Secure File Sharing System

Syed Mujtaba Haider
BS(Data Science)
FAST NU
21L-5613

Kamran Ishtiaq
BS(Data Science)
FAST NU
21L-6253

Khurram Imran
BS(Data Science)
FAST NU
21L-6256

***Abstract*—This paper presents a secure web-based file sharing system that uses client-side AES encryption to protect data confidentiality during transmission and storage. Using Flask and Python, the system ensures that uncompressed data remains inaccessible to the server. Users can easily upload and download files through an intuitive web interface, where the data is stored and extracted only from client machines. The approach addresses today's challenges in data security and demonstrates the feasibility of providing client-side encryption for privacy-focused applications.**

***Index Terms*—AES, Flask, Secure File Sharing, Cryptography, Client-Side Encryption**

I. Introduction

In today's digital world, rapidly advancing technology has increased the need for secure exchange and storage of information. Individuals and organizations increasingly rely on digital communication, where data is shared, stored and accessed in constant formats. However, these developments have created new security challenges, as sensitive data is at high risk of unauthorized access, interception and theft. To address these challenges, encryption—turning readable data into unreadable by ensuring privacy—has become a cornerstone of data security Efficient encryption algorithms provide a means to encrypt data protect its integrity and confidentiality, especially in the face of malicious attempts to obtain or manipulate information.

Advanced Encryption Standard (AES) has emerged as one of the most reliable encryption methods, widely accepted in the industry due to its strength and versatility AES, a symmetric block cipher designed as will replace the older Data Encryption Standard (DES), the Rijndael algorithm for encrypting and encrypting data in 128-bit chunks using a key-size of 128, 192, or 256 bits AES structure and design for highly secure and resistant to most known attacks, so it can meet the stringent security requirements of modern networks Unlike DES , which due to its short key length for brute-force attacks is vulnerable , and unlike Triple DES (3DES), which is slow and inefficient, AES offers both speed and security, making it suitable for a wide range of applications, from personal smartphones to large enterprise systems.

The motivation for this research was to investigate the use of AES in a secure, web-based file sharing system, ensuring that files remain encrypted during both transmission and storage by encrypting files on the client side before it is transmitted to the server, this access is due to unauthorized parties including servers The only thing that prevents access to unencrypted data is the intended recipient, whose privilege a Decryption key exists, he can retrieve and decrypt the files at their end. This client-side encryption model not only enhances data security but also solves privacy concerns, as sensitive resources are never encountered in central systems.

The following paper depicts the design and implementation of the file sharing system using AES encryption in Flask—a web based application. The system encrypts files locally on the user's machine before transmission, ensuring that only encrypted content is stored on the servers. The receiver can then safely download the file to their device. The solution demonstrates the feasibility and practicality of secure file sharing with client-side encryption and provides the foundation for further development of privacy-focused applications.

II. Related Work

The need for data security increases with the advancement of digital communication, due to which encryption has become an important means of protecting information in data transmission and storage. There are several studies that have examined its strengths and weaknesses into the various encryption algorithms available with a special focus on the Advanced Encryption Standard (AES).

In some recent years, secure digital communication transformation has required reliable cryptographic algorithms that can handle rapidly evolving information technologies and research in this area have shown that AES, a symmetric block cipher algorithms, is very effective in data security. AES was a replacement for Data Encryption Standard (DES) and Triple DES (3DES), which were found not good for present security due to the small DES keys and slower performance. AES is offering more safety profiles with higher processing speeds and flexibility across different platforms.

AES can handle three different key sizes i.e. 128, 192, and 256 bits, each contains a 128 bit block size. AES provides different levels of security depends on the needs of the application. The system is built to make it difficult for attackers to decrypt the encrypted data. It is a fact that no AES instances have been compromised till date. These things have made AES one of the most trusted encryption algorithm in the world.

There are several people who have encouraged the use of AES in files and its advantages over alternatives such as DES, 3DES, and Blowfish. Unlike DES, which suffers from a few key length weaknesses, and 3DES, which is comparatively slow, AES offers both high security and efficiency, making it suitable for high throughput environments. AES ability to handle a variety of files-text, documents, PDF, and image files - makes it is versatile for various types of secure data. Recent implementations, such as file sharing applications and smartphone encryption apps, have used AES to encrypt files before they are sent over the network, ensuring data privacy even in transit.

