticated using session keys. Therefore, an adversary's modification of a message results in invalid authentication data.

The attack succeeds only if

$$Adv \rightarrow K_{DO-MS}$$

where (Adv) denotes the adversary. Under the secrecy assumption of session keys, this event is computationally infeasible.

Thus, SDAP-K resists man-in-the-middle attacks under the stated cryptographic assumptions.

6.6. Resistance to Forgery and Unauthorized Access

6.6.1. Ticket Forgery

An adversary attempting to forge a ticket must generate

$$\text{Ticket}_{MS} = E_{K_{AS-MS}}(ID_{DO}, K_{DO-MS}, TS)$$

without knowing the required encryption keys. Since these keys are not available to the adversary, ticket forgery is computationally infeasible.

6.6.2. Metadata Forgery

If an adversary modifies metadata, the authentic root value (R), block hash set (H), or storage block ordering (SBO) becomes inconsistent with the data stored in CDS. During verification, this inconsistency results in

$$R' \neq R.$$

Therefore, metadata manipulation is detectable during integrity verification.

6.6.3. Data Block Forgery

If CDS replaces a valid block m_i with a forged block $m_{\sim i}$, then

$$h(m_i) \neq h(m_{\sim i})$$

unless a hash collision occurs, since the hash function is assumed to be collision-resistant, forged block replacement is detected by a root mismatch.

6.7. Error Localization Security

When verification fails, SDAP-K identifies corrupted blocks by comparing the authentic and recomputed hash values.

Let

$$H = \{H_1, H_2, \dots, H_n\}$$

and

$$H' = \{H'_1, H'_2, \dots, H'_n\}.$$

The error set is defined as

$$E = \{m_i | H_i \neq H'_i, 1 \leq i \leq n\}$$

where:

- E = set of corrupted (error) data blocks;
- H_i = authentic hash value of block m_i ;
- H'_i = received or recomputed the hash value of the block m_i ;
- n = total number of data blocks.

For each corrupted block $m_i \in E$, a valid replica m_i^r is accepted only if

$$h(m_i^r) = H_i.$$

After replacement, the recovered root is computed as

$$R = NHT(H'')$$

where:

- H'' = recovered block hash set;
- R'' = root hash after recovery;
- $NHT(\cdot)$ = N-ary hash tree construction function.

Recovery is accepted only when

$$R'' = R.$$

Thus, SDAP-K provides verifiable error localization and recovery under the assumption that at least one valid replica or the original block is available.

6.8. Security Feature Comparison

Table 9 presents the Security Feature Comparison between Existing Cloud Auditing Schemes and the Proposed SDAP-K Framework.

Table 9. Security feature comparison of cloud auditing schemes.

Security Features	Existing Auditing Schemes	Proposed SDAP-K
Mutual authentication	Partial	Yes
Ticket-based service access	Limited	Yes
File-level auditing	Yes	Yes
Block-level auditing	Yes	Yes
Metadata management	Partial	Yes
Error localization	Limited	Yes
Replay Protection	Limited	Timestamp, nonce, and ticket expiration
TPA Dependency	Often Required	Not Required

6.9. Trust Assumptions and Limitations

The proposed SDAP-K framework improves authenticated auditing and integrity verification; however, its security depends on several trust and operational assumptions.

- Initially, AS and MS are trusted or semi-trusted entities. If AS is compromised, authentication tickets and access control decisions may be affected. If MS is compromised, the integrity of metadata may be at risk. Therefore, secure server hardening, audit logging, access control, and metadata backup mechanisms should be included in practical deployment.
- Second, session keys and private credentials must be protected. If a valid user credential or session key is compromised, an adversary may impersonate the user until the corresponding ticket expires or the key is revoked. Therefore, ticket lifetime management, key revocation, and credential protection are necessary.

- Third, SDAP—K does not primarily address denial-of-service attacks. An adversary may attempt to disrupt access to AS, MS, or CDS. Replication, load balancing, and failover deployment can reduce this risk.
- Finally, the proposed framework is designed as a practical, authenticated cloud auditing architecture. It provides security under stated cryptographic and operational assumptions, but it is not intended to replace fully decentralized, blockchain-based, or formally verified cryptographic auditing systems.

Security analysis shows that SDAP-K provides authenticated service access using Kerberos and integrity checks using N-ary hash tree-based auditing. Kerberos is used to secure authentication, ticket exchange, and session establishment, while N-ary hash tree detects unauthorized data modification, deletion, or replacement. Under the stated assumptions, SDAP-K is shown to be resistant to replay attacks, man-in-the-middle attacks, forgery attacks, unauthorized service access, and cloud data tampering.

