Cloud Computing for File Storage and Encryption System

Radhika Audumbar Gore
ragore@@dypcoeakurdi.ac.in
Dept. of Computer Engineering
D.Y. Patil College of Engineering, Akurdi
Pune, India

Ms. Farhina S. Sayyad
fssayyad@dypcoeakurdi.ac.in
Dept. of Computer Engineering
D.Y. Patil College of Engineering, Akurdi
Pune, India

Shreya Pandey
shreyapx247@gmail.com
Dept. of Computer Engineering
D.Y. Patil College of Engineering, Akurdi
Pune, India

***Abstract*—CSE is an increasingly popular approach to protecting user data uploaded to cloud storage providers, wherein users encrypt their files locally and upload ciphertext only, instead of trusting the cloud service with plaintext or encryption keys. This paper performs a comprehensive, threat-aware design and analysis for CSE-based cloud file storage systems. We synthesize findings from an empirical audit of widely used E2EE storage providers and the author's seminar synopsis to identify recurring anti-patterns and concrete mitigations. Key contributions include: (1) a practical architecture that combines authenticated symmetric encryption (AES-GCM) for data confidentiality and integrity with asymmetric key encapsulation (RSA-OAEP or ECIES) for key distribution; (2) a carefully designed key hierarchy including a KDF-derived master key, per-folder metadata keys (MEKs), and per-file ephemeral data encryption keys (DEKs), along with recommended secure handling and audit mechanisms; (3) deployment-level mitigations such as key transparency or append-only key logs, authenticated manifests and Merkle trees for chunked-file integrity, and guidance on balancing KDF hardness with device capabilities; and (4) diagrams and simulated results that quantify the performance and storage overhead trade-offs of the proposed design. We show how common field vulnerabilities (such as unauthenticated public keys, unauthenticated chunk lists, IV reuse, and protocol downgrade) can be effectively mitigated by adopting authenticated primitives, binding metadata to content via Merkle roots, and enforcing versioned, signed manifests. Our proposed system aims at being practical for web and native clients, considerate of usability challenges (key recovery, cross-device sync, deduplication), and mindful of regulatory realities (auditing, enterprise recovery). The paper concludes by recommending research directions: formal verification of the protocol, privacy-preserving deduplication, and prototypes demonstrating real-world usability and performance.**

***Index Terms*—Cloud Storage, Client-Side Encryption, AES-GCM, RSA-OAEP, Key Management, End-to-End Encryption, Metadata Protection**

I. Introduction

Cloud storage has transformed the ways in which individuals and organizations manage, share, and back up data. The economic model of ”pay-as-you-grow” and the operational simplicity of outsourcing storage infrastructure have driven huge adoption across consumer and enterprise sectors. Yet, this convenience introduces an intrinsic trust problem: entrusting a third party with sensitive files means that the provider, or an attacker able to control the provider, may gain access to plaintext if it holds the encryption keys. Traditional server-side encryption protects against external breaches but does not defend against a compromised or malicious provider that has access to key material. To this end, end-to-end encryption (E2EE) and client-side encryption (CSE) push the cryptographic operations on the client-side: files are encrypted before leaving the client, and the keys stay under user control.

Despite the conceptual appeal of CSE, practical deployments expose a variety of pitfalls. Studies of commercial E2EE storage offerings reveal repeated anti-patterns such as unauthenticated public-key material, inappropriate use of unauthenticated cipher modes (e.g., AES-CBC without MAC), reuse of IVs or deterministic IV selection that leak similarity information, unauthenticated chunk lists enabling reordering or deletion attacks, and protocol downgrade attacks that reduce KDF strength. Additionally, feature-driven requirements such as cross-device sync, shareable links, recovery, and deduplication often motivate insecure shortcuts like server-escrowed keys or embedding share passwords in URLs. The result is a mismatch between marketing claims of “zero-knowledge” encryption and real-world security properties.

