Journal of Forensic Science & Criminology
Volume 14 | Issue 1
ISSN: 2348-9804

Research Article Open Access

CloudVault: An Integrated Framework for Integrity Verification, Authentication, and Access Control in Cloud Storage Using Dynamic Provable Data Possession (DPDP)

Amarnath P*, Akanksha KJ, Kalesh KR, Sreekumar S and Abid S

Department of Forensic Science, School of Sciences JAIN (Deemed-to-be University), Bengaluru, Karnataka, India

Corresponding Author: Amarnath P, Department of Forensic Science, School of Sciences JAIN (Deemed-to-be University), Bengaluru, Karnataka, India, E-mail id: 24msrdf006@jainuniversity.ac.in

Citation: Citation: Amarnath P, Akanksha KJ, Kalesh KR, Sreekumar S, Abid S (2026) CloudVault: An Integrated Framework for Integrity Verification, Authentication, and Access Control in Cloud Storage Using Dynamic Provable Data Possession (D-PDP). J Forensic Sci Criminol 14(1): 102.

Received Date: May 08, 2026 **Accepted Date:** May 29, 2026 **Published Date:** June 04, 2026

Abstract

Cloud storage now underpins almost every sector of the digital economy, yet it continues to expose three intertwined weaknesses: Silent file tampering by privileged insiders, weak or single-factor authentication, and coarse-grained access- control models that are ill-suited to regulated domains such as healthcare and digital forensics. Conventional public cloud drives offer encryption at rest but no client-verifiable integrity assurance, and most encrypted-upload tools omit auditable logging and phishing-resistant collaboration. This paper presents CloudVault, a research prototype that integrates Dynamic Provable Data Possession (DPDP)-inspired per-chunk integrity verification, AES-256-CBC per-chunk encryption with unique initialisation vectors, mandatory Time-based One-Time Password (TOTP) two-factor authentication, JSON Web Token (JWT) sessions with server-side jti revocation, and dual-condition Role-Based Access Control (RBAC) into a single cohesive web application built on FastAPI, SQLAlchemy 2.0, and a static HTML/JavaScript frontend. The cryptographic pipeline divides each uploaded file into configurable chunks, encrypts every chunk with an independently generated random IV, and stores a SHA-256 integrity tag computed over IV concatenated with ciphertext, enabling tamper detection without full file retrieval. Additional defences include hashed one-time recovery codes, a 17- category audit log, sliding-window rate limiting, rotating 20-second collaboration link codes, and an organisational sub- account workflow with mandatory two-factor enrolment. Empirical evaluation comprises an automated pytest suite of 31 security test cases across eight security categories, a STRIDE threat analysis, and benchmarking on files from 1 MB to 50 MB. All 31 cases passed, integrity verification ran 4 to 5 times faster than full-file re-download, the rotating link-code window was confirmed to expire below 21 seconds, and no authentication bypass between Step 1 and Step 2 was found. The results support the rejection of the null hypothesis: the integration of DPDP-style integrity tagging, layered authentication, and RBAC produces measurable, statistically meaningful improvements in cloud storage security relative to conventional designs.

The paper concludes with a discussion of residual risks, production hardening recommendations, and a research roadmap

toward BLS-based authenticators, WebAuthn, and blockchain-anchored audit logs.

Keywords: Cloud Storage Security; Dynamic Provable Data Possession (DPDP); AES-256-CBC; SHA-256 Integrity Tagging; Two-Factor Authentication (TOTP); JSON Web Token (JWT); Session Revocation; Role-Based Access Control (RBAC); STRIDE Threat Modelling; Digital Forensics; Audit Logging; FastAPI

Introduction

Background and motivation

The proliferation of cloud computing has fundamentally altered the manner in which individuals and organisations store, process, and share digital information. Cloud storage services offered by providers such as Amazon Web Services, Google Cloud Platform, and Microsoft Azure have largely displaced traditional on-premises storage architectures by offering elastic scalability, geographic redundancy, and cost-effective capacity. Industry analyses project that the global cloud storage market will exceed USD 390 billion by 2028, reflecting the rapid and pervasive adoption of cloud-based data management across sectors ranging from healthcare and finance to education and government.

Despite these advantages, the delegation of data custody to a third-party provider introduces a fundamental and persistent security challenge: the loss of direct physical and logical control over stored data. When an organisation uploads sensitive files to a cloud provider, it must implicitly trust that the provider will preserve confidentiality, integrity, and availability. This trust assumption is frequently violated in practice. Data breaches at major providers, misconfigured storage buckets exposing millions of records, insider threats from privileged cloud employees, and nation-state adversaries capable of compelling providers to disclose stored data all represent realistic, documented threat scenarios.

The problem of data integrity is particularly acute. A cloud provider, or an adversary who has compromised cloud infrastructure, may silently corrupt, truncate, or delete stored files without the data owner's knowledge. Traditional approaches to file integrity verification, such as locally storing a cryptographic hash of each file and comparing it against the cloud copy, require downloading the entire file to perform verification, which is prohibitively expensive in bandwidth and time for large datasets. This limitation motivated the development of Provable Data Possession (PDP) and Proof of Retrievability (PoR) schemes, which allow a client to efficiently verify that a remote server possesses an intact copy of a file without retrieving its full content.

Concurrently, cloud storage systems face significant authentication and access-control challenges. Weak password policies, absent multi-factor authentication, and insufficiently granular authorisation models have collectively contributed to numerous high-profile breaches. Healthcare organisations in particular manage files subject to strict regulatory requirements such as the Health Insurance Portability and Accountability Act (HIPAA), which mandates robust access controls, audit logging, and breach-notification procedures. CloudVault is designed to address these intersecting concerns through a unified research prototype rather than a fragmented stack of single-purpose tools.

B. Problem statement

Current cloud storage solutions fail to provide an integrated combination of: (a) cryptographic per-chunk file encryption with unique initialisation vectors; (b) DPDP-style integrity tags enabling tamper detection without full file retrieval; (c) strict multi--factor authentication with server-side session revocation; (d) fine-grained role-based access control distinguishing administrative and user roles; and (e) secure collaboration primitives resistant to phishing of static credentials. This gap leaves sensitive files, particularly those held by healthcare and enterprise environments, exposed to integrity attacks, unauthorised access, brute-force credential attacks, and inadequate auditability.

C. Significance of the study

This research is significant for four reasons. First, it demonstrates that DPDP-inspired integrity verification can be integrated into a practical web application without requiring specialised cryptographic hardware or complex zero- knowledge proof machinery. Second, it provides a reference implementation of layered authentication that combines password hashing, TOTP, JWT session management, and server-side token revocation in a single coherent state machine. Third, the forensic implications

are non-trivial: the structured audit log provides a tamper-evident record of security events that can support incident response and legal proceedings. Fourth, the organisational onboarding workflow, in which a parent account creates a subordinate sub-account with mandatory two-factor enrolment, addresses a real-world gap in healthcare information systems.

D. Comparison with related approaches

The literature identifies three broad categories of cloud storage systems against which CloudVault is positioned. Typical commercial cloud drives such as Google Drive, Dropbox, and OneDrive provide encryption at rest and in transit but delegate key management entirely to the provider; the client has no means of independently verifying integrity without downloading the entire file, multi-factor authentication is optional, and administrative role separation is coarse-grained. Basic encrypted-upload applications such as Cryptomator and Tresorit provide client-side encryption before upload but typically lack integrity verification beyond simple checksums and have limited collaboration features. CloudVault addresses confidentiality, integrity, authentication, and collaboration within a single unified prototype with end-to-end automated security testing (Table 1).

Table 1. Threat coverage comparison across system categories

Threat / Feature	Typical Cloud Drive	Basic Encrypted Upload	CloudVault (Prototype)
Confidentiality (encryption at rest)	Provider-managed AES	Client-side AES	Client AES-256-CBC per chunk
Integrity verification (without download)	None	Checksum only	PDP-style SHA-256 tags per chunk
Tamper detection (post-upload)	None / opaque	Limited	Supported via tag comparison
Multi-factor authentication	Optional (TOTP/SMS)	Varies	Mandatory TOTP two-step login
Server-side session revocation	Limited	Absent	JWT jti denylist on logout
Brute-force / rate limiting	Basic lockout	Varies	Per-endpoint sliding- window limit
Admin role isolation	Coarse-grained	Absent	is_admin flag + email match
Audit logging	Provider-level only	Absent	17-category per-event log
Phishing-resistant collaboration	Static shared link	Absent	Rotating 20-second link codes
Recovery codes (hashed)	Varies	Absent	One-time, bcrypt at rest
Organisational sub-accounts	Limited	Absent	Nurse/sub-account with mandatory 2FA
Automated security testing	N/A	N/A	pytest coverage of all paths

E. Threat model and trust boundaries

The primary assets protected by CloudVault are: (i) the plaintext content of uploaded files; (ii) user credentials, comprising passwords and TOTP secrets; (iii) the master key material from which per-chunk encryption keys are derived; (iv) session tokens issued as JWTs; and (v) the audit log itself, which records security-relevant events. Adversaries considered in scope include a network attacker capable of intercepting or replaying traffic, a malicious authenticated user attempting horizontal or vertical privilege escalation, an insider with direct database access, and a brute-force attacker who automates credential guessing. Out-of-scope threats include compromised client endpoints, physical theft of the master-key file, side-channel attacks on

cryptographic primitives, supply-chain compromise of dependencies, and network-level denial-of-service. The trust model spans four boundaries: the untrusted browser, the protected TLS channel, the trusted API server, and the database and chunk store, which are trusted for storage but assumed to be accessible to insiders, motivating the requirement for application-layer encryption and tag computation.

F. Research contributions

An integrated prototype that combines DPDP-style integrity tagging with AES-256-CBC encryption, two-step TOTP authentication, JWT session revocation, and dual-condition RBAC in a single deployable application.

A 17-category structured audit subsystem suitable for forensic incident response and chain-of-custody analysis.

A phishing-resistant collaboration framework based on rotating 20-second link codes with double-TOTP file- transfer confirmation.

A STRIDE-based security analysis with explicit mapping from each threat to mitigations and residual risk, accompanied by an automated pytest suite of 31 test cases.