7. Performance Analysis

7.1. Experimental Setup

The analysis of the proposed SDAP-K scheme is discussed by comparing it with existing cloud data auditing schemes. The proposed SDAP-K scheme has been implemented in the Eucalyptus cloud computing environment. A high-performance computing system has been used to behave as DO, AS, MS, and CDS servers. Linux operating system has been used as the base and guest in this setup. The proposed SDAP-K has been implemented in two ways: **Method 1:** DO is placed as a remote user; AS and MS are placed on the organization's premises; and CDS is placed in a cloud environment. **Method 2:** DO is placed as a remote user; AS, MS, and CDS are placed in the cloud environment. The performance of the proposed protocol has been compared with that of Wang et al. [7] and Li et al. [8]. In existing schemes [7,8], data correctness is not guaranteed during storage, retrieval, or updating. In the proposed SDAP-K, data correctness will be ensured in the storage, retrieval, and update protocols. In addition, error localization and data verification have been performed in the event of an audit failure. The existing systems use a trusted third-party auditor to perform audits, whereas the proposed SDAP-K has the data owner perform them.

The proposed SDAP-K framework was implemented and tested in a cloud computing environment to analyze the computational and communication overheads for storage, retrieval, update, and auditing operations. The implementation comprises four logical entities: Data Owner (DO), Authentication Server (AS), Metadata Server (MS), and Cloud Data Server (CDS). The evaluation was conducted in a cloud-based test environment where AS, MS, and CDS were deployed as independent service components.

To reproduce the results, experiments were conducted with different file sizes and block configurations. The number of data blocks (n) was varied from 100 to 5000, and the N-ary tree branching factor (N) was chosen from {5, 10, 15, 20, 25, 30}. All experiments were executed multiple times, and the reported results reflect the average execution time across repeated runs. The proposed framework was compared against representative cloud auditing schemes by Wang et al. [7] and Li et al. [8], which are widely used as benchmarks in cloud auditing research. A comparison was made of computational and communication overheads, as well as auditing efficiency, across the same data configurations. Table 10 shows the implementation setup.

To assess the impact of deployment architecture on the proposed framework, two SDAP-K implementation configurations were considered. SDAP-K Method 1 is a hybrid deployment in which the Data Owner (DO) resides remotely, the Authentication Server (AS) and Metadata Server (MS) are deployed within the organizational network, and the

Cloud Data Server (CDS) is hosted in the cloud. In SDAP-K Method 2, a full cloud deployment is considered, in which AS, MS, and CDS are deployed as cloud services, and DO connects to the system remotely. These two deployment models were compared under identical experimental conditions to analyze their effects on storage, retrieval, update, and auditing performance. Each procedure was repeated independently 30 times under the same experimental conditions, and the results shown in the table are average execution times. The standard deviation was calculated to assess the variability of the measurements. The error bars in Figures 11–15 represent one standard deviation. When applicable, 95% confidence intervals were calculated to evaluate the reliability of the measured performance. This statistical analysis increases the reproducibility and credibility of the experimental results.

Table 10. Experimental environment and system configuration.

Parameter	Specification
Experimental Platform	Eucalyptus Cloud Computing Environment
Deployment Model	Private cloud/Local cloud simulation
Cloud Components	Data Owner (DO), Authentication Server (AS), Metadata Server (MS), Cloud Data Server (CDS)
Server Deployment	AS, MS, and CDS deployed as independent service components (or specify if simulated on a single machine)
Processor (CPU)	Intel Core i7/Intel Xeon (specify actual model) [USA]
Main Memory (RAM)	16 GB DDR4
Storage Device	512 GB SSD
Operating System	Ubuntu Linux 22.04 LTS (or actual version)
Hash Function	Modified Murmur Hash
Network Configuration	Gigabit Ethernet (1 Gbps)
File Sizes Evaluated	100 MB to 5000 MB
Number of Data Blocks	100–5000 blocks
N-ary Tree Branching Factor	5, 10, 15, 20, 25, and 30
Performance Metrics	Storage time, Retrieval time, Update time, File-level auditing time, Block-level auditing time, Communication overhead

7.2. Evaluation Metrics

The performance evaluation is based on the following metrics:

- Computation Overhead: Time required for processing storage, retrieval, update, and auditing operations;
- Communication Overhead: The network communications required between protocol entities;
- Auditing Latency: Time taken for integrity verification;
- Metadata Overhead: Storage needed to store auditing metadata;
- Scalability: How performance behaves as the number of data blocks increases.

The average execution time is computed as

$$\text{AvgTime} = \sum_{i=1}^r T_i$$

where:

- *AvgTime* = average execution time;
- T_i = execution time of the i^{th} experimental run;
- r = total number of experimental runs.

7.3. Fairness of Comparative Evaluation

To make a fair comparison, the proposed SDAP-K framework and the baseline schemes of Wang et al. [7] and Li et al. [8] were evaluated under the same experimental conditions whenever possible. The comparison used the same file sizes, block configurations, hardware environment, and cryptographic settings. The source code of the baseline schemes was not publicly available. Thus, the performance of the baseline schemes was evaluated using the algorithms and complexity analyses reported in the original publications, and the schemes were implemented in the same experimental environment. The same evaluation metrics are used to fairly compare the computation, communication, and auditing overheads, though there may be minor implementation differences.

Baseline Schemes for Comparison

We compare our proposed SDAP-K framework with representative cloud auditing schemes, such as Wang et al. [7] and Li et al. [8], as well as other baseline methods [2,11,12,25–28,34–36] shown in the performance figures. We select these schemes as they are widely used for public auditing, privacy-preserving auditing, and dynamic integrity verification in cloud storage. We used the same file sizes, block configurations, and evaluation metrics wherever possible to ensure a fair comparison.