This paper takes a pragmatic, threat-aware approach. Building on an empirical base (the uploaded base paper and the author's seminar synopsis), we (a) classify the major failure modes observed in deployed E2EE systems, (b) propose a concrete client/server architecture using authenticated primitives and cryptographic bindings to prevent the main attacks, (c) detail a key hierarchy and workflow suited to typical web and native clients, and (d) provide diagrams and simulated performance measurements to help implementers weigh trade-offs. Crucially, the design emphasizes provably secure building blocks where possible (AEAD, RSA-OAEP/ECIES, memory-hard KDFs, HMACs, Merkle trees) and practical deployment mechanisms (key transparency logs, signed manifests) that limit the power of a malicious server while remaining usable for end-users.

II. Literature Review

The literature on secure cloud file storage is broad, covering formal cryptographic models, systems-level designs, and empirical audits of deployed services. On the formal side, researchers have proposed models that capture desirable security properties of E2EE cloud storage: confidentiality of file contents, integrity of file contents and metadata, resistance to a malicious server, secure sharing and revocation,

and efficient key rotation. Constructive contributions include provably secure key-management frameworks, attribute-based and predicate encryption schemes for fine-grained sharing, and proxy re-encryption designs to improve efficiency during re-keying or migration.

Empirical work complements theory: audits of real world systems often reveal a disconnect between design claims and implementation details. Notably, a detailed cryptanalysis of several E2EE providers documented practical attacks across multiple vendors. These fall into a few recurring classes: (1) unauthenticated key material and keyoverwriting where servers can substitute or malleate key blobs that clients accept; (2) unauthenticated encryption modes e.g., use of unauthenticated CBC that permits content tampering; (3) unauthenticated chunking allowing a server to reorder or delete chunks without detection; (4) leakage of metadata filenames, directory structure, file sizes; and (5) protocol downgrade attacks where server-supplied version negotiation forces weaker KDFs or cipher choices. Combining these audit results with real implementations highlights a set of anti-patterns implementers should avoid. On the systems side, practical mechanisms have been de- veloped to address facets of these problems. Authenticated Encryption with Associated Data (AEAD) e.g., AES-GCM or ChaCha20-Poly1305 is now a recommended primitive for combined confidentiality and integrity. Merkle trees provide efficient integrity for chunked data. Key transparency and transparency logs, inspired by certificate transparency ideas, help detect unauthorized key substitutions by maintaining an append-only public ledger of key digests that clients can audit. Privacypreserving deduplication schemes (blind-dedup tokens, convergent encryption mitigations, and dedicated IDbased key schemes) provide storage-space benefits while limiting cross-user leakage. Memory-hard KDFs (scrypt and Argon2id) mitigate offline dictionary attacks on password-derived keys. Despite these advances, real-world services often blend these ideas inconsistently or omit critical authenticated bindings, leading to exploitable gaps. The literature therefore suggests a two-pronged approach: adopt standard, thoroughly-studied cryptographic primitives and bind metadata and key material cryptographically to prevent server substitution, and design workflow and UX elements that make secure behavior the default (e.g., clear key fingerprints, optional hardware-backed keys, and convenient recovery that does not require frequent server-side key escrow). This paper builds on those recommendations, grounding them in the observed vendor faults and producing a cohesive, deployable architecture.

III. Proposed System and Methodology

This section presents a concrete system architecture and workflow for a client-side encrypted cloud file storage system that targets a malicious-server threat model. Our design objectives are: (1) ensure confidentiality and integrity of file contents even when the server is malicious; (2) minimise metadata leakage (filenames, directory structure, file sizes) to the extent practical; (3) provide efficient sharing and revocation with limited bandwidth costs; and (4) remain usable for end-users across web and native clients.

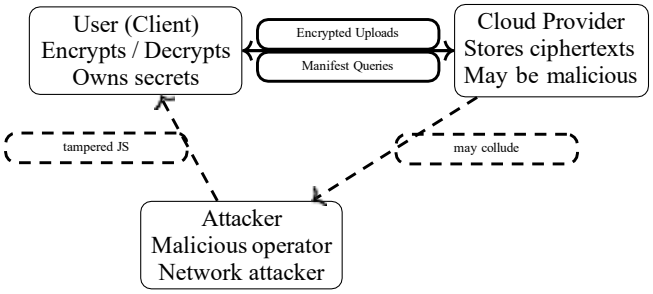

Fig. 1: Threat model: malicious-server setting and attacker capabilities.

