COMP3334 Project Report

Group 10		
Name:	Student ID:	Contribution:
HARYOTO Dominicus Dylan	22099885D	25%
FU Tao	22097327D	25%
NING Weichen	22097838D	25%
BAI Haoran	22097746D	25%

Abstract

With the increasing reliance on cloud-based services, online storage systems have become essential. However, these systems often face significant security threats. This project proposes a secure online storage system designed to address these security challenges through a combination of cryptographic techniques and the access control system structure. The system features client-side AES and RSA encryption, log auditing, and robust protection against SQL injection and metadata leakage. Additionally, the system supports multi-factor authentication and implements rapid password modification using the BitLocker-like security system structure. By implementing this whole system, the project significantly enhances user privacy and system integrity. The project demonstrates a practical and extensible approach to constructing a secure online storage solution suitable for real-world deployment.

I. Introduction

Background

Nowadays, due to the rapid development of network and cloud technology, online storage systems [1] have become widespread and essential. Basically, these systems provide functionalities such as data upload, remote data management, and instant data retrieval. These functions enable users' management of numerous data across multiple devices. Most of the systems implement the functions by deploying a centralized client-server system, which ensures efficient data management, processing, and synchronization. These mature and practical implementation of the online storage system results in its irreplaceability for large data management of both individuals and organizations.

However, the reliance on centralized system structures creates multiple security risks, including unauthorized access, data interception, and compromised user credentials. With the rapid rise of computer security-related cases, the security concerns about online storage systems have become a critical issue. In order to encounter these challenges, this project focuses on developing a secure and robust online storage system.

II. Threat Models

System Overview

The online storage system consists of three components, which are client, server, and network communication, enabling users to upload, store, and retrieve data. The assumptions are: Server does not alter stored data but may try to decrypt it; Attackers can access a legitimate user's computer.

Threat Analysis

Table 1 lists all the assets that require protection from potential threats. The stored data is the files or other user information that would be stored on the server. The encryption key is the crucial information for stored data encryption and decryption. User credentials are assets such as user passwords for authentication. Log records are log information of essential user operations and should only be accessed by the administrator.

In Table 2, which lists all the stakeholders, including those who need protection and also those who may carry out attacks. Legitimate users are those who have passed authentication and probably have their data stored on the server. The administrator can manage the whole system, such as reading the log of user operations. Server is the subject that contains all the user data. However, it is assumed that the server cannot be completely trusted, which means the server will not modify stored data but is curious about the data. Unauthorized users can be attackers, but can also be the users who attempt to log into other users' accounts.

Table 1. Assets List

Assets to Protect
Stored Data
Encryption Keys
User Credentials
Log Records

Table 2. Stakeholder List

Possible Stakeholders
Legitimate Users
Administrator
Server (curious about data)
Unauthorized Users

Based on the previous analysis of the assets and stakeholders, the team combined the information and resulted in an analysis of potential threats, which can be shown in the Table. 3 below. Password brute force and credentials lost in plaintext can directly lead to security problems on the user accounts. Other than unauthorized access by attackers trying to view user plaintext, the metadata exposure can also be an issue. Metadata such as file names and author can also expose sensitive data to the attackers. Besides the attackers who aim at user data and sensitive information, the server is also curious about the stored user data and is willing to view plain text. While information is being transported between the client and the server, attackers may eavesdrop on the transmitted data. At last, attackers may reveal sensitive information from logs, which

record essential user operations.

Table 3. Identified Threats

Authentication and Account Security	
Password Attacks	Attackers may guess weak passwords
Credentials Lost	Attackers may directly access users' devices and steal passwords
File Storage and Access Control	
Unauthorized Access	Unauthorized users try to access and view other users' stored data
Directory Traversal Attack	Accessing sensitive data through filenames
SQL Injection	Code injection technique for retrieving database information
Metadata Exposure	Although files are encrypted, attackers may still infer sensitive information from metadata such as file names
Server Security	
Untrusted Server	Server will not modify user data, but is curious about the data information
Network Security	
Session Hijacking	Attackers eavesdrop on transmitted data
Logging	
Invalid Log Access	Attacker access log for sensitive data

The analysis of assets (Table 1), stakeholders (Table 2), and potential threats (Table 3) highlights the critical security issues the online storage system faces. Security issues eventually aim at user data, benefiting not only unauthorized users but also the untrusted server, and are involved in every aspect of data transportation, processing, and storage. The threats to confidence and the requirement for availability emphasize the need for a robust security system.