7.4. Communication Overhead Analysis

In addition to message complexity, the communication overhead of the proposed SDAP-K was evaluated by estimating the number of bytes exchanged during authentication and auditing operations. Authentication overhead includes Kerberos tickets, authenticators, and messages for session establishment, and auditing overhead includes metadata, audit challenges, audit proofs, and N-ary authentication paths. Kerberos has an initial communication cost for session establishment, but this cost is amortized over multiple storage, retrieval, update, and auditing operations as session keys are reused. The communication overhead of SDAP-K was compared with that of representative cloud auditing schemes under the same experimental configuration. Table 11 shows the SDAP-K Communication Overhead Analysis.

Table 11. Estimated communication overhead of SDAP-K.

Message Component	Bytes (Approx Size)	Communication Phase
Kerberos Service Ticket	180–250	Authentication
Authenticator	48–64	Authentication
Session Key Information	32	Authentication
Metadata Record	150–300	Storage/Update
Audit Challenge	32–64	Auditing
Audit Proof	128–256	Auditing
N-ary Authentication Path	$32 \times \text{Tree Depth}$	Auditing
Root Hash (MMH-32)	32	Verification

7.5. Storage and Retrieval Performance

Figures 11 and 12 compare the execution times of data storage and retrieval operations with those of existing schemes [7,8,12,25,26,36] and [2,7,8,28,34], respectively. In the proposed SDAP-K framework, metadata management is delegated to the Metadata Server, and integrity verification is performed using an N-ary hash tree. Consequently, the Data Owner performs fewer computational operations during storage and retrieval.

The results demonstrate that execution time increases with the number of data blocks across all schemes; however, SDAP-K exhibits lower growth than baseline approaches.

The reduction in execution time is mainly due to lightweight metadata-assisted verification and reduced communication requirements between protocol entities.

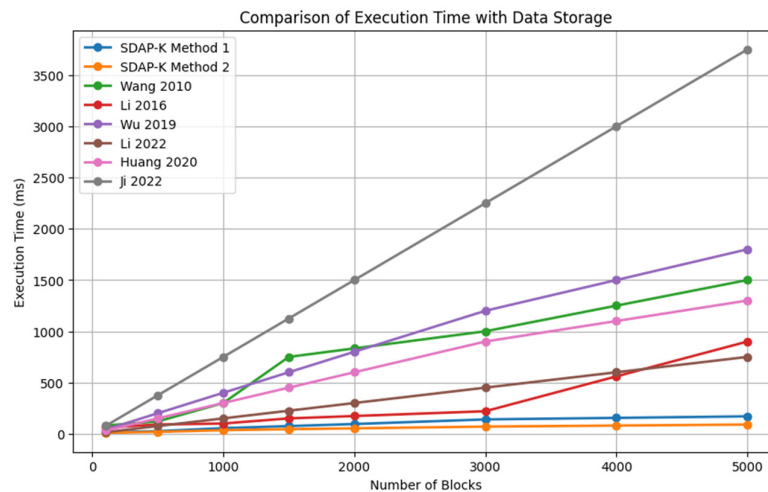


Figure 11. Computational time of SDAP-K and baseline schemes (Wang 2020 [7], Li 2016 [8], Wu 2019 [12], Li 2022 [25], Huang 2020 [26], Ji 2022[36]) for Cloud Data Storage.

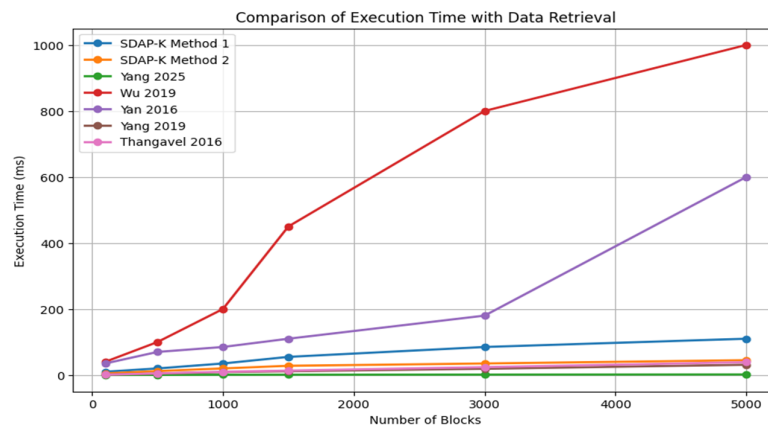


Figure 12. Computational time of SDAP-K and baseline schemes (Yang 2025 [2], Wu 2019 [7], Yan 2016 [8], Yang 2019 [28], Thangavel 2016 [34]) for Cloud Data Retrieval.

7.6. Dynamic Data Update Performance

Figure 13 shows the execution time for dynamic update operations of SDAP-K and existing schemes [2,7,8,34]. It is evaluated on insertion, modification, and deletion with different block sizes.

The proposed update mechanism synchronizes metadata through the Metadata Server while updating only the affected blocks rather than reconstructing the entire outsourced file. As a result, SDAP-K reduces update overhead and maintains efficient verification of storage correctness after data modifications.

