# 

# **SECURE SHARE**

# 

**BY**

**ROHITH VARMA DATLA(AM.EN.U4CSE22119)**

**SHAIK REHAAN ASHRAF(AM.EN.U4CSE22153)**

# 

# **I. Introduction**

## **Brief Overview**

The project "Secure Share" is designed to facilitate secure file sharing between users by leveraging advanced cryptographic techniques. It ensures confidentiality, integrity, and accessibility of shared data through encryption and decryption processes, employing both symmetric and asymmetric cryptographic methods.

## **Objective**

The primary objective of this project is to provide a secure platform for users to share files. By using RSA for key exchange and AES for file encryption, the project ensures that data is protected from unauthorized access and interception during both storage and transfer.

# **II. Implementation**

## **Tools and Software Used**

1. **Programming Language**: JavaScript

2**. Framework**: Express.js for server-side logic  
 3. **Database**: MongoDB for storing user and file metadata  
 4. **Libraries and Modules**:  
 - **jsonwebtoken** for user authentication  
 - **bcrypt** for password hashing  
 - **crypto** for encryption and decryption  
 - **multer** for file handling  
 - **node-rsa** for RSA key generation  
 5. **Environment**: Node.js

6. **API testing:** Postman  
 7. **Storage**: Local directory for file storage

## **Description of Steps Taken**

**1. User Authentication**:  
 - Implemented signup and login functionalities using **JWT(SHA-256)** for token-based authentication.  
 - Passwords are securely hashed using **bcrypt** before storage.  
 - RSA key pairs are generated during user registration and stored securely in the database.

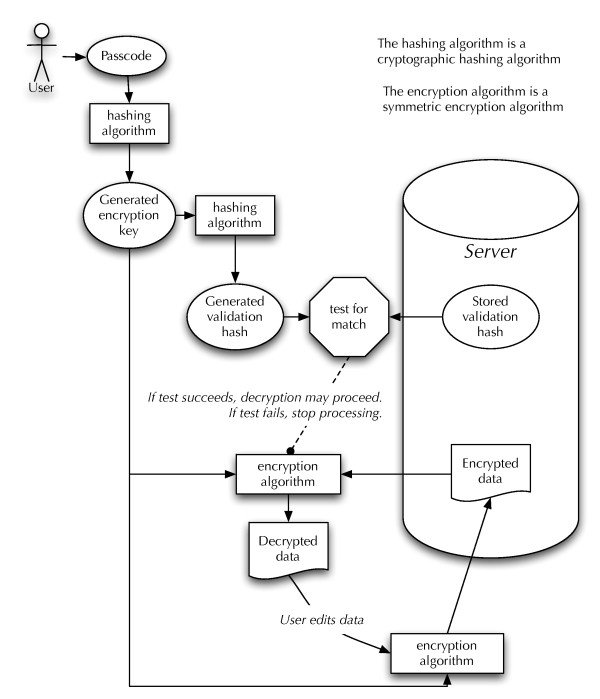
**2. File Upload**:  
 - Users upload files via a secure API endpoint.  
 - Files are encrypted using **AES-256** with randomly generated keys and IVs.  
 - The AES key and IV are further encrypted using the receiver’s RSA public key.  
 - Encrypted files and metadata are stored in the server.

**3. File Download**:  
 - Users can download files shared with them by providing a valid file ID.  
 - The receiver’s private RSA key is used to decrypt the AES key and IV.  
 - The decrypted AES key and IV are used to decrypt the file, which is then provided to user.

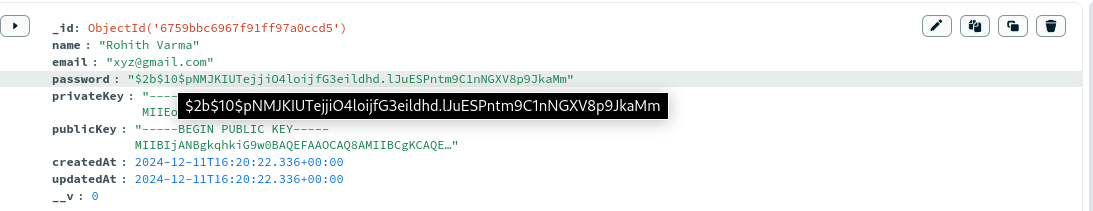
**4. Authorization and Security**:  
 - Middleware ensures only authorized users can access specific files.  
 - JWT tokens are validated for every request to ensure user authenticity.

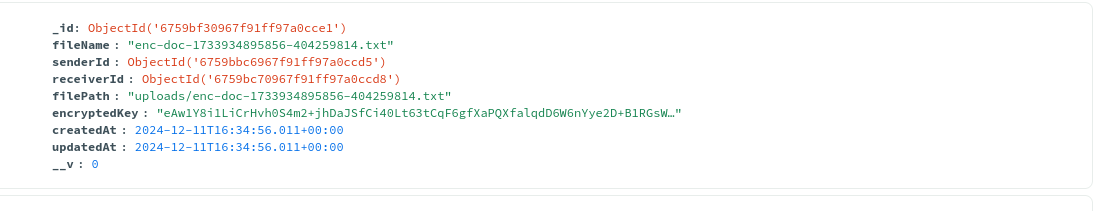
## **Screenshots and Diagrams**

**1. Architecture Diagram** - Diagram showing the flow of data and encryption steps.



**2. Screenshots** - Signup and login API tests.  
 - File upload and download endpoints.  
 - Encrypted file structure.  
 - Decrypted file download result.





# **III. Results**

## **Output or Results**

1. Secure file upload and download workflows were successfully implemented.  
 2. Files were encrypted and decrypted correctly, ensuring data integrity and confidentiality.  
 3. The RSA-based key exchange ensured secure distribution of AES keys.

## **Observations and Conclusions**

1. **Performance**: The system performs efficiently for files up to 50 MB. Larger files may require optimization.  
 2. **Security**: The combination of AES and RSA effectively secures files against unauthorized access.  
 3. **Ease of Use**: The API-based design provides a flexible interface for integration with other systems.

# **IV. Conclusion**

This project demonstrates the effectiveness of combining symmetric and asymmetric cryptographic techniques for secure file sharing. By implementing robust encryption mechanisms and user authentication, the platform ensures that sensitive data is protected from unauthorized access. Future enhancements may include scalability improvements, support for larger file sizes, and cloud integration for storage and sharing.