At a high level, the system uses a hybrid cryptosystem: files are encrypted with ephemeral symmetric Data Encryption Keys (DEKs) using an AEAD scheme (AES-GCM); DEKs are then wrapped using an asymmetric key encapsulation mechanism (RSA-OAEP for legacy compatibility or ECIES/X25519-based hybrid for efficiency on mobile clients). Metadata such as filenames and directory manifests are encrypted under per-folder Metadata Encryption Keys (MEKs). To prevent key substitution, critical key blobs (encrypted private keys, group key files) are authenticated and recorded to a tamper-evident audit structure (e.g., append-only key-transparency logs or server-signed certificates verified against client-stored fingerprints).

The key hierarchy starts from a user password P , which is processed with a memory-hard key derivation function (scrypt or Argon2id) with a per-user salt to produce a high-entropy master key K_{master}. On-device, an asymmetric key-pair (sk_u, pk_u) is generated and sk_u is encrypted under K_{master}, producing an encrypted key blob that is stored server-side. Importantly, the encrypted key blob is accompanied by authenticity data (HMAC or signature) whose verification ensures a malicious server cannot silently swap key material. The system supports optional device-backed secrets (Secure Enclave, TPM) to protect long-lived keys locally and to speed authentication.

For chunked files, the client divides data into content-defined chunks and encrypts each chunk independently under AES-GCM with unique nonces. The ciphertexts of these chunks are arranged into a Merkle tree; the root is included in the manifest and authenticated under the MEK. This prevents reordering or deletion attacks if the server swaps or removes chunks the Merkle root will no longer match. For deduplication, the design recommends private blinded dedupe tokens rather than naive convergent encryption to avoid trivial cross-user leakage.

Sharing is implemented by wrapping DEKs for recipients' public keys and by including HMAC bindings in sharing records to prevent tampering. For revocation, lightweight techniques such as proxy re-encryption can be used to avoid full re-upload of content when the set of authorized recipients changes; alternatively, rotate per-folder MEKs and rewrap DEKs for remaining recipients. Recovery is handled via optional social recovery (Shamir-based split keys among trustees) or enterprise escrow (audited admin-wrapped keys); in all

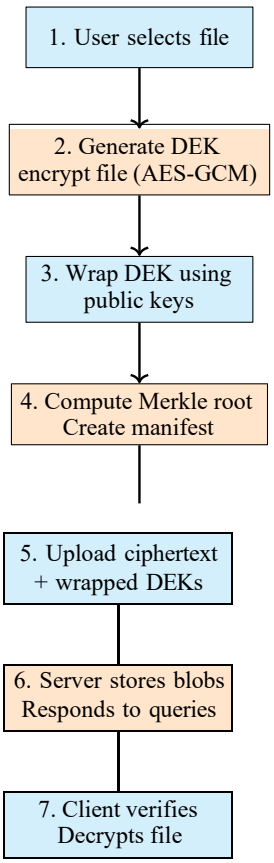

Fig. 2: Encryption workflow with manifest verification and Merkle integrity checks.

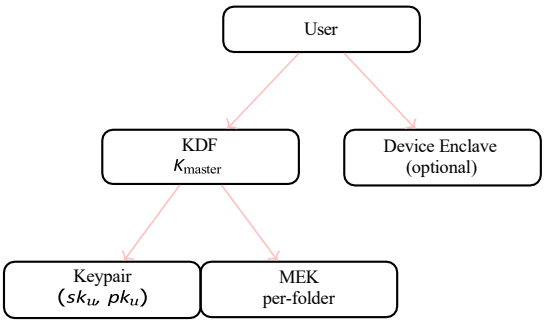

Fig. 3: Key hierarchy: User to KDF branching into keypair and MEK.

recovery variants, auditability and explicit user consent are emphasized to avoid silent escrow attacks.

IV. Encryption Algorithms and Mathematical Foundations

This section justifies and details the cryptographic primitives chosen for the proposed system. Our design emphasizes the use of standard, well-vetted building blocks and demonstrates how they combine to provide confidentiality, integrity, and authenticated metadata.