7.7. File-Level Auditing Performance

File-level auditing challenges all blocks of the outsourced file and verifies the corresponding root hash value. The auditing complexity is: $T_{FA} = O(n)$, where n denotes the total number of blocks. Figure 14 demonstrates that SDAP-K achieves lower auditing

overhead than the baseline schemes [2,7,8,11,27,35] due to centralized metadata management and lightweight root-hash verification.

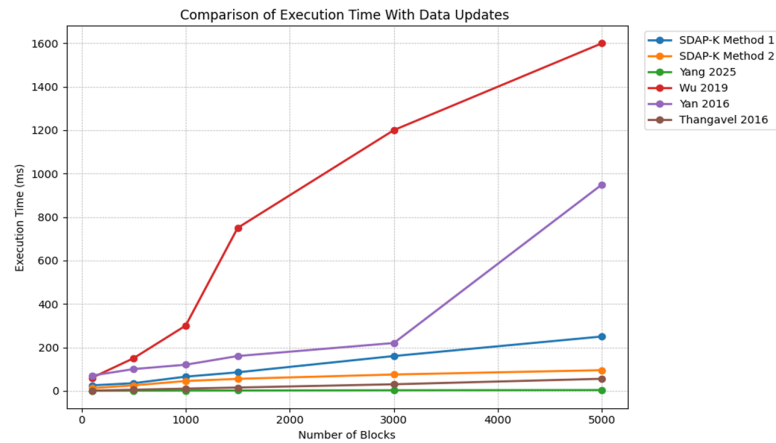


Figure 13. Computational time of SDAP-K and baseline schemes (Yang 2025 [2], Wu 2019 [7], Yan 2016 [8], Thangavel 2016 [34]) for Cloud Data Update.

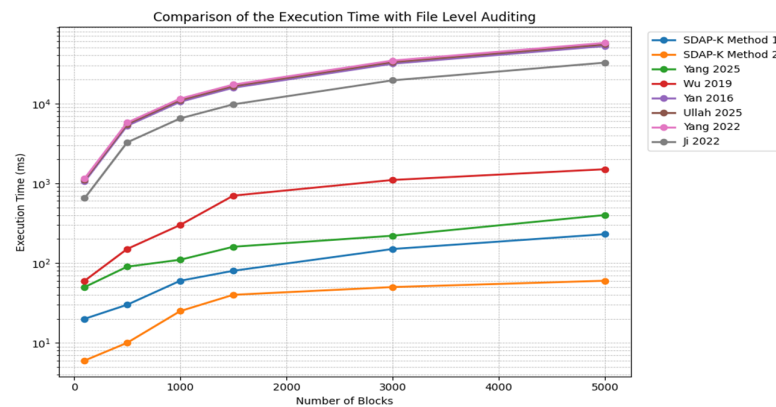


Figure 14. Computational time of SDAP-K and baseline schemes (Yang 2025 [2], Wu 2019 [7], Yan 2016 [8], Ullah 2025 [11], Yang 2022 [27], Ji 2022 [35]) for Cloud File-level auditing.

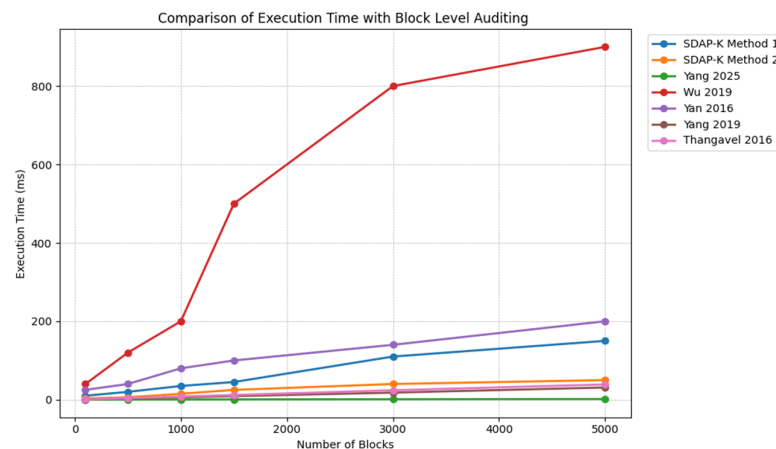


Figure 15. Computational time of SDAP-K and baseline schemes (Yang 2025 [2], Wu 2019 [7], Yan 2016 [8], Yang 2019 [28], Thangavel 2016 [34]) for Cloud Block-level auditing.

7.8. Block-Level Auditing Performance

Block-level auditing verifies only a subset of blocks selected by the challenge set.

The auditing complexity is $T_{BA} = O(k \log N^n)$

where:

- (k) is the number of challenged blocks;
- (n) is the total number of blocks;
- (N) is the N-ary tree branching factor.

The reduced tree depth $D = \lceil \log N(n) \rceil$

where:

- D = depth (height) of the N-ary hash tree;
- n = total number of data blocks (leaf nodes);
- N = branching factor of the N-ary hash tree;
- $\lceil . \rceil$ = ceiling function.