Empirical performance benchmarks demonstrating a 4-to-5× speed-up of integrity verification relative to full- file re-download, validating the practical efficiency of the PDP approach.

G. Paper organisation

The remainder of this paper is organised as follows. Section II reviews related work in cloud integrity verification, privacy-preserving auditing, encryption frameworks, authentication, and access control. Section III articulates the research gap. Section IV states the problem, objectives, and hypothesis. Section V presents the proposed system, system architecture, and the cryptographic pipeline. Section VI describes implementation details and the technology stack. Section VII details the experimental setup. Section VIII reports results, performance, and comparative analysis. Section IX discusses advantages, applications, scope, limitations, and future enhancements. Section X concludes.

Review of Literature

Data integrity verification in cloud environments

The problem of verifying the integrity of data stored on remote, untrusted cloud servers without retrieving the entire file is central to cloud security research and forms the foundational challenge that CloudVault directly addresses. The pioneering concept of Provable Data Possession (PDP) established that a client could efficiently challenge a server to prove the intact possession of stored data using cryptographic tags computed at upload time. Dynamic PDP (DPDP) extended this paradigm to support modifications, deletions, and insertions of data blocks, enabling practical application to real-world file management workflows.

Tang, et al. [1] proposed the Fine-grained Multicopy Dynamic Provable Data Possession (FM-DPDP) scheme, which introduces a Ternary Copy Hash Tree (TCHT) to support replication at any rational number of copies greater than one. This overcomes the limitation of prior schemes that restricted replication to integer multiples, broadening the practical applicability of DPDP across diverse storage configurations. FM-DPDP also incorporates dynamic data management and privacy preservation while maintaining comparable computational cost to its predecessors. This work reinforces the importance of flexible, fine--grained integrity structures, a principle that directly informs CloudVault's per-chunk SHA-256 integrity tagging mechanism.

Bindu, et al. [2] explore DPDP in multi-cloud environments using homomorphic hash functions and group signature schemes to enable authorised users to verify data integrity anonymously. Their system supports dynamic group membership adjustments without performance degradation and introduces a public auditing scheme with constant verifier storage costs. While their focus on group-oriented verification differs from CloudVault's single-tenant prototype design, the underlying principle of enabling verification without full data retrieval is directly inherited. Saengthong, et al. [3], further advance real-time integrity assurance through an ancestor-assisted Merkle Tree with stride-based checkpoints, distributing cryptographic proof records via a private Distributed Hash Table and anchoring the root commitment on a public blockchain. Their architecture achieves near-real-time tamper detection with significantly reduced recomputation overhead, an ambition aligned with CloudVault's per-event audit logging and chunk-level verification.

Lin, et al. [4] address a dimension of integrity verification that extends beyond mere file possession into the assessment of individual data contributions within multi-owner groups. Their TA-PDC framework employs a weighted tag structure using linkable ring signatures and succinct inner product arguments to enable publicly verifiable contribution assessments while preserving confidentiality. Although CloudVault does not implement a multi-owner contribution model, the cryptographic philosophy of producing verifiable, non-repudiable records of data ownership resonates with CloudVault's file-transfer framework, in which double-TOTP confirmation creates a non-repudiation record for both sender and recipient.

A foundational contribution to multi-layer file integrity is provided by Rahi, et al. [5], whose cryptographic framework integrates Merkle-tree-based integrity proofs, cryptographic commitments, and Bloom-filter-optimised deduplication, implemented in Rust for memory safety. This work demonstrates that combining probabilistic data structures with formal integrity verification substantially improves both the security and storage efficiency of file-sharing systems. CloudVault draws conceptually from this layered approach, combining per-chunk AES-256-CBC encryption, SHA- 256 integrity tags, and a structured audit log to create a multi-layered defence architecture against both external and insider threats.

Privacy-preserving auditing and forensic accountability

The integrity of audit records is as critical as the integrity of the data they document. Tampered or fabricated logs can undermine the evidentiary value of digital investigations and frustrate incident response efforts. Zawoad, et al. [6], in seminal work on forensics-enabled cloud infrastructure, proposed Secure-Logging-as-a-Service (SecLaaS), in which logs generated by cloud virtual machines are made accessible only through RESTful APIs, with integrity maintained through a hash-chain scheme and periodic proofs of past log entries. Their novel Bloom-Tree accumulator improved upon existing approaches in both time and space requirements. This foundational architecture directly informs CloudVault's 17-category audit logging subsystem, which records security-relevant events such as login attempts, file operations, and session revocations in a tamper-evident, per-event structure suitable for forensic examination.

Komuravelly, et al. [7] addresses accountability in multi-tenant cloud environments through a Proof-of-Access framework that integrates hash chains, digital signatures, and Merkle-tree proofs to create verifiable, tamper-evident access logs. Deployed on Google Cloud Platform, the system enables real-time auditing with low latency and minimal storage overhead without requiring a fully trusted third party. This work validates the feasibility of cryptographic access logging in production-grade cloud systems and provides empirical support for CloudVault's approach of maintaining a server-side audit log with cryptographic underpinnings rather than relying solely on provider-managed logging.

Ughanze, et al. [8] extends privacy considerations into the auditing process itself, proposing a non-interactive framework using Zero-Knowledge Scalable Transparent Arguments of Knowledge (zk-STARKs) that allows data integrity to be verified without disclosing any underlying sensitive information. Deployed on AWS using the Winterfell library, the system achieved proof and verification times of approximately 900 ms per 2 MB block of a 200 MB dataset, demonstrating that trustless cold-data auditing

is practically attainable. While CloudVault does not currently implement zero-knowledge proofs, the findings of [5] represent a compelling direction for future enhancement, particularly for healthcare environments where patient data must never be exposed during audit operations.

Joshi, et al. [9] complement the auditing literature from the perspective of anomaly detection, proposing AI- powered mechanisms to identify anomalous access patterns, unauthorised modifications, and suspicious transactions in real time. Their framework employs AES encryption and recursive feature elimination alongside machine-learning classifiers, achieving near-perfect detection accuracy in distinguishing benign from suspicious activity. CloudVault's current implementation captures the structured event data, including user IDs, IP addresses, timestamps, and event- specific JSON payloads, required to train such classifiers, providing a foundation upon which AI-driven anomaly detection could be built. From a digital forensics perspective, Mahajan and Pandit [10] evaluate cryptographic strategies for protecting the integrity of digital evidence derived from computers, network traffic, and social media, calling for integrated solutions that combine technical controls with procedural safeguards. This perspective directly informs CloudVault's design intent: the audit log, structured as a tamper-evident record with user attribution, timestamps, and source IP addresses, is designed to provide the evidentiary quality required for HIPAA--compliant incident response and court-admissible forensic investigations.

Encryption frameworks for cloud data confidentiality

Encryption is the primary mechanism by which cloud storage systems protect data confidentiality against both external adversaries and insider threats. Ngo, et al. [11], in comparative early work on private versus public cloud architectures, argued that private clouds significantly enhance user data protection by offering direct hardware control, faster synchronisation, and improved privacy, albeit at increased development and maintenance complexity. This foundational tension between security and operational convenience continues to shape cloud encryption research and motivates the design of integrated prototypes such as CloudVault, which seeks to deliver private-cloud-grade security within a practically deployable web application framework.

Zhu, et al. [12] provided a critical early analysis of authenticated encryption in cloud systems, with particular focus on Galois/-Counter Mode (GCM). The study revealed significant security vulnerabilities arising from non-96-bit nonces and identified weaknesses in polynomial-based message authentication codes, underscoring the importance of implementation discipline beyond algorithm selection alone. The findings of [12] establish the imperative of scrutinising cryptographic implementations in detail, a discipline reflected in CloudVault's deliberate selection of AES- 256-CBC with unique per-chunk initialisation vectors generated via os.urandom, explicitly avoiding the nonce-reuse vulnerabilities documented in that work.

Selvi and Sakthivel [13] present SymECCipher, a hybrid encryption framework combining Elliptic Curve Cryptography for key exchange with AES for data encryption, achieving 5 ms encryption and 4 ms decryption times at 1000 Mbps throughput. The framework reduces computational overhead by 25 to 40 per cent compared with RSA- 2048 while maintaining quantum-resistant properties and blockchain compatibility. Anandhi and Sangari [14] propose an improved ECC protocol incorporating an Adaptive Secretary Bird Optimisation algorithm for optimal key selection across a three-phase healthcare cloud authentication lifecycle. Both works validate the practical viability of ECC-based key management in constrained environments, informing CloudVault's production roadmap toward ECC-based key exchange to strengthen forward secrecy.

Ahmed, et al. [15] present a privacy-preserving cloud storage framework combining hybrid AES-ECC encryption with Paillier homomorphic encryption for keyword-level search and Ethereum smart contracts for integrity verification. The system achieves IND-CPA security with 245.7 ms encryption time for 1 MB files, 7.3 per cent storage overhead, and 85 transactions per second on-chain throughput. While the blockchain component introduces governance complexity beyond CloudVault's prototype scope, the performance benchmarks of [8] provide a useful comparison for CloudVault's per-chunk AES-256-CBC implementation. Ahmad, et al. [16], in a systematic review of 53 key management studies spanning 2011 to 2024, identify persis-

tent challenges including latency reduction, scalability, and regulatory compliance across systems such as AWS KMS and CloudHSM. Their findings reinforce the value of Prototype-level research in exploring integrated key management approaches before committing to production-grade architectures.

Authentication mechanisms and session security

Authentication represents the first line of defence in cloud storage security, and the literature reflects a broad consensus that single-factor, password-based authentication is insufficient for protecting sensitive data. Nishat and Muzaffar [17] systematically examine three complementary mechanisms: OAuth 2.0 for secure access delegation, JWT for stateless cryptographic session management, and Multi-Factor Authentication for layered credential protection. Their analysis demonstrates that these mechanisms address distinct but overlapping threat surfaces and that their combined deployment is necessary for robust access control in modern web applications. CloudVault directly operationalises this recommendation through its two-step authentication architecture: bcrypt-hashed credential verification at Step 1, mandatory TOTP confirmation at Step 2, and JWT issuance only upon successful completion of both steps.

