IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Secure File Storage Using Hybrid Cryptography

Dr. Chandrakala B M, Aaryan Singh Rajput, Ananya M, Abhijeet Kumar, Aditya Sinha

Department of Information Science Engineering
Dayananda Sagar College of Engineering, Bangalore

Abstract- In today's digital world, when sensitive data is frequently in danger of being read by unauthorized parties, secure file storage is a crucial concern. Hybrid cryptography, which combines the advantages of many encryption methods in both a vertical and horizontal way, is one approach to solving this issue. In hybrid cryptography, the original symmetric key is encrypted using a different symmetric key while the real data is encrypted using a different symmetric key. This permits the data to be encrypted and decrypted quickly and effectively, while also adding the extra protection of several encryption methods for authentication. Organizations may guarantee the safe storage of their critical data while utilizing hybrid cryptography, shielding it from illegal access and any data breaches.

Keywords - Hybrid Cryptography, Symmetric Encryption, Cipher Block Chaining, Data Security, Data Protection, Information Security, Cryptographic Techniques, Cloud.

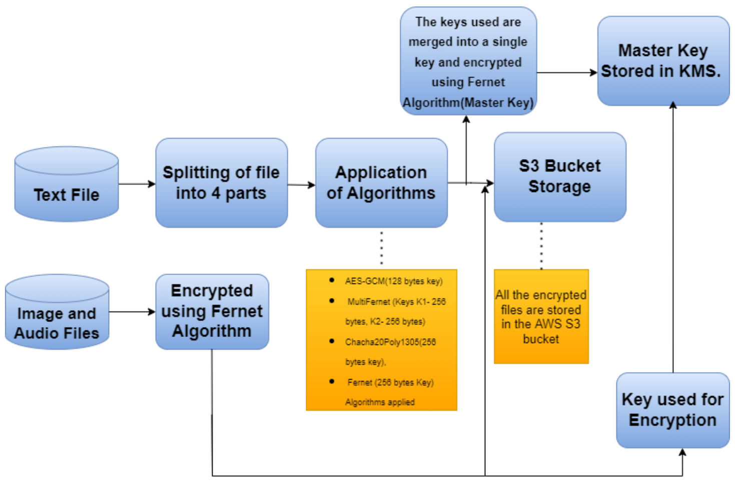
I. INTRODUCTION

A strong security system is required to handle our data and prevent malicious hackers from accessing it, which has made the demand for safe file storage more pressing in recent years as our digital world continues to grow tremendously. These files must be secured against unauthorized access, and a strong security system must be implemented to safeguard critical user data and make it extremely challenging for threat actors to acquire unauthorized access. The title concept of this project is named Secure File Storage using Hybrid Cryptography. To safely store and transmit the data on the Flask Web application framework, we are using the encryption algorithms AESGCM, Fernet, Multi-Fernet, and ChaChaPoly-1305 to safely store and send the data. Flask is a compact Python web framework that offers helpful tools and functionalities for developing web applications. In order to establish an impenetrable environment where the contents of the file would remain encrypted even in the event of a breach or leak, files are encrypted utilizing algorithms in many rounds.

II. PROPOSED METHODOLOGY

Select the particular file to encrypt. It can be either a text file or a multimedia file.
Divide the file into 4 parts and encrypt each part applying a particular algorithm to one part of the file AES-GCM(128 bytes key), Multi-Fernet (Keys K1- 256 bytes, K2- 256 bytes), Chacha20 Poly1305(256 bytes key), Fernet (256 bytes Key) algorithms are applied. The keys used are merged into a single key and encrypted using the Fernet Algorithm(Master Key) which enables multilayer encryption of the key.
Get the encrypted file and send it to AWS S3 Storage Bucket using the Boto S3 API kit.
To decrypt the file, the authorized user needs to download the file and use the master symmetric key to decrypt the keys, and using the keys they can decrypt and obtain the divided parts of the file that is later merged into the original file through our web application.