III. System Design

System Structure

The overall system structure consists of the client and the server. The users directly communicate with the client, and the client communicates with the server. As seen in Fig. 1, the information and files are stored in the server, securely. The user will directly communicate with the client, which is the user I/O, and if the actions need information or files from the server, the client will extract them from the server. All the information flowing between the client and the server is encrypted. The user information and files are all recorded in tables on the server for better user management and client request handling.

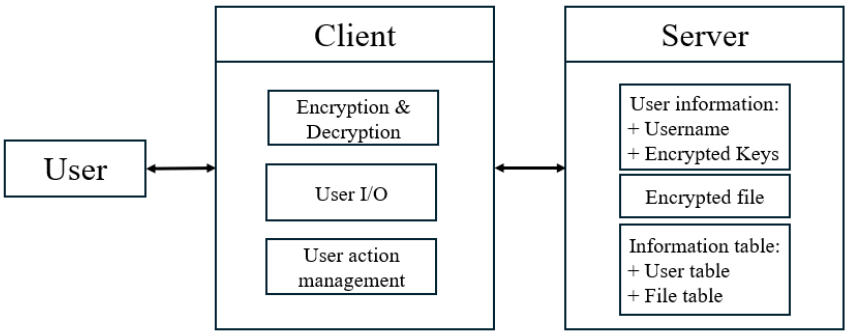

Fig. 1. System Structure Overview

IV. Detailed Design

Function I: User Management

User management is responsible for handling user authentication and user credential management. Balancing user convenience with security principles, the user management design ensures that users' availability while minimizing risks of unauthorized account data access. It includes functions for new user registration, user login with previous credentials, and user password changing.

For user registration, the system requires the users to input their usernames, which are also their email addresses, and their passwords. After successfully recording user information, the system generates the users' recovery keys and private keys and provides them to the user. To avoid conflicts, the system checks whether the chosen username already exists in the database. If the username is found to be unique, the registration process proceeds; otherwise, the user is required to choose a different name. The password provided during registration is not stored in plaintext. Instead, it is hashed using a cryptographically secure algorithm (bcrypt hash) [2]. This hash is then stored in the server database. Hashing ensures that even if the database is compromised, attackers cannot retrieve actual user passwords.

For user login, the user needs to input their previous username and password for authentication. The system retrieves the hashed password associated with the provided username. The input password is hashed using the same algorithm, and the two hashes are compared. If they match, the login is successful. Otherwise, the system denies access.

For user password reset, after choosing to change the password, the system will require recovery key input for authentication and AES key decryption (details about the AES key and password will be introduced in the data encryption part). After recovery verification, the system will allow the user to input a new password and update the password hash in the server database. Then the user will be able to log in with the new password.

Function II: Data Encryption

Data encryption is mainly focused on file upload, file extraction, and file sharing. All files should be plain text when communicated between the client and users. However, all the important information should be encrypted while being transported between the client and server. Also, all the important information saved on the server should be encrypted. For data upload and download, the symmetric encryption method AES-CBC [3] is used. For data sharing, the asymmetric encryption method RSA [4] is implemented.

The process structure can be separated into three different parts. Key generation, AES encryption and decryption, and RSA-based sharing. The key generation part, as shown in Fig. 2, is for every account registration, an AES key and RSA private and public keys are generated. The AES key is encrypted by a password hash (PKDF2) [5] using AES-CBC encryption. The encrypted AES key and the public key are sent to the server.

