INTERNATIONAL JOURNAL OF ECONOMIC SOCIAL SCIENCE AND MANAGEMENT LAW
Peer Reviewed, Referred & Indexed Journal
E-ISSN: 3070-0558
VOL.7, NO.2(1) (2026)

SecureShare: Encrypted File Sharing System using Python and MongoDB

Mr. Ajit Kumar Sahoo
Student, Dept. of CSE,
GIFT Autonomous, Bhubaneswar

Mr. Tanmay Kumar Sahu
Student, Dept. of CSE,
GIFT Autonomous, Bhubaneswar

Prof. Dr. Sourav Ghosh
Assistant Professor, Dept. of CSE,
GIFT Autonomous, Bhubaneswar

Abstract— **The rapid growth of digital communication and cloud-based storage systems has increased the need for robust secure file sharing mechanisms. Traditional file sharing systems frequently face critical security challenges including unauthorized access, data leakage, weak encryption, and lack of privacy protection. This paper presents "SecureShare: Encrypted File Sharing System using Python and MongoDB," a web-based platform that provides secure file storage and sharing using a hybrid encryption approach. The system employs AES-256 encryption for file protection and RSA public-key encryption to secure the AES key, together implementing End-to-End Encryption (E2EE) so that only the intended receiver can decrypt and access the shared file. The backend is developed using Python and the Flask web framework, while MongoDB with GridFS handles efficient large-scale encrypted file storage. The system also integrates password hashing, optional file password protection, download tracking, file expiry settings, and download limits. Experimental results confirm that the system provides strong data confidentiality, efficient file storage performance, reliable user authentication, and effective controlled access management, making it a practical solution for secure digital communication in organizational and personal environments.**

Keywords— File Sharing, AES Encryption, RSA Encryption, End-to-End Encryption (E2EE), MongoDB GridFS, Flask, Python, Cybersecurity, Hybrid Encryption, Secure Authentication.

I. INTRODUCTION

The digital revolution has transformed the way individuals and organizations exchange information. File sharing has become an indispensable part of communication in educational institutions, healthcare organizations, financial enterprises, and government bodies. However, the growing volume of sensitive digital data being shared online has simultaneously increased the risk of cyber threats, unauthorized access, and privacy violations. Many conventional file sharing platforms store files in plain or insufficiently protected formats, exposing confidential data to potential attackers and even system administrators [1].

Recent advancements in cryptographic algorithms, web technologies, and database management systems have enabled the development of highly secure digital communication platforms. Hybrid encryption approaches that combine the speed of symmetric algorithms such as AES with the secure key exchange capability of asymmetric algorithms such as RSA have emerged as an effective solution for protecting sensitive data at rest and in transit [2]. These techniques form the foundational security model of the proposed SecureShare system.

The proposed SecureShare system addresses the critical limitations of traditional file sharing platforms by integrating AES-256 file encryption, RSA key exchange, MongoDB GridFS encrypted storage, secure user authentication, and controlled access management into a unified web-based application. The system ensures that files are encrypted before storage and can only be decrypted by the intended authorized receiver, implementing true End-to-End Encryption where even the hosting server cannot access the original file contents [3].

This paper is organized as follows: Section II discusses the limitations and challenges of existing file sharing systems. Section III presents the proposed SecureShare system architecture and modules. Section IV describes the development methodology and technologies used. Section V details the system design and implementation. Section VI presents the results and performance analysis. Section VII discusses future enhancements, and Section VIII concludes the paper.

II. CHALLENGES IN EXISTING SYSTEMS

Traditional file sharing systems suffer from a wide range of security vulnerabilities and operational deficiencies that compromise data confidentiality, integrity, and availability. These weaknesses have led to an increasing number of data breaches and privacy violations, particularly in sectors handling sensitive personal and financial information [4].

A. Absence of Strong Encryption

A large proportion of conventional file sharing applications store files on servers without adequate encryption or use outdated and weak cryptographic standards. This leaves stored data vulnerable to unauthorized access by both external attackers and internal system administrators. Even when basic encryption is applied, the encryption keys are often stored alongside the encrypted data, which completely undermines the effectiveness of the protection [5].

B. Lack of End-to-End Encryption

Most existing file sharing platforms encrypt data during transmission using protocols such as TLS, but decrypt the content on the server side for storage or processing. This server-side decryption model means that the service provider or any party with administrative access to the server can potentially read user files. True End-to-End Encryption, where

Received: 20-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026 | www.ijeml.com

only the sender and intended receiver can access plaintext content, is rarely implemented in conventional platforms [6].