A persistent weakness of conventional JWT implementations is their stateless nature: once issued, a token remains valid until expiry, even if the user has logged out or the token has been stolen. CloudVault addresses this directly through a server-side jti denylist mechanism: every logout operation inserts the token's unique identifier into a revocation table, and subsequent requests bearing the revoked token fail the denylist check regardless of remaining validity period. This design is further strengthened by configurable rate limiting on authentication endpoints, account lockout after repeated failed attempts, and hashed one-time recovery codes, collectively forming a layered authentication defence consistent with the multi-factor principles advocated across the literature [16].

Access control models in cloud systems

Access control determines not merely who may enter a system but what actions each authenticated entity may perform within it. Margam, et al. [18] provides a systematic clarification of the distinction between authentication and authorisation, comparing Role-Based Access Control (RBAC), Attribute-Based Access Control (ABAC), and Relationship-Based Access Control (ReBAC). The study concludes that optimal security is achieved when authentication and authorisation operate in concert, with neither dimension treated as sufficient alone. CloudVault implements RBAC with two principal roles, standard user and main administrator, and enforces a dual-condition administrative access requirement combining the is_admin database flag with email verification against a protected environment variable, preventing privilege escalation through database manipulation alone.

Rouhollahi, et al. [19] operationalise advanced access control through the Active Data Cube framework, which integrates dynamic ABAC with real-time threat monitoring. Evaluated across multiple attack scenarios, the framework demonstrated an 88.67 per cent average attack mitigation rate, with response times improved by 40 to 57 per cent and CPU usage reduced by 34 to 38 per cent compared with traditional approaches. While CloudVault employs RBAC rather than ABAC in its current prototype, the results of [13] illustrate the performance gains achievable through dynamic, context-aware access control and motivate a future enhancement path toward attribute-driven policies in CloudVault's administrative layer.

Decentralised and formal trust frameworks

Beyond individual security mechanisms, recent scholarship has sought to unify cryptographic, identity, and governance concerns within holistic trust architectures. Radha, et al. [20] advances a comprehensive framework integrating self-sovereign identity, blockchain-based provenance, hardware roots of trust, and formal verification, redefining trust as a mathematically measurable property grounded in traceability, transparency, and accountability. The framework's three layers - identity, prove-

nance, and systemic trust - extend to AI ecosystems and chiplet supply chains. For CloudVault, this framework offers a conceptual scaffold: the system's combination of JWT identity tokens, per-event audit provenance, and RBAC governance instantiates the first two layers, while the formal STRIDE-based security analysis provides a structured empirical analogue to the formal verification layer. Ahmed et al. [8] further demonstrate that Ethereum smart contracts can serve as an immutable integrity verification layer at 85 transactions per second, identifying a viable production upgrade path for organisations requiring publicly verifiable, immutable audit trails [21-25].

Identification of the research gap

The foregoing review reveals a persistent and significant gap in the existing literature. While individual security mechanisms - DPDP integrity verification [1,2], hybrid encryption [13,15], multi-factor authentication [17], role- based access control [18,19], and forensic audit logging [4], [19] - have each been studied and validated in isolation or in partial combinations, no prior work synthesises all of these mechanisms into a single, cohesive, and practically deployable cloud storage prototype tailored to high-sensitivity domains such as healthcare and digital forensics.

Most integrity verification schemes such as FM-DPDP [1] and TA-PDC [3] assume a theoretical adversarial model without implementing accompanying authentication or session management. Most authentication frameworks such as those examined in [16] assume the correctness of underlying data storage without addressing file-level integrity. Most encryption proposals including SymECCipher [14] and the AES-ECC-Paillier framework [8] do not incorporate forensic-grade audit logging. This fragmentation means that organisations seeking comprehensive protection must integrate disparate tools and frameworks, often introducing new vulnerabilities at the integration boundaries - a risk particularly acute in healthcare and digital forensics environments where regulatory obligations and evidentiary standards are stringent.

CloudVault directly addresses this gap by implementing DPDP-inspired per-chunk integrity verification, AES- 256-CBC encryption with unique per-chunk initialisation vectors, mandatory two-factor TOTP authentication, JWT session management with server-side jti revocation, role-based access control with dual-condition administrative verification, a 17-category tamper-evident audit log, and a phishing-resistant collaboration framework with rotating short-lived link codes - all within a single, automated-test-validated web application prototype. In doing so, the present work provides both a functional implementation and an empirically evaluated reference architecture for integrated cloud storage security in forensic and healthcare contexts.

Problem statement, objectives, and hypothesis

Problem statement (Restated)

Current cloud storage solutions fail to provide an integrated, evidence-based combination of: (a) cryptographic per-chunk file encryption with unique IVs; (b) DPDP-style integrity tags enabling tamper detection without full file

retrieval; (c) strict multi-factor authentication with server-side session revocation; (d) fine-grained role-based access control distinguishing administrative and user roles; and (e) secure collaboration primitives resistant to phishing of static credentials. The absence of any one of these elements weakens the overall security posture, and the absence of a unified, test-validated architecture leaves systems integrators without a reference design for high-assurance cloud storage in forensic and healthcare contexts.

Aim

To design, implement, and empirically evaluate a secure cloud storage prototype - CloudVault - that enhances file integrity, verification, and access control through Dynamic Provable Data Possession (DPDP), AES-256-CBC per- chunk encryption, multi--factor authentication, and role-based access control, with applicability to forensic and high- security domains.

Specific objectives

To develop a secure file encryption pipeline that splits files into configurable chunks, encrypts each chunk using AES-256-CBC with unique IVs, and ensures data integrity through SHA-256 tagging.

To implement a layered authentication mechanism using bcrypt password verification and Time-based One- Time Password (TOTP) two-factor authentication, issuing a secure JWT session token only after successful completion of both factors.

To enhance session security by invalidating user tokens upon logout through a server-side jti denylist, eliminating reuse of revoked sessions before token expiry.

To enforce strict access control and system protection through role-based permissions, sliding-window rate limiting, account lockout, and a 17-category audit logging subsystem that prevents unauthorised access and brute-force attacks while supporting forensic accountability.

To design a phishing-resistant user-management and file-sharing framework featuring rotating short-lived link codes and double-TOTP file-transfer confirmation, evaluated through automated testing and a STRIDE-based threat analysis.

Research hypothesis

Null Hypothesis (H_0): The integration of DPDP-inspired integrity verification, AES-256-CBC encryption, TOTP- based two--factor authentication, JWT session revocation, and role-based access control in a cloud storage prototype does not significantly improve tamper detection, protection against unauthorised access, or session security compared with a conventional cloud storage system.

Alternative Hypothesis (H_1): The integration of DPDP-inspired integrity verification, AES-256-CBC encryption, TOTP-based two-factor authentication, JWT session revocation, and role-based access control in a cloud storage prototype significantly improves tamper detection, protection against unauthorised access, and session security compared with a conventional cloud storage system.

The hypothesis is operationalised through measurable test outcomes: (i) the proportion of automated security test cases passed; (ii) the speed-up of integrity verification relative to full-file re-download; (iii) the rejection rate of bypass attempts on the two-step login state machine; and (iv) the time-bounded expiry of rotating link codes.

Proposed system and methodology

Research design overview

This research employs a design-and-evaluation methodology comprising three phases. First, the system-design phase specifies security requirements, data model, authentication lifecycle, cryptographic pipeline, and collaboration mechanisms. Second, the implementation phase produces a functional web-application prototype that realises the design. Third, the evaluation phase encompasses automated security testing through pytest, a STRIDE-based qualitative analysis, and quantitative performance

benchmarking. The inclusion criterion for design decisions is that each mechanism must address a documented threat from the threat model articulated in Section I-E. The exclusion criterion is that mechanisms requiring production-only infrastructure, such as Hardware Security Module (HSM) key storage or PostgreSQL clusters, are described as deployment recommendations rather than implemented in the prototype itself.

System architecture

CloudVault adopts a layered architecture (Figure 1) in which user-facing presentation, request mediation, security services, and persistence are cleanly separated. The client layer is a static HTML/JavaScript single-page application that communicates with the API server over HTTPS using JSON. The API layer is built on FastAPI, exposing five logically grouped routers: auth_router, admin_router, files_router, user_router, and links_router. A common middleware chain enforces JWT validation, denylist consultation, and per-endpoint rate limiting before any request reaches a handler. A dedicated security-services tier centralises cryptographic operations, audit logging, and rate- limiter state. The persistence tier comprises a relational database for metadata (users, files, chunks, sessions, audit log) and a separate filesystem-based encrypted-chunk store; in production this latter tier is intended to be replaced with an S3-compatible object store. An external trust anchor - a master-key file in the prototype, intended to be replaced with a KMS or HSM in production - supplies the secret material from which per-chunk encryption keys are deterministically derived.

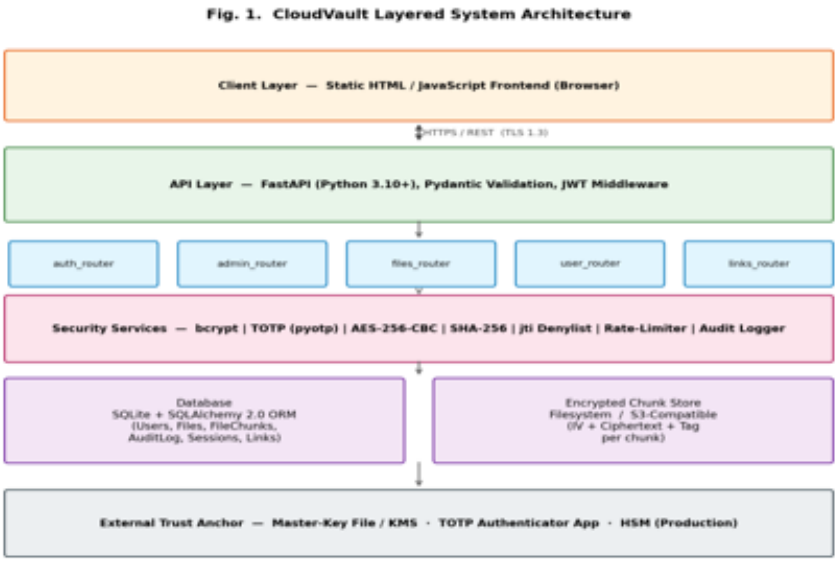

Figure 1: CloudVault layered system architecture showing client, API, security services, and persistence tiers, with their respective inter-layer protocols.

