JIK: Jurnal Ilmu Komputer
Vol. xx, No. x, Bulan Tahun, pp. xx~xx p-ISSN: 2527-9653
DOI: 10.47007/komp.v7i01.xxxxx e-ISSN: 2797-8389

PERANCANGAN APLIKASI ENKRIPSI DAN DEKRIPSI FILE MENGGUNAKAN ALGORITMA KRIPTOGRAFI AES-256 BERBASIS STREAMLIT

Arditya Adjie Rosandi[1], Dhicky Hariyadi Supriyono[2], Teguh Tegar Aulia[3], Jefry Sunupurwa Asri[4]

[1,2,3,4,] Teknik Informatika, Fakultas Ilmu Komputer, Universitas Esa Unggul
Email: [1]ardityaadjierosandi18@student.esaunggul.ac.id , [2]dhickyharyadi5462@student.esaunggul.ac.id, [3]tegart39@student.esaunggul.ac.id, [4]jefry.sunupurwa@esaunggul.ac.id

Abstrak

Akses yang semakin luas terhadap perangkat komputasi pribadi membuat banyak dokumen penting disimpan dalam bentuk file digital tanpa perlindungan kriptografi yang memadai, sehingga menimbulkan risiko kebocoran dan penyalahgunaan data. Advanced Encryption Standard (AES) dengan panjang kunci 256 bit merupakan algoritma kriptografi kunci simetris yang direkomendasikan untuk menjaga kerahasiaan data karena kekuatan keamanan dan performanya. Penelitian ini merancang dan mengimplementasikan aplikasi web sederhana untuk enkripsi dan dekripsi file menggunakan algoritma AES-256 berbasis password dengan framework Streamlit. Proses pembentukan kunci dilakukan menggunakan PBKDF2-HMAC-SHA256 dengan penambahan salt dan iterasi tertentu, sedangkan enkripsi data menggunakan mode AES-GCM yang menyediakan kerahasiaan sekaligus autentikasi integritas. Aplikasi memungkinkan pengguna mengunggah file, mengenkripsi menjadi berkas berekstensi .enc, serta mengembalikan file ke bentuk semula melalui antarmuka web tanpa perlu menulis perintah di terminal. Pengujian dilakukan menggunakan beberapa file uji dengan ukuran berbeda untuk mengamati keberhasilan enkripsidekripsi dan estimasi waktu proses, serta uji negatif dengan password yang salah untuk memastikan data tidak dapat dikembalikan tanpa kunci yang benar. Hasil menunjukkan bahwa seluruh file uji dapat dienkripsi dan didekripsi dengan benar, file terenkripsi tidak dapat dibaca secara langsung, dan overhead waktu enkripsidekripsi masih berada pada rentang yang dapat diterima untuk penggunaan sehari-hari, sehingga prototipe aplikasi dinilai layak sebagai solusi sederhana untuk meningkatkan keamanan penyimpanan file lokal.

Kata kunci: *kriptografi, AES-256, enkripsi file, Streamlit, keamanan data, PBKDF2*

DESIGN OF FILE ENCRYPTION AND DECRYPTION APPLICATION USING AES-256 CRYPTOGRAPHY ALGORITHM BASED ON STREAMLIT

Abstract

The widespread use of personal computing devices has led to an increasing amount of sensitive documents being stored as digital files without adequate cryptographic protection, which raises the risk of data leakage and misuse. The Advanced Encryption Standard (AES) with a 256-bit key length is a symmetric-key cryptographic algorithm recommended for protecting data confidentiality due to its strong security and good performance. This study designs and implements a simple web-based application for file encryption and decryption using the AES-256 algorithm with a password-based key, built on the Streamlit framework. Key generation is performed using PBKDF2-HMAC-SHA256 with a random salt and a sufficient number of iterations, while data encryption uses the AES-GCM mode, which provides both confidentiality and integrity authentication. The application allows users to upload files, encrypt them into .enc ciphertext files, and restore them to their original form through a web interface without interacting directly with the command line. Experiments are conducted using several test files of different sizes to observe the success of the encryptiondecryption process and estimate processing time, as well as negative tests with incorrect passwords to ensure that data cannot be recovered without the correct key. The results show that all test files can be correctly encrypted and decrypted, the encrypted files cannot be read directly, and the encryptiondecryption overhead remains acceptable for everyday use, indicating that the prototype application is suitable as a simple solution to enhance local file storage security.