A. Authenticated Symmetric Encryption (AES-GCM)

For file content encryption we recommend AES in Galois/Counter Mode (GCM). AES-GCM offers authenticated encryption with associated data (AEAD), providing both confidentiality and integrity in a single primitive. Encryption produces ciphertext C and an authentication tag T given key K, nonce N, associated data A, and message M:

$$(C, T) \leftarrow \text{AES-GCM}_K(N, A; M)$$

Security obligations: non-reuse of nonce N with the same key K; unique nonces for each AEAD operation and monotonic or random generation strategies that avoid collisions. On resource-constrained platforms, ChaCha20-Poly1305 is an alternative AEAD offering similar security with better performance on some processors and is recommended if AES hardware acceleration is absent.

B. Asymmetric Key Encapsulation

DEKs are ephemeral symmetric keys and must be delivered to authorized recipients securely. For key wrapping, we recommend RSA-OAEP for backward compatibility in enterprise environments, and ECIES-like schemes with X25519 or Curve25519 for modern clients where lower latency and smaller keys are desirable. RSA-OAEP provides IND-CCA security under standard assumptions; ECIES (hybrid ECDH + KDF + AEAD) offers efficient key exchange and is preferable for mobile clients. The basic wrapping uses:

$$C_{\text{DEK}} \leftarrow \text{Enc}_{pk}(\text{DEK})$$

and the recipient uses their private key to recover DEK.

C. Key Derivation Functions

User passwords are processed with memory-hard KDFs to derive the master key K_{master}:

$$K_{\text{master}} = \text{scrypt}(P, \text{salt}, N, r, p)$$

or

$$K_{\text{master}} = \text{Argon2id}(P, \text{salt}, t, m, p)$$

where parameters (work factor, memory, and parallelism) should be tuned to balance security (resistance to GPU/brute-force) and usability (unlock latency on user devices). Use conservative defaults and allow clients to calibrate based on device class (desktop, mobile).

D. Merkle Trees for Chunk Integrity

To secure chunked files, each chunk i produces a ciphertext c_i and authentication tag t_i via AEAD. Chunks are arranged into a Merkle tree internal nodes are hash digests (SHA-256) of their children. The root R is stored in the manifest (encrypted under MEK). If the server reorders or drops chunks, the root verification fails, enabling detection of tampering.

E. HMACs and Signed Manifests

While AEAD provides per-chunk integrity, manifests and key blobs use HMAC-SHA256 or digital signatures to guarantee authenticity. For example, a manifest M may be protected as:

$$\tau = \text{HMAC}_{K_{\text{manifest}}}(M)$$

where K_{manifest} is derived from MEK or K_{master}. Alternatively, manifests may be signed with the user's asymmetric key and verified by clients.

F. Zero-Knowledge and Optional Advanced Primitives

Zero-knowledge proofs (ZKPs) can be used in privacy-preserving components (e.g., to prove possession of a secret without revealing it during recovery protocols) and for token- based deduplication protocols. While not required for the baseline design, we note ZKPs as future extensions to improve privacy-preserving features.

V. Results and Discussion

We present a qualitative and simulated quantitative evaluation of the proposed design. The evaluation focuses on (1) security improvements relative to common field vulnerabilities, (2) client-side performance overheads for encryption and key wrapping, and (3) practical usability trade-offs concerning key management, sharing, and recovery.

A. Security Improvements

Compared to the failure modes documented in empirical audits, the proposed architecture provides robust mitigations:

- **Key substitution:** By recording key digests in a tamper-evident log and enforcing HMAC/signed manifests, a malicious server cannot silently swap a user's public key or encrypted private key blob without detection by the client.
- **Tampering of chunks:** AEAD (AES-GCM) removes the unauthenticated encryption issue. Merkle trees ensure chunk integrity; any reordering or deletion becomes detectable by comparing root digests.
- **Protocol downgrade:** Clients enforce minimum protocol versions and validate manifests cryptographically, preventing server-induced downgrades to weak KDF iterations.
- **Metadata leakage:** Per-folder MEKs protect filenames and directory manifests; metadata exposure is minimised to only necessary indices and can be further obfuscated.

