**File Storage and Sharing Using Google Storage**

**Sudhiksha Aradhyula, Kuheli Das, Aabha Bhide, Niharika Bandekar**

**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING ( AI & ML )**

**A . P SHAH INSTITUTE OF TECHNOLOGY , THANE , MAHARASHTRA**

---------------------------------------------------------------------\*\*\*--------------------------------------------------------------------

**ABSTRACT**

In today’s digital age, secure storage of sensitive data is more critical than ever. With the proliferation of data breaches, insecure access, and increasing reliance on cloud storage, traditional systems often fall short in providing end-to-end data protection. Our project addresses these challenges by developing a **Secure File Storage System** that employs **Hybrid Cryptography** and **Google Cloud Storage (GCS)** to ensure data confidentiality, integrity, and accessibility.

The system utilizes **AES** for encrypting file content due to its efficiency in handling large data, and **RSA** to securely encrypt the AES key, ensuring safe key distribution. Encrypted files are uploaded to Google Cloud Storage using the google-cloud-storage SDK, and users can retrieve and decrypt them locally via a secure dashboard. User credentials are managed through a Flask backend integrated with a MySQL database. Though the prototype stores plaintext passwords for simplicity, enhancements such as bcrypt hashing and cloud-based key management are suggested for production deployment.

By combining cloud storage, hybrid encryption, and secure authentication, the system ensures that only authorized users can access and decrypt their files. This report elaborates on the system architecture, technologies used, encryption methodology, security considerations, limitations, and future improvements.

**1. INTRODUCTION**

As organizations and individuals increasingly depend on cloud technologies for data storage, the need for robust security mechanisms has intensified. While cloud providers offer basic security measures, such as HTTPS and server-side encryption, these do not fully protect user data against unauthorized access or data breaches. Users have limited control over how their data is encrypted and managed in the cloud, leaving a gap for security-conscious applications that require end-to-end data protection.

To address these issues, this project introduces a **client-side encryption model** using **hybrid cryptography**, where the file is encrypted on the user's end before being uploaded to the cloud. By integrating **AES (Advanced Encryption Standard)** and **RSA (Rivest–Shamir–Adleman)** encryption, the system ensures both fast data processing and secure key transmission. AES efficiently encrypts the file, while RSA secures the AES session key by encrypting it with the recipient’s public key.

Furthermore, the integration with **Google Cloud Storage** offers reliability, scalability, and global accessibility for storing encrypted files. The backend is developed using Flask, a lightweight Python web framework, and MySQL handles user data and logs. This system not only guarantees data confidentiality and controlled access but also ensures that the encryption keys remain protected from exposure.

**2. LITERATURE SURVEY**

Prior work in the domain of secure file storage has explored both symmetric and asymmetric encryption methods, each with its advantages and limitations. AES, being a symmetric cipher, provides high-speed encryption but faces challenges in secure key distribution. RSA, on the other hand, excels in secure key exchange but is computationally expensive for large data sets.

A study by Bruce Schneier on cryptographic algorithms highlights the necessity of combining encryption types for optimal security. Hybrid cryptography, which leverages both AES and RSA, has been shown to provide a secure and scalable solution for cloud file storage systems. However, many of these systems either neglect cloud integration or provide limited automation in file handling and user access control.

Recent models have explored Google Cloud Storage as a potential backend for encrypted data due to its object storage capability and integration with client libraries. Still, a fully integrated system that combines **hybrid encryption, secure cloud storage, and user authentication** in a single web-based interface is rare. Our project closes this gap by offering a streamlined, cloud-integrated file storage platform with user-level control over encryption and access.

**3. EXISTING SYSTEM**

Existing file storage systems typically offer either local encryption or rely on cloud vendors to handle encryption at the backend. In most mainstream services, users upload unencrypted files, which are then encrypted server-side using provider-controlled keys. While this does offer protection from external threats, it doesn’t protect data from internal misuse or access by the cloud provider itself.

There are several standalone encryption tools (e.g., VeraCrypt, 7-Zip, BitLocker) that users can utilize before uploading files. However, these tools require manual processes, are prone to user error, and don’t scale well in web-based applications. Moreover, such systems don't integrate with cloud platforms or offer automated decryption/download features.