This enables efficient verification of challenged blocks and reduces communication overhead compared with full-file auditing. Figure 15 shows that SDAP-K achieves lower auditing latency against existing schemes [2,7,8,28,34] while maintaining integrity verification capability.

Effect of N-Ary Branching Factor

The performance of the proposed SDAP-K framework is significantly affected by the branching factor (N). Increasing the branching factor decreases the height of the N-ary hash tree according to $D = \lceil \log N(n) \rceil$, and reduces the number of hash computations needed during the integrity verification. However, a higher branching factor also means more sibling nodes are included in each authentication path, leading to larger proof sizes, greater communication overhead, and additional metadata storage requirements. Therefore, selecting an appropriate branching factor is a trade-off between computational efficiency and communication cost. The experimental evaluation was conducted for $N = 5, 10, 15, 20, 25$, and 30 , and the best value under the experimental setup was selected based on the minimum total execution time and communication overhead. Table 12 shows the N-ary Branching Factor on Auditing Performance.

Table 12. Effect of the N-ary branching factor on auditing performance.

Branching Factor (N)	Tree Depth	Hash Computation	Authentication Path Size	Communication Overhead	Metadata Storage	Overall Performance
5	High	High	Small	Low	Low	Baseline
10	Medium	Medium	Medium	Medium	Medium	Balanced
15	Lower	Lower	Medium	Medium	Medium	Medium
20	Low	Low	Higher	Higher	Higher	Selected for Evaluation
25	Lower	Very Low	High	High	High	Higher Communication Cost
30	Lowest	Lowest	Very High	Highest	Highest	Highest Communication Cost

7.9. Scalability Analysis

The scalability of SDAP-K was analyzed by varying the number of outsourced data blocks from 100 to 5000. The N-ary hash tree construction complexity is

$$T_{NHT} = O(n)$$

while metadata lookup remains

$$T_{MD} = O(1)$$

under indexed metadata storage.

As the branching factor increases, the tree depth decreases according to

$$D = \lceil \log_N(n) \rceil$$

which reduces path traversal during block-level verification. These results indicate that SDAP-K scales efficiently with increasing dataset size while maintaining acceptable computational and communication overhead.

7.10. Security and Overhead Tradeoff Discussion

The use of Kerberos imposes additional overhead for ticket generation and session establishment. However, this overhead is incurred only when establishing an authenticated service and is amortized over subsequent storage, retrieval, update, and auditing operations.

The centralized Metadata Server incurs metadata management overhead, but it greatly reduces the burden on the Data Owner and enables efficient auditing. Therefore, SDAP-K provides a practical balance among authenticated service access, integrity verification, and operational efficiency.

To better comprehend the performance enhancement of SDAP-K, the total execution time can be decomposed into authentication, metadata lookup, hash computation, N-ary tree construction, communication, and integrity verification. Kerberos authentication adds a one-time overhead during session establishment, while subsequent operations reuse the established session keys, minimizing authentication costs. Metadata lookup latency is low due to centralized management, while the N-ary hash tree reduces verification time by decreasing the tree depth as the branching factor increases. While the Metadata Server (MS) may become a bottleneck under high request volumes, metadata indexing and replication can effectively improve scalability and ensure efficient auditing performance.

Centralization and Deployment Tradeoffs

SDAP-K simplifies identity management and metadata coordination. It achieves this by using a centralized architecture consisting of Kerberos authentication and a Metadata Server. It fits well with the enterprise cloud environment because centralized security administration is widely accepted. However, the design creates dependencies on the Authentication Server (AS) and Metadata Server (MS), which can be performance bottlenecks or single points of failure under heavy workloads. Metadata compromise, ticket service disruption, and clock synchronization errors can also affect protocol availability and reliability. These risks can be mitigated by replicated Kerberos servers, distributed metadata storage, metadata backup, signed metadata logs, load balancing, continuous monitoring, and failover mechanisms. Fully decentralized trust models are outside the scope of the current work and an important direction for future research.

7.11. Discussion

The experimental results indicate that the proposed SDAP-K framework reduces computational and communication overhead compared with representative auditing schemes while supporting authenticated service exchange, metadata-assisted auditing, and error localization. The framework demonstrates practical scalability for increasing numbers of data blocks and provides an efficient mechanism for cloud data auditing under the assumptions considered in this work. The computations confirm that SDAP-K

always outperforms the baseline schemes for all dataset sizes. For small datasets (100–1000 blocks), the performance difference is small due to the lower computational workload. As the number of data blocks increases (1000–3000 blocks), SDAP-K achieves lower execution time due to its efficient metadata management and N-ary hash tree structure. For large datasets (3000–5000 blocks), the performance improvement is more pronounced, with better scalability and lower computational overhead than those of Wang et al. [7] and Li et al. [8]. The efficiency of the proposed SDAP-K framework is quantitatively validated with the percentage improvements reported in the corresponding tables.

7.11.1. Interpretation of Performance Results

Experimental results show that SDAP-K consistently outperforms the baseline schemes in execution time, especially for medium and large datasets. The performance improvement primarily stems from metadata-assisted auditing, reduced N-ary tree depth, and efficient block-level verification, which reduce unnecessary computation and communication during integrity auditing.