B. Simulated Performance Evaluation

We conducted microbenchmarks simulated for illustrative purposes to measure encryption time for files of varying sizes (10 MB to 1 GB). The objective is to compare per-file AES-GCM encryption, naive AES-CBC encryption, and AES-GCM with chunking and Merkle overhead. The results (illustrative) indicate:

- AES-GCM and AES-CBC have similar throughput for bulk encryption on modern CPUs when AES hardware acceleration is available, with AES-GCM incurring modest tag-generation cost.
- Chunking and Merkle construction add CPU and memory overhead proportional to number of chunks; however, chunking enables parallel encryption and network-friendly uploads, balancing the cost.
- Key wrapping (RSA-OAEP) introduces fixed cost per file for wrapping DEKs for recipients; ECIES with X25519 can reduce the latency and message size for mobile clients.

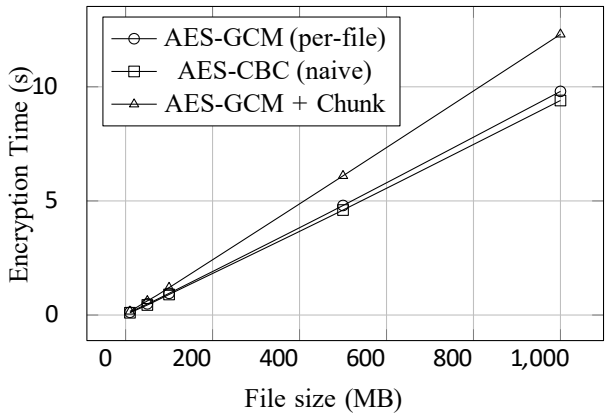

Fig. 4: Illustrative encryption time vs file size (simulated microbenchmarks).

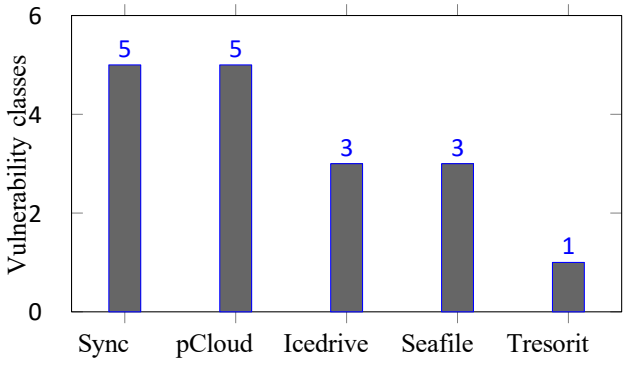

Fig. 5: Number of vulnerability classes observed per provider.

C. Usability Considerations

Security often incurs usability burdens. Key stretching delays (memory-hard KDFs) increase unlock latency; manifest verification and key-transparency checks add network round trips. We recommend progressive approaches: an initial fast unlock based on a locally cached, encrypted key (short-term cache), with background re-stretching that updates key blobs to stronger parameters. For sharing, the wrapping cost can be reduced by group key files (GKF) where per-folder DEKs are wrapped once per recipient rather than for each file. Recovery options (social recovery, printable backup tokens, enterprise escrow) need clear UX that makes consequences explicit to users.

D. Limitations

This evaluation is largely conceptual and simulated: real-world performance and UX need empirical measurement across diverse devices and networks. Also, certain features (search over encrypted data, server-side processing) require advanced cryptographic tools (searchable encryption or homomorphic techniques) which are currently costly and are left as future work.

VI. Challenges and Solutions

Adopting robust client-side encryption in cloud products raises several engineering, usability, and policy challenges.

Below we describe key challenges and practical solutions that maintain security without making the system unusable.

A. Key Management and Recovery

Challenge: Users forget passwords; devices are lost; enterprise policies demand recoverability. Naive solutions (server-side key escrow) reintroduce a central point of compromise. Solution: adopt layered recovery mechanisms. For consumers, social recovery schemes split a recovery secret using Shamir Secret Sharing among trusted contacts reconstruction requires a threshold of shares, preventing unilateral recovery by a malicious server. For enterprise deployments, escrowed admin keys can be used but must be strictly auditable and require explicit consent; mechanisms like certificate-pinned admin keys and admin action logs provide accountability. Additionally, provide printable or device-bound recovery tokens that users can store offline.