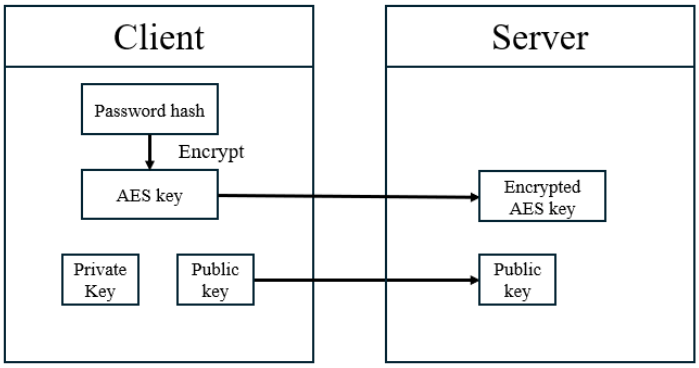

Fig. 2. Key Generation

The AES encryption and decryption part, as shown in Fig. 3, can be described as follows: Client requests the encrypted AES key and decrypts it using the user password; Then the client encrypts the user file with the AES key and sends the file to the server; After downloading the encrypted file, the client again request the encrypted AES key and decrypt the file; At last, the data is sent back to the users.

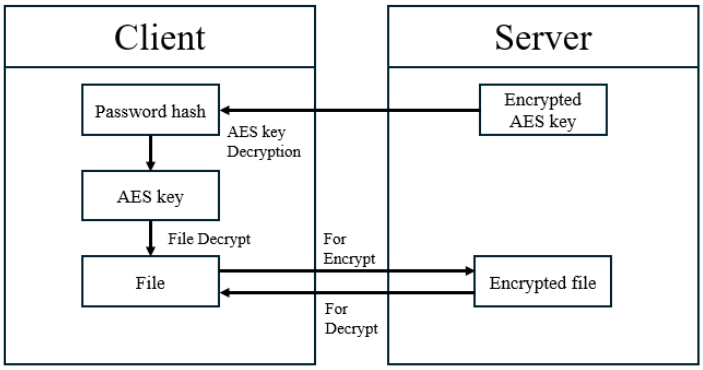

Fig. 3. AES Encryption and Decryption

For the RSA-based sharing, as shown in Fig. 4, it can be described as the following data flow: The share frontend user requests the public key of the backend user; Then the

frontend user generates a random AES key, encrypts the file with the AES key, and encrypts the AES key with the public key; After sending the encrypted file and the encrypted AES key to the backend user, the backend user will decrypt the encrypted AES key with private key, and decrypt the encrypted file with the AES key.

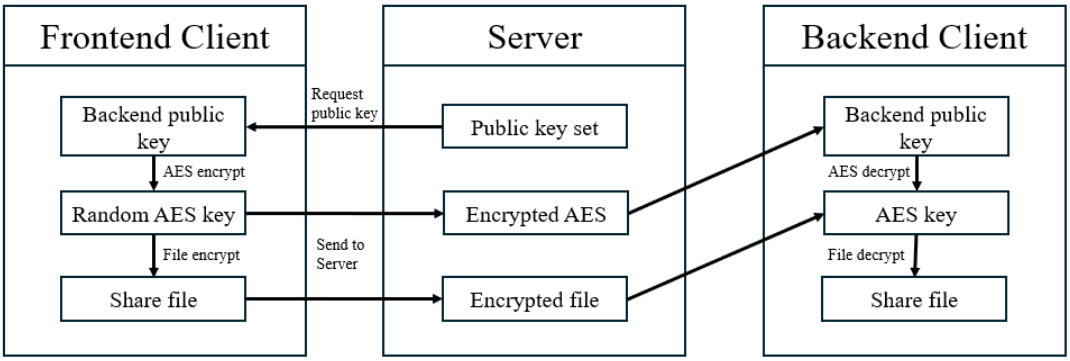

Fig. 4. RSA-based sharing

While combining all these encryption and decryption methods, it can be ensured that the server can only get encrypted data, and the client can always provide plain text to users. Also, all the data transferred between the server and the client is through HTTPS [6] secure transmission. Thus, all the functionality for data encryption is satisfied.

Function III: Access Control

In our design, almost everything is authenticated or accessed using the password. When a user registers, the system will do an authentication using OTP, as shown in Fig. 5, to ensure that both the email for registration exists and the email belongs to the user. This design provides protection in the scenario that the user is unaware of typing a wrong email, and the server sends the recover key to others, who may access the user's account using the recover key received from the server.

