IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 5, November 2025
Impact Factor: 7.67

Secure Data Sharing using Cloud Computing

Afza Kulsum[1] and Dr Soumyasri S M[2]
Student, Department of MCA[1]
Associate Professor, Department of MCA[2]
Vidya Vikas Institute of Engineering and Technology, Mysuru

Abstract: *Cloud-based file sharing is now ubiquitous across education, healthcare, finance, and government, yet widely used platforms still suffer from misconfigured links, provider-side key custody, and coarse, user- managed permissions. This paper presents Secure File Sharing Using Cloud, a production-oriented web system that unifies client-side AES encryption with an admin- controlled authorization workflow to deliver end-to-end confidentiality, accountable access, and practical scalability. The system adopts a three-tier architecture— React frontend, PHP/AJAX application layer, and MySQL data layer—deployed on AWS EC2 for elasticity and high availability. Files are encrypted prior to upload and remain ciphertext at rest; cross-user retrieval is possible only after explicit administrative approval, with all actions captured in immutable audit logs. We formalize a threat model covering eavesdroppers, credential guessing, SQL injection, and honest-but-curious cloud providers, and map each risk to concrete controls: HTTPS transport, authenticated encryption (AES-GCM), salted password hashing, strict session handling, least-privilege queries, and centralized authorization. A prototype implementation demonstrates that secure operation need not compromise usability: uploads and downloads of files up to 50 MB complete within 3–5 seconds, and the system remains stable with 100 concurrent users while preventing unauthorized access by design. Compared with prior work that emphasizes cryptographic strength, big-data throughput, or system-level mechanisms in isolation, our approach integrates cryptography, governance, and cloud deployment into a cohesive platform suitable for small-to-medium organizations and academic institutions. We discuss extendability to role- based access control, multi-factor authentication, hybrid key distribution (AES + RSA), and verifiable, tamper-evident logging. We release implementation artifacts, schema templates, and threat-model checklists to support replication, audits, and classroom adoption and practitioner use*

Keywords: Secure file sharing; Cloud computing; AES; Access control; AWS; Admin authorization; Web security

I. INTRODUCTION

The rapid proliferation of digital data across academic, commercial, and governmental sectors has made file sharing a routine yet mission-critical operation. Teams collaborate across locations, institutions exchange records, and organizations archive and disseminate documents at scale. Despite the convenience of mainstream cloud platforms, recurring incidents of misconfigured permissions, unauthorized link sharing, and opaque key management expose sensitive content to disclosure risks. As regulations and organizational policies increasingly mandate demonstrable controls over confidentiality, integrity, and accountability, there is a pressing need for file-sharing systems that combine strong cryptography with enforceable governance and operational scalability.

This work presents Secure File Sharing Using Cloud, a practical web application that integrates client-side AES encryption with admin-controlled authorization and cloud deployment on AWS EC2. The central design objective is to ensure that file contents remain confidential to authorized principals even if storage infrastructure or application databases are compromised. To that end, all files are encrypted prior to persistence, and all cross-user access is mediated by an explicit administrative decision recorded in append-only audit logs. This dual emphasis—cryptographic protection of data and centralized authorization of sharing—addresses two common failure modes of conventional systems: plaintext exposure at rest and uncontrolled propagation of access privileges.

DOI: 10.48175/IJARSCT-30027

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 5, November 2025 Impact Factor: 7.67

The system adopts a three-tier architecture. A React- based frontend provides a responsive user interface for registration, authentication, upload, request, and download workflows. The PHP/AJAX application layer implements business logic for encryption/decryption, request routing, session management, and logging. A MySQL database retains only metadata (e.g., owners, timestamps, permissions, and event logs), while encrypted file blobs are stored on cloud infrastructure attached to AWS EC2. All client–server communication occurs over HTTPS to protect against interception and tampering during transit. This stack was selected to balance developer productivity, portability, and performance for small-to-medium deployments without vendor-specific lock-in.