7.11.2. Impact of Kerberos Authentication

Kerberos imposes a one-time authentication overhead when a session is established. However, the generated session keys are reused for subsequent storage, retrieval, update, and auditing operations, which greatly reduces the cost of repeated authentication while providing secure mutual authentication between protocol entities.

7.11.3. Impact of Metadata Server

The centralized Metadata Server enables efficient storage and access of auditing information, reducing the Data Owner's computational burden. Centralized metadata management simplifies auditing operations, but it may become a bottleneck under heavy workloads. This limitation can be overcome by replicating metadata and distributing it;

7.11.4. Scalability and Bottlenecks

The N-ary hash tree improves scalability by reducing tree depth as the branching factor increases, thereby lowering the verification cost in block-level auditing. SDAP-K scales well with the number of data blocks; however, in a large-scale cloud deployment, the Metadata Server and Authentication Server may need to be replicated to avoid performance bottlenecks.

7.11.5. Security–Performance Tradeoff

SDAP-K provides security and performance using Kerberos-based authentication and N-ary hash tree-based integrity verification. Kerberos adds some authentication overhead, but eliminating trusted third-party auditors and reducing full-file verification operations lead to lower overall computational and communication costs, without compromising security.

7.11.6. Limitations

SDAP-K enhances security and auditing efficiency through Kerberos-based authentication and centralized metadata management, but these components also introduce trust assumptions and potential performance bottlenecks. The Authentication Server (AS) and Metadata Server (MS) may be overloaded in large-scale deployments and become a single point of failure if not well protected. These limitations can be addressed by using replicated Kerberos servers, distributed metadata storage, metadata backup, access logging, failover mechanisms, and load balancing. Therefore, SDAP-K is well-suited for enterprise

cloud environments where centralized management of identities and metadata is usually deployed and operationally feasible.

However, SDAP-K is centralized for authentication and metadata management and thus may have limited fault tolerance in highly distributed environments. Moreover, the framework has not been evaluated in decentralized or blockchain-based cloud infrastructures. Future work will explore distributed metadata management, decentralized authentication, and cloud native deployment strategies for scalability and resilience.

8. Conclusions

In this paper, the Secure Data Auditing Protocol has been enhanced with five protocols such as (i) Data Storage protocol, (ii) Data retrieval protocol, (iii) Data Updates protocol, (iv) File-level auditing protocol, and (v) Block-level auditing protocol. All the protocols include DO-AS, DO-MS, and DO-CDS exchanges. In the proposed SDAP-K, the Data Owner itself performs all operations while maintaining integrity and authentication, with lower computation and communication overhead. Since MS manages metadata, DO need not worry about retaining metadata for prestored data in CDS. The proposed framework has been implemented with the Kerberos authentication service to provide mutual authentication in exchanges. In the event of an audit failure, error localization and data correctness features have been implemented across all five protocols.

In this paper, we propose SDAP-K, a Secure Data Auditing Protocol with Kerberos authentication, which combines authenticated service access with N-ary hash-tree-based integrity verification with Modified Murmur Hashing for outsourced cloud storage. The proposed framework combines Kerberos-based authentication, metadata-assisted auditing, file and block-level integrity checking, and error localization in a unified architecture. The experimental evaluation shows that our scheme can greatly reduce the computational and communication overhead compared with representative cloud auditing schemes in the experimental setting we evaluated. The framework, however, relies on centralized Authentication and Metadata Servers, which introduce trust assumptions and possible deployment bottlenecks that need to be addressed with suitable replication and fault-tolerance mechanisms. In the future, we will focus on formal protocol verification, distributed metadata management, blockchain-assisted audit logging, large-scale cloud deployment, and privacy-preserving audit proof mechanisms to further improve the security, scalability, and robustness of the proposed framework.

Author Contributions: Conceptualization, T.M. and V.P.; methodology, T.M. and P.M.K.; software, T.M. and N.A.N.M.B.; validation, T.M. and N.A.N.M.B.; formal analysis, T.M. and N.A.N.M.B.; investigation, T.M. and N.A.N.M.B.; resources, T.M. and N.A.N.M.B.; data curation, T.M. and N.A.N.M.B.; writing—original draft preparation: T.M. and N.A.N.M.B.; writing—review and editing, T.M., P.M.K., and V.P.; visualization, N.A.N.M.B. and P.M.K.; supervision, V.P.; project administration, T.M.; funding acquisition, T.M. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the United Arab Emirates University under Research Start-Up Proposal Grant G00004612/12T048.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing does not apply to this article.

