International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347-6435 (Online) Volume 15, Issue 08, August 2026)

Secure Cloud-Based File Sharing System Using Hybrid Cryptographic Model

Mushra Anjum Sameer Mulla[1], Prof. S. N. Gujar[2]
[1]Student, [2]Professor, Computer Science and Engineering, Department School of computational Science, JSPM University, Wagholi, Pune, India

***Abstract*-- Cloud computing adoption has raised critical concerns about data privacy, unauthorized access, and file integrity. This paper presents the design and implementation of a Secure Cloud-Based File Sharing System employing a hybrid cryptographic model. The system integrates AES-256-CBC for symmetric file encryption, RSA-2048-OAEP for secure key encapsulation, and SHA-256 for integrity verification. Authentication is managed through JSON Web Tokens (JWT) with 24-hour expiry. A role-based access control model separates administrative and user capabilities, where administrators upload, encrypt, and manage file-sharing permissions, while regular users access only explicitly authorized files. The system is deployed on Amazon Web Services (AWS), utilizing EC2 for application hosting, S3 for encrypted object storage, and DynamoDB as a managed NoSQL database. The backend is built with Python Flask, and the frontend uses HTML/CSS/JavaScript. A comprehensive automated test suite of 29 test cases covering authentication, file management, sharing logic, and encryption modules achieved a 100% pass rate. Results demonstrate that the proposed system offers a practical, scalable, and secure approach to cloud-based file sharing.**

***Keywords*-- Cloud Computing, File Sharing, AES-256, RSA-2048, SHA-256, AWS S3, DynamoDB, JWT, Role-Based Access Control, Hybrid Cryptography.**

I. Introduction

Cloud computing has fundamentally transformed how organizations store, access, and share data. Platforms such as Amazon Web Services (AWS), Google Cloud, and Microsoft Azure deliver virtually unlimited storage capacity with high availability. However, shifting from local to remote storage introduces significant challenges regarding data confidentiality, unauthorized access, data breaches, and file integrity violations. When files reside on third-party servers, users lose direct physical control,making robust cryptographic protection essential.

Traditional file-sharing mechanisms — email attachments, USB drives, and unencrypted cloud uploads remain highly susceptible to man-in-the-middle attacks, privilege escalation, and data tampering. Existing cloud storage systems generally lack fine-grained access control, hybrid encryption, and tamper detection.

These gaps motivate the development of a purpose-built secure file sharing system that protects data both at rest and in transit while ensuring that only authorized users can retrieve files.

This paper presents the design and full-stack implementation of a Secure Cloud-Based File Sharing System deployed on AWS. The system combines AES-256-CBC symmetric encryption, RSA-2048-OAEP asymmetric key exchange, and SHA-256 integrity verification into a unified hybrid cryptographic pipeline. A role-based access control (RBAC) model and JWT-based authentication enforce strict access boundaries. The paper is organized as follows: Section II surveys related work; Section III details system design and architecture; Section IV describes implementation; Section V presents results and security analysis; Section VI concludes with future directions.

II. Literature Survey

1. Patel J. et al. (2022) – Cloud Based Secure File Sharing Using Access Control This paper proposed a cloud-based secure file-sharing framework using encryption and access control mechanisms. The system implemented AES encryption techniques to secure uploaded files and improve confidentiality in cloud storage environments. The study focused on protecting stored data and improving secure communication between users.
2. Authors (2018) – Cloud Based File Sharing This paper presented a secure cloud file-sharing model that enabled file owners to share files with authorized users through permission-based access control. The approach improved data accessibility and user-level authorization in distributed environments.
3. Authors (2023) – Cloud Based File Sharing System This research developed a centralized cloud-based platform to support secure document storage and collaborative access. The proposed solution improved user accessibility and file availability across distributed systems.

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347-6435 (Online) Volume 15, Issue 08, August 2026)