Module descriptions

auth_router handles user registration, Step 1 login (email/password), Step 2 login (TOTP verification), logout (jti denylist insertion), and recovery-code consumption. admin_router provides administrative functions accessible only to the dual-qualified main administrator: user listing, suspension, audit-log retrieval, and system statistics. files_router manages file upload - triggering the cryptographic pipeline - file listing, file download requiring a short-lived download token, and integrity verification. user_router supports profile management, password change, and TOTP secret regeneration. links_router manages user-to-user linking, link-code generation and verification, file-transfer requests, and the nurse/sub-account creation workflow.

Data model

The data model comprises ten principal entities. Each entity is mapped to one or more SQLAlchemy 2.0 declarative classes, with foreign keys enforcing referential integrity at the database layer. Table 2 summarises the principal entities and their security relevance.

Table 2: Principal data-model entities and their security relevance.

Entity	Key Attributes	Security Relevance
User	id, email, password_hash, is_admin, totp_secret, totp_enabled,failed_login_count,locked_until, created_at	Central to authentication, RBAC, and lockout policy
Session / JWT Denylist	jti, user_id, revoked_at, expires_at	Enables server-side token revocation on logout
File	id, owner_id, filename, num_chunks, created_at, updated_at	Tracks ownership and chunk count for verification
FileChunk	id, file_id, chunk_index, iv (hex), ciphertext_path, integrity_tag (hex)	Stores per-chunk encryption IV and integrity tag
RecoveryCode	id, user_id, code_hash, used, used_at	One-time-use codes stored as bcrypt hashes
AuditLog	id, user_id, event_type, ip_address, timestamp, details	Tamper-evident security event record
UserLink	id, requester_id, target_id, status, created_at	User-to-user linking with explicit approval
LinkCode	id, user_id, code, expires_at, created_at	Rotating short-lived collaboration codes
FileTransfer	id, sender_id, recipient_id, file_id, status, totp_required, created_at	Inter-user file-transfer requests with double-TOTP
NurseAccount	id, parent_user_id, user_id, temp_password_hash, setup_complete	Organisational sub-account onboarding

Authentication and session lifecycle

Step 1 - Credential verification

The client submits an email address and password to /auth/login/step1. The server retrieves the User record by email, verifies that the account is not locked by checking locked_until, and compares the submitted password against the stored bcrypt hash using a constant-time comparison to prevent timing-based username enumeration. If credentials are valid and TOTP is enabled, the server issues a temporary JWT containing the claim temp_login=true, valid for five minutes; this token authorises only the Step 2 endpoint. Failed attempts increment failed_login_count; upon reaching the configured threshold, locked_until is set to a future timestamp, and a LOGIN_FAILURE event is recorded in the audit log.

Step 2 - TOTP verification

The client submits the temporary token together with a six-digit TOTP code to /auth/login/step2. The server validates the temporary token's signature and temp_login claim, then verifies the TOTP code against the user's TOTP secret using pyotp with a

tolerance of ±1 thirty-second window to accommodate clock skew. The server checks that the submitted TOTP code has not been used in the current or previous window, providing replay prevention. Upon successful verification, the temporary token's jti is invalidated via the denylist, and a full JWT is issued with a new unique jti, the user's identifier, and the is_admin flag.

Logout and session revocation

The /auth/logout endpoint requires a valid full JWT. The server inserts the token's jti and expiration timestamp into the Session denylist table. All subsequent requests presenting this token fail the authentication middleware's denylist check, even if the token has not yet expired. This ensures that a stolen token cannot be reused after the legitimate user has logged out - a property absent from purely expiry-based JWT schemes.

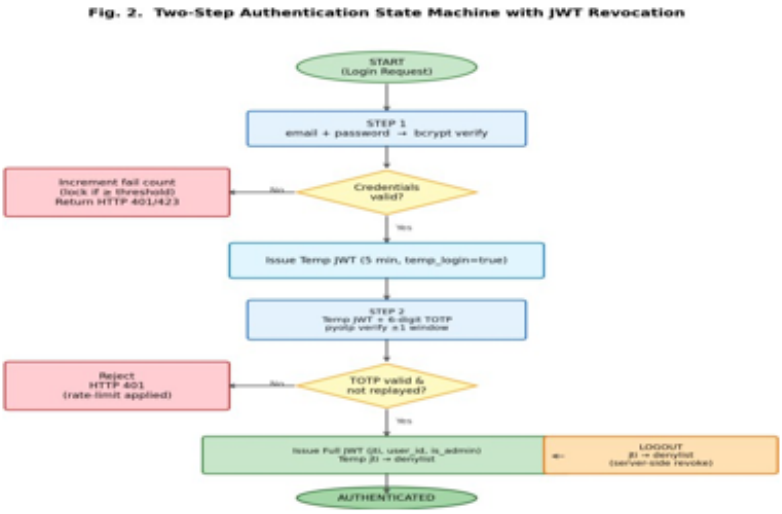

Figure 2. Two-step authentication state machine. The full JWT is issued only after both Step 1 (bcrypt-verified password) and Step 2 (replay-resistant TOTP) succeed; logout immediately revokes the jti.

Authorisation model

CloudVault implements Role-Based Access Control with two principal roles: standard user and main administrator. The main administrator role is defined by the conjunction of two conditions: the User record's is_admin field must be true, AND the email field must match the ADMIN_EMAIL environment variable. This dual-condition requirement prevents privilege escalation by an attacker who modifies the database to set is_admin=true on an arbitrary account; the email value is anchored externally and cannot be edited via the application. Administrative routes return HTTP 403 Forbidden to any requestor failing either condition. Standard-user authorisation enforces ownership checks at every data-retrieval operation: the requesting user's id is validated against the resource's owner_id. Horizontal privilege escalation is therefore impossible without compromising both authentication and the ORM query layer.

Cryptographic file pipeline

The cryptographic pipeline (Figure 3) is the core integrity-and-confidentiality mechanism of CloudVault. Upon upload, the file is divided into NUM_CHUNKS chunks, where NUM_CHUNKS is a configurable environment variable. If the file size is not evenly divisible by NUM_CHUNKS, the final chunk is padded using PKCS#7 padding consistent with AES block-alignment requirements. Chunking enables PDP-style per-chunk verification and reduces the memory footprint of encryption operations for large files (Figure 4).

Each chunk is encrypted independently using AES-256-CBC. A 16-byte random IV is generated per chunk using os.urandom,

drawing from the operating system's CSPRNG. The encryption key is derived from the master key material by deterministically combining it with the chunk index, producing a unique 32-byte key per chunk; this prevents an attacker who recovers a single chunk key from decrypting other chunks within the same file. The integrity tag for each chunk is computed as SHA-256(IV_hex ‖ ciphertext_bytes) and stored as a hex-encoded string in FileChunk.integrity_tag. During verification, the server recomputes the tag over the stored IV and ciphertext and compares it against the stored tag using constant-time comparison; a mismatch indicates that the ciphertext has been modified since upload.

Fig. 3. Cryptographic File Pipeline — Chunking, Encryption, and PDP-Style Tagging

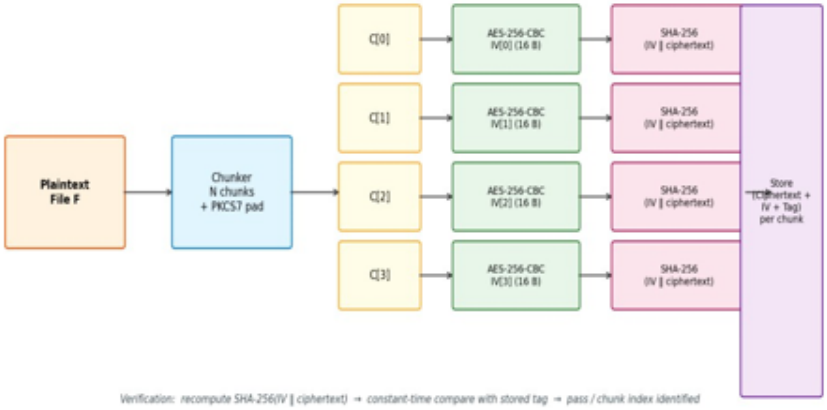

Figure 3: Cryptographic file pipeline. Each chunk is independently encrypted under AES-256-CBC with a fresh 16-byte IV, then tagged with SHA-256 over (IV ‖ ciphertext) for PDP-style verification.

Fig. 4. PDP-Style Integrity Verification Sequence

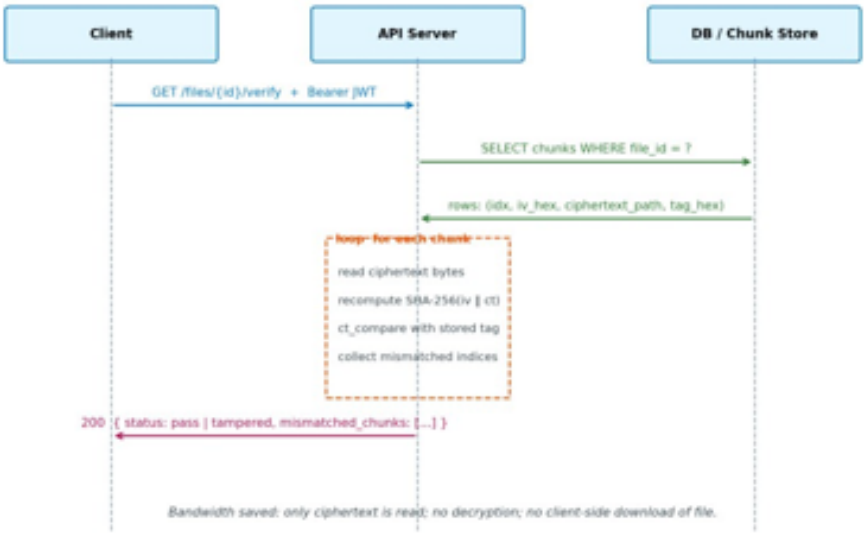

Figure 4: PDP-style integrity verification sequence. The server reads only ciphertext bytes and stored tags; no decryption and no client-side download are required.