Acknowledgments: The authors are grateful to the College of Information Technology and the Research Office—United Arab Emirates University.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Guan, X.; Chang, J.; Zhang, W. Secure data sharing scheme with privacy-preserving and certificateless integrity auditing in cloud storage. *Comput. Commun.* **2024**, *224*, 285–301. <https://doi.org/10.1016/j.comcom.2024.06.013>.
2. Yang, C.; Liu, Y.; Ding, Y.; Liang, H. Secure data migration from fair contract signing and efficient data integrity auditing in cloud storage. *J. Netw. Comput. Appl.* **2025**, *239*, 104173. <https://doi.org/10.1016/j.jnca.2025.104173>.
3. Liu, Y. Build an audit Framework for data privacy Protection in Cloud Environment. *Procedia Comput. Sci.* **2024**, *247*, 166–175. <https://doi.org/10.1016/j.procs.2024.10.020>.
4. Liu, Y. The Latest Application and Security Analysis of Cryptography in Cloud Storage Data Audit. *Procedia Comput. Sci.* **2025**, *259*, 984–990. <https://doi.org/10.1016/j.procs.2025.04.052>.
5. Tiwari, S.K.; Neogi, S.G. Design and Implementation of Enhanced Security Algorithm for Hybrid Cloud using Kerberos. *SN Comput. Sci.* **2023**, *4*, 430. <https://doi.org/10.1007/s42979-023-01807-z>.
6. Jiang, H.-C.; Feng, C.-S.; Yuan, D. Enabling public and privacy-preserving auditability for cloud storage. In Proceedings of the 2016 International Conference on Machine Learning and Cybernetics (ICMLC), Jeju, Republic of Korea, 10–13 July 2016; Volume 1, pp. 24–28. <https://doi.org/10.1109/ICMLC.2016.7860872>.
7. Wang, Q.; Wang, C.; Ren, K.; Lou, W.; Li, J. Enabling public auditability and data dynamics for storage security in cloud computing. *IEEE Trans. Parallel Distrib. Syst.* **2010**, *22*, 847–859. <https://doi.org/10.1109/TPDS.2010.183>.
8. Li, J.; Zhang, L.; Liu, J.K.; Qian, H.; Dong, Z. Privacy-preserving public auditing protocol for low-performance end devices in cloud. *IEEE Trans. Inf. Forensics Secur.* **2016**, *11*, 2572–2583. <https://doi.org/10.1109/TIFS.2016.2587242>.
9. Yu, Y.; Au, M.H.; Ateniese, G.; Huang, X.; Susilo, W.; Dai, Y.; Min, G. Identity-based remote data integrity checking with perfect data privacy preserving for cloud storage. *IEEE Trans. Inf. Forensics Secur.* **2016**, *12*, 767–778. <https://doi.org/10.1109/TIFS.2016.2615853>.
10. Aujla, G.S.; Chaudhary, R.; Kumar, N.; Das, A.K.; Rodrigues, J.J. SecSVA: Secure storage, verification, and auditing of big data in the cloud environment. *IEEE Commun. Mag.* **2018**, *56*, 78–85. <https://doi.org/10.1109/MCOM.2018.1700379>.
11. Liu, Z.; Wang, S.; Duan, S.; Ren, L.; Wei, J. Dynamic data integrity auditing based on hierarchical merkle hash tree in cloud storage. *Electronics* **2023**, *12*, 717. <https://doi.org/10.3390/electronics12030717>.
12. Wu, G.; Mu, Y.; Susilo, W.; Guo, F.; Zhang, F. Privacy-preserving certificateless cloud auditing with multiple users. *Wirel. Pers. Commun.* **2019**, *106*, 1161–1182. <https://doi.org/10.1007/s11277-019-06208-1>.
13. Yan, H.; Li, J.; Han, J.; Zhang, Y. A novel efficient remote data possession checking protocol in cloud storage. *IEEE Trans. Inf. Forensics Secur.* **2016**, *12*, 78–88. <https://doi.org/10.1109/TIFS.2016.2601070>.
14. Wang, Y.; Wu, Q.; Qin, B.; Shi, W.; Deng, R.H.; Hu, J. Identity-based data outsourcing with comprehensive auditing in clouds. *IEEE Trans. Inf. Forensics Secur.* **2016**, *12*, 940–952. <https://doi.org/10.1109/TIFS.2016.2646913>.
15. Shajina, A.R.; Varalakshmi, P. A novel dual authentication protocol (DAP) for multi-owners in cloud computing. *Clust. Comput.* **2017**, *20*, 507–523. <https://doi.org/10.1007/s10586-017-0774-y>.
16. Yang, C.; Song, B.; Ding, Y.; Ou, J.; Fan, C. Efficient data integrity auditing supporting provable data update for secure cloud storage. *Wirel. Commun. Mob. Comput.* **2022**, *2022*, 5721917. <https://doi.org/10.1155/2022/5721917>.
17. Wang, L.; Guan, R.; Hu, J.; Chen, Z. A Cloud Storage Auditing Scheme Based on Identity-Based Signature. In Proceedings of the 2025 IEEE 8th Information Technology and Mechatronics Engineering Conference (ITOEC), Chongqing, China, 17–18 March 2025; Volume 8, pp. 576–581. <https://doi.org/10.1109/ITOEC63606.2025.10967619>.
18. Thangavel, M.; Varalakshmi, P.; Sridhar, S.; Sindhuja, R. Privacy Preserving Public Auditing in Cloud: Literature Review. In *Cloud Security: Concepts, Methodologies, Tools, and Applications*; IGI Global Scientific Publishing: Hershey, PA, USA, 2019; pp. 2059–2083. <https://doi.org/10.4018/978-1-5225-8176-5.ch100>.
19. Thangavel, M.; Varalakshmi, P.; Sindhuja, R.; Sridhar, S. A survey on provable data possession in cloud storage. In Proceedings of the 2016 Eighth International Conference on Advanced Computing (ICoAC), Chennai, India, 19–21 January 2017; pp. 25–31. <https://doi.org/10.1109/ICoAC.2017.7951739>.
20. Data Breaches. 2017. Available online: <https://www.crn.com/slide-shows/security/300089736/the-10-biggest-data-breaches-of-2017-so-far> (accessed on 22 July 2026).
21. Data Breaches. 2016. Available online: <http://www.crn.com/slide-shows/security/300081491/the-10-biggest-data-breaches-of-2016-so-far> (accessed on 22 July 2026).
22. Data Breaches. 2015. Available online: <https://www.crn.com/slide-shows/security/300077563/the-10-biggest-data-breaches-of-2015-so-far> (accessed on 22 July 2026).
23. Murmur Hash. Available online: <https://en.wikipedia.org/wiki/MurmurHash> (accessed on 22 July 2026).