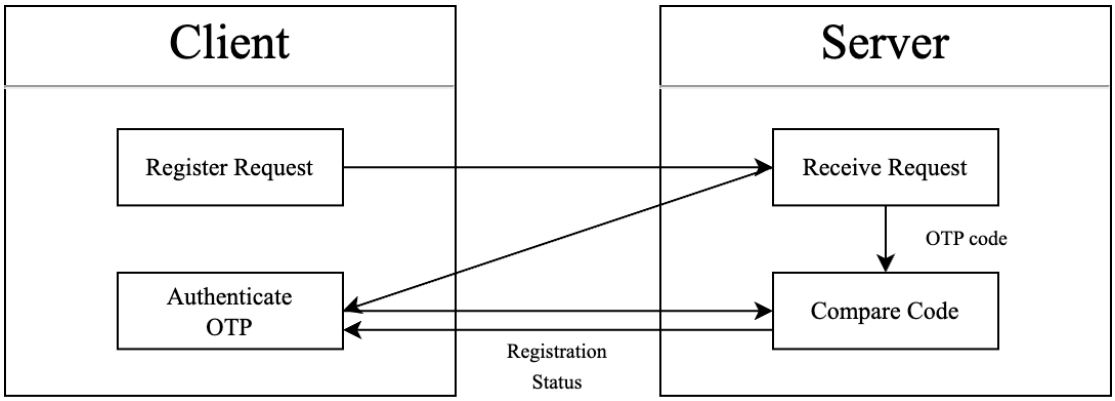

Fig. 5. OTP authentication

For file access, once the user successfully registers, the password hash, encrypted AES key, and public key, which are allowed to be directly read by the server, are sent to the

server. At the same time, a recovery key (another password hash using a different method) and secret key (paired with the public key) are kept by the user. In this case, the files are still protected from both other users and the server.

Owned files can only be decrypted by AES, and AES can only be obtained by decrypting the ‘encrypted AES’ using the recovery key. For shared files, they are encrypted by the ‘pk’ of the user being shared to, therefore, it can only be decrypted by the ‘sk’ of the user. As there’s always one key needed while it’s kept by the user him/herself, the files are well protected and cannot be accessed by others.

When a user resets their password, the encrypted AES is re-encrypted on the client side using the new password, therefore, others and the server will still not have access to the plaintext of AES.

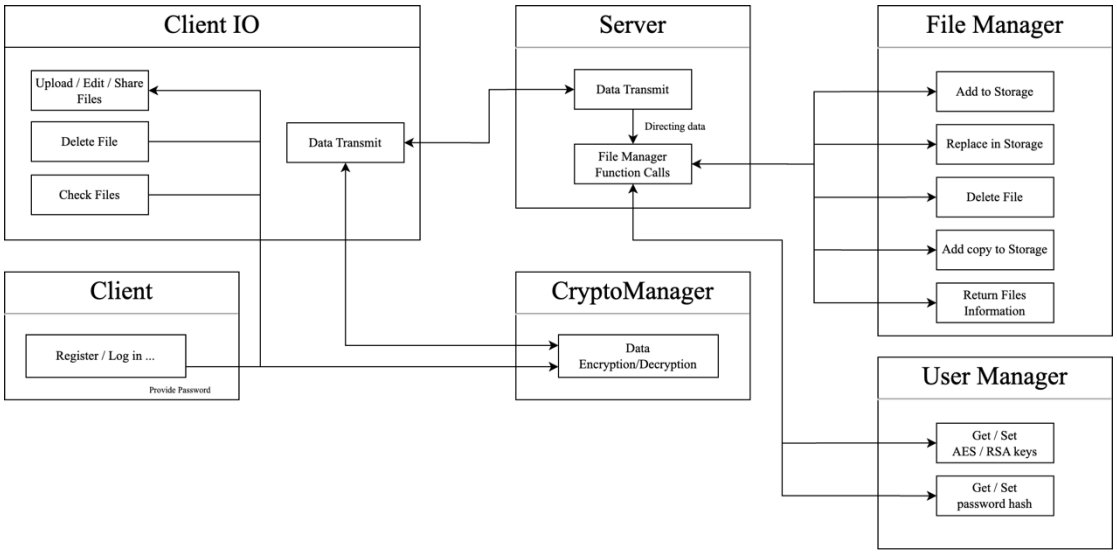

Fig. 6. Access Control Overview

Function IV: Log Auditing

Log auditing mechanism records and tracks critical user operations. This enhances the stability of the system and helps the administrators to track down any abnormal operation.