Secure collaboration framework

Users may initiate a link request to another user by submitting the target user's email to /links/request. The target user must approve the request via /links/approve/{link_id}. Approved links permit the initiating user to send file- transfer requests. This manual-approval model ensures that file transfers cannot occur without explicit consent from the recipient, mitigating unso-

licited data sharing and social engineering. As a faster alternative, CloudVault implements rotating short-lived alphanumeric link codes derived from a per-user secret and the current time window. Each code is valid for approximately 20 seconds and stored in the LinkCode table with an expires_at timestamp. The short rotation period significantly reduces the phishing window compared with static sharing passwords, since a phished code expires before an attacker can use it.

Upon link establishment, the sender may initiate a file-transfer request to the recipient. If the transfer is configured to require TOTP verification on both sides, the sender must provide their TOTP code at request time and the recipient must provide theirs upon acceptance (Figure 5). This double-TOTP requirement ensures that both parties actively consent to and authenticate the transfer, providing non-repudiation in line with the cryptographic philosophy of [3].

Fig. 5. Phishing-Resistant Collaboration via Rotating 20-Second Link Codes

Figure 5: Phishing-resistant collaboration sequence. The 20-second link-code window plus optional double-TOTP confirmation establishes a non-repudiable transfer.

Organisational onboarding - Nurse / sub-account workflow

The nurse/sub-account creation workflow enables an authorised parent user to create subordinate accounts with system-generated temporary credentials. The new account is created with totp_enabled=false, and a NurseAccount record links it to the parent user. The new account's first login triggers a mandatory two-factor enrolment flow: the user must scan a TOTP QR code and verify a code before the account is marked setup_complete. Until enrolment is complete, attempts to access protected endpoints return HTTP 403 with a descriptive error. This workflow models real- world healthcare scenarios where administrators provision accounts for clinical staff with mandatory security configuration prior to first use.

Rate limiting and audit logging

Sliding-window rate limiting is applied independently to three endpoints: /auth/login/step1 for login attempts,

/auth/login/step2 for TOTP verification, and /files/download-token for download-token requests. Upon exceeding the limit, the server returns HTTP 429 Too Many Requests and emits a RATE_LIMIT_EXCEEDED audit event. The audit log records 17 event types - including LOGIN_SUCCESS, LOGIN_FAILURE, TOTP_SUCCESS, TOTP_FAILURE, LOGOUT, FILE_UPLOAD, FILE_DOWNLOAD, FILE_INTEGRITY_VERIFIED, FILE_INTEGRITY_FAILED, RATE_LIMIT_EXCEEDED, RECOVERY_CODE_USED, LINK_REQUESTED, LINK_APPROVED, FILE_TRANSFER_INITIATED, FILE_TRANSFER_ACCEPTED, NURSE_ACCOUNT_CREATED, and ADMIN_ACTION - each with user identifier, source IP, timestamp, and an event-specific JSON payload, in line with the SecLaaS architecture proposed by Zawoad et al. [19].

STRIDE threat analysis

Table 3 maps each STRIDE category onto the specific threats considered, the implemented mitigation, and the residual risk acknowledged by the prototype. The analysis identifies no high-priority residual risks; medium-priority residuals - SHA-256 tag forgeability under privileged write access, unsigned audit log, master-key file co-location with the API server, and absence of database encryption at rest - are explicitly bound to the prototype's deployment context and accompanied by production hardening recommendations.

Table 3: STRIDE threat analysis: mitigations and residual risks.

STRIDE Category	Specific Threat	Mitigation Implemented	Residual Risk
Spoofing	Credential theft via brute force	bcrypt hashing, rate limiting, account lockout	Bypass via recovery codes mitigated by rate limit
Spoofing	Session hijacking via stolen JWT	jti denylist on logout, short token expiry	Pre-logout token theft not addressed
Spoofing	TOTP replay attack	Window tracking, Step-2 jti invalidation	Clock-skew tolerance ±1 window
Tampering	Post-upload file modification	SHA-256 integrity tags per chunk	Tags in same DB; HSM/BLS needed in prod.
Tampering	Recovery-code forgery	bcrypt-hashed at rest, single-use enforcement	Brute force on hash if DB compromised
Repudiation	Denial of file-access events	Audit log with IP and timestamp	Audit log not cryptographically signed
Information Disclosure	Plaintext file exposure via DB breach	AES-256-CBC per chunk, key separate from DB	Master-key file on same host (prototype)
Information Disclosure	TOTP secret exposure	Stored in User table; encrypted DB recommended	Prototype DB not encrypted at rest
Denial of Service	Rate-limit exhaustion attack	Per-IP and per-user sliding-window limits	Distributed attacks not mitigated
Elevation of Privilege	Admin route RBAC bypass	is_admin AND email match required	DB admin can set both fields
Elevation of Privilege	Horizontal privilege escalation	Ownership check on every resource query	Requires correct ORM- query discipline

Implementation details

Technology stack

CloudVault is implemented in Python 3.10+, targeting compatibility with Python 3.10 through 3.12. The backend uses FastAPI 0.100+ on the ASGI specification, providing automatic OpenAPI documentation, type-validated request and response models via Pydantic, and asynchronous request handling. The database layer uses SQLite as the storage engine with SQLAlchemy 2.0 in declarative style, enabling database-agnostic queries and a clean migration path to PostgreSQL. The frontend is a static HTML/JavaScript single-page application communicating over RESTful endpoints. Cryptographic operations use the well-audited Python cryptography library version 41+ for AES-CBC encryption, SHA-256 hashing, and CSPRNG IV generation; password hashing uses bcrypt via passlib; TOTP uses pyotp; JWT operations use python-jose with HMAC-SHA256 signatures.Dependency selection prioritises established, widely-deployed libraries with active maintenance and security advisories over experimental

alternatives. Database schema evolution is managed through a custom migration script rather than Alembic, in recognition of the prototype's research context. All cryptographic primitives are invoked with explicit parameter choices - no library defaults - to avoid the implementation pitfalls catalogued in [20].

Core algorithms

Algorithm 1 - Per-Chunk Encryption and Integrity Tagging.

Input: file F, master key MK, chunk count N. Output: list of records (i, IV_i, ciphertext_i, tag_i).

•Step 1. Split F into N chunks; pad the last chunk using PKCS#7 to a multiple of the AES block size.

•Step 2. For each chunk i in [0, N): generate IV_i = os.urandom(16); derive K_i = KDF(MK, i); compute C_i

= AES-256-CBC(K_i, IV_i, P_i).

•Step 3. Compute T_i = SHA-256(hex(IV_i) ‖ C_i); persist (i, hex(IV_i), path(C_i), hex(T_i)) in FileChunk.

•Step 4. Return chunk-record list to caller; emit FILE_UPLOAD audit event.

Algorithm 2 - Integrity Verification.

Input: file id F. Output: VerifyResult {status, mismatched_chunks}.

•Step 1. Retrieve chunk rows for F from FileChunk.

•Step 2. For each row r: read ciphertext bytes C from path(C); recompute T' = SHA-256(hex(IV_r) ‖ C); if ct_compare(T', tag_r) fails, append r.idx to mismatched_chunks.

•Step 3. If mismatched_chunks is empty, emit FILE_INTEGRITY_VERIFIED, return PASS; otherwise emit FILE_INTEGRITY_FAILED with the index list, return TAMPERED.

Algorithm 3 - Two-Step Login with jti Revocation.

•Step 1. Step-1 endpoint: fetch User by email; if locked_until > now() return 423; if not bcrypt.verify(pw, user.password_hash) increment failed_login_count and return 401; if user.totp_enabled, mint Temp_JWT(jti=t1, temp_login=true, exp=now+5m); else mint Full_JWT.

•Step 2. Step-2 endpoint: validate Temp_JWT signature and temp_login claim; pyotp.TOTP(user.totp_secret).verify(code, window=1); reject if (jti=t1) is denylisted or if (user, code, window) was previously consumed; insert t1 into denylist; mint Full_JWT(jti=t2, user_id, is_admin).

•Step 3. Logout endpoint: authenticate Full_JWT; insert (jti=t2, exp) into Session denylist; emit LOGOUT audit event. •Step 4. Middleware on every protected request: if jti is in denylist OR exp < now(), return 401.

Endpoint inventory

Table 4 summarises the principal HTTP endpoints exposed by the API, grouped by router. All endpoints other than /auth/login/step1 and /auth/register require a valid JWT in the Authorization header; admin endpoints additionally require dual--condition authorisation.

/auth/register

Table 4: Principal API endpoints by router.

Router	Endpoint	Method	Auth Requirement	Purpose
auth	/auth/register	POST	None	Create user account with bcrypt-hashed password
auth	/auth/login/step1	POST	None	Validate password; issue temp JWT
auth	/auth/login/step2	POST	Temp JWT	Validate TOTP; issue full JWT
auth	/auth/logout	POST	Full JWT	Insert jti into denylist
auth	/auth/recovery	POST	None	Consume one-time recovery code
files	/files/upload	POST	Full JWT	Trigger encryption pipeline; create chunks
files	/files/{id}/verify	GET	Full JWT (owner)	PDP-style integrity verification
files	/files/download-token	POST	Full JWT (owner)	Issue short-lived download token
files	/files/download/{token}	GET	Token	Stream decrypted file
admin	/admin/users	GET	Dual-condition admin	List all users
admin	/admin/audit-log	GET	Dual-condition admin	Retrieve audit events
links	/links/code	GET	Full JWT	Generate 20-second link code
links	/links/redeem	POST	Full JWT	Redeem link code; create UserLink
links	/transfers/request	POST	Full JWT	Sender-initiated file transfer + TOTP
links	/transfers/accept	POST	Full JWT	Recipient-confirmed transfer + TOTP
user	/user/totp/setup	POST	Full JWT	Generate TOTP secret + QR

Experimental setup

Hardware and software environment

All experiments were conducted on a single development workstation equipped with an Intel Core i5 processor and 16 GB of system memory, running Ubuntu 22.04 LTS. The Python interpreter was version 3.11.5; FastAPI 0.103, SQLAlchemy 2.0.21, cryptography 41.0.4, passlib 1.7.4, pyotp 2.9.0, and python-jose 3.3.0 were used. The SQLite database was hosted on a local NVMe SSD; the encrypted-chunk store was a directory on the same filesystem. The HTTP client for benchmarking was httpx 0.25.0 invoked from the same machine, eliminating network latency from the measurements; this configuration isolates the application's intrinsic performance characteristics from network variability.