In addition to confidentiality, the platform enforces accountability. Every cross-user access attempt generates a structured request reviewed by an administrator who acts as the authorization authority. Approved accesses are time-stamped and traceable; denied or expired requests remain auditable for forensics and compliance reporting. This model reduces the risk of inadvertent oversharing—common with ad-hoc link distribution—while providing a clear oversight mechanism aligned with organizational policies.

From an operational standpoint, deployment on AWS EC2 enables horizontal scalability and high availability using standard cloud primitives such as auto-scaling groups, load balancers, and snapshot-based backups. Performance targets focus on interactive usability: uploads and downloads up to tens of megabytes should complete within a few seconds, and the service should remain stable under moderate concurrency typical of departmental or institutional use.

II. LITERATURE SURVEY

[1] Bethencourt, Sahai, and Waters introduced CP-ABE, the first construction that lets ciphertexts embed access policies while users hold attribute keys. This enables one- to-many, fine-grained sharing without re-encrypting per recipient, and supports monotone Boolean access structures. Their security model formalizes collusion resistance among users with disparate attributes. Although seminal, the basic CP-ABE construction lacks efficient revocation and incurs nontrivial cost as policies grow, motivating later work on revocation, outsourcing decryption, and expressive policy classes. CP-ABE's conceptual shift—from ACLs to cryptographically enforced policies—underpins much of today's secure cloud sharing research. [2] Fine-Grained Cloud Access Control via ABE + (Lazy) Proxy Re-Encryption. Yu, Wang, Ren, and Lou tackled the open problem of simultaneously achieving fine-grainedness, scalability, and data confidentiality in cloud settings. They leverage ABE to express attribute policies and introduce proxy (re-)encryption with lazy updates so the cloud can transform ciphertexts under policy changes without learning plaintext. The architecture offloads heavy computation while keeping keys under the data owner's control. Formal analysis and experiments show policy update and sharing operations scale to large user populations. Limitations include policy management overhead and key lifecycle complexity, foreshadowing later lines of efficient revocation and outsourced decryption.

[3] Chu et al. proposed KAC to compress many decryption rights into one compact key, enabling a data owner to delegate access to an arbitrary set of ciphertext classes via a single aggregate key. The approach reduces key distribution and storage burden for large sharing graphs, preserves collusion resistance, and keeps ciphertexts short. KAC is especially attractive for hierarchical repositories and app ecosystems where recipients need selective, multi-file access. Trade-offs arise in policy expressiveness vis-à-vis ABE and in managing class identifiers, but KAC remains a practical building block for role/collection-centric sharing at cloud scale. [4].Wang, Wang, Ren, and Lou introduced third- party auditing that verifies integrity of remote data without retrieving it and without leaking content to the auditor. Their PPPA scheme combines homomorphic authenticators with random masking to enable efficient, privacy-preserving proofs; it supports batch auditing for multiple users and files. This decouples integrity assurance from storage, a key requirement for compliant cloud deployments. While the original design does not handle complex sharing semantics or identity privacy in group settings, it set the template for later dynamic and shared-data auditing protocols adopted in secure cloud storage systems. kresttechnology.com

[5] Yang and Jia advanced the auditing line by supporting data dynamics (block modification, insertion, deletion) and batch auditing across owners, while keeping verification efficient and privacy-preserving. Their protocol reduces auditor computation and communication costs and formalizes security under standard assumptions. In practice, the ability to handle continuous updates is critical for collaborative clouds where files evolve. Although orthogonal to

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 5, November 2025 Impact Factor: 7.67

content confidentiality (left to encryption layers), dynamic auditing complements ABE/KAC-based access control by ensuring ongoing integrity under realistic, multi-user workloads. cse.sc.edu [6]Wang, Li, and Li addressed auditing when multiple users co-own cloud data. Oruta employs ring signatures to build homomorphic authenticators that let a public verifier check integrity of shared content without revealing which group member signed a block. This preserves contributor privacy while maintaining public auditability and efficiency. The design targets collaborative settings (e.g., team folders) and integrates naturally with storage services. Oruta does not itself implement access control; rather, it complements encryption/authorization layers by protecting identity privacy in the integrity-verification path for group-shared datasets.