All critical operations, including logging in, logging out, uploading files, deleting content, and sharing data, are recorded in a log file. Each log entry includes essential metadata such as the username, timestamp, and operation type. To ensure *non-repudiation*, users cannot modify or delete log records. This guarantees that users cannot deny the actions they have performed, as all activities are verifiably recorded and linked to their accounts. However, the system provides an administrative interface that allows the authorized administrator account to access and review the logs. This access enables the administrator to view user activities, investigate incidents, and

ensure system integrity.

Function V: General Security Protection

To ensure the overall security of the system, the metadata should be protected, and the database should be secured. All the file names are stored with the file content in file data to prevent direct extraction using the file name. Thus, directory traversal attacks and metadata attacks are prevented. The system transfers data into bytes and encrypts the bytes before loading and uses parameterized SQL queries [7] to save them to the server database to prevent SQL injection attacks during data upload and download. User input is never directly input into SQL statements. This ensures that even if a user attempts to inject SQL code, the query structure remains unaffected, and the system remains secure.

Extended Function I: Multi-Factor Authentication (MFA)

To enhance account security beyond traditional username-password authentication, the system implements Multi-Factor Authentication (MFA) based on the One-Time Password (OTP) [8] algorithm. Every time before the user logs in, the system will send the user OTP through the user's email (user ID). The user needs to input the data to the client, and the system will verify the OTP for authentication.

Extended Function II: Rapid Password Modification

The system implemented the BitLocker-like encryption architecture [9], which supports rapid password modification. For the traditional password change method, the database will need to encrypt all the previous user files, which is much more complicated and exposes plain text for a longer time. On the contrary, the system uses a password hash (PKDF2) as the recovery key, and also to encrypt the AES key. In this condition, each time for password modification, the system only needs to verify the recovery key for authentication and then encrypt the AES key with the new password hash. No previous files in the server database are required to be modified in this structure.

V. Test Case

Test Case 1

User Registration, File Confidentiality, and Upload Files:
When a user registers accounts, the password and AES are hashed and sent to the server.

```
Enter your email address (or type "q" to EXIT):
> tonyfu812@gmail.com
[STATUS] Email does not exist yet.
Enter a password with at least 8 characters (or type "q" to EXIT, "b" to BACK):
> tonytony
Confirm your password (or type "q" to EXIT, "b" to BACK):
> tonytony
[STATUS] OTP sent to your email. Please check it.
Enter the OTP sent to your email (or type "q" to EXIT, "b" to BACK):
> 645843
[STATUS] Email 'tonyfu812@gmail.com' registered successfully.
```

Fig. 7. Example registration using password ‘tonytony’

Fig. 8. Stored password hash and encrypted AES key

In this case, the plaintext of the password and AES are protected. For AES key, it is decrypted using the password on the client side and used for file encryption and decryption, the process and details for which are unknown to others. The encrypted file can neither be read by others directly nor decrypted, as shown in Fig. 9 and Fig. 10.