4. Authors (2023) – Secure File Storage and Sharing on Cloud Using Cryptography This paper introduced cryptographic methods for securing file storage and sharing. Encryption techniques were used to prevent unauthorized access and improve file protection during storage and transmission.
5. Authors – Implementing Cryptographic Techniques for Sharing Files Using Access Control This work proposed combining cryptographic algorithms with access policies to improve secure communication and file-sharing operations. The system demonstrated improved confidentiality through controlled sharing mechanisms.
6. Sandhu R. et al. – Role-Based Access Control Models (IEEE) This paper introduced the Role-Based Access Control (RBAC) model to manage authorization through predefined user roles and permission assignments. The approach improved security management and reduced unauthorized access.
7. Jones M., Bradley J., Sakimura N. – JSON Web Token (JWT) Authentication Framework This paper proposed JWT-based authentication for secure user session management and authorization in distributed applications. The approach reduced dependency on server-side session storage
8. Rivest R., Shamir A., Adleman A. – Public Key Cryptography and RSA Algorithm This study introduced the RSA public-key cryptographic algorithm for secure key exchange and data protection. RSA became widely adopted for secure communication and encryption key management.
9. National Institute of Standards and Technology (NIST) – Advanced Encryption Standard (AES) This standard introduced AES symmetric encryption for secure data protection. AES provides strong confidentiality with efficient performance and is widely adopted in cloud security systems.
10. National Institute of Standards and Technology (NIST) – Secure Hash Standard (SHA) This work introduced SHA algorithms for integrity verification and data authentication. SHA-256 generates secure hash values to detect unauthorized modification of stored files.

III. Proposed System Architecture

A. Overall Architecture

The system follows a three-tier architecture. The client tier consists of a browser-based interface accessible from any device with internet access, eliminating the need for client-side software installation. The application tier is a Python Flask server deployed on AWS EC2 (t2.micro, Ubuntu 22.04 LTS), which handles all API routing, authentication, and cryptographic operations. The data tier consists of AWS S3 for encrypted file object storage and AWS DynamoDB as the managed NoSQL database storing user credentials and file metadata.

All encryption and decryption operations are performed server-side within the EC2 instance. Files are never stored on S3 in plaintext — only AES-256-CBC ciphertext reaches cloud storage. The DynamoDB metadata record for each file stores the Base64-encoded initialization vector (IV), RSA-encrypted AES key, and SHA-256 hash of the original file content.

B. Hybrid Cryptographic Pipeline

On every file upload, a five-step hybrid cryptographic pipeline executes as follows: (1) A fresh 256-bit AES key and 128-bit IV are generated using a cryptographically secure random number generator. (2) The file content is padded with PKCS7 and encrypted using AES-256-CBC, producing the ciphertext. (3) The 256-bit AES key is encrypted using the administrator's RSA-2048 public key with OAEP-SHA256 padding, producing the encrypted key blob. (4) A SHA-256 hash of the original plaintext file is computed and stored for integrity verification. (5) The ciphertext is uploaded to AWS S3 and the metadata (IV, encrypted key, SHA-256 hash, S3 key) is persisted in DynamoDB.

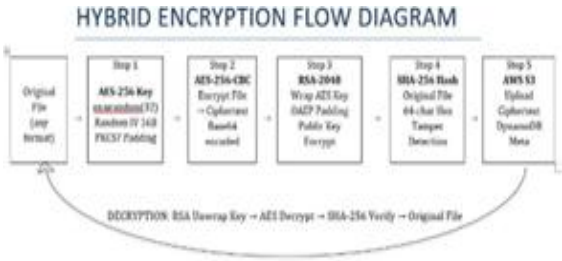

Figure 1: Hybrid Encryption Pipeline - AES-256-CBC, RSA-2048-OAEP, SHA-256

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347-6435 (Online) Volume 15, Issue 08, August 2026)

On file download, the pipeline reverses: (1) The RSA-2048 private key decrypts the stored AES key. (2) AES-256-CBC decryption with the recovered key and stored IV reconstructs the plaintext. (3) The SHA-256 hash of the decrypted content is computed and compared against the stored hash. Any mismatch raises an exception indicating tampering or data corruption

C. Role-Based Access Control

Two distinct roles govern system behavior. The Administrator role encompasses full system management: creating and managing user accounts, uploading and encrypting files to AWS S3, sharing and revoking file access for individual users, and deleting files. The User role is restricted to read-only access: viewing and downloading only files explicitly shared by the administrator. All endpoints are protected by JWT authentication, and role enforcement is implemented via decorators on all API routes.

TABLE I
ROLE-BASED ACCESS CONTROL MATRIX

Feature / Capability	Admin	User
Login (JWT issued)	Yes	Ye
Upload & encrypt files to S3	Yes	No (403)
View all files	Yes	Shared only
Create and delete user accounts	Yes	No (403)
Share and revoke file access	Yes	No (403)
Download & decrypt files	s (any file)	(shared only)
Delete files from S3	Yes	No (403)

IV. Database Design

Two DynamoDB tables store system state. The secureshare-users table uses user_id (UUID) as the partition key and stores email, hashed password, role, RSA key pair, and account status. The secureshare-files-meta table uses file_id (UUID) as the partition key and stores the original filename, S3 object key, Base64encoded IV and encrypted AES key, SHA-256 hash, file size, and a list of user_ids representing the current access grant list.