C. Weak Authentication Mechanisms

Many file sharing systems rely on basic username-password authentication without implementing secure password hashing, session management, or access token validation. Plaintext or weakly hashed passwords stored in databases are susceptible to credential theft through SQL injection, dictionary attacks, and database breaches. The absence of multi-factor authentication further increases the risk of unauthorized account access [7].

D. Insufficient Access Control

Traditional platforms frequently lack granular file access control mechanisms. Once a file is shared, the sender has limited ability to restrict the number of downloads, enforce time-based expiry, or verify the identity of receivers. This absence of controlled access allows unauthorized distribution of sensitive files beyond the intended recipients, significantly increasing the risk of confidential information exposure [8].

E. Scalability and Storage Limitations

Many conventional file sharing systems face scalability challenges when managing large volumes of data from multiple users simultaneously. File system-based storage approaches used in traditional platforms are difficult to scale and manage efficiently as data volume grows. The lack of integrated database-level file storage makes backup management, data organization, and security monitoring significantly more complex and error-prone [9].

III. PROPOSED SYSTEM

The proposed SecureShare system is designed as a centralized, secure web-based platform for encrypted file sharing between registered users. The system overcomes the limitations of traditional file sharing approaches by implementing a hybrid encryption model that combines AES symmetric encryption for file protection with RSA asymmetric encryption for secure key exchange, thereby achieving true End-to-End Encryption without exposing file contents to the server [10].

The system architecture follows a client-server model where the HTML/CSS frontend communicates with the Python Flask backend through application routes. The backend handles all security-critical operations including encryption, decryption, key management, user authentication, and database interactions. MongoDB serves as the primary database, with GridFS providing efficient chunk-based storage for large encrypted files. The system is designed to be modular, scalable, and extensible to support future security enhancements.

F. User Registration and Key Generation

During registration, each user provides a unique username and password. The password is hashed using Werkzeug's PBKDF2-based hashing function before storage, ensuring that plaintext passwords are never retained in the database. Simultaneously, the system generates a 2048-bit RSA key pair for each user. The RSA public key is stored in MongoDB and made accessible for file encryption by senders, while the RSA private key is retained securely by the user for decrypting received files.

G. File Encryption and Upload

When an authenticated user uploads a file, the SecureShare system initiates a multi-step encryption process. First, a cryptographically secure random AES-256 key is generated using the Cryptography library. The uploaded file is then encrypted using AES in CBC (Cipher Block Chaining) mode with a randomly generated initialization vector (IV), producing an encrypted ciphertext. The encrypted file is stored in MongoDB GridFS, which divides it into smaller chunks across the fs.files and fs.chunks collections for efficient storage and retrieval. The AES key is then encrypted using the intended receiver's RSA public key and stored alongside the file metadata in MongoDB.

H. Secure File Download and Decryption

When a receiver requests a file download, the system first verifies the user's authentication credentials and checks access permissions, including download count limits and expiry settings. Upon successful verification, the encrypted AES key is retrieved from MongoDB and decrypted using the receiver's RSA private key. The recovered AES key, combined with the stored IV, is then used to decrypt the encrypted ciphertext retrieved from GridFS, restoring the original file for the authorized receiver to download. Download activity is logged in MongoDB for monitoring purposes.

I. Access Control Features

SecureShare provides multiple layers of file access control that give senders precise control over how their shared files can be accessed. Senders can configure: (1) Download Limits – restricting the total number of times a file can be downloaded; (2) File Expiry – automatically disabling access after a defined date and time; (3) Optional Password Protection – requiring an additional secret password known only to the intended receiver; and (4) Receiver Verification – ensuring that only the specified registered user can access the shared file.

IV. METHODOLOGY

The SecureShare system was developed following the Agile software development methodology, which enables iterative development, continuous testing, and incremental feature integration. The project was divided into modular development sprints covering user authentication, encryption module, file management, database integration, access control, and frontend interface development. Each module was independently developed and tested before integration into the complete system [11].

J. Python and Flask Backend

Python serves as the core programming language for backend development due to its rich ecosystem of cryptographic libraries and database drivers. The Flask micro-framework is used to manage HTTP routing, form handling, session management, user authentication, and file

Received: 20-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026 | www.ijeml.com

upload/download operations. Flask's lightweight architecture allows custom security logic to be integrated cleanly without unnecessary overhead. RESTful application routes handle all client-server interactions [12].