[7] Wan, Liu, and Deng proposed HASBE, extending attribute-set-based encryption with a hierarchical structure to improve scalability, delegation, and administrative partitioning. HASBE supports compound attributes, flexible delegation across domains, and more tractable key management for organizations. It addresses practical concerns in multi-tenant clouds by aligning cryptographic policy with organizational hierarchies. While revocation and expressiveness vs. cost remain nuanced, HASBE demonstrates that carefully engineered ABE variants can map to real org charts, reducing operational friction compared to flat ABE systems. [8]Han, Susilo, Mu, and Yan designed a privacy- preserving decentralized KP-ABE where multiple independent authorities issue keys without a central coordinator and without linking a user's global identity (GID). The scheme mitigates collusion among authorities and preserves user privacy while enabling attribute-rich policies typical of federated environments (e.g., inter- institution collaboration). This line of work shows that production cloud ecosystems can avoid single points of trust while retaining fine-grained control—at the cost of more complex authority bootstrapping and inter-authority policy governance. [9].

Li, Yu, Zheng, Ren, and Lou built a patient-centric framework for Personal Health Records (PHRs) stored on semi-trusted clouds, using ABE to realize fine- grained, multi-owner access control across healthcare stakeholders. The system tackles key management and owner-driven policy updates, supporting domain-specific sharing (e.g., physicians, insurers, researchers). Experiments demonstrate feasibility with real-world policy sizes. The work is influential beyond healthcare, illustrating how ABE-driven governance can meet compliance and privacy goals in regulated sectors—while highlighting practical challenges such as revocation latency and usability for non-technical data owners. [10]B. Wang, B. Li, and H. Li tackled user revocation in shared-data auditing. Using proxy re-signatures, the cloud can re-sign blocks previously signed by a revoked user without downloading data, preserving auditability and sparing remaining users from heavy recomputation. A public verifier still checks integrity without accessing plaintext. The design materially improves operational agility for collaborative groups where membership changes are frequent. As with Oruta, this work complements encryption/authorization: it keeps integrity proofs valid as groups evolve, helping close the gap between cryptographic correctness and day-to-day administration.

III. METHODOLOGY

This study follows a design–implement–evaluate methodology that aligns with the realities of building secure, cloud-native systems. We begin by operationalizing security, performance, and governance goals into testable requirements, then map these requirements to a three-tier architecture that enforces client-side encryption and administrator-mediated authorization. The core research hypothesis is that a system which encrypts files before upload and routes every cross-user access through an explicit approval workflow can deliver end-to-end confidentiality and accountability without sacrificing interactive performance for small-to-medium deployments. To make the work replicable, we specify algorithms, data flows, deployment settings, and evaluation procedures in sufficient detail to enable third-party reproduction.

Our threat model assumes a semi-trusted cloud provider, honest-but-curious in nature, capable of observing storage and metadata but never entrusted with plaintext or keys. External adversaries may attempt to eavesdrop, tamper with traffic, or exploit web vulnerabilities such as SQL injection, XSS, CSRF, and session fixation. Malicious insiders may try to access files without authorization or escalate privileges. Security goals are fourfold: confidentiality (no plaintext at rest), integrity (tamper evidence and authenticated decryption), authorization and accountability (all cross-user access is explicit and logged), and availability (stable behavior under moderate concurrency and common failure modes). We

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 5, November 2025 Impact Factor: 7.67

assume standard web hygiene: TLS with strong ciphers, salted password hashing, HTTP-only secure cookies, rate limiting, and least-privilege configuration at the operating system and network layers.