I. Project Architecture


The keys used are merged into a single key and encrypted using Fernet Algorithm(Master Key)
Master Key Stored in KMS.
Text File
Splitting of file into 4 parts
Application of Algorithms
S3 Bucket Storage
AES-GCM(128 bytes key)
MultiFernet (Keys K1- 256 bytes, K2- 256 bytes)
Chacha20Poly1305(256 bytes key),
Fernet (256 bytes Key)
Algorithms applied
All the encrypted files are stored in the AWS S3 bucket
Image and Audio Files
Encrypted using Fernet Algorithm
Key used for Encryption

PROPOSED ALGORITHMS

Four different methods are used in our file encryption process to offer security and effectiveness. We use several encryption methods based on the kind of files that are uploaded. We employ the straightforward and quick symmetric-key Fernet encryption technique for media data including pictures, movies, and music. We use the combination of **AES-GCM (Galois Counter Mode), ChaCha20-Poly1305 algorithms and Fernet, Multi-Fernet** library modules—for text files. An authorized encryption method called **AES-GCM** guarantees the data's secrecy and integrity. The **Fernet** standard uses AES in CBC mode and HMAC to encrypt and sign messages. Another authenticated encryption method that makes use of a stream cipher and a MAC based on polynomial evaluation is **ChaCha20-Poly1305**. **Multi-Fernet** is a way of applying multiple Fernet keys in sequence to increase the security level.

AES in counter mode for encryption and **G-HASH** for authentication are combined to form **AES-GCM**.
AES is a block cipher that uses a secret key to encrypt data in 128-bit blocks. By encrypting a series of numbers (known as counters) and XORing them with plaintext, counter mode converts a block cipher into a stream cipher. This prevents keystream bytes from being repeated while using the same key to encrypt data of any length.
Using the ciphertext and optional associated data (AD) that is not encrypted but has to be validated, the GHASH function generates a message authentication code (MAC). The MAC, also known as an authentication tag, makes sure that the ciphertext and AD are not tampered. **GHASH** basically uses Galois Field Arithmetic to perform fast and parallelizable computations that help with the efficiency and speed of the algorithm.

Encryption and Decryption algorithms are as follows:

Encryption: $C = P \oplus E(K, IV \,||\, i)$
where C is the ciphertext, P is the plaintext, K is the key, IV is the initialization vector, i is the block number, and E is the AES encryption function.

Decryption: $P = C \oplus E(K, IV \,||\, i)$
where P is the plaintext, C is the ciphertext, K is the key, IV is the initialization vector, i is the block number, and E is the AES encryption function.

$Tag = E(K, Y0) \oplus GHASH(H, A \,||\, C \,||\, len(A) \,||\, len©)$
where Tag is the authentication tag, K is the key, Y0 is an initial counter block, GHASH is the hash function, H is the key-dependent value, A is the additional authenticated data (not encrypted), C is the ciphertext, and len() is a function that returns the bit length of its argument.

AES-GCM has these advantages over normal AES implementations -

1) **AES-GCM** provides both encryption and authentication in one operation, while other AES modes may require a separate MAC algorithm to ensure data integrity and authenticity.
2) **AES-GCM** is faster and more parallelizable than other AES modes, especially when a large portion of the plaintext is treated as Associated Data (AD). This saves processing time and power.
3) **AES-GCM** is more secure than other AES modes against various attacks like **Chosen Plaintext Attack (CPA)** and **Chosen Ciphertext Attack (CCA)**. It's because AES-GCM uses a unique nonce for each message.

CHACHA20-POLY1305 is an authenticated encryption with additional data (AEAD) algorithm that combines the ChaCha20 stream cipher with the Poly1305 message authentication code. It encrypts plaintext using a 256-bit key and a 96-bit nonce, with a 128-bit ciphertext expansion (the tag size).

The ChaCha20-Poly1305 algorithm is used in IETF protocols and is standardized in RFC 84391. It has fast software performance without hardware acceleration. Poly1305 can be used as a one-time message authentication code to authenticate a single message using a key shared by the sender and recipient.

Encryption and Decryption algorithm is as follows:
Encryption: $C = P \oplus ChaCha20(K, N \,||\, Ctr)$
where C is the ciphertext, P is the plaintext, K is the key, N is the nonce, Ctr is the counter, and ChaCha20 is the stream cipher function.
Decryption: $P = C \oplus ChaCha20(K, N \,||\, Ctr)$
where P is the plaintext, C is the ciphertext, K is the key, N is the nonce, Ctr is the counter, and ChaCha20 is the stream cipher function.

Tag = $Poly1305(K', A \,||\, C \,||\, len(A) \,||\, len©)$
where Tag is the authentication tag, K' is a one-time key derived from ChaCha20(K, N || 0), A is the additional authenticated data (not encrypted), C is the ciphertext, len() is a function that returns the bit length of its argument, and Poly1305 is the message authentication code function.