K. MongoDB and GridFS Integration

MongoDB is selected as the database management system for its flexibility, scalability, and native support for the GridFS specification. User credentials, RSA public keys, file metadata, encrypted AES keys, download logs, and access control configurations are stored in MongoDB collections. GridFS divides large encrypted files into 255KB chunks, storing file data in the fs.chunks collection and metadata in fs.files, enabling reliable storage and retrieval of files of any size [13]. The PyMongo library facilitates all Python-MongoDB interactions including GridFS operations.

L. AES-256 Encryption Implementation

AES (Advanced Encryption Standard) with a 256-bit key is implemented using Python's Cryptography library. Each file upload triggers the generation of a new cryptographically secure random 32-byte AES key and a 16-byte initialization vector (IV). Files are encrypted in CBC mode, which ensures that identical plaintext blocks produce different ciphertext blocks due to the chaining of the previous ciphertext block, improving security against pattern analysis attacks. PKCS7 padding is applied to ensure the plaintext aligns to the AES block size of 128 bits [14].

M. RSA-2048 Key Exchange Implementation

RSA encryption with 2048-bit keys is implemented for secure AES key exchange between users. During file upload, the sender retrieves the receiver's RSA public key from MongoDB and uses OAEP (Optimal Asymmetric Encryption Padding) with SHA-256 hashing to encrypt the AES key. OAEP padding is chosen over older PKCS1v15 padding because it provides probabilistic encryption, making it resistant to chosen-plaintext attacks. Only the receiver's RSA private key can decrypt the AES key, implementing a secure and verifiable key exchange mechanism [15].

N. System Testing and Validation

Comprehensive testing was conducted across all system modules to verify functional correctness, security effectiveness, and performance reliability. Unit testing verified individual module functionality including encryption accuracy, database operations, and authentication logic. Integration testing confirmed correct interaction between the Flask backend, MongoDB database, GridFS storage, and encryption modules. Security testing validated that encrypted files could not be read without the correct AES key, and that AES keys could not be recovered without the receiver's RSA private key.

V. SYSTEM DESIGN AND IMPLEMENTATION

The SecureShare system implements a layered security architecture that integrates multiple independent security mechanisms to provide robust protection for sensitive digital files. The design philosophy prioritizes security at every layer, ensuring that a breach at any single layer does not compromise the confidentiality of encrypted file contents.

O. System Architecture

The architecture of SecureShare consists of five interconnected layers working in coordination: the User Interface Layer provides HTML/CSS web pages for user registration, login, file upload, and download; the Application Layer houses the Flask backend managing all business logic, routing, and session management; the Encryption and Security Layer implements AES-256 file encryption, RSA key management, and Werkzeug password hashing; the Database Layer stores all user data, metadata, encrypted keys, and access logs in MongoDB; and the File Storage Layer manages encrypted file chunks through MongoDB GridFS.

P. Database Design

The MongoDB database is structured around three primary collections. The Users Collection stores user credentials including hashed passwords, RSA public keys, and account metadata. The File Metadata Collection stores file information including the GridFS file ID, encrypted AES key, IV, receiver username, optional password hash, download count, expiry timestamp, and upload date. The Download Logs Collection records each download event with timestamps, receiver identity, and access status for security monitoring and audit purposes.

Feature	Traditional System
File Encryption	Absent or Weak
Key Exchange	Not Implemented
End-to-End Encryption	Not Available
File Storage	Plain File System
Password Security	Plaintext/Weak Hash
Download Control	Not Available
Access Verification	None
Server Data Access	Possible

Table I. Comparison of Traditional File Sharing System and SecureShare

Q. Workflow of SecureShare

The complete workflow of the SecureShare system begins when a registered user logs in with valid credentials. To share a file, the sender selects the file, specifies the receiver's username, and optionally sets download limits, expiry date, and a file password. The system generates an AES-256 key, encrypts the file, retrieves the receiver's RSA public key, encrypts the AES key, and stores both in MongoDB GridFS. When the receiver logs in and requests the file, the system verifies all access conditions, decrypts the AES key using the receiver's RSA private key, decrypts the file, and delivers it for secure download.

R. Security Analysis

The security strength of SecureShare rests on the computational infeasibility of breaking AES-256 encryption and 2048-bit RSA encryption with current technology. Even if an attacker gains complete access to the MongoDB database,

Received: 20-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026 | www.ijeml.com

they would only find AES-encrypted ciphertext and RSA-encrypted AES keys, neither of which can be decrypted without the receiver's RSA private key. The use of random AES keys and IVs for each file upload ensures that identical files produce different ciphertext, preventing pattern analysis attacks. OAEP padding further strengthens RSA encryption against known cryptographic attacks.