Keywords: *cryptography, AES-256, file encryption, Streamlit, data security, PBKDF2*

1. *INTRODUCTION*

Advances in information technology have led to an increase in the number of important documents, such as academic assignments, work reports, and personal data, being stored in digital file form on personal computers. Without adequate protection, these files are vulnerable to being copied, altered, or accessed by unauthorized parties if the device is lost, stolen, or attacked by malware. Therefore, file security mechanisms that are easy to use but still cryptographically strong are urgently needed by general users. [1]

The Advanced Encryption Standard (AES) is a symmetric-key cryptography algorithm established as a standard by NIST to replace DES and is widely used to secure data on various platforms. The 256-bit key length variant of AES (AES-256) offers a very large key space that is practically resistant to brute-force attacks, while also providing sufficient performance for everyday file encryption applications. A number of studies have applied AES to file security systems and shown that this algorithm is capable of maintaining data confidentiality with acceptable time overhead.

Some existing file encryption implementations are still command line-based, making them less user-friendly for non-technical users. Streamlit is a Python framework that simplifies the development of interactive web applications with simple syntax and supports direct file upload and download components via the browser. The combination of AES-256 with the Streamlit interface has the potential to produce a file encryption application that is both secure and easy to use without complex installation on the user's side.

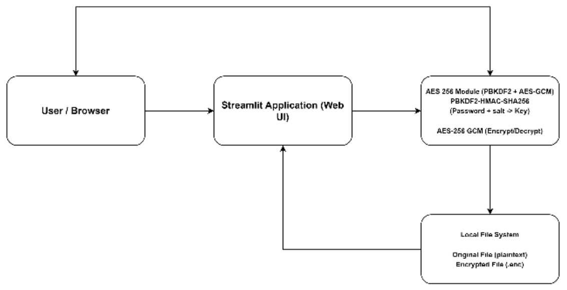
This study aims to design and implement a simple web application for file encryption and decryption using a password-based AES-256 algorithm with a Streamlit interface. In addition, testing was conducted on the success of the encryption and decryption process on several test files of different sizes and a simple analysis of the processing time to assess the feasibility of the application for daily use.

2. *RESEARCH METHOD*

This research uses a simple software engineering approach in the form of designing and implementing a web-based file encryption application prototype without large-scale system development. The cryptographic algorithm used is AES-256 in Galois/Counter Mode (GCM), which provides both confidentiality and data integrity authentication in a single process. The AES-256 key is obtained from the password entered by the user using the PBKDF2-HMAC-SHA256 key derivation function with the addition of salt and a sufficiently high number of iterations to increase resistance to brute-force attacks.

The application was built using the Python programming language with the Streamlit framework as the web interface, allowing users to access encryption and decryption functions through a local browser without interacting directly with the command line. The prototype runs on a single personal computer that acts as both a server and a client, where uploaded and downloaded files are located on the user's local file system.

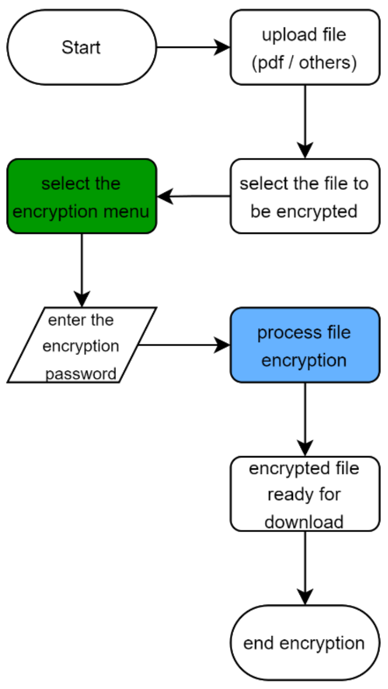
Figure 1. Architecture of a Streamlit-based file encryption and decryption application using AES-256.