Another drawback of traditional systems is **key management**. Most users lack secure mechanisms to store and share encryption keys. This often results in insecure sharing via email or chat platforms, defeating the purpose of encryption. Our system automates key generation, encryption, storage, and decryption, thereby ensuring a complete and secure file handling workflow.

**3.1 SCOPE**

The system is designed to serve a wide range of use cases where secure file handling is critical, such as academic institutions, healthcare systems, financial firms, and law offices. It ensures that files can be uploaded, encrypted, stored in the cloud, and later retrieved and decrypted by authorized users.

Key components within scope include:

* **Hybrid encryption (AES + RSA)** for robust file protection.
* **Google Cloud Storage integration** for scalable file storage.
* **User dashboards** for managing uploads, downloads, and encrypted file views.
* **Secure file handling** from browser to cloud, ensuring no file remains unencrypted during transit.
* **Pluggable architecture** that can integrate future modules like Google Cloud KMS, biometric login, or file expiration rules.

While the initial implementation targets academic use with basic access control, the system can be extended into enterprise environments with more advanced security and compliance requirements, including audit logging, digital signatures, and role-based access control.

**3.2 PROPOSED SYSTEM**

The proposed system comprises three main components: a **Flask-based backend**, a **cloud storage client**, and a **hybrid encryption engine**. The system allows users to register and authenticate through a web interface. Once authenticated, users can upload files that are encrypted locally on the server using AES and RSA before being uploaded to Google Cloud Storage.

During the upload process, the system:

* Generates a **random AES session key**.
* Encrypts the file content using AES in **EAX mode**, which provides both **confidentiality and data integrity**.
* Encrypts the AES key with the **user’s RSA public key**.
* Combines the encrypted file, encrypted key, nonce, and tag into a single archive.
* Uploads this archive to a specified **Google Cloud Storage bucket**.

Upon file retrieval:

* The archive is downloaded from GCS.
* The AES session key is decrypted using the **user’s RSA private key**.
* The original file is then decrypted and served to the user.

All operations are logged in a MySQL database, and in future iterations, access analytics and automated expiration can be incorporated for better file lifecycle control.

**4. METHODOLOGY FOR IMPLEMENTATION**

The system's development followed a modular and agile methodology, starting with requirement identification and culminating in functional deployment. The design revolves around enabling secure user interaction with the file storage process, where encryption and decryption are performed automatically with minimal user intervention.

**1. Requirement Analysis:**  
This stage involved identifying essential features such as user registration/login, secure file upload and download, and encryption-decryption workflow. Non-functional goals like security, scalability, and simplicity were also defined.

**2. System Design:**  
The architecture follows a layered approach:

* **Presentation Layer:** HTML/CSS/JavaScript-based frontend.
* **Application Layer:** Flask handles the routing, encryption logic, and interaction with GCS.
* **Storage Layer:** GCS stores encrypted files; MySQL holds user data and metadata.  
  ER diagrams were created to map user, file, and key relationships. Flowcharts were drawn to define encryption and storage flows.

**3. Development:**  
The backend was developed using Flask and Python modules like pycryptodome for AES/RSA and google-cloud-storage for GCS interaction. Separate utility scripts (crypto\_utils.py, gcs\_utils.py) were created for encryption and storage functions, improving code modularity.

**4. Testing & Validation:**  
The system underwent unit testing for cryptographic operations and integration testing for file storage and retrieval. Edge cases like large file sizes, corrupted downloads, and missing keys were tested.