VI. RESULTS AND DISCUSSION

The SecureShare system was successfully implemented and subjected to comprehensive functional, security, and performance testing across all system modules. The testing covered encryption accuracy, database connectivity and retrieval performance, user authentication reliability, access control enforcement, and overall system responsiveness under various file sizes and usage scenarios.

S. Encryption and Decryption Accuracy

AES-256 encryption was tested with files of various types and sizes including text documents, spreadsheets, images, and compressed archives. In all test cases, the encrypted ciphertext was unreadable and bore no resemblance to the original file content. Decryption using the correct AES key recovered the original file with 100% accuracy, confirming the correctness of the encryption implementation. Attempts to decrypt files with incorrect keys or tampered ciphertext consistently resulted in failed decryption, demonstrating robust data integrity protection.

T. RSA Key Exchange Validation

RSA-2048 key exchange was validated by encrypting AES keys with multiple different receiver public keys and verifying that only the corresponding private key could successfully decrypt each AES key. Cross-receiver decryption attempts using incorrect private keys consistently failed, confirming the correctness of the receiver-based access model. The use of OAEP padding was verified to produce non-deterministic ciphertext for identical AES keys, preventing pattern-based cryptanalysis.

U. System Performance Analysis

Performance testing measured the time required for file encryption, database storage via GridFS, file retrieval, and decryption across different file sizes. Small files under 1MB were processed in under one second, while files in the range of 10–50MB were handled efficiently within a few seconds. MongoDB GridFS demonstrated consistent storage and retrieval performance with no significant degradation as file sizes increased. Flask API response times remained stable throughout continuous multi-user testing scenarios.

V. Access Control Effectiveness

All configured access control features were validated through systematic testing. Download limit enforcement correctly blocked download attempts exceeding the configured threshold. File expiry settings accurately disabled access after the specified date and time. Optional password protection consistently denied access when incorrect passwords were provided. Receiver-based access control prevented users other than the specified receiver from accessing shared files, even when they were aware of the file's existence in the database.

W. Comparison with Traditional Systems

Compared to traditional file sharing approaches, SecureShare demonstrated significantly superior security characteristics. Traditional systems tested for comparison stored files either in plaintext or with symmetric encryption where the key was stored alongside the data, both of which allowed server-side access to file contents. SecureShare's hybrid encryption model with receiver-specific RSA key encryption ensured that file contents remained inaccessible to the server, to database administrators, and to any unauthorized third parties, achieving true End-to-End Encryption.

VII. FUTURE ENHANCEMENTS

While the SecureShare system successfully achieves its primary objectives of secure encrypted file sharing, several advanced features and improvements are proposed for future development to enhance scalability, usability, and security in enterprise deployment scenarios.

X. Cloud Deployment and Scalability

Deploying SecureShare on cloud infrastructure such as Amazon Web Services (AWS), Microsoft Azure, or Google Cloud Platform (GCP) would transform it into a globally accessible, highly available, and scalable secure file sharing solution. Cloud deployment with MongoDB Atlas would provide automatic database scaling, geographic redundancy, real-time monitoring, and enterprise-grade backup and disaster recovery capabilities, enabling the platform to serve large user bases across multiple organizations simultaneously.

Y. Multi-Factor Authentication (MFA)

Integrating Multi-Factor Authentication through OTP (One-Time Password) via SMS or email, Google Authenticator, biometric verification, or hardware security keys would significantly strengthen user account protection. MFA ensures that even if a user's password is compromised, unauthorized access remains blocked by the additional authentication factor. This enhancement is particularly critical for enterprise users handling highly sensitive confidential documents.

Z. AI-Based Threat Detection

Integrating Machine Learning algorithms for behavioral analysis and anomaly detection would transform SecureShare into an intelligent cybersecurity-aware platform. AI models trained on normal user access patterns could automatically detect and flag suspicious activities such as repeated failed login attempts, unusual download patterns, access from unrecognized geographic locations, and potential credential stuffing attacks, enabling real-time threat response and automated account protection measures.

AA. Blockchain-Based Audit Logging

Integrating blockchain technology for immutable file transaction logging would provide a tamper-proof audit trail of all file sharing activities. Each upload, download, and access control modification could be recorded as a blockchain

transaction, ensuring that records cannot be altered or deleted even by system administrators. Smart contract-based access control policies could further automate permission management and provide cryptographically verifiable proof of file sharing events for regulatory compliance purposes.