CHACHA20-POLY1305 has these advantages-

1) **CHACHA20-POLY1305 t**ypically outperforms the AES-GCM method. On platforms where the processor lacks the AES-NI instruction set expansion, ChaCha20-Poly1305 typically outperforms the AES-GCM method.
2) **CHACHA20-POLY1305** offers 256 bits of security, which is regarded as adequate for future-proofing communication.
3) **CHACHA20-POLY1305** also offers authentication, preventing attackers from injecting fake messages into a secure stream.

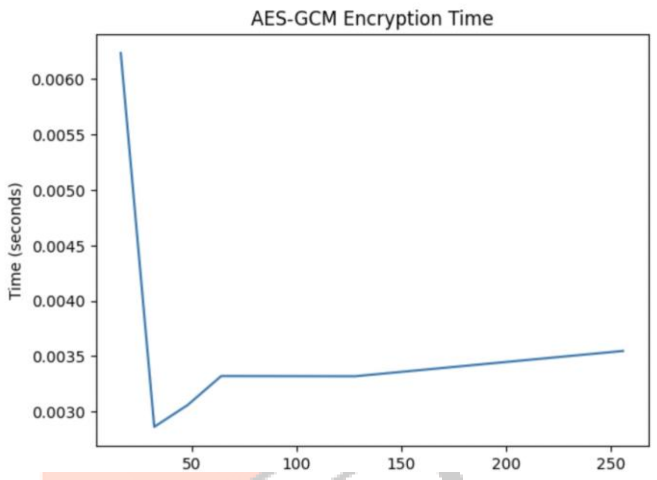
FERNET AND MULTI-FERNET are Python cryptography library modules that provide symmetric data encryption and decryption. Fernet encrypts and decrypts data with a single key, whereas Multi-Fernet encrypts and decrypts data with a list of keys. Multi-Fernet encrypts using the first key in the provided list. Multi-Fernet decrypts tokens one at a time using each key. A cryptography.fernet.InvalidToken exception is thrown if the correct key is not found in the provided list. Multi-Fernet attempts to decrypt the data using all of the Fernets in the list.

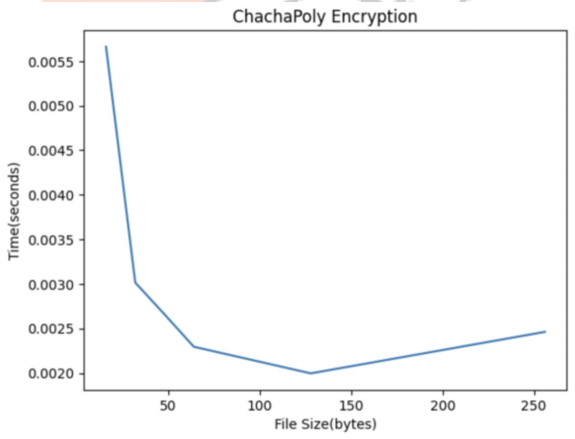
FERNET AND MULTI-FERNET cryptography library modules have the following advantages:

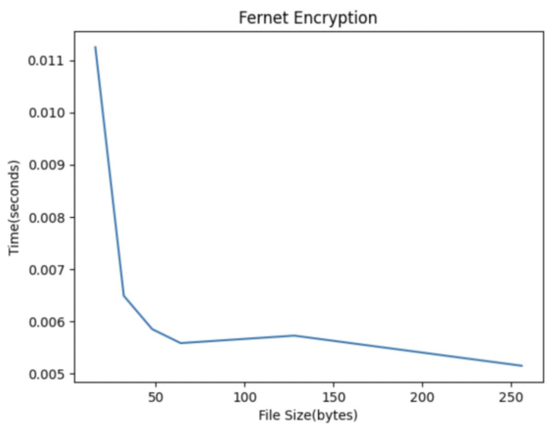
1) **FERNET & MULTI-FERNET LIBRARY MODULE** enables encryption as well as authentication, ensuring data confidentiality and integrity.
2) **FERNET & MULTI-FERNET LIBRARY MODULE** is simple to use and integrate with Python's cryptography library.
3) Since **FERNET & MULTI-FERNET LIBRARY MODULE** employs AES in CBC mode with HMAC-SHA2562, it is less vulnerable to side-channel attacks than other algorithms.