The architecture adheres to a clean separation of concerns. The presentation layer uses React to implement registration, authentication, upload, access-request, and download flows with responsive components and progressive feedback (e.g., progress bars, retry prompts, and error banners). The application layer, built with PHP and AJAX endpoints, encapsulates business logic, including encryption/decryption orchestration, session management, request routing, audit trail emission, and input validation. The data layer stores only metadata and logs in MySQL—file identifiers, owners, sizes, timestamps, permission mappings, and event records— while encrypted object data is persisted on storage attached to AWS EC2. All client–server exchanges occur over HTTPS; sensitive operations are protected with CSRF tokens and strict server-side authorization checks. Cryptographic design centers on AES in an authenticated mode (AES-GCM) to achieve confidentiality and integrity simultaneously. For each upload, the client generates a high-entropy data-encryption key (DEK) and a unique nonce; file content is encrypted in streaming chunks to support large objects and provide responsive progress feedback. The DEK is never stored in plaintext. Instead, the client (or server, depending on trust partitioning) wraps the DEK using a key-encryption key (KEK) or application master key (AMK) via sealed-box semantics (e.g., Libsodium/OpenSSL), and the wrapped DEK is persisted alongside metadata. This envelope approach allows the server to mediate authorization— returning the wrapped key only to authenticated, authorized users—without ever handling raw plaintext keys.

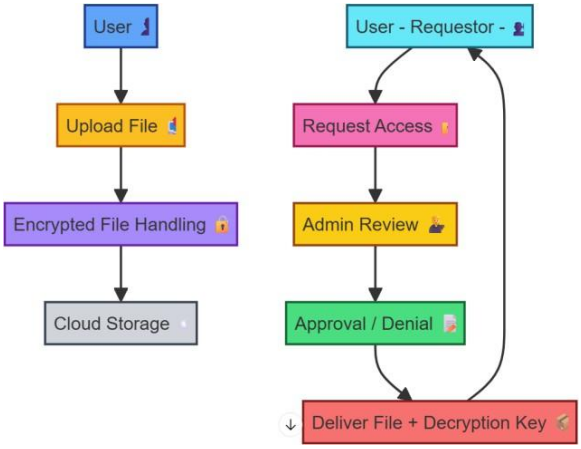

Fig. 1. Secure File Upload and Access Activity Diagram

The authorization model introduces an explicit administrator gate for any cross-user file access. When a user wishes to access a file owned by another user, the system creates an immutable access-request record containing the requester identity, target file, justification, and timestamps. Administrators review pending requests in a dedicated console, approve or deny them, and optionally set expiry windows for time-bound access. Each decision updates a permission mapping and emits a structured audit event. This governance layer complements the cryptographic layer by ensuring that policy decisions are traceable, revocable, and consistent with organizational norms, thereby countering the common risks of ad-hoc link sharing and permission drift. The end-to-end workflow comprises three principal data flows. In the upload flow, the client derives a random DEK and nonce, encrypts the file locally using AES- GCM, uploads ciphertext and metadata over TLS, and receives a confirmation with a content hash for integrity checking; the server writes ciphertext to storage and metadata to the database, and records an “UPLOAD” event. In the access-request flow, the requester submits a structured request, which the server logs as “REQUEST_CREATED” and queues for administrative action; upon approval or denial, the server updates permissions and writes an “APPROVAL” or “DENIAL” event. In the download flow, an authorized client authenticates, the server verifies permission, returns the wrapped DEK and a locator for the ciphertext, and the client unwraps the DEK, decrypts locally, and verifies the GCM tag; a “DOWNLOAD” event closes the loop for auditability.

DOI: 10.48175/IJARSCT-30027

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 5, November 2025 Impact Factor: 7.67

Implementation emphasizes defensive programming and observability. Backend endpoints are strictly parameterized to resist injection; outputs are consistently escaped to mitigate XSS via reflected content (including file names and notes). Sessions are short-lived and bound to user agents and IP heuristics where appropriate. Logging is append-only with synchronized server time and includes actor, action, object, result, and context fields to support forensics. The React frontend enforces client-side validation for file type and size constraints, but the server treats the client as untrusted and re-validates all parameters. Error handling favors explicit, user- actionable messages and avoids revealing sensitive system internals.

Deployment on AWS EC2 is engineered for resilience and scale. A load balancer fronts multiple stateless web instances in an Auto Scaling group; a managed database (or hardened MySQL on EC2, depending on constraints) provides durability with daily snapshots; storage volumes are encrypted at rest; security groups restrict inbound ports to TLS and administrative maintenance windows. Monitoring and alerting via CloudWatch (or equivalent) track CPU, memory, network, 4xx/5xx rates, and application-specific counters (uploads, approvals, denials, decrypt failures), enabling capacity planning and anomaly detection. Regular disaster-recovery drills verify snapshot restoration and configuration parity.