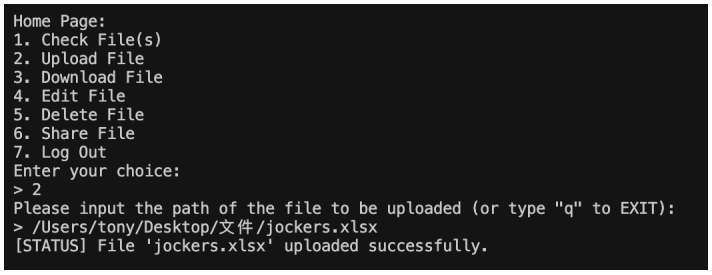

Fig. 9. Excel file being encrypted and sent

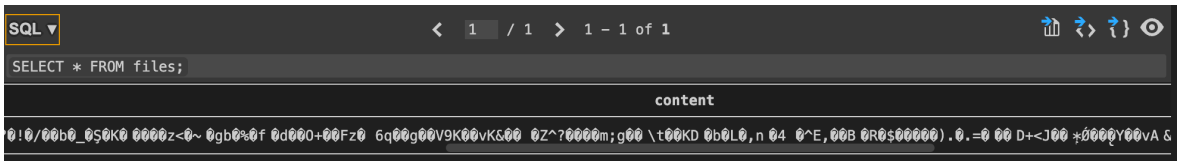

Fig. 10. Unreadable content in server

Test Case 2

Protect our System from SQL Injection:
In our design of database data retrieval, to get information for a user, we query by the user's username and password hash.
To test for avoidance of SQL injection, we can test by passing "1' OR '1'='1" as both username and other parameters. In trivial design, the condition is usually coded as "WHERE username = 'param1', in this case, the resulting SQL sentence will be "… WHERE username = '1' OR '1'='1' which always returns true.
In our design, the inputs are parameterized, as shown in Fig. 11, therefore, the attack mentioned above will be avoided, tested results in Fig. 11.

```
cursor.execute(
    "UPDATE users SET password = ?, encrypted_aes_key = ? WHERE username = ?",
    (new_password, new_aes, username)
)
db_conn.commit()
cursor.execute("SELECT password FROM users WHERE username = ?", (username,))
result = cursor.fetchone() is not None
```

Fig. 11. Parameterization of inputs

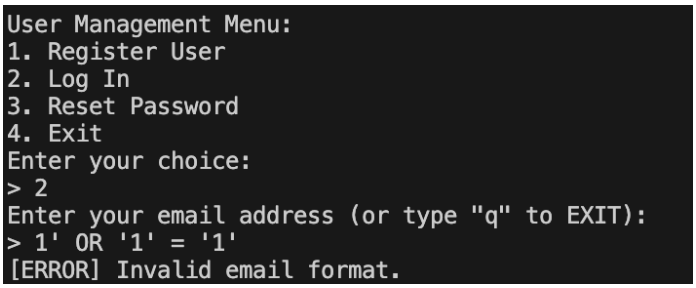

Fig. 12. SQL injection rejected

VI. Future Works

Possible Future Improvement

While the current system effectively addresses the security challenges of online storage, there remain several aspects for future improvements. For the server, the system will develop decentralized storage support to eliminate single points of failure and reduce trust dependencies on centralized servers. Automated audit tools will be developed to help the administrator check system behavior. Cross-device synchronization will be improved to support faster file updates.

Reference

[1] Q. He, Z. Li, and X. Zhang, "Study on Cloud Storage System Based on Distributed Storage Systems," *IEEE Xplore*, Dec. 01, 2010. https://ieeexplore.ieee.org/abstract/document/5709530/ (accessed Nov. 04, 2021).

[2]"Bcrypt Algorithm," www.usenix.org. https://www.usenix.org/legacy/events/usenix99/provos/provos_html/node5.html

[3] S. Heron, "Advanced Encryption Standard (AES)," Network Security, vol. 2009, no. 12, pp. 8–12, Dec. 2009, doi: https://doi.org/10.1016/s1353-4858(10)70006-4.

[4] T. Davis, "RSA Encryption," 2003. Available: http://www.geometer.org/mathcircles/RSA.pdf

[5] Choi, H., & Seo, S. C. (2021). Optimization of PBKDF2 using HMAC-sha2 and HMAC-LSH families in CPU environment. IEEE Access, 9, 40165–40177. https://doi.org/10.1109/access.2021.3065082

[6] Z. Durumeric et al., "The Security Impact of HTTPS Interception," Proceedings 2017 Network and Distributed System Security Symposium, 2017, doi: https://doi.org/10.14722/ndss.2017.23456.

[7] L. K. Shar and H. B. K. Tan, "Defeating SQL Injection," Computer, vol. 46, no. 3, pp. 69–77, Mar. 2013, doi: https://doi.org/10.1109/mc.2012.283.

[8] H. Choi, H. Kwon, and J. Hur, "A secure OTP algorithm using a smartphone application," IEEE Xplore, Jul. 01, 2015. https://ieeexplore.ieee.org/abstract/document/7182589?casa_token=r6esUD7z7_YAAAAA:srQrGY1hOXi0z2dKAbJQpjzOhmhIQPdHaCyf56r_aVukE1hSpXPTS8RnsjM0ACb2Bza7X6p0gQ (accessed May 09, 2022).

[9] S. G. Lewis and T. Palumbo, "BitLocker Full-Disk Encryption," Proceedings of the 2018 ACM SIGUCCS Annual Conference, Sep. 2018, doi: https://doi.org/10.1145/3235715.3241363.