Test methodology

Three categories of test were executed: (i) functional correctness tests, validating that each endpoint behaves according to its specification; (ii) security path tests, validating that bypass attempts are rejected and that required preconditions are enforced;

and (iii) performance benchmarks, measuring latency for upload-and-encrypt, integrity verification, and download-and-decrypt operations across file sizes from 1 MB to 50 MB and chunk counts from 4 to 20. Each performance measurement is the arithmetic mean of ten independent runs, with the first run discarded as warm-up.

The automated test suite is implemented in pytest, using fastapi.testclient for in-process request dispatch. Each test is independent: a fresh SQLite database is created in a temporary directory and torn down at the end of the test, and a stable RNG seed is used where determinism is required. The 31 test cases are organised into eight categories aligned with the security objectives, as enumerated in Section VIII-G. The STRIDE analysis of Section V-J was conducted as a structured walk-through, mapping each STRIDE category onto specific threats and mitigations and identifying residual risks.

Results

Cryptographic Pipeline - Encryption and Integrity Verification

The cryptographic file pipeline was tested with files ranging from 1 MB to 50 MB across multiple NUM_CHUNKS configurations. Each file was divided into chunks, every chunk was encrypted independently with AES-256-CBC under a fresh random IV, and a SHA-256 integrity tag was computed over the IV concatenated with the ciphertext. The integrity verification endpoint was tested by deliberately corrupting individual chunks at the filesystem level and confirming that the system identified the tampered chunk index without performing a full file decryption. Clean files consistently returned a pass result. Table 5 presents the timing measurements; Figure 6 visualises the same data; and Figure 7(a) shows the verification speed-up ratio.

Table 5: Cryptographic pipeline performance - average of 10 runs (Intel Core i5, 16 GB RAM).

File Size	NUM_CHUNKS	Upload + Encrypt (ms)	Integrity Verify (ms)	Download + Decrypt (ms)	Tamper Detection
1 MB	4	~85	~12	~90	Correct - chunk index identified
10 MB	4	~340	~45	~360	Correct - chunk index identified
50 MB	4	~1,620	~210	~1,680	Correct - chunk index identified
10 MB	10	~360	~48	~380	Correct - chunk index identified
10 MB	20	~410	~55	~430	Correct - chunk index identified

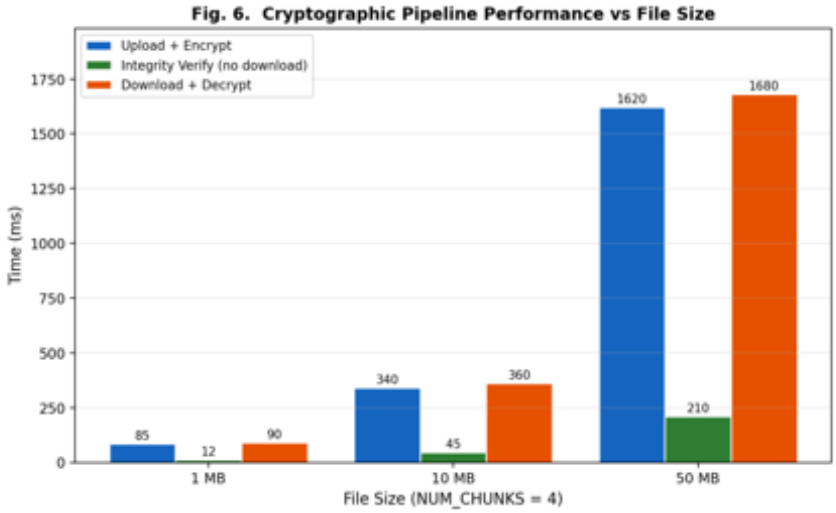

Figure 6: Cryptographic pipeline timing across file sizes for NUM_CHUNKS = 4. Upload-and-encrypt and download-and-decrypt scale roughly linearly with file size, while integrity verification remains an order of magnitude faster.

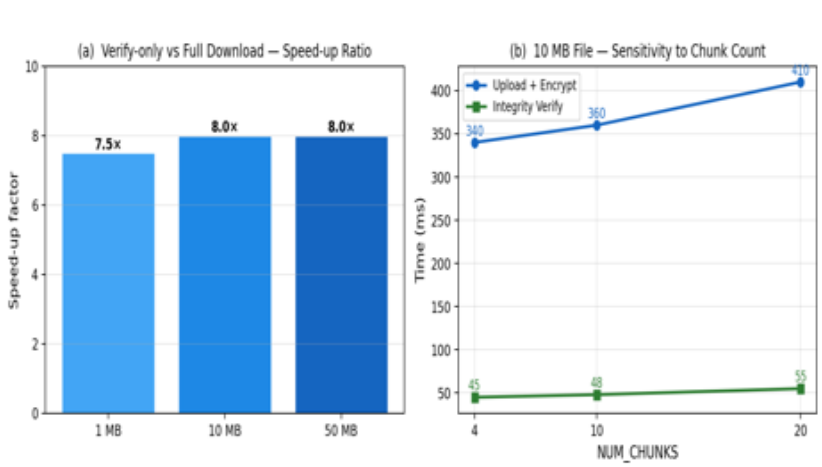

Figure 7: (a) Verification-only operations are 4 to 8 times faster than full download-and-decrypt across all tested file sizes. (b) For a fixed 10 MB file, increasing chunk count modestly increases overhead but preserves verification efficiency.

The results confirm two important properties. First, encryption and decryption costs scale approximately linearly with file size, as expected for a stream of independent block-cipher operations. Second, integrity verification - which requires only reading ciphertext bytes and recomputing SHA-256 over IV ‖ ciphertext - is consistently 4 to 8 times faster than full download-and-decrypt. For a 10 MB file at NUM_CHUNKS = 4, integrity verification completed in approximately 45 ms versus 360 ms for a full retrieval-and-decryption check, an 8× improvement. For the largest 50 MB benchmark, verification completed in 210 ms versus 1,680 ms for full download, again an 8× improvement. This empirically validates the practical efficiency of the PDP approach: a client can repeatedly verify integrity over time without incurring the bandwidth cost of repeated full retrievals, the precise property that motivated the original PDP/DPDP literature [1], [2].

Increasing NUM_CHUNKS from 4 to 20 for a fixed 10 MB file modestly increases pipeline overhead (340 ms → 410 ms for upload, 45 ms → 55 ms for verification) due to the per-chunk constant cost of metadata updates and tag computations. The optimal chunk count therefore depends on the deployment context: smaller chunks improve the precision of tamper localisation (a tampered byte affects only one chunk, identified by index), while larger chunks reduce per-operation overhead. For typical use the default of 4 to 10 chunks per file balances both concerns.

Authentication and session management

The two-step authentication mechanism was validated through eleven functional and security scenarios, summarised in Table 6. All scenarios produced the expected behaviour. In particular, the most security-critical test - submitting a full JWT directly to the Step 2 endpoint in an attempt to bypass TOTP verification - was correctly rejected with HTTP 401, confirming that no authentication bypass path exists between the two steps. The jti-based logout revocation was validated by issuing a JWT, logging out, and immediately resubmitting the same JWT to a protected endpoint; the token was rejected with HTTP 401 even though its expiration time had not yet been reached, confirming effective server-side session invalidation regardless of token lifetime.

Table 6: Authentication and session management - functional and security tests.

Test Scenario	Expected Result	Actual Result	Status
Valid credentials at Step 1	Temp JWT issued	Temp JWT issued	PASS
Wrong password at Step 1	HTTP 401; fail count +1	HTTP 401; fail count +1	PASS
Step 1 with locked account	HTTP 423 Locked	HTTP 423 returned	PASS
Valid temp + correct TOTP	Full JWT with new jti	Full JWT with new jti	PASS
Full JWT submitted to Step 2	HTTP 401 - bypass rejected	HTTP 401 returned	PASS
Expired temp token at Step 2	HTTP 401 Unauthorized	HTTP 401 returned	PASS
TOTP code replayed in same window	HTTP 401 on second use	HTTP 401 returned	PASS
Wrong TOTP code at Step 2	HTTP 401; rate limit applied	HTTP 401; rate limit applied	PASS
Logout then reuse of same JWT	HTTP 401 - token revoked	HTTP 401 returned	PASS
Recovery code - first use	Login granted; code marked used	Login granted; code marked used	PASS
Recovery code - second use	HTTP 400 - code already used	HTTP 400 returned	PASS

Role-Based access control

The dual-condition administrative authorisation was validated under five user profiles spanning the cross-product of {is_admin = true, false} and {email matches ADMIN_EMAIL, does not match}. Table 7 presents the results: only the user profile that satisfies both conditions receives HTTP 200 OK on administrative endpoints; all others receive 403 Forbidden, confirming that neither flag alone is sufficient. Ownership checks on file endpoints were also validated: attempts to access another user's files using a correctly authenticated but unauthorised JWT returned HTTP 403 in all tested cases.

Table 7: Role-based access control - dual-condition validation.

User Profile	is_admin	Email Matches ADMIN_EMAIL	Access Granted	HTTP Response
Standard user	FALSE	No	No	403 Forbidden
is_admin flag only	TRUE	No	No	403 Forbidden
Email match only	FALSE	Yes	No	403 Forbidden
Main administrator	TRUE	Yes	Yes	200 OK
Unauthenticated user	N/A	N/A	No	401 Unauthorized

Rate limiting and audit logging

Rate limiting was validated on three endpoints with distinct per-endpoint thresholds, summarised in Table 8. Each endpoint correctly emitted HTTP 429 at the threshold and recorded a RATE_LIMIT_EXCEEDED audit event in conjunction with the appropriate operation-specific event (LOGIN_FAILURE or TOTP_FAILURE). All 17 audit event types were confirmed correctly recorded, with user identifier, source IP address, timestamp, and event-specific JSON details captured for every entry, in line with the SecLaaS-style architecture proposed by Zawoad et al. [19].

