p-ISSN 2827-8585
e-ISSN 2827-8062

International Journal of Multidisciplinary Research and Literature
IJOMRAL
Vol. 5, No. 1, January 2026 pp. 162-168
Journal Page is available at http://ijomral.esc-id.org/index.php/home

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

Arditya Adjie Rosandi[1*], Dhicky Hariyadi Supriyono[2], Teguh Tegar Aulia[3], Jefry Sunupurwa Asri[4]
[1,2,3,4]Informatics Engineering, Faculty of Computer Science, Universitas Esa Unggul, Indonesia
Email: *ardityaadjierosandi18@student.esaunggul.ac.id[1]*, *dhickyharyadi5462@student.esaunggul.ac.id[2]*, *tegart39@student.esaunggul.ac.id[3]*, *jefry.sunupurwa@esaunggul.ac.id[4]*

Abstract
The widespread use of personal computing devices has led to an increasing number of sensitive documents, including digital files, lacking adequate cryptographic protection, raising 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 excellent 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 uses PBKDF2-HMAC-SHA256 with a random salt and a sufficient number of iterations, while data encryption uses AES-GCM, which provides confidentiality and integrity authentication. The application enables users to upload files, encrypt them into .enc ciphertext files, and restore them to their original form via a web interface, eliminating the need for direct command-line interaction. Experiments using several test files of varying sizes to assess the success of the encryption/decryption processes and estimate processing times. Furthermore, to ensure that data could not be accessed without the correct key, negative tests were conducted using erroneous passwords. All test files can be successfully encrypted and decrypted; encrypted files cannot be read directly; and the overhead of encryption and decryption is manageable for everyday use. The prototype app is a suitable solution to enhance the security of local file storage.
Keywords: cryptography, AES-256, file encryption, Streamlit, data security, PBKDF2

INTRODUCTION

Advances in information technology have led to an increase in the number of essential documents, such as academic assignments, work reports, and personal data, being stored digitally on personal computers. Without adequate protection, these files are vulnerable to unauthorized copying, alteration, or access if the device is lost, stolen, or infected with malware. Therefore, file security mechanisms that are easy to use but still cryptographically strong are urgently needed by general users (Stallings, 2017).

The Advanced Encryption Standard (AES) is a symmetric-key cryptographic algorithm established by NIST to replace DES and is widely used to secure data across various platforms. The 256-bit key length variant of AES (AES-256) offers a huge key space that is practically resistant to brute-force attacks, while also providing sufficient performance for everyday file encryption applications. Several studies have applied AES to file security systems, demonstrating that this algorithm can maintain data confidentiality with an 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, with minimal installation required 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. Additionally, testing to evaluate the success of the encryption and decryption processes on various test files of different sizes, and to perform a simple analysis of processing time to assess the application's feasibility for daily use.

LITERATURE REVIEW

Advanced Encryption Standard (AES-256)

Advanced Encryption Standard (AES) is a fundamental symmetric block cipher widely used in modern cryptography to secure digital information. AES performs encryption and decryption using a secret key, converting plaintext into ciphertext and vice versa. The algorithm supports multiple key lengths, with AES-256 using a 256-bit key that offers enhanced security compared to shorter key sizes. In the context of secure file storage and communication, AES-256 is particularly valued for its resistance against brute-force attacks and its suitability for protecting sensitive data (National Institute of Standards and Technology, 2023).

As noted, “the AES algorithm is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits” (National Institute of Standards and Technology, 2023), emphasizing the modular design and flexibility of AES. Furthermore, research in cryptography highlights that “AES’s ascendancy to the forefront of encryption can be attributed to its unparalleled security features… ensuring the confidentiality of encrypted data remains intact” (Ganesh, 2025). These theoretical perspectives explain why AES-256 is considered a robust choice for building secure file encryption and decryption applications.

Cryptographic Systems and Application Frameworks

Effective encryption extends beyond the algorithm itself to encompass secure system integration, proper key management, and user-friendly application design. The theoretical base of modern cryptography originates from key principles in communication and secrecy systems that formalize symmetric key cryptography, shaping standards like AES (Shannon, 1949). According to this foundational work, secure cryptographic systems must balance confidentiality, integrity, and usability to prevent unauthorized data access.

Additionally, deploying encryption applications through reliable frameworks enhances usability and accessibility, particularly for non-technical users. As described in the literature, “Streamlit enables developers to deliver interactive data applications with security and usability in mind” (Streamlit, 2024). With Streamlit, developers can build Python-based interactive interfaces for encryption processes, enabling users to perform secure file encryption and decryption without deep programming knowledge.

Thus, the combination of AES-256 core theory and practical application frameworks like Streamlit supports both theoretical rigor and real-world usability in secure software design.