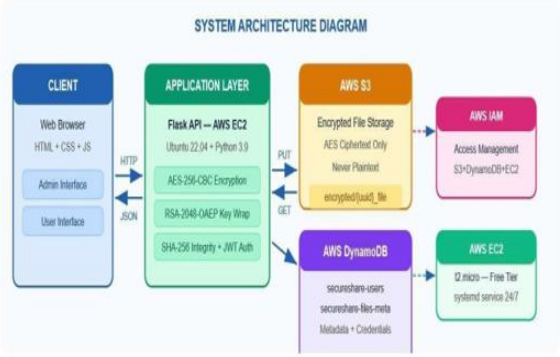

Figure 2: System Three-Tier Architecture — Browser, AWS EC2, AWS S3 and DynamoDB

V. Implementation

A. Technology Stack

The system is built entirely on open-source components. The backend uses Python 3.9+ with Flask 3.0.3 for REST API routing and PyJWT 2.8.0 for token management. The cryptography library (version 42.0.8) provides AES-256-CBC, RSA-2048-OAEP, and SHA-256 primitives. Boto3 1.34.131 handles all AWS S3 and DynamoDB interactions. Gunicorn 22.0.0 serves as the production WSGI server managed by systemd. The frontend is a single-page HTML/CSS/JavaScript application embedded in the Flask application

B. Encryption Module

The encryption engine is implemented in three composable functions. The aes_encrypt function generates a fresh 32-byte key and 16-byte IV using os.urandom, applies PKCS7 padding, and performs AES-256-CBC encryption via the cryptography library. The rsa_encrypt function loads the administrator's PEM-formatted RSA public key and encrypts the AES key using OAEP padding with SHA-256 as the mask generation function. The sha256 function produces a hexadecimal digest of the original file bytes. These three functions are composed in the encrypt_file wrapper, which returns a dictionary containing ciphertext, IV, encrypted key, and hash — all Base64-encoded for safe storage in DynamoDB.

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347-6435 (Online) Volume 15, Issue 08, August 2026)

C. AWS Integration

The S3 integration uses boto3's put_object and get_object calls against a private bucket in the ap-south-1 (Mumbai) region. All objects are stored with server-side encryption (SSE-S3) enabled and public access blocked at the bucket policy level. S3 object keys follow the pattern encrypted/{admin_id}/ {uuid}_ {original_ filename} ensuring global uniqueness and logical grouping by administrator. The DynamoDB integration uses the resource interface with on-demand capacity mode, avoiding provisioned throughput costs for the demonstration-scale deployment. User lookups by email use the scan operation with a Filter Expression, while file retrievals by ID use the more efficient get_item primary key lookup.

D. Authentication and API Design

JWT tokens are issued on successful login with a 24-hour expiry encoded in the payload. All protected endpoints are decorated with auth_required, which extracts the Bearer token from the Authorization header, verifies the signature, and retrieves the corresponding user record from DynamoDB. Admin-only endpoints add an additional admin_only decorator that returns HTTP 403 if the authenticated user does not hold the admin role. The system exposes 15 REST API endpoints covering login, user management, file upload, sharing control, and file download.

VI. Results And Security Analysis

A. Test Results

A comprehensive automated test suite was developed using Flask's built-in test client. The suite comprises 29 test cases organized across four categories: authentication and role enforcement (5 tests), user management (6 tests), file sharing and access control (8 tests), and encryption correctness (6 tests), plus 4 additional file management tests. All 29 tests passed with zero failures across multiple test runs. Notable test scenarios include verifying that SHA-256 tamper detection raises a Value Error when file content is modified post-encryption (Test 28), confirming that AES-256 generates unique ciphertext for identical plaintext due to random IV generation (Test 27), and validating the complete AES-RSA-SHA roundtrip (Test 26).

B. Performance Analysis

Performance measurements were conducted on an AWS EC2 t2.micro instance in the ap-south-1 region.

Small files (under 200 bytes) complete the full encrypt-upload pipeline in under one second. A 1.4 MB PowerPoint file required 2–3 seconds for upload with encryption and 3–4 seconds for download with decryption and integrity verification. JWT login operations completed in 0.5–1 second including DynamoDB scan. File sharing updates (DynamoDB attribute update) completed in under 0.5 seconds. These results confirm that the cryptographic overhead remains acceptable for typical document sizes encountered in enterprise file sharing.