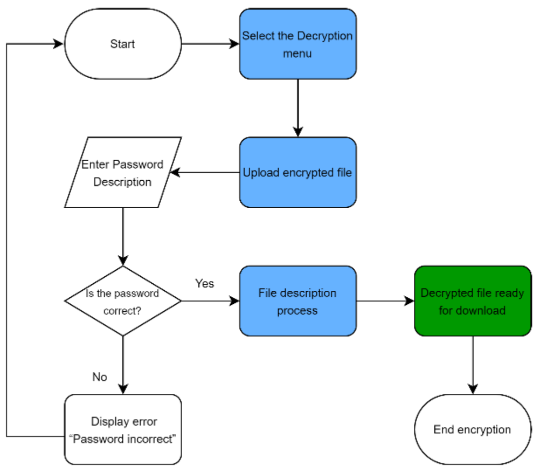

User / Browser
Streamlit Application (Web UI)
AES 256 Module (PBKDF2 + AES-GCM)
PBKDF2-HMAC-SHA256
(Password + salt -> Key)
AES-256 GCM (Encrypt/Decrypt)
Local File System
Original File (plaintext)
Encrypted File (.enc)

The application provides two main operating modes, namely Encrypt and Decrypt, which are selected via radio buttons on the main page. In Encrypt mode, users upload a file and enter a password; the system then generates a random salt, derives an AES-256 key from the password and salt using PBKDF2, creates a nonce for AES-GCM, and then encrypts the file contents into ciphertext. The salt, nonce, and ciphertext are combined and sent back to the user as an .enc file via a download button.

In Decrypt mode, users upload the .enc file and enter the password they believe to be the same as the one used previously. The system separates the salt and nonce from the encrypted data, recalculates the AES-256 key from the password and salt, then runs AES-GCM decryption to recover the original file contents. If the internal AES-GCM authentication process fails due to an incorrect password or corrupted data, the system displays an error message to the user without generating the plaintext file. [2]

Figure 2. Flowchart of the encryption and decryption process.

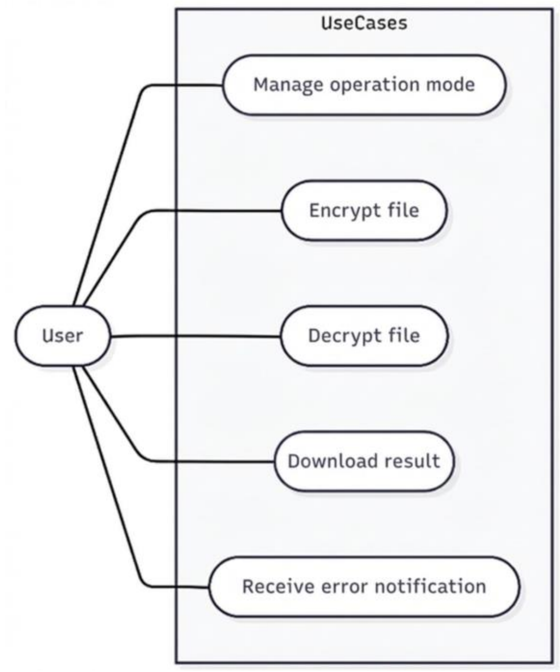

Start
upload file (pdf / others)
select the file to be encrypted
select the encryption menu
enter the encryption password
process file encryption
encrypted file ready for download
end encryption


Start
Select the Decryption menu
Upload encrypted file
Enter Password Description
Is the password correct?
Yes
No
File description process
Decrypted file ready for download
Display error "Password incorrect"
End encryption

In the encryption business process, users access the application through a browser, select Encrypt mode, then upload the files they want to protect and enter the password that will be used as the basis for generating the key. The system then randomly generates a salt and nonce, generates an AES-256 key using PBKDF2, encrypts the file contents with AES-GCM, wraps the result in the order [salt][nonce][ciphertext], and finally displays a Download button so that users can download the .enc file. Meanwhile, in the decryption business process, the user selects Decrypt mode, uploads the .enc file, and enters the password. The system then extracts the salt and nonce, reforms the AES-256 key with PBKDF2, and runs AES-GCM decryption. If the authentication process is successful, the system generates a plaintext file and provides a download button. If authentication fails, the system only displays an error message without ever releasing the original file.[3]

In the decryption business process, users first select Decrypt mode, then upload the .enc file and enter what they believe to be the correct password. The system then extracts the salt and nonce values from the file, derives the AES-256 key using PBKDF2, and performs the decryption process with AES-GCM. If authentication is successful, the system generates a plaintext file and displays a download button, whereas if authentication fails, the system only displays an error message without generating a plaintext file.

Figure 3. Use Case Diagram



The testing was conducted experimentally by running the application on a single personal computer and using several test files of varying sizes. The test files consisted of small text files, PDF documents around 1 MB, and media files several megabytes in size to represent common usage scenarios. Each file was encrypted and decrypted through the Streamlit interface, while the processing time was measured manually using a stopwatch from the moment the process button was pressed until the download button appeared.