METHOD

This research employs a straightforward software engineering approach, involving the design and implementation of a web-based file-encryption application prototype rather than undertaking 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 derived from the user-entered password using the PBKDF2-HMAC-SHA256 key derivation function, supplemented with a salt and a sufficiently high number of iterations to enhance resistance to brute-force attacks.

The application was built using the Python programming language and the Streamlit framework for the web interface, allowing users to access encryption and decryption functions in a local browser without requiring direct interaction with the command line. The prototype runs on a single personal computer that serves as both a server and a client, with uploaded and downloaded files stored on the user's local file system.

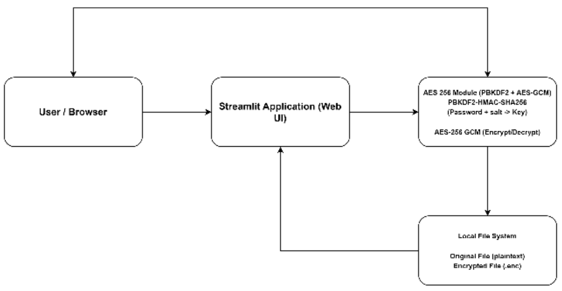

User / Browser
Streamlit Application (Web UI)
AES 256 Module (PBKDF2 + AES-GCM)
PBKDF2-HMAC-SHA256
(Password + salt -> Key)
AES-256 GCM (Encrypt/Decrypt)
Local File System
Original File (plaintext)
Encrypted File (.enc)

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

The application offers two primary operating modes: Encrypt and Decrypt, which are available 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, generates a nonce for AES-GCM, and encrypts the file contents. 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 matches 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, and 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 (Ferguson et a., 2010).

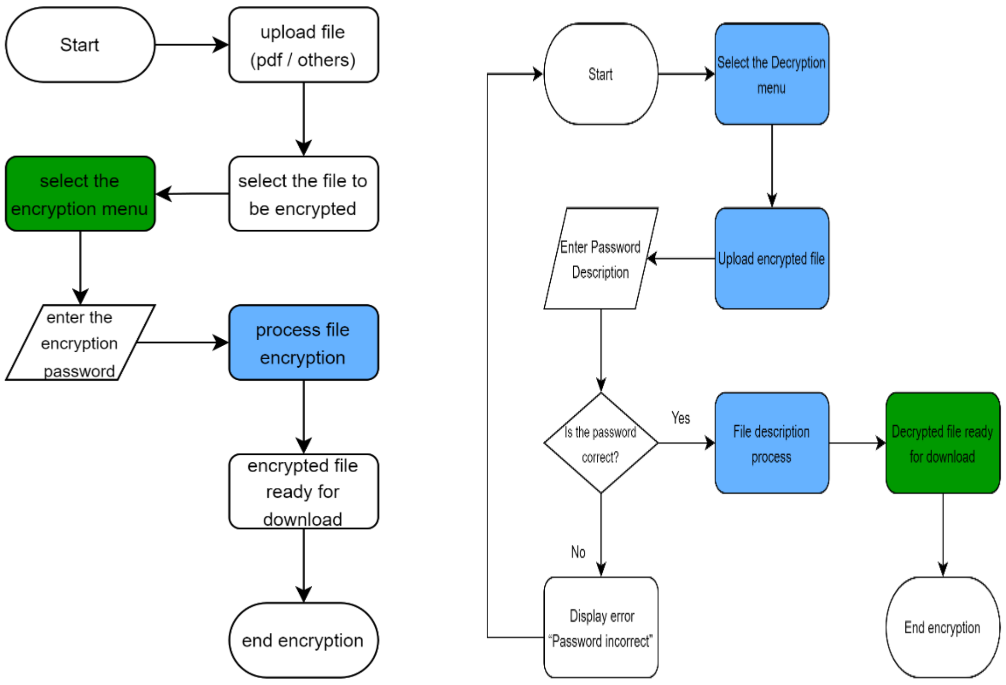

Start
upload file (pdf / others)
select the file to be encrypted
select the encryption menu
enter the encryption password
process file encryption
encrypted file ready for download
end encryption
Start
Select the Decryption menu
Upload encrypted file
Enter Password Description
Is the password correct?
Yes
No
File description process
Decrypted file ready for download
Display error "Password incorrect"
End encryption

Figure 2. Flowchart of the encryption and decryption process.