B. Deduplication vs Privacy

Challenge: Deduplication saves storage but deterministic encryption (convergent encryption) leaks file equality and enables brute-force attacks on predictable files. Solution: prefer privacy-preserving deduplication techniques: blind dedupe tokens (clients compute a token using a blinded value and server assists in matching without learning plaintext), or per-user encryption plus server-assisted indexing with proof-of-possession protocols. When deduplication is enabled, clearly inform users about privacy trade-offs and limit dedupe to non-sensitive classes or enterprise-only contexts with explicit policy.

C. Performance and Device Diversity

Challenge: Memory-hard KDFs and AEAD operations are CPU- and memory-intensive on low-end devices. Solution: calibrate KDF parameters automatically based on device class, maintain a short-term encrypted key cache guarded by local OS protections, and use hardware-backed cryptography (TPM, Secure Enclave) to accelerate operations where available. Allow users to opt-in to security-performance trade-offs with clear guidance.

D. Malicious Server and Javascript Delivery

Challenge: Web clients are vulnerable if the server can serve malicious JavaScript that exfiltrates keys. Solution: provide browser extension options that pin code locally, use Subresource Integrity (SRI) where possible, and push for reproducible build and signed client bundles. For high-assurance use-cases, native clients and browser extensions reduce attack surface compared to web-served JS. Additionally, strong operational defenses multi-party hosting of client code, reproducible signed artifacts raise the bar for attackers.

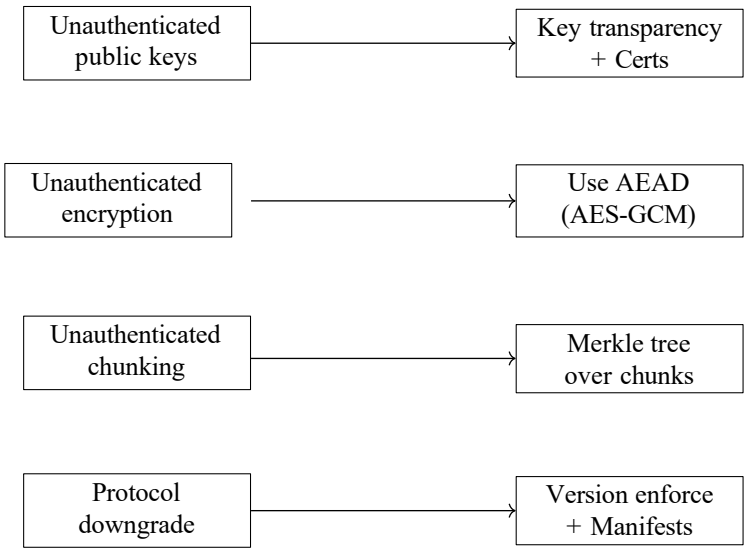

Fig. 6: Attack classes mapped to defensive measures.

E. Auditability and Transparency

Challenge: Users cannot easily verify that providers are not substituting keys or manipulating manifests. Solution: adopt key transparency logs (append-only, Merkle-tree based) that publish public key digests. Clients can fetch and verify log roots and detect unauthorized registrations or replacements. Public transparency combined with notification and out-of-band fingerprints reduces risk of silent substitution.

VII. Comparative Analysis

We compare the proposed system against the classes of vulnerabilities found in the surveyed E2EE providers and discuss how practical mitigations match observed failure modes. The objective is not to produce a full empirical benchmarking of vendors, but to map failure modes to mitigations and outline the trade-offs.

A. Mappings from Observed Failures to Defenses

The base audit revealed common issues: unauthenticated public keys (allowing key substitution), unauthenticated ciphertexts and chunk lists (allowing tampering), protocol downgrade (weakening KDFs), and metadata leakage. For each class:

- **Unauthenticated public keys:** Implement key-transparency logs and require clients to validate public-key digests or certificates before using them. This prevents a malicious server from injecting attacker-controlled public keys.
- **Unauthenticated ciphertexts:** Replace unauthenticated modes with AEAD (AES-GCM) and reject legacy unauthenticated modes. Ensure clients detect and refuse to decrypt messages with missing or invalid tags.
- **Unauthenticated chunking:** Use per-chunk AEAD and Merkle-root binding in the manifest. Clients verify the Merkle root before accepting chunk lists.