Given its strengths, AES has become the de facto standard of modern cryptography, supporting the secure transfer, storage, and integrity of sensitive information. All of this work collectively builds on the use of AES size and adequacy emphasis, especially in situations where data security is paramount and computational efficiency is required. They provide a solid foundation for further research and development of AES-based encryption systems.

III. System Architecture and Design

The proposed file-sharing scheme is designed to ensure that files remain encrypted throughout their lifetime on the server, providing end-to-end security by encrypting files on the client side before delivering them to the server on the This option means that the server cannot transmit unencrypted messages.

A. System Overview

The architecture has three main components: client-side encryption, server-side storage, and client-side decryption. They are as follows:

1) Client-Side Encryption: The user picks the file on his system and uploads it, then the system uses Advanced Encryption Standard (AES) algorithm to encrypt it before uploading it to the server. This will ensure that the user's file is safe from unauthorized.

2) Server-Side Storage: The file that is encrypted, with its metadata, is stored on the server. The server only keeps encrypted data, which means that even if the attacker accesses the server, he will only have access to encrypted files.

3) Client-Side Decryption: When the user downloads the file, the unique decryption key is used to restore it to its original format on their system. Only the authorized users with a valid decryption key can access the original file.

B. Technology Stack

This project uses many technologies to create secure file sharing process. Each component supports a specific functionality for the application, ensuring security, performance, and ease of use.

1) Python: It is used as the application's main backend language, because it supports libraries for cryptographic operations and database management.

2) Flask: It is a small web-based framework, used to process HTTP requests. It simplifies the process of developing web applications and simplifies the integration of encryption and database components.

3) Cryptography Library: It is implemented with AES encryption, which provides secure algorithms for key generation, encryption, and decryption. Advanced Encryption Standard (AES) is chosen for its security and high performance, which makes it suitable for encrypting files.

4) SQLite: This database is used to store data and metadata of file. It is a small database, used for these types of web applications because it provides reliable storage with minimal configurations, which makes it suitable for smaller applications.

5) HTML/CSS: The interface of this system is designed with HTML and CSS, which provides users with an attractive interface for secure file uploading and downloading. The HTML is used for page elements, and CSS is used to enhance the visual appearance of the interface.

After all these technologies are combined, the system ensures that files are handled safely and efficiently. It also focuses on preserving user's privacy and data throughout this process.

C. Design

Following is the design that separates the role of the server and client. The user uploads the file which gets encrypted and then stores in the server. When user clicks on download then it decrypts it and downloads it on the user's machine. The diagram is as follows:

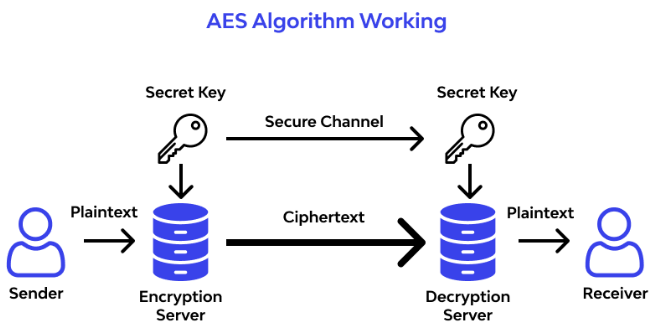


Fig. 1. Secure File-Sharing System.

IV. Encryption and Decryption Process

A. AES Encryption

1) AES for Security and Performance: We chose Advanced Encryption Standard (AES), as it has strong encryption capabilities, and wide acceptance throughout the world as a secure encryption method. It is a symmetric encryption algorithm, which means that it uses the same key for encryption and decryption. Mostly, it uses 256-bit encryption key which provides a higher level of security and a larger key space which is not feasible for attackers to use brute force technique. This makes it well-suited to protect data, such as larger files, from unauthorized access.