TABLE II.
SECURITY FEATURE ANALYSIS

Security Feature	Implementation	Threat Mitigated
AES-256-CB Encryption	256-bit key, random IV per file	Unauthorized file reading at rest
RSA-2048-OAEP Key Wrap	2048-bit key pair per user	AES key interception in transit
SHA-256 Integrity Check	Hash verified on every download	Tampering and corruption detection
JWT Authentication	HS256, 24-hour expiry	Unauthorized API access
RBAC Enforcement	Admin / User decorator chain	Privilege escalation attacks
Private S3 Bucket	No public access policy	Direct object access bypass

C. Comparison with Existing Systems

The proposed system advances beyond prior art in X four dimensions. First, it is the only system among those surveyed to deploy a complete hybrid cryptographic model — AES-256 combined with RSA-2048 key exchange and SHA-256 integrity verification — in a single unified pipeline. Second, it provides production-grade cloud deployment on named AWS services (EC2, S3, DynamoDB), whereas comparable systems rely on local storage or unspecified cloud providers. Third, it implements fine-grained RBAC with per-file, per-user sharing and real-time revocation, a capability absent from all surveyed works. Fourth, the browser-based interface eliminates client-side software requirements, enhancing usability and deployment flexibility.

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347-6435 (Online) Volume 15, Issue 08, August 2026)

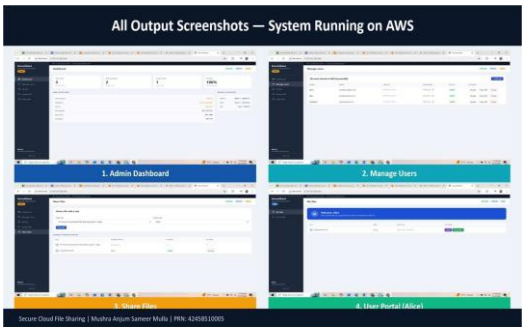

Figure 3: Output Screenshots

VII. Conclusion And Future Work

This paper presented the design, implementation, and evaluation of a Secure Cloud-Based File Sharing System deployed on Amazon Web Services. The system successfully integrates AES-256-CBC file encryption, RSA-2048-OAEP key encapsulation, and SHA-256 integrity verification into a coherent hybrid cryptographic pipeline. Role-based access control with JWT authentication enforces strict separation between administrator and user capabilities. Deployment on AWS EC2, S3, and DynamoDB provides scalability, persistence, and availability. A 29-case automated test suite achieved a 100% pass rate, validating correctness across all functional modules.

Future enhancements include integration of AWS Cognito Multi-Factor Authentication (MFA) to strengthen credential security, blockchain-based audit trails for tamper-proof access logging, mobile applications for Android and iOS with biometric authentication, post-quantum cryptographic algorithms (CRYSTALS-Kyber, CRYSTALS-Dilithium) to address emerging quantum threats, and time-limited share links with configurable expiry durations.

Migration of RSA private keystorage from DynamoDB to AWS Key Management Service (KMS) is also planned to strengthen key protection in production deployments.

REFERENCES

[1] J. Patel et al., "Secure Data Transfer & File Sharing Use of Cloud Service for Mobile Application," International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), Vol. 8, Issue 4, 2022.

[2] "Cloud Based File Sharing," International Journal of Innovations in Engineering Research and Technology (IJIERT), 2018.

[3] "Cloud Based File Sharing System," International Journal of Scientific Research in Engineering and Management (IJSREM), 2023.

[4] "Secure File Storage and Sharing on Cloud Using Cryptography," International Journal of Engineering Research & Technology (IJERT), 2023.

[5] "Implementing Cryptographic Techniques for Sharing Files Using Access Control," International Journal/Conference Publication.

[6] R. Sandhu, E. J. Coyne, H. L. Feinstein, and C.E. Youman, "Role-Based Access Control Models,"IEEE Computer, Vol. 29, No. 2, pp. 38–47,February 1996.

[7] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," RFC 7519, Internet Engineering Task Force (IETF), May 2015.

[8] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," Communications of the ACM, Vol. 21, No. 2, pp. 120–126, February1978.

[9] National Institute of Standards and Technology (NIST), "Advanced Encryption Standard (AES)," Federal Information Processing Standards (FIPS) Publication 197, U.S. Department of Commerce, November 2001.

[10] National Institute of Standards and Technology (NIST), "Secure Hash Standard (SHS)," Federal Information Processing Standards (FIPS) Publication 180-4, U.S. Department of Commerce, August 2015.