III. EFFICIENCY OF THE ALGORITHMS USED

The algorithms used of AES-GCM, Chacha20Poly1305, Fernet and MultiFernet algorithms have their efficiency calculated based on the speed with which the algorithm is encrypting the files. The efficiency is depicted in the graphs given below where the x-axis represents the time(seconds) variable and the y-axis represents the file size(bytes) variable.







There can be an observation made that as the size of the file increases the the time taken to encrypt the file decreases.

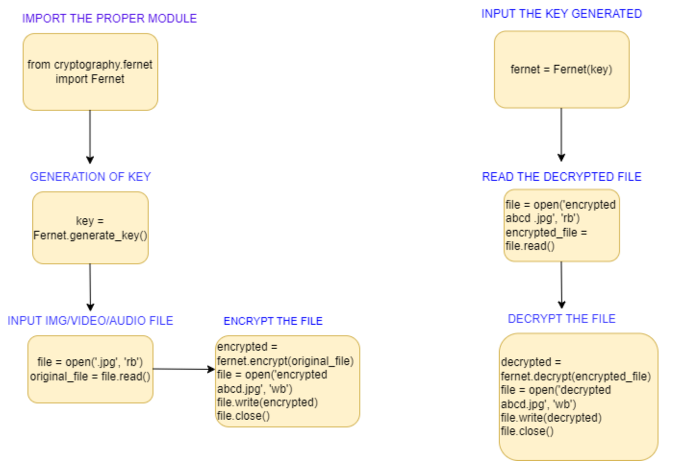
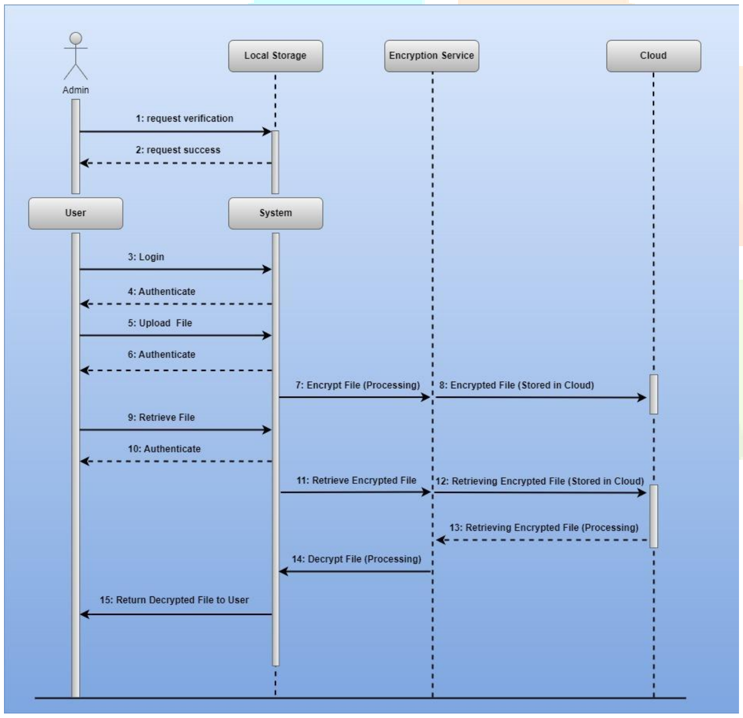

IMPORT THE PROPER MODULE
from cryptography.fernet import Fernet
GENERATION OF KEY
key = Fernet.generate_key()
INPUT IMG/VIDEO/AUDIO FILE
file = open('.jpg', 'rb')
original_file = file.read()
ENCRYPT THE FILE
encrypted = fernet.encrypt(original_file)
file = open('encrypted abcd.jpg', 'wb')
file.write(encrypted)
file.close()
INPUT THE KEY GENERATED
fernet = Fernet(key)
READ THE DECRYPTED FILE
file = open('encrypted abcd .jpg', 'rb')
encrypted_file = file.read()
DECRYPT THE FILE
decrypted = fernet.decrypt(encrypted_file)
file = open('decrypted abcd.jpg', 'wb')
file.write(decrypted)
file.close()