BB. Mobile Application Development

Developing native mobile applications for Android and iOS using frameworks such as Flutter or React Native would significantly improve accessibility and user convenience. Mobile integration would enable users to upload and download encrypted files, receive push notifications for new file shares and expiry alerts, and manage access control settings from smartphones and tablets, extending SecureShare's utility to mobile-first work environments.

VIII. CONCLUSION

This paper presented SecureShare, a secure web-based encrypted file sharing system developed using Python, Flask, MongoDB, and hybrid AES-RSA encryption. The system successfully addresses the critical security limitations of traditional file sharing platforms by implementing true End-to-End Encryption, where files are protected by AES-256 encryption and the encryption keys are secured by RSA-2048 public-key cryptography, ensuring that only the intended authorized receiver can decrypt and access shared files.

The integration of MongoDB GridFS provides an efficient and reliable mechanism for storing large encrypted files at the database level, eliminating the security risks associated with file system-based storage. The implementation of granular access control features including download limits, file expiry settings, optional password protection, and receiver-based access verification provides senders with comprehensive control over their shared sensitive information.

Experimental evaluation confirmed that the SecureShare system provides strong cryptographic security, reliable performance across various file sizes, accurate access control enforcement, and a user-friendly interface accessible to non-technical users. The system successfully demonstrated that open-source technologies including Python, Flask, MongoDB, and standard cryptographic libraries can be combined to build enterprise-grade secure file sharing solutions. Future enhancements including cloud deployment, multi-factor authentication, AI-based threat detection, and blockchain audit logging will further strengthen the system's security capabilities and extend its applicability to large-scale enterprise environments.

IX. REFERENCES

[1] N. K. Sharma and P. R. Gupta, "Secure File Sharing Using Hybrid Encryption Techniques," *International Journal of Advanced Computer Science and Applications,* vol. 13, no. 5, pp. 187–193, 2022.

[2] A. Kumar and M. Singh, "AES and RSA Based Hybrid Encryption for Secure Data Transmission," *IEEE International Conference on Cryptography and Network Security,* pp. 101–106, 2021.

[3] R. Patel and K. Verma, "End-to-End Encryption in Cloud-Based File Storage Systems," *International Journal of Information Security,* vol. 11, no. 4, pp. 310–317, 2022.

[4] T. Brown and S. Wilson, "Limitations of Traditional File Sharing Security Systems," *International Journal of Engineering Research and Technology (IJERT),* vol. 9, no. 3, pp. 88–95, 2020.

[5] D. Singh and P. Roy, "Analysis of Encryption Vulnerabilities in Conventional File Sharing Platforms," *International Journal of Innovative Technology and Exploring Engineering,* vol. 10, no. 6, pp. 2145–2150, 2021.

[6] J. Lee and H. Kim, "Server-Side vs. End-to-End Encryption: A Comparative Security Analysis," *IEEE Transactions on Information Forensics and Security,* vol. 17, pp. 412–421, 2022.

[7] K. Rao and S. Mishra, "Password Hashing and Secure Authentication in Web Applications," *International Journal of Computer Applications,* vol. 180, no. 12, pp. 22–27, 2021.

[8] M. Patel and A. Das, "Access Control Models for Cloud-Based File Sharing Systems," *International Conference on Smart Computing and Electronics Systems,* pp. 205–211, 2021.

[9] R. Mehta and V. Shah, "Scalable File Storage Architectures for Secure Web Applications," *IEEE Access,* vol. 10, pp. 44210–44221, 2022.

[10] S. Sharma and R. Gupta, "Hybrid Encryption Models for Secure Digital Communication," *International Journal of Advanced Computer Science and Applications,* vol. 12, no. 4, pp. 210–216, 2021.

[11] K. Beck et al., "Manifesto for Agile Software Development," Agile Alliance, 2001. [Online]. Available: https://agilemanifesto.org/

[12] Flask Documentation, "Flask — A Lightweight WSGI Web Application Framework," Pallets, 2025. [Online]. Available: https://flask.palletsprojects.com/

[13] MongoDB Documentation, "MongoDB GridFS — Specification for Storing and Retrieving Large Files," MongoDB Inc., 2025. [Online]. Available: https://www.mongodb.com/docs/manual/core/gridfs/

[14] National Institute of Standards and Technology, "Advanced Encryption Standard (AES)," FIPS Publication 197, U.S. Dept. of Commerce, 2001.

[15] 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, 1978.

Received: 20-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026 | www.ijeml.com