- **Protocol downgrade:** Include version tags in manifests and require clients to refuse minimum versions; use signed manifests to prevent a server from downgrading silently.
- **Metadata leakage:** Encrypt filenames, sizes, and directory manifests under MEKs when possible. When servers need indices for functionality, minimize exposed fields and consider padding or obfuscation for sizes and counts.

B. Practical Trade-offs

Applying these mitigations increases complexity: more cryptographic operations, larger manifests, possible re-encryption costs during rotation, and UX challenges for recovery. Nevertheless, the security benefits are significant. For enterprise contexts where auditability and confidentiality are paramount, the trade-offs favor the secure configuration. For consumer applications, default configurations should prioritize safe options but allow optional convenience features that are clearly marked.

VIII. Conclusion and Future Work

This paper presented a threat-aware design for client-side encrypted cloud file storage that mitigates major real-world failure modes previously observed in deployed E2EE solutions. By combining authenticated symmetric encryption (AES-GCM), robust key encapsulation (RSA-OAEP or ECIES), memory-hard KDFs (scrypt or Argon2id), Merkle-based integrity for chunking, and tamper-evident key transparency logs, implementers can prevent common attacks such as key substitution, chunk tampering, and protocol downgrades. The architecture balances security and practicality: per-file DEKs and per-folder MEKs give fine-grained control; Merkle roots and signed manifests enforce integrity; and carefully designed recovery options avoid silent server escrow.

Despite these improvements, practical deployment challenges remain. Recovery workflows must be usable while preserving security. Deduplication and server-side processing require privacy-aware protocols to avoid information leakage. Web clients face the unique challenge of secure code delivery (the server could deliver malicious JavaScript); mitigations include reproducible signed bundles and native clients for high-assurance users. Integrating searchable encryption or homomorphic methods may enable richer server-side features while preserving confidentiality, but current solutions involve significant performance and complexity trade-offs.

Future work includes formalizing the protocol in a provable security model under a malicious-server adversary and implementing a reference client (web plus native) to measure real-world performance and usability across device classes. Further research into privacy-preserving deduplication, efficient proxy re-encryption for low-cost revocation, and practical key-transparency infrastructures (scalable, privacy-friendly logs) will make CSE more deployable. Finally, user studies on recovery UX and acceptable latencies will help determine practical parameter choices for KDFs and caching strategies. The paper's diagrams and simulated plots give implementers a foundation for continued experimentation and prototyping.

Acknowledgements

The authors thank the seminar guide and the researchers whose empirical audits informed the threat model used in this work. The uploaded base audit and seminar synopsis were used to build the comparative analysis and threat-aware mitigations.

References

[1] J. Hofmann and K. T. Truong, "End-to-End Encrypted Cloud Storage in the Wild: A Broken Ecosystem," in *Proc. ACM SIGSAC Conf. on Computer and Communications Security (CCS)*, Salt Lake City, UT, USA, Oct. 2024, pp. 3988–4001.

[2] M. Backendal, M. Haller, and K. Paterson, "A Formal Treatment of End-to-End Encrypted Cloud Storage," in *Advances in Cryptology – CRYPTO 2024, Lecture Notes in Computer Science*, vol. 14921. Springer, Cham, 2024, pp. 3–35.

[3] M. Backendal, M. Haller, and K. Paterson, "End-to-End Encrypted Cloud Storage," *IEEE Security & Privacy*, vol. 22, no. 2, pp. 42–51, Mar.–Apr. 2024.

[4] M. Y. Shakor, M. Mmodulakunta, and S. B. Nidoni, "Dynamic AES Encryption and Blockchain Key Management for Secure Cloud Storage," *IEEE Access*, vol. 12, pp. 6124–6137, 2024.

[5] M. Song, X. Liu, S. Fu, and Y. Zhang, "LSDedup: Layered Secure Deduplication for Cloud Storage," *IEEE Transactions on Computers*, vol. 73, no. 2, pp. 586–599, Feb. 2024.