**5. SYSTEM REQUIREMENTS**

**Software Requirements:**

* Operating System: Linux/Windows/macOS
* Programming Language: Python 3.9+
* Web Framework: Flask
* Frontend: HTML5, CSS3, JavaScript
* Cloud SDK: google-cloud-storage
* Cryptography Library: PyCryptodome
* Database: MySQL

**Hardware Requirements:**

* Processor: Intel i3 or higher
* RAM: Minimum 4 GB
* Disk: 1 GB free for local testing
* Internet connection for GCS access
* Google Cloud Platform (GCP) account with billing enabled

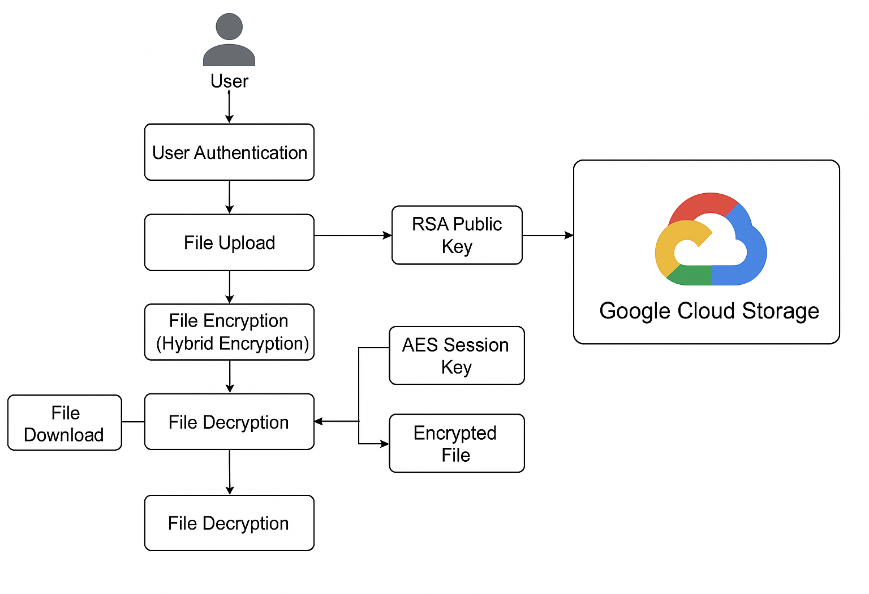
**6. SYSTEM FLOW**

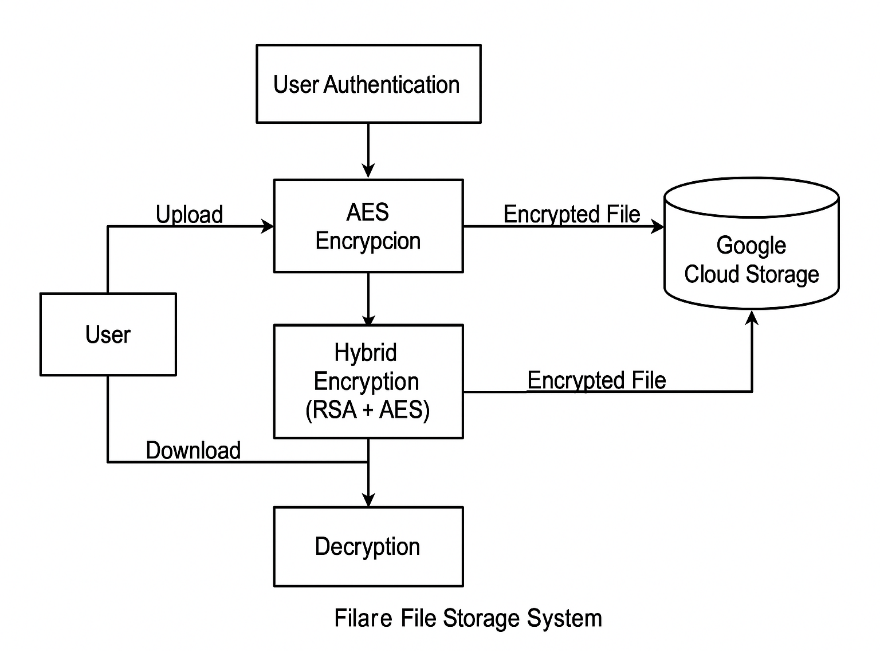
**A. Encryption and Upload Workflow**

1. User logs into the web dashboard.
2. Uploads a file via browser interface.
3. Server generates a unique AES session key.
4. File is encrypted using AES in EAX mode.
5. AES key is encrypted using RSA public key of the user.
6. Nonce, tag, encrypted file, and encrypted AES key are bundled into a single package.
7. Package is uploaded to **Google Cloud Storage (GCS)** using the GCP bucket.
8. File metadata is stored in MySQL for reference.