In the encryption business process, users access the application via a browser, select Encrypt mode, then upload the files they want to protect and enter a password to generate 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, reformulates the AES-256 key using PBKDF2, and performs 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 displays only an error message and never releases the original file (National Institute of Standards and Technology, 2010)

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

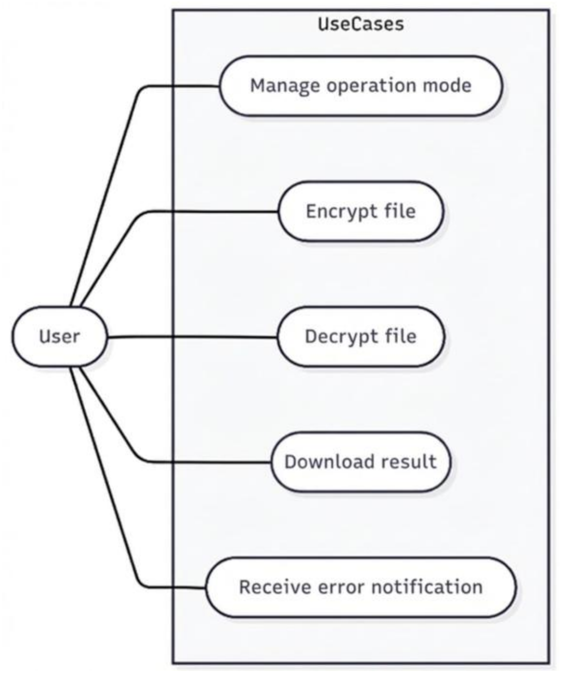


Figure 3. Use Case Diagram

Several test files of varied sizes were run on a single personal computer to undertake the experimental testing. To simulate typical use cases, the test files ranged in size from a few megabytes to several gigabytes, encompassing both small text files and larger PDFs (approximately 1 MB). Using the Streamlit interface, we encrypted and decrypted every file. Additionally, the processing time, from the moment the process button was clicked until the download button appeared, was manually measured using a stopwatch.

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.

Table 1. File Encryption and Decryption Testing Scenarios

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

RESULT AND DISCUSSION

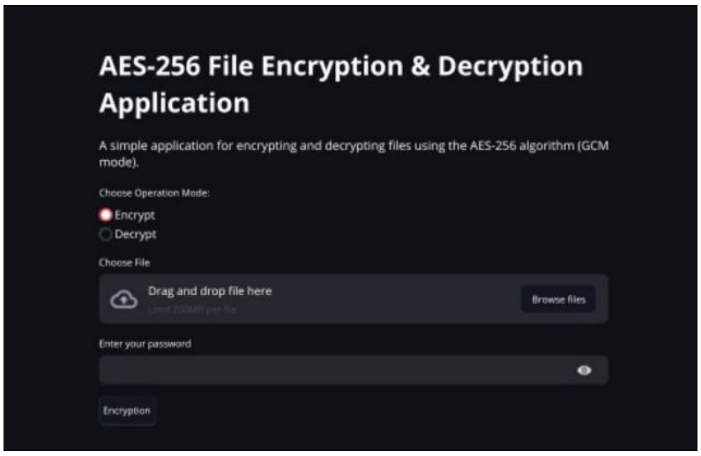
The testing was conducted according to the scenario outlined in Table 1, using three test files of varying sizes: small, medium, and large. Each file was encrypted and decrypted once, and then the file size and processing times for encryption and decryption were recorded to provide an overview of the effect of file size on application performance. Verification was performed by opening the decrypted file and confirming its consistency with the original file.

Table 2. Encryption and Decryption Time Based on File Type and Size

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 with file size, 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, allowing users to perform encryption and decryption without needing 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.



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
Limit 200MB per file
Browse files

Enter your password

Encryption

Figure 4. Application interface display in encryption mode

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 indicate that the encryption-decryption process operates correctly on various file types and sizes, with an acceptable time overhead that meets the needs of individual users. Additionally, decryption attempts with incorrect passwords did not yield plaintext, indicating that the implemented security mechanism maintains file confidentiality as long as the password is not known to other parties.

In the future, the application can be further developed to include more secure password management, separate key configuration storage, and integration with cloud storage services to support backup and encrypted file sharing scenarios (Bishop, 2018).

REFERENCES

Bishop, M. (2018). *Computer security: Art and science* (2nd ed.). Addison-Wesley.

Ferguson, N., Schneier, B., & Kohno, T. (2010). *Cryptography engineering: Design principles and practical applications*. Wiley.

Ganesh, R. (2025). A panoramic survey of the advanced encryption standard. *Security and Communication Networks*. https://doi.org/10.1007/s10207-025-01116-x

National Institute of Standards and Technology. (2007). *Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) (NIST SP 800-38D)*. U.S. Department of Commerce.

National Institute of Standards and Technology. (2010). *Recommendation for password-based key derivation (NIST SP 800-132)*. U.S. Department of Commerce.

National Institute of Standards and Technology. (2023). *FIPS PUB 197: Advanced Encryption Standard (AES)*. U.S. Department of Commerce. https://doi.org/10.6028/NIST.FIPS.197-2023

Shannon, C. E. (1949). Communication theory of secrecy systems. *Bell System Technical Journal*.

Stallings, W. (2017). *Cryptography and network security: Principles and practice* (7th ed.). Pearson.

Streamlit. (2024). *Streamlit documentation and framework overview*.