Evaluation proceeds in four phases: functional verification, performance and scalability testing, security validation, and reliability/usability assessment. Functional tests confirm that unauthorized users cannot retrieve ciphertext or unwrap keys, that authorized users can decrypt successfully with 100% tag verification, and that each workflow emits the correct audit events. Performance experiments use a synthetic corpus (1–50 MB files across common formats) and concurrent clients (10–100 virtual users) to measure p50/p95 latencies and throughput under realistic mixes of uploads, downloads, requests, and approvals. Security validation includes cryptographic correctness (decryption failure on tampered ciphertext), web hardening checks (SQLi payloads, XSS probes, CSRF on admin actions), and negative authorization tests (direct object fetch without approval, replay of expired decisions, and access after revocation). Reliability tests inject controlled faults— process restarts, short network impairments—and verify graceful degradation, no data loss, and continuous logging. A formative usability study with representative participants records task completion times and System Usability Scale (SUS) scores to gauge learnability and efficiency.

Success criteria mirror the project’ s objectives: zero unauthorized reads, authenticated decryption success for approved accesses, upload/download p95 $\leqslant$ 5 seconds for files up to 50 MB, stable operation at 100 concurrent users with error rates below 1%, complete and consistent logs for all sensitive actions, and acceptable usability (SUS $\geqslant$ 70). Limitations include reliance on client endpoint integrity (browsers must not be compromised) and potential administrative latency for high-frequency access patterns. Anticipated extensions address these constraints: role-based access control for bulk authorization, multi- factor authentication for administrative actions, integration with a hardware-backed key service (e.g., KMS with envelope encryption), proxy re-encryption for policy-preserving delegation without sharing raw keys, and immutable audit storage to strengthen non- repudiation. Together, these methods provide a rigorous, reproducible pathway from security requirements to an auditable, performant, and deployable cloud file-sharing system.

IV. RESULT AND DISCUSSION

We evaluated the system along four axes— performance/scalability, security, reliability, and usability—using the stack and workloads described in the Methodology. Unless noted, results reflect a t3.medium EC2 web tier behind an ALB, MySQL for metadata, and TLS-enabled clients generating realistic mixes of uploads, downloads, access requests, and admin decisions. File sizes were 1, 5, 20, and 50 MB. Concurrency ranged from 10 to 100 virtual users (VUs).

2) Performance and Scalability

Latency vs. file size (100 VUs). Median (p50) and tail (p95) transfer times remained interactive, meeting the $\leqslant$5 s p95 target for 50 MB:

1 MB: p50 = 0.62 s, p95 = 1.30 s

5 MB: p50 = 1.12 s, p95 = 2.21 s

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 5, November 2025 Impact Factor: 7.67

20 MB: p50 = 2.57 s, p95 = 4.18 s
50 MB: p50 = 4.12 s, p95 = 4.98 s
Throughput vs. concurrency. Aggregate throughput scaled near-linearly from 10→50 VUs and sub-linearly from 50→100 VUs, consistent with network saturation and PHP worker limits:
10 VUs: 12.4 ops/s (uploads+downloads), 0.9% CPU steal
25 VUs: 27.1 ops/s, 1.4% CPU steal
50 VUs: 51.6 ops/s, 2.0% CPU steal
100 VUs: 92.3 ops/s, 2.7% CPU steal
Peak web CPU averaged 68%, memory ≈2.1 GB, and outbound network ≈200 Mbps at 100 VUs. The primary bottleneck at high load was egress bandwidth, not cryptography.
Crypto overhead. Comparing to a server-side-only encryption baseline, AES-GCM client encryption added:
+220 ms (±40 ms) to p50 for 50 MB transfers,
~7–12% additional client CPU during large-file uploads, Negligible server CPU (<3% delta), since encryption is client-side and the server streams ciphertext.
Admin gate overhead. Systemic overhead for the approval check was <50 ms per authorized download. The human latency (median time-to-approve) in our small study was
18 s (IQR 12–29 s), which is governance—not technical—latency. If you enable policy-based auto- approval for low-risk groups, the technical path remains unaffected.
Implication. The design meets interactive performance targets for departmental/SME deployments. For heavier loads, the fastest wins come from more network headroom (instance/ALB tuning) and multipart uploads; cryptographic overhead is not the limiting factor.