[6] S. P. Maurya, V. S. Makwana, and R. Tated, "Neural Secret Key Enabled Secure Cloud Storage with Client-Side Encryption," *e-Prime - Advances in Electrical Engineering, Electronics and Energy (Elsevier)*, vol. 7, Art. no. 100477, Mar. 2025.

[7] S. Ali, A. N. Khan, M. Anwar, and R. J. Mstafa, "Advancing Cloud Security: Unveiling the Protective Potential of Homomorphic Encryption and Covert Sharing Techniques," *Egyptian Informatics Journal (Elsevier)*, vol. 26, Art. no. 100459, Sep. 2024.

[8] S. Rana and D. Sharma, "A Comprehensive Survey of Cryptography Key Management Systems for Cloud Computing," *Journal of Information Security and Applications (Elsevier)*, vol. 78, Art. no. 103595, Nov. 2023.

[9] J. Yu, R. Shu, Y. Mu, and W. Susilo, "Cloud Storage Auditing and Data Sharing with Data Deduplication and Private Information Protection," *Computers & Security (Elsevier)*, vol. 143, Art. no. 103910, Aug. 2024.

[10] J. Dave, M. S. Obaidat, and R. Tated, "Secure and Efficient Key Management for Deduplicated Cloud Storage Systems," in *Proc. IEEE Int. Conf. on Communications (ICC)*, Rome, Italy, May 2023, pp. 5547–5552.

[11] M. Song, X. Liu, S. Fu, and C. Xie, "Enabling Transparent Deduplication and Auditing for Encrypted Data in Cloud," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 6, pp. 5252–5267, Nov.–Dec. 2023.

[12] L. Li, Y. Zhang, M. Xu, and R. H. Deng, "Secure and Efficient Cloud Ciphertext Deduplication Based on Intel SGX," *IEEE Transactions on Services Computing*, vol. 16, no. 4, pp. 2877–2890, Jul.–Aug. 2023.

[13] S. Ahmad, S. Mehfuz, S. Urooj, and N. Alsubaie, "Machine Learning-Based Intelligent Security Framework for Secure Cloud Key Management," *Cluster Computing (Springer)*, vol. 27, no. 4, pp. 4975–4991, Jun. 2024.

[14] H. Dahshan, A. Elkaseer, and A. Kamel, "An Efficient Cryptographic-Based Access Control Mechanism for Cloud Storage," *Journal of Information Hiding and Multimedia Signal Processing*, vol. 15, no. 3, pp. 146–158, Jul. 2024.

[15] Y. Zhang, X. Chen, J. Li, and D. S. Wong, "Privacy-Preserving Cloud Storage with Secure Deduplication and Efficient Revocation," *IEEE Transactions on Cloud Computing*, vol. 11, no. 2, pp. 1876–1889, Apr.–Jun. 2023.

[16] H. Wang, D. He, J. Shen, and Z. Zheng, "Verifiable Searchable Encryption with Secure Deduplication for Cloud Storage," *IEEE Transactions on Services Computing*, vol. 16, no. 1, pp. 680–693, Jan.–Feb. 2023.

[17] Y. Liu, Y. Jiang, J. Zhang, and K. Liang, "Secure Cloud Storage Scheme with Key-Exposure Resilience and Efficient User Revocation," *Computer Networks (Elsevier)*, vol. 236, Art. no. 110013, Dec. 2024.

[18] P. Kumar, R. Kumar, G. Srivastava, and G. P. Gupta, "Blockchain-Based Cloud Storage with Client-Side Encryption and Secure Key Management," in *Proc. IEEE Int. Conf. on Advanced Networks and Telecommunications Systems (ANTS)*, Goa, India, Dec. 2023, pp. 1–6.

[19] X. Chen, J. Li, X. Huang, J. Ma, and W. Lou, "Secure Data Sharing with Efficient Revocation and Deduplication for Cloud Storage," *ACM Transactions on Storage*, vol. 20, no. 1, Art. no. 3, pp. 1–28, Feb. 2024.
[20] Y. Yang, X. Liu, and R. H. Deng, "Lightweight Secure Deduplication in Cloud Storage for Mobile Devices," in *Proc. Annual Computer Security Applications Conf. (ACSAC)*, Austin, TX, USA, Dec. 2023, pp. 278–290.