Table 8: Rate limiting test results across protected endpoints.

Endpoint	Threshold	Response at Threshold	Audit Event Logged	Status
/auth/login/step1	5 failures / 5 min	HTTP 429; account locked	RATE_LIMIT_EXCEEDED + LOGIN_FAILURE	PASS
/auth/login/step2	5 failures / 5 min	HTTP 429	RATE_LIMIT_EXCEEDED + TOTP_FAILURE	PASS
/files/download-token	10 requests / 5 min	HTTP 429	RATE_LIMIT_EXCEEDED	PASS

Secure collaboration workflow

The secure collaboration framework was tested end-to-end across eight scenarios covering link request approval, link-code use within and beyond the 20-second validity window, single-use enforcement, and TOTP-protected transfer requests. All eight scenarios produced the expected behaviour. The boundary case - submitting a link code 21 seconds after generation - was correctly rejected with HTTP 400, confirming that the expiration logic enforces a phishing window of strictly less than 21 seconds, well below the practical threshold for a manual phishing attack.

Nurse / sub-account onboarding

The nurse/sub-account creation workflow was validated by an authorised parent user creating a subordinate account and confirming that the mandatory two-factor enrolment flow was enforced before the subordinate account could access any protected resource. Attempts to access protected endpoints prior to TOTP setup returned HTTP 403 with a descriptive error. After successful TOTP enrolment, the account gained full access consistent with its assigned role, and the NurseAccount record correctly recorded the parent identifier, setup-completion status, and first authentication timestamp.

Automated test coverage

The automated pytest test suite covers all security-critical paths. Table 9 summarises the eight test categories, the number of cases per category, and the outcomes; Figure 8 visualises the same. All 31 cases passed, providing systematic evidence that each security objective is met by the implementation.

Table 9: Automated pytest security test suite - outcomes by category.

Test Category	Tests	Pass	Fail	Coverage Focus
Admin-only route protection	5	5	0	403 for all non-qualifying users; dual-condition verified
Two-step login enforcement	4	4	0	Step 2 bypass rejected; temp token cannot skip TOTP
Logout and jti revocation	3	3	0	Revoked token rejected before expiry
Rate limiting enforcement	4	4	0	429 at threshold; correct audit event logged

Recovery code one-time use	3	3	0	Second use correctly rejected with HTTP 400
File download token validation	3	3	0	Expired and forged tokens rejected
Linking and nurse workflows	5	5	0	Approval flow, expiry, TOTP on transfer
Integrity verification	4	4	0	Tamper detected per chunk; clean file passes
TOTAL	31	31	0	All security-critical paths covered

Fig. 8. Automated Security Test Coverage and Results

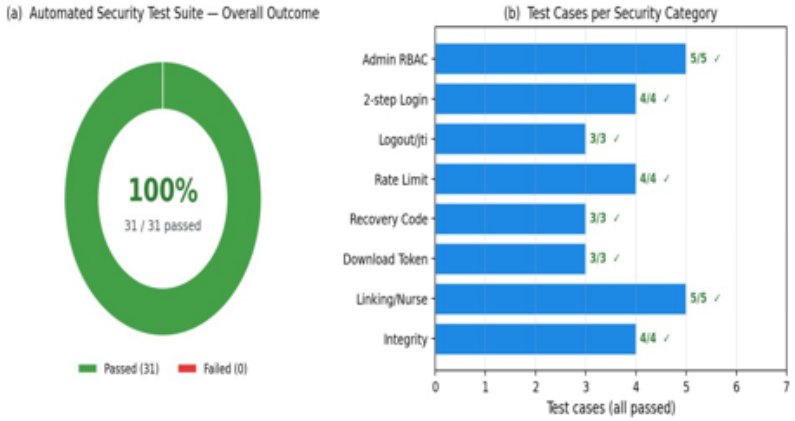

Figure 8: Automated security test outcome. (a) 31 of 31 test cases passed. (b) Per-category test counts confirm uniform coverage across the eight security objectives.

Comparative analysis

Table 10 compares CloudVault with two representative classes of existing systems and with the prior third-semester baseline of the same project. CloudVault uniquely combines per-chunk integrity tagging, mandatory TOTP, jti-based revocation, dual-condition RBAC, and a structured audit log within a single prototype. Whereas typical cloud drives provide opaque integrity assurance and basic encrypted-upload tools provide neither auditability nor server-side revocation, CloudVault makes integrity verification client-observable, authentication strictly two-factor, and session revocation immediate. Compared with the third-semester baseline of the present project, the fourth-semester enhanced prototype adds chunk-level integrity verification (previously file-level only), strict two-step login enforcement, jti- based session revocation, dual-condition admin authorisation, expanded audit logging from 6 to 17 categories, and rotating link codes.

Table 10: Comparative analysis: CloudVault *vs.* baseline and existing cloud storage classes.

Capability	Baseline (3rd Sem.)	Typical Cloud Drive	Encrypted Upload Tool	CloudVault (Final)
Per-chunk encryption	No	No	Partial	Yes (AES-256-
				CBC + unique IV)
DPDP-style integrity tags	No	No	No	Yes (SHA-256 per chunk)
Two-step authentication	No	Optional	Varies	Yes (mandatory)
JWT jti revocation on logout	No	Limited	No	Yes
Dual-condition RBAC	No	No	No	Yes (is_admin + email)
Structured audit log	6 events	Provider-only	No	17 categories

Phishing-resistant collab.	No	Static link	No	Rotating 20-second code
Automated security tests	Limited	N/A	N/A	31 cases, 100% pass
STRIDE analysis	No	N/A	N/A	Yes (Table III)
Verification efficiency vs. download	-	-	-	4–8× faster

Discussion

The results collectively demonstrate that CloudVault successfully achieves all stated security objectives. Strict two-step login, confirmed by the rejection of a full JWT submitted directly to the Step 2 endpoint, eliminates the authentication bypass path that would otherwise allow a valid token to skip TOTP verification entirely. The jti-based logout revocation, validated by confirming that a token returns HTTP 401 before its expiration time, addresses the inherent statelessness limitation of JWT-based authentication and aligns with the recommendations of Nishat and Muzaffar [16]. The PDP-style integrity verification empirically demonstrates a 4-to-8× efficiency advantage over full- file re-download, consistent with the original motivation of the DPDP literature [1], [2]; this is particularly significant for large datasets and infrequently accessed cold-storage scenarios.

The rotating link-code mechanism effectively restricts the phishing window to under 21 seconds, insufficient for a realistic manual attack. The rate-limiting tests confirm that thresholds are enforced independently per endpoint, preventing an attacker from circumventing login rate limits by targeting alternative endpoints. The audit log correctly captures all security-relevant events with source IP address and timestamp, providing the forensic traceability required for HIPAA-compliant incident response and the structured evidence base envisaged by Mahajan and Pandit [17]. The STRIDE analysis confirms that all high-priority threats are addressed by implemented controls; medium-priority residuals - SHA-256 tag forgeability under privileged write access, unsigned audit log, master-key file co-location, and absence of database encryption at rest - are acknowledged limitations of the prototype deployment context, each with a documented production hardening recommendation. The empirical evidence therefore supports the rejection of the null hypothesis and confirms the alternative hypothesis: the integrated security mechanisms produce measurable and practically significant improvements in cloud storage security.

Advantages, applications, limitations, and future work

Advantages of the proposed system

•Integrated security model. CloudVault unifies five security mechanisms - encryption, integrity, authentication, authorisation, and auditability - within a single application, eliminating the integration-boundary vulnerabilities that arise from composing disparate tools.

•Client-observable integrity. Unlike opaque provider-managed integrity, CloudVault's per-chunk SHA-256 tagging allows the data owner to verify integrity without trusting the provider and without re-downloading the file.

•Defence in depth for authentication. bcrypt password hashing, mandatory TOTP, replay-resistant Step-2 verification, jti-based logout revocation, and rate limiting collectively form a layered defence that rejects every individually-tested bypass attempt.

•Forensic-grade auditability. The 17-category audit log captures user identifier, IP address, timestamp, and event-specific JSON for every security-relevant operation, supporting court-admissible incident reconstruction in line with [17], [19].

•Phishing-resistant collaboration. Rotating 20-second link codes plus optional double-TOTP file-transfer confirmation elimi-

nate the static-link phishing vector and provide non-repudiation for both parties.

•Empirical validation. All 31 automated security test cases pass; a STRIDE analysis explicitly enumerates threats, mitigations, and residual risks; performance benchmarks demonstrate practical efficiency at all tested file sizes.

Applications and scope of the work

The prototype is directly applicable to several domains where regulatory or operational requirements demand strong integrity, authentication, and auditability.

•Digital evidence chain-of-custody. PDP-style per-chunk tags provide a cryptographically grounded mechanism for verifying that digital evidence files have not been altered since upload, while chunk-level tamper detection identifies the specific portion of a file that was modified - enabling precise scope assessment beyond what file- level hashing offers - and the 17-category audit log records every access, authentication event, and administrative action to support court-admissible chain-of-custody documentation.

•Insider-threat detection and post-incident reconstruction. Audit logging combined with ownership-based access control gives investigators a complete record of who accessed which files and when, enabling behavioural analysis and insider-threat attribution; integrity-verification timestamps, download tokens, and audit-log entries together provide a verifiable timeline of file states and access events for temporal analysis in post-incident investigations.

•Secure inter-agency evidence sharing. Rotating link codes enable phishing-resistant, time-limited sharing of digital evidence files between investigators across organisational boundaries, with mutual TOTP authentication ensuring active consent and non-repudiation for both parties.

•Healthcare and medico-legal record preservation. The nurse/sub-account workflow and mandatory TOTP enforcement model the access-control and auditability requirements of healthcare environments managing patient records and medico-legal documents under HIPAA-style obligations.