In addition, negative testing was performed by attempting to decrypt encrypted files using incorrect passwords to ensure that the application did not return readable data when the key did not match. This test assessed whether the AES-GCM authentication mechanism and key derivation scheme had been implemented correctly.

ID scenario	**File type**	**Estimated size**	**Test objectives**	**ID scenario**
S1	Text (.txt)	< 100 KB	Testing the basic success of small file encryption and decryption	S1
S2	PDF document	± 1 MB	Testing performance on commonly used working documents	S2
S3	Image / video	5–20 MB	Testing the effect of large file sizes on processing time	S3

3. *RESULT AND ANALYSIS*

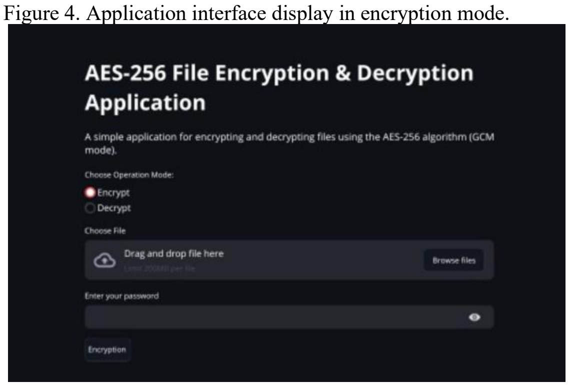
The testing was conducted according to the scenario in Table 1 with three test files of small, medium, and large sizes. Each file was encrypted and decrypted once, then the file size and encryption and decryption processing time were recorded to provide an overview of the effect of file size on application performance. Verification was carried out by opening the decrypted file and ensuring its conformity with the original file.

File ID	File type	Size (MB)	Encryption time (seconds)	Decryption time (seconds)
F1	Text (.txt)	0,10	0,3	0,2
F2	PDF document	1,25	0,8	0,7
F3	Image (.jpg)	5,40	1,9	1,7
F4	Short video (.mp4)	18,70	5,4	5,0

Based on Table 2, encryption and decryption times increase as file size increases, but the entire process still takes only a few seconds, making it suitable for everyday use with small to medium-sized files. No encryption or decryption failures were found in tests with the correct password, where all decrypted files could be opened and used again as the original files.

The Streamlit application interface displays mode options, file upload components, password input fields, and process buttons so that users can perform encryption and decryption without having to write commands in the terminal. Once the process is complete, the application displays a success message and a Download results button that allows users to download ciphertext files and decrypted files via their browser.

Figure 4. Application interface display in encryption mode.



AES-256 File Encryption & Decryption Application

A simple application for encrypting and decrypting files using the AES-256 algorithm (GCM mode).

Choose Operation Mode:

[✓ Encrypt]
Decrypt

Choose File

Drag and drop file here

4. *CONCLUSION*

This research successfully designed and implemented a Streamlit-based file encryption and decryption application that utilizes the AES-256 cryptographic algorithm with PBKDF2-HMAC-SHA256 key derivation. The application allows users to upload files, encrypt them into .enc files, and restore them to their original form using the same password through a simple web interface. Test results show that the encryption-decryption process runs correctly on several file types and sizes, with acceptable time overhead for individual user needs. In addition, decryption attempts with incorrect passwords did not produce plaintext, so it can be concluded that the implemented security mechanism is capable of maintaining file confidentiality as long as the password is not known to other parties.

In the future, the application can be developed with the addition of more secure password management, separate key configuration storage, and integration with cloud storage services to support backup and encrypted file sharing scenarios.[4]

5. *REFERENCES*

[1] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 7th ed. Boston, MA, USA: Pearson, 2017.

[2] N. Ferguson, B. Schneier, and T. Kohno, *Cryptography Engineering: Design Principles and Practical Applications*. Indianapolis, IN, USA: Wiley Publishing, 2010.

[3] National Institute of Standards and Technology, *Recommendation for Password-Based Key Derivation*, NIST SP 800-132, 2010; dan
National Institute of Standards and Technology, *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM)*, NIST SP 800-38D, 2007.

[4] M. Bishop, *Computer Security: Art and Science*, 2nd ed. Boston, MA, USA: Addison-Wesley, 2018.