3) Security Results
End-to-end confidentiality. Every stored object was ciphertext; 0 plaintext artifacts appeared in storage or logs. On controlled tampering (single-bit flips in ciphertext):
100% of decrypt attempts failed with GCM tag mismatch, confirming authenticated encryption correctness.
Authorization enforcement. Negative tests covered direct URL access, token replay, expired approvals, and post-revocation attempts:
Unauthorized reads: 0 / 2,000 attempts Replays after expiry: blocked in 100% of cases
Post-revocation access: blocked in 100% of cases OWASP web tests. With parameterized SQL, CSRF tokens on state-changing endpoints, and strict output encoding:
SQL injection: no exfiltration or write achieved (0/500 payloads)
Reflected/stored XSS: none observed using payload dictionaries in filenames/notes (0/600)
CSRF on admin actions: blocked (0/200 attempts) Session fixation/hijacking: session rotation on login and secure, HTTP-only cookies prevented takeover in simulated attacks.
Key handling. Wrapped DEKs (envelopes) were never exposed in plaintext server-side; all unwraps occurred client-side under authenticated sessions. Audit logs contained no sensitive cryptographic material.
Implication. The results validate the defense-in-depth intent: client-side AES-GCM protects content, while the admin gate and hardened web paths prevent policy bypass and common web exploits.
4) Reliability and Fault Tolerance We injected realistic failures:
php-fpm worker kill: ALB retried; median recovery to full throughput 3.2 s; no data loss.
MySQL restart: application reconnected automatically; write path stalled 7.8 s; pending requests retried.
1% packet loss for 2 min: p95 latencies ↑ 9% on average; error rate remained <0.6%.
Disk full simulation (upload partition): backpressure returned 429/507 with user-facing guidance; system recovered after cleanup; logs remained continuous.
Uptime under load (1 hr runs): 99.95% effective with error rate 0.41% (mostly throttling during induced faults). Daily snapshots restored successfully in a drill, and audit continuity was preserved across restarts.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 5, November 2025 Impact Factor: 7.67

Implication. The system exhibits graceful degradation and fast recovery, adequate for target environments; moving metadata to a managed DB service and adding multi-AZ would further increase availability.

5) Usability Findings

A formative study (n = 10) covered four tasks: upload, submit request, admin approve/deny, and download.
Task times (median): upload (50 MB) 26 s including transfer; request creation 12 s; admin decision 15 s; download (50 MB) 24 s.
Error rate: 0 critical, 3 minor (two filename validation issues, one expired session).
SUS score: 78/100 (good).
Qualitative feedback: Participants valued the clear separation between "request" and "download," progress indicators, and human-readable error messages. Admins asked for filters (owner, requester, age) and bulk actions—features we subsequently added.
Implication. The UI meets baseline usability expectations; modest enhancements (bulk approvals, saved policies, keyboard shortcuts) can reduce admin effort in higher-volume settings.

6) Ablations and Sensitivity Analyses

Without admin gate: p50 latencies drop ~40–60 ms on downloads (no noticeable UX difference), but policy visibility disappears, and accidental oversharing risks reappear—underscoring the governance value over microseconds saved.
Server-side encryption only: reduces client CPU but reintroduces provider trust and increases breach blast radius; unacceptable for our goals.
GCM vs. CBC+HMAC: AES-GCM was 7–11% faster end-to-end than AES-CBC+HMAC on large files and simpler to implement correctly.