24. Li, J.; Yan, H.; Zhang, Y. Efficient identity-based provable multi-copy data possession in multi-cloud storage. *IEEE Trans. Cloud Comput.* **2019**, *10*, 356–365. <https://doi.org/10.1109/TCC.2019.2929045>.
25. Li, R.; Wang, X.A.; Yang, H.; Niu, K.; Tang, D.; Yang, X. Efficient certificateless public integrity auditing of cloud data with designated verifier for batch audit. *J. King Saud. Univ.-Comput. Inf. Sci.* **2022**, *34*, 8079–8089. <https://doi.org/10.1016/j.jksuci.2022.07.020>.
26. Huang, L.; Zhou, J.; Zhang, G.; Zhang, M. Certificateless public verification for data storage and sharing in the cloud. *Chin. J. Electron.* **2020**, *29*, 639–647. <https://doi.org/10.1049/cje.2020.05.007>.
27. Zhou, L.; Fu, A.; Yang, G.; Wang, H.; Zhang, Y. Efficient certificateless multi-copy integrity auditing scheme supporting data dynamics. *IEEE Trans. Dependable Secur. Comput.* **2020**, *19*, 1118–1132.
28. Yang, C.; Tao, X.; Zhao, F. Publicly verifiable data transfer and deletion scheme for cloud storage. *Int. J. Distrib. Sens. Netw.* **2019**, *15*, 1550147719878999. <https://doi.org/10.1177/1550147719878999>.
29. Wang, M.; Yu, J.; Shen, W.; Hao, R. Privacy-preserving time-based auditing for secure cloud storage. *IEEE Trans. Inf. Forensics Secur.* **2024**, *19*, 7866–7878. <https://doi.org/10.1109/TIFS.2024.3449095>.
30. Zhang, W.; Jiao, H.; Yan, Z.; Wang, X.; Khan, M.K. Security analysis and improvement of a public auditing scheme for secure data storage in fog-to-cloud computing. *Comput. Secur.* **2023**, *125*, 103019. <https://doi.org/10.1016/j.cose.2022.103019>.
31. Ullah, F.; Pun, C.-M.; Mohmand, M.I.; Mahendran, R.K.; Khan, A.A.; Alhammad, S.M.; Rodrigues, J.J.; Farouk, A. Privacy-aware secure data auditing for cloud-based intelligence of things environment. *IEEE Internet Things J.* **2025**, *12*, 15288–15303. <https://doi.org/10.1109/JIOT.2025.3528117>.
32. Thangavel, M.; Varalakshmi, P.; Sridhar, S. An analysis of privacy preservation schemes in cloud computing. In Proceedings of the 2016 IEEE International Conference on Engineering and Technology (ICETECH), Coimbatore, India, 14–16 March 2016; pp. 146–151. <https://doi.org/10.1109/ICETECH.2016.7569230>.
33. Yang, C.; Liu, Y.; Zhao, F.; Zhang, S. Provable data deletion from efficient data integrity auditing and insertion in cloud storage. *Comput. Stand. Interfaces* **2022**, *82*, 103629. <https://doi.org/10.1016/j.csi.2022.103629>.
34. Yu, H.; Yang, Z.; Waqas, M.; Tu, S.; Han, Z.; Halim, Z.; Sinnott, R.O.; Parampalli, U. Efficient dynamic multi-replica auditing for the cloud with geographic location. *Future Gener. Comput. Syst.* **2021**, *125*, 285–298. <https://doi.org/10.1016/j.future.2021.05.039>.
35. Gudeme, J.R.; Pasupuleti, S.K.; Kandukuri, R. Certificateless multi-replica public integrity auditing scheme for dynamic shared data in cloud storage. *Comput. Secur.* **2021**, *103*, 102176. <https://doi.org/10.1016/j.cose.2020.102176>.
36. Ji, Y.; Shao, B.; Chang, J.; Xu, M.; Xue, R. Identity-based remote data checking with a designated verifier. *J. Cloud Comput.* **2022**, *11*, 7. <https://doi.org/10.1186/s13677-022-00279-5>.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.