Fig. Encryption and Decryption of Image/Video/Audio Files

IV. SEQUENCE DIAGRAM OF THE PROCESS


Admin
Local Storage
Encryption Service
Cloud
1: request verification
2: request success
User
System
3: Login
4: Authenticate
5: Upload File
6: Authenticate
7: Encrypt File (Processing)
8: Encrypted File (Stored in Cloud)
9: Retrieve File
10: Authenticate
11: Retrieve Encrypted File
12: Retrieving Encrypted File (Stored in Cloud)
13: Retrieving Encrypted File (Processing)
14: Decrypt File (Processing)
15: Return Decrypted File to User

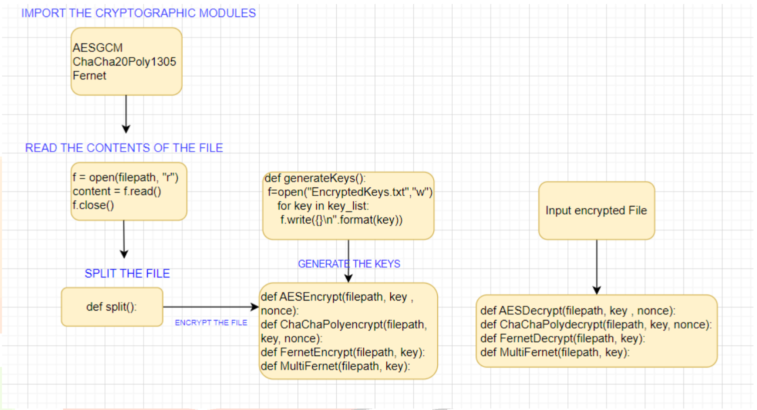

IMPORT THE CRYPTOGRAPHIC MODULES
AESGCM
ChaCha20Poly1305
Fernet
READ THE CONTENTS OF THE FILE
f = open(filepath, "r")
content = f.read()
f.close()
SPLIT THE FILE
def split():
ENCRYPT THE FILE
def generateKeys():
f=open("EncryptedKeys.txt","w")
for key in key_list:
f.write({}\n".format(key))
GENERATE THE KEYS
def AESEncrypt(filepath, key , nonce):
def ChaChaPolyencrypt(filepath, key, nonce):
def FernetEncrypt(filepath, key):
def MultiFernet(filepath, key):
Input encrypted File
def AESDecrypt(filepath, key , nonce):
def ChaChaPolydecrypt(filepath, key, nonce):
def FernetDecrypt(filepath, key):
def MultiFernet(filepath, key):

Fig. Encryption and Decryption of Text files

V. USER MODULEs

IV. CONCLUSION

This project aimed to contribute to the creation of a secure environment for the exchange of media and information in the cyber world. To achieve this goal, a combination of encryption algorithms was utilized, which proved to be efficient, reliable, and fast during multiple test cases conducted. The project can be further expanded by incorporating the feature of Amazon Web Services' Key Management Service (KMS). This addition would enhance the security of the system by encrypting the master key, making it even more secure and difficult to compromise. By leveraging KMS, the project can benefit from a managed service that provides strong encryption capabilities and simplifies key management. Moreover, a mobile application can be developed to facilitate easier access to this secure service. By creating a mobile app, users would have the convenience of securely exchanging media and information directly from their mobile devices, making the service more accessible and user-friendly. Overall, this application was developed with the intention of contributing to the field of cybersecurity.

It serves as a stepping stone for further research and experimentation, particularly in exploring and testing new combinations of cryptographic algorithms. The ultimate aim is to develop a robust file transfer architecture that can withstand potential threats and provide a safe environment for users to exchange sensitive data in the cyber-world.

REFERENCES

[1] G. O. Young, “Synthetic structure of industrial plastics (Book style with paper title and editor),” in *Plastics*, 2nd ed. vol. 3, J. Peters, Ed. New York: McGraw-Hill, 1964, pp. 15–64.

[2] W.-K. Chen, *Linear Networks and Systems* (Book style). Belmont, CA: Wadsworth, 1993, pp. 123–135.

[3] H. Poor, *An Introduction to Signal Detection and Estimation*. New York: Springer-Verlag, 1985, ch. 4.

[4] B. Smith, “An approach to graphs of linear forms (Unpublished work style),” unpublished.

[5] E. H. Miller, “A note on reflector arrays (Periodical style—Accepted for publication),” *IEEE Trans. Antennas Propagat.*, to be published.

[6] J. Wang, “Fundamentals of erbium-doped fiber amplifiers arrays (Periodical style—Submitted for publication),” *IEEE J. Quantum Electron.*, submitted for publication.