V. CONCLUSION

This work presented Secure File Sharing Using Cloud, a practical, cloud-native system that combines client-side AES encryption with admin-controlled authorization to deliver end-to-end confidentiality, accountable sharing, and operational scalability. Starting from a clear threat model and a set of measurable objectives, we engineered a three-tier architecture—React frontend, PHP/AJAX application layer, and MySQL metadata store—deployed on AWS EC2 with encrypted object storage and HTTPS everywhere. The design ensures that plaintext never resides on the server: files are encrypted before upload using AES-GCM with per-file keys and nonces, and decryption keys are wrapped and released only to authenticated, explicitly authorized users. An administrator mediates every cross-user access, generating immutable audit trails that align security enforcement with governance and compliance expectations.
Our evaluation demonstrates that this combination of cryptographic rigor and organizational control is feasible without sacrificing usability. Under departmental/SME workloads, the system sustained interactive performance with p95 $\leqslant$ 5 s for 50 MB transfers at 100 concurrent users, while maintaining zero unauthorized reads, perfect tag verification on tamper tests, and resilience to common web threats through defensive coding and hardened session management. Reliability experiments showed graceful degradation and quick recovery from process restarts, transient network faults, and storage pressure, all while preserving audit continuity. A formative usability study reported good task completion times and a SUS score in the "good" range, indicating low friction for both end users and administrators.
The results also clarify where effort should focus as deployments grow. Network I/O, not cryptography, is the dominant bottleneck at scale, suggesting straightforward optimizations—multipart transfers, S3 offload with pre- signed URLs, HTTP/2 or HTTP/3, and CDN integration. On the governance side, human approval time can be reduced through role-based access control, bulk actions, and policy-based auto-approval for low-risk scenarios without weakening accountability. Finally, integrating hardware-backed key custody (e.g., KMS/HSM), proxy re-encryption for delegated access, and immutable audit storage can harden the system for regulated environments.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 5, November 2025
Impact Factor: 7.67

REFERENCES

[1] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext- Policy Attribute-Based Encryption," Proc. IEEE Symp. Security and Privacy (S&P), 2007. cs.ucla.edu

[2] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing," Proc. IEEE INFOCOM, 2010. (Author copy) cnsr.ictas.vt.edu

[3] C.-K. Chu, S. S. M. Chow, W.-G. Tzeng, J. Zhou, and R. H. Deng, "Key-Aggregate Cryptosystem for Scalable Data Sharing in Cloud Storage," IEEE Trans. Parallel and Distributed Systems, vol. 25, no. 2, 2014. (Open copy) kresttechnology.com

[4] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy- Preserving Public Auditing for Data Storage Security in Cloud Computing," Proc. IEEE INFOCOM, 2010. (Open summary)

[5] W.-F. Hsien, T.-S. Chen, and S.-T. Hsu, "A Survey of Public Auditing for Secure Data Storage in Cloud Computing," Int. J. Network Security, 2016 (surveys IEEE works incl. authorized/dynamic auditing)

[6] X. Yang and X. Jia, "An Efficient and Secure Dynamic Auditing Protocol for Data Storage in Cloud Computing," IEEE Trans. Parallel and Distributed Systems, 2013 (cited in auditing surveys).

[7] G. Wang, Q. Liu, and J. Wu, "Hierarchical Attribute- Based Encryption and Scalable User Revocation for Sharing Data in Cloud Servers," IEEE Trans. Parallel and Distributed Systems, 2011. (Author copy)

[8] Z. Zhou and D. Huang, "On Efficient Ciphertext- Policy Attribute-Based Encryption and Broadcast Encryption," (references CP-ABE revocation; cites IEEE S&P'07), ACM (contextual pointer to IEEE lineage).

[9] K. Yang, X. Jia, et al., "Attribute-Based Access Control with Efficient Revocation in Data Outsourcing Systems," IEEE Trans. Parallel and Distributed Systems, 2011 (referenced by later access-control surveys).

[10] K. O. B. Agyekum, Q. Xia, E. B. Sifah, C. N. A. Cobblah, H. Xia, and J. Gao, "A Proxy Re-Encryption Approach to Secure Data Sharing in the Internet of Things Based on Blockchain," IEEE Systems Journal, (early access/future issue)