2) Client-Side Encryption Process: Advanced Encryption Standard (AES) encryption technique takes place entirely on the user's device, which ensures that the plaintext never leaves the user's device. When the user selects a file to upload, the system then generates a random AES key specific to that session or file. Using this key, the file is encrypted that can only be decrypted by the user with the decryption key. In this encryption technique, the file is divided into 128-bit chunks, and each chunk is filtered, modified, and manipulated using the Rijndael algorithm. After the encryption is complete, the ciphertext is then uploaded to the server, and the AES key stays safe in the user's device.

B. File Handling Workflow

1) Process of Uploading File: When the user uploads a file, the following actions takes place to secure the file:

Encryption: File is encrypted with a 256-bit key using AES.

Upload: Encrypted file is then uploaded to the server, with some metadata of file.

Server Storage: Server then receives only the encrypted file and metadata of file and stores it, without knowing the encryption key. This will ensure that the files remain unreadable.

2) Downloading Process of File: The receiver performs the following actions to download the file and view:

Download: The user clicks on the download button which sends the request to the server, then file is decrypted and downloaded on the user's system.

Decryption: The receiver requests to download the file then it decrypts the file with the AES key, and it is decrypted and downloaded.

Access: File is downloaded, the user can open it.

This maintains the confidentiality of the file, because the encryption and decryption are done on the user's side.

V. Implementation

This file sharing system has three main components: user authentication and session management, AES encryption and decryption of files, web interface for file upload and download. Each component is designed to ensure that files will remain encrypted when stored and transmitted.

A. User Authentication and Session Management

1) Authentication: Flask is used to handle user authentication. It has a some libraries for login and registration system that verifies the user by hashing the password through secure password storage and session management.

Password hashing: The werkzeug library is used to hash user passwords before storing them in the database, ensuring plaintext passwords are never shown.

2) Session Management: The Flask session object maintains secure sessions, and each authenticated session associates a unique session token to identify the user without revealing their credentials.

B. AES Encryption and Decryption Logic

AES encryption and decryption logic is implemented on the client side using the Python cryptography library. This ensures that the files are encrypted before they are uploaded to the server and decrypted when downloaded. The following use case describes how to process files securely.

1) Key generation: Generate a 256-bit AES key for each file or session using a cryptographically secure random function.

2) File encryption: File fragments are read and encrypted before transmission. The AES encryption mode used is CBC (Cipher Block Chaining) with an initialization vector (IV) for additional security.

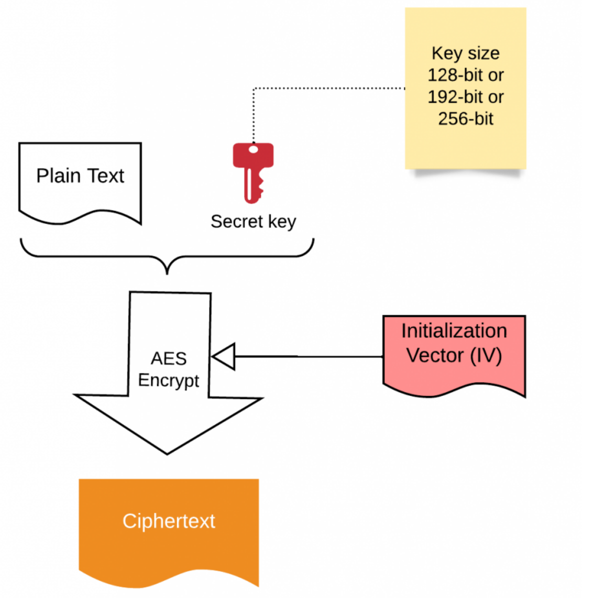
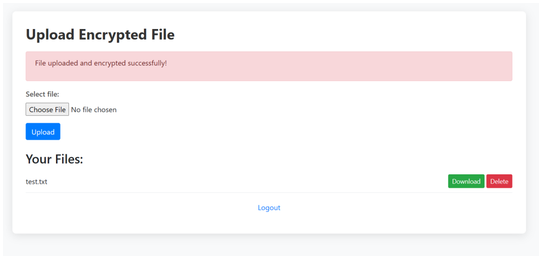


Fig. 2. Encryption and Decryption process using AES. The data is encrypted and decrypted in ciphertext using the same key.