**B. Download and Decryption Workflow**

1. User selects a file to download.
2. Flask backend fetches the encrypted package from GCS.
3. The user’s RSA private key decrypts the AES key.
4. AES key is then used to decrypt the file.
5. Decrypted file is made available for user download.





**7. CODE STRUCTURE**

* **app.py**:  
  Handles Flask routes (/login, /upload, /download) and session management.  
  Calls utility functions for encryption, decryption, and GCS upload/download.
* **crypto\_utils.py**:  
  encrypt\_file() – Generates AES key, encrypts file, returns encrypted archive.  
  decrypt\_file() – Extracts AES key, decrypts file using RSA and AES.
* **gcs\_utils.py**:  
  upload\_to\_gcs() – Uploads file to cloud using google-cloud-storage.  
  download\_from\_gcs() – Fetches file from GCS and stores it locally for decryption.
* **dashboard.html**:  
  Provides interface for users to upload/download files. Simple and intuitive design using Bootstrap or Tailwind CSS.

**8. SECURITY CONSIDERATIONS**

**Private Key Management:**  
In this prototype, RSA private keys are stored locally on the server. For production, using **Google Cloud KMS** or **hardware security modules (HSMs)** is recommended to avoid key leakage.

**Password Storage:**  
Currently, passwords are stored in plaintext. This must be replaced with **hashed passwords using bcrypt** or Argon2 to prevent password leaks in case of a database compromise.

**Encryption Mode:**  
AES in **EAX mode** is used because it provides both **encryption and authentication**. This prevents not only unauthorized reading but also tampering with file contents.

**Transmission Security:**  
All communications between users and the server should be conducted over **HTTPS** to prevent man-in-the-middle attacks during file uploads and downloads.

**Access Control and Audit Logs:**  
The system logs user activities (uploads, downloads, and failed login attempts). These logs can later be used for audit or anomaly detection.

**9. LIMITATIONS**

* **No Centralized Key Management**:  
  Private keys are manually stored and managed. Implementing GCP's Key Management Service would provide more scalable and secure key handling.
* **Plaintext Passwords**:  
  A major security concern. Should be replaced with salted password hashing mechanisms.
* **No Multi-user Access Control**:  
  The current system does not differentiate between users beyond login. Role-based access control (RBAC) could enhance security.
* **Limited UI**:  
  The web interface is minimal and lacks features such as file versioning, file sharing, and preview options.

**10. CONCLUSION**

The project delivers a functional and secure file storage platform by combining **hybrid cryptographic techniques** with **Google Cloud Storage**. Through AES encryption for content and RSA for key handling, the system ensures strong encryption with practical file sharing. Integration with GCS enables scalable and cost-effective storage, while Flask provides a lightweight, flexible backend.

Though the prototype demonstrates strong core functionality, improvements such as hashed passwords, managed key storage, enhanced UI, and role-based access are needed for enterprise-grade deployment. This project forms a solid foundation for cloud-integrated secure storage and can evolve into a robust SaaS application for privacy-focused file handling.

**11. ACKNOWLEDGMENT**

We extend our heartfelt gratitude to our project mentor, **Prof. Viki Patil**, for her invaluable support, constructive criticism, and technical insights throughout the development of this project. Her guidance helped us better understand encryption systems and cloud integration.

We also thank our faculty and the **Department of Computer Science and Engineering (AI & ML)** at **A. P. Shah Institute of Technology** for facilitating this project and encouraging our exploration into practical cybersecurity.

Special thanks to **Google Cloud** for providing the free-tier resources that enabled seamless testing and deployment of our cloud-integrated file storage application.

**12. REFERENCES**

1. William Stallings, *Cryptography and Network Security*, Pearson Education.
2. Google Cloud Documentation – https://cloud.google.com/storage/docs
3. Bruce Schneier, *Applied Cryptography*, Wiley Publishing.
4. R. Rivest, A. Shamir, and L. Adleman, “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems,” Communications of the ACM.
5. “Hybrid Encryption Techniques in Secure Cloud Storage” – IEEE Cloud Computing Journal, 20