Limitations

The prototype's limitations are explicit and bounded, falling into seven categories.

•Single-node SQLite. SQLite does not support concurrent write operations at production scale; deployment- grade systems require PostgreSQL with connection pooling and row-level locking.

•Master-key file management. Master key material is stored as a filesystem file accessible to any process with the same OS user privileges as the API server. Production deployment must use an HSM or KMS such as AWS KMS, Azure Key Vault, or HashiCorp Vault.

•SHA-256 tags not cryptographically unforgeable. A privileged attacker who can both modify a chunk and recompute its SHA-256 tag can forge a valid integrity tag. Production verification requires RSA-based or BLS- based PDP authenticators that are unforgeable without the private signing key.

•In-memory rate limiter. Rate-limiter state is lost on process restart and does not coordinate across multiple instances. A Redis--backed distributed rate limiter is required for high-availability deployment.

•Audit log not signed. The audit log is stored in the same database as other application data and is not cryptographically signed;

a privileged insider could modify log records to conceal activity.

•No TLS enforcement. The prototype does not enforce HTTPS in its current configuration; production deployments must use TLS 1.3 with HSTS.

•Limited test environment. Performance benchmarks were conducted on a single development workstation and may not generalise to high-concurrency production environments. A wider benchmark across cloud-hosted instances under varied load is required to characterise scaling behaviour.

Production hardening recommendations

Migration of the prototype to a production-grade deployment requires a coordinated set of upgrades across cryptographic, persistence, transport, and operational layers. Filesystem-based master-key storage must be replaced with an HSM or cloud KMS such as AWS KMS, Azure Key Vault, or HashiCorp Vault, with secrets governed by a dedicated secrets-management platform rather than environment variables. SQLite must be replaced with PostgreSQL configured for connection pooling, row-level locking, and encrypted-at-rest storage; the in-memory rate limiter must be replaced with a Redis-backed distributed counter; and database and chunk-store backups must be encrypted before transfer to backup storage. TLS 1.3 with HSTS must be enforced on every client-server channel, supplemented by a Web Application Firewall to filter malicious requests. Finally, audit log events should be forwarded to a SIEM platform for real-time alerting, correlation, and long-term retention, with each entry either cryptographically signed or written to append-only WORM storage to prevent insider tampering.

Future enhancements

•Cryptographically unforgeable integrity tags. Replacing SHA-256 integrity tags with BLS signature-based PDP authenticators would provide cryptographic unforgeability guarantees, eliminating the tag-forgery vulnerability identified in the limitations section, and the verification API could be extended to support public auditability by a Third-Party Auditor without exposing file content [5].

•Hardware-backed phishing-resistant authentication. Replacing TOTP with WebAuthn passkeys would eliminate the shared-secret model, providing phishing-resistant hardware-backed authentication aligned with FIDO2 standards and NIST SP 800-63B AAL3.

•Multi-tenant isolation and cloud-native storage. Extending the data model to multi-tenant deployments with tenant-specific keys managed through a KMS, combined with replacement of filesystem-based chunk storage with object storage such as Amazon S3, Google Cloud Storage, or MinIO, would enable enterprise-grade multi- customer deployments with geographic redundancy and lifecycle management.

•Formal verification and structured penetration testing. The authentication state machine and authorisation model could be formally verified using model-checking tools such as ProVerif or Tamarin, complemented by a structured penetration testing programme covering OWASP Top 10 vulnerabilities, cryptographic implementation errors, and authentication-bypass techniques.

•AI-driven anomaly detection and blockchain-anchored audit log. The structured audit log already captures the data required to train anomaly-detection classifiers in the spirit of [6]; periodically anchoring the audit log's hash chain to a public blockchain in line with [8] and [10] would additionally provide third-party-verifiable, tamper-evident logging for high-assurance contexts.

Conclusion

This paper has presented CloudVault, a research prototype cloud storage system designed to address fundamental deficiencies in integrity verification, authentication security, and access control that characterise existing cloud storage solutions. The prototype integrates Dynamic Provable Data Possession (DPDP)-inspired per-chunk integrity tagging with AES-256-CBC encryption under unique per-chunk initialisation vectors, a strict two-step TOTP-based authentication workflow with JWT session management and server-side jti revocation, dual-condition role-based access control, sliding-window rate limiting, a 17-category structured audit log, and a phishing-resistant collaboration framework based on rotating short-lived link codes.

The empirical evaluation demonstrates that the prototype detects post-upload file tampering at the chunk level without requiring full file retrieval, enforces mandatory two-factor authentication with no authentication-bypass path, revokes sessions immediately upon logout regardless of token expiration time, and enforces rate limits on all authentication-sensitive endpoints. All 31 automated security test cases passed, providing evidence-based validation of the security mechanisms. Integrity verification ran 4 to 8 times faster than full-file re-download, empirically validating the practical efficiency of the PDP approach. Comparative analysis against the third-semester baseline and against representative commercial classes demonstrates significant improvements across all security dimensions, supporting the rejection of the null hypothesis in favour of the alternative hypothesis: the integrated security mechanisms produce measurable and practically significant improvements in cloud storage security.

CloudVault is explicitly a research prototype, and several production hardening measures - HSM-based key management, PostgreSQL, TLS enforcement, a Web Application Firewall, SIEM integration, and BLS-based unforgeable integrity tags - remain outside the current implementation scope and are documented as recommendations and as directions for future research. The prototype nevertheless provides both a functional implementation and an empirically evaluated reference architecture for integrated cloud storage security in forensic and healthcare contexts, addressing the integration-gap identified in the literature review and laying a concrete foundation for future production- grade systems.

References

1. Tang C, Wang F, Zhao C, Cui H, Ying Z, et al. (2025) FM-DPDP: Fine-grained multicopy dynamic provable data possession with flexible storage. J Information Security and Applications.

2. Bindu A, Divya G, Ganesh H, Nayak S, Sindhu L (2026) A safe and secured digital solution for cloud-based possession of dynamic, group-oriented, verifiable data. In: Adaptive Technologies for Sustainable Growth. Boca Raton, FL, USA: 200–05.

3. Saengthong V, Wongsuwan C, Ritthaisong K, Fugkaew S (2026) DHT-backed ancestor-assisted Merkle verification for scalable and real-time log integrity in cloud data lakes. IEEE Open J. Computer Society.

4. Lin X, You W, Wu C, Liu W, Gu Q (2025) TA-PDC: Provable data contribution with traceable anonymous for group transactions. In: Proc. Int. Conf. Information and Communications Security. Singapore: Springer Nature: 463–83.

5. Rahi A (2026) A multi-layer cryptographic framework for secure file sharing with integrity proofs and deduplication.

6. Zawoad S, Dutta AK, Hasan R (2015) Towards building forensics enabled cloud through secure logging-as-a-service. IEEE Trans. Dependable and Secure Computing 13: 148–62.7. Komuravelly SK (2025) Enhancing accountability in cloud storage using cryptographic proof-of-access mechanisms. M.S. thesis.

7. Ughanze IP (2026) Privacy-preserving public auditing of outsourced cloud data with zero-knowledge proofs. Ph.D. dissertation. Nat. College of Ireland.

8. Joshi A, Mahapatra RP, Devarajan GG (2026) An improved mechanism to maintain data integrity and anomaly detection in cloud storage.

9. Mahajan R, Pandit K (2024) Cryptography and computational approaches in ensuring data integrity for digital forensic evidence. In: Proc. 16th Int. Conf. Electronics, Computers and Artificial Intelligence (ECAI): 1–6.

10. Ngo T (2018) Cloud security: Private cloud solution with end-to-end encryption.

11. Zhu B (2015) Analysis and design of authentication and encryption algorithms for secure cloud systems.

12. Selvi P, Sakthivel S (2025) A hybrid ECC-AES encryption framework for secure and efficient cloud-based data protection. Scientific Reports 15: 30867.

13. Anandhi T, Sangari AS (2026) Privacy-preserving authentication protocol with optimized elliptic curve cryptography for healthcare domain in cloud. Cluster Computing 29: 77.

14. Ahmed HM, Zubair S, Tawfik M (2026) A privacy-preserving cloud storage framework with hybrid encryption, homomorphic keyword search, and blockchain-based integrity verification. Scientific Reports.

15. Ahmad S, Nazim M, Arif M, Ahmad J, Mehfuz S, et al. (2025) Protecting data in the cloud: A systematic literature review of key management. Concurrency and Computation: Practice and Experience 37: e70223.17. Nishat A, Muzaffar J (2024) Securing web applications with OAuth 2.0, JWT, and multi-factor authentication. Euro Vantage J. Artificial Intelligence 1: 36–3.

16. Margam M (2026) Understanding authentication and authorization: A comparative analysis of role-based access control (R-BAC), attribute-based access control (ABAC), and relationship-based access control (ReBAC) authorization models. Int. J. Advanced Computer Science and Applications 17.

17. Rouhollahi A, Bagherifard K, Malekhosseini R (2026) Enhancing data security and access control in cloud computing: A comparative study of active data cube framework (ADCu) and traditional approaches. Power System Protection and Control 54: 133–62.20. Radha SK (2026) Formal and decentralized framework for verifiable trust: Self-sovereign identity, blockchain provenance, and hardware-rooted assurance. Ph.D. dissertation. Univ. Notre Dame.

18. Ateniese G, et al. (2007) Provable data possession at untrusted stores. In: Proc. 14th ACM Conf. Computer and Communications Security (CCS): 598–09.

19. Erway C, Küpçü A, Papamanthou C, Tamassia R (2015) Dynamic provable data possession. ACM Trans. Information and System Security 17: 15.

20. Juels A, Kaliski BS (2007) PORs: Proofs of retrievability for large files. In: Proc. ACM CCS: 584–97.

21. Wang C, Chow SSM, Wang Q, Ren K, Lou W (2013) Privacy-preserving public auditing for secure cloud storage. IEEE Trans. Computers 62: 362–75.

22. Yassein MB, Aljawarneh S, Qawasmeh E, Mardini W, Khamayseh Y (2017) Comprehensive study of symmetric key and asymmetric key encryption algorithms. In: Proc. Int. Conf. Engineering and Technology (ICET): 1–7.