C. Web Interface

The web interface was created using HTML and CSS, while backend methods in Flask handle file uploads and downloads.

1) Upload Route: Users upload a file through an HTML form, and Flask background calls the function before saving the file.


Upload Encrypted File

File uploaded and encrypted successfully!

Select file:

Your Files:

test.txt

Fig. 3. Upload File

2) Download Route: When a user downloads a file, the backend retrieves the encrypted file and sends it to the user. The user then decrypt the file using the key saved in the user's system.

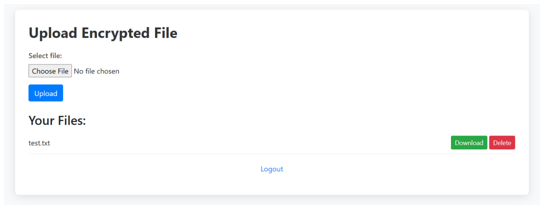


Fig. 4. Download File

VI. Security Analysis

The security of this file sharing system is maintained through many different mechanisms that ensures to protect data from unauthorized access and modifications.

A. Protection against unauthorized access

All the encryption techniques are important in protecting sensitive files from unauthorized access. All the data on the user's device is encrypted before sending it to the server, which ensures that only user with a valid decryption key can download the file. This also prevents the server from viewing the user's data, which adds privacy and security for the users.

B. Preventing data misuse and interference

This system transfers data between user and the server using SSL/TLS encryption. This also stops third parties from interfering during upload and download process. This ensures that the integrity of the encrypted data is safe, because this encryption technique makes it difficult for the attacker to make any changes in the file.

C. Mitigation of Potential Security Challenges

When the client-side encryption is strong, security depends on the responsibility of user keys. Users are encouraged to store keys securely to reduce the risk of key loss or exposure, user should use multifactor authentication for additional security. Also, periodic and internal security checks and updating cryptographic standards ensure that the system remains resilient to changing threats.

VII. Results and Testing

We tested the performance and security of this system by using different testing methods and results were quite impressive.

A. Test Environment

We uploaded different files of different types and sizes. It was done on our systems locally. This Advanced Encryption Standard (AES) algorithm is the only algorithm used in our project for uploading and downloading the files. This also ensures that the encryption and decryption process remain same and no unauthorized person can gets access.

B. Description

The basic focus of encryption and decryption during file uploading and downloading was perfromance. By uploading different files of different sizes, we saw that files smaller than 50 MB took minimal time to process. However, larger files showed slightly increase in processing time. System was stable during during simultaneous file transfers, which tells that the system can handle load efficiently.

C. Security Verification

During these tests while uploading and downloading we saw that the server was not able to see the original contents of the files. These tests confirm that only archived files are stored on the server and no plaintext can be retrieved from the server. If attacker tries any attempt to delete files then there is no way to do it without knowing the key.

VIII. Conclusion

This project provides a secure file sharing system which ensures the secrecy of the files of the user. Using the Advanced Encryption Standard (AES) technique, the files are encrypted and decrypted on the user's side which means that the key always remains in the user's system. This system stops unauthorized access to information even from the server, as there is no key stored in it.

This design helps the user to securely upload the file and share it. This can also be expanded by allowing very large files like heavy games of 200 GB or 300 GB, by adding larger databases to the system. We can add more complexity and integrate additional security to secure user files which meet user needs.

There are significant implications for secure data sharing, especially where privacy is critical. This design can be expanded to support larger databases, add more complex authentication mechanisms, and integrate with additional security features to meet user needs.

References

[1] Muttaqin, Khairul, and Jefril Rahmadoni. "Analysis and design of file security system AES (advanced encryption standard) cryptography based." Journal of Applied Engineering and Technological Science (JAETS) 1.2 (2020): 113-123.

[2] Tayde, Suchita, and Seema Siledar. "File encryption, decryption using AES algorithm in android phone." International Journel of Advanced Research in computer science and software engineering 5.5 (2015).

[3] Abdullah, Ako Muhamad. "Advanced encryption standard (AES) algorithm to encrypt and decrypt data." Cryptography and Network Security 16.1 (2017): 11.