**Advanced Encryption Standard (AES)**

**1. Introduction**

The Advanced Encryption Standard (AES) is a symmetric-key encryption algorithm widely used for secure data transmission. It operates on blocks of data and follows multiple encryption rounds involving substitution, permutation, mixing, and key addition.

This report details the implementation of AES in Python, explaining the encryption, decryption, and key management processes.

**2. Implementation**

The AES implementation uses the Python cryptography library to perform encryption and decryption with AES-256 in Cipher Feedback (CFB) mode. The implementation includes:

* Key Generation: A 256-bit encryption key and a 16-byte IV.
* Encryption: Converts plaintext into ciphertext using AES in CFB mode.
* Decryption: Recovers plaintext from ciphertext using the same key and IV.
* Base64 Encoding: Used for displaying encrypted text in a readable format.

**3. Python Code:**

from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes

from cryptography.hazmat.backends import default\_backend

import os

import base64

# AES Encryption Function

def aes\_encrypt(plaintext, key):

iv = os.urandom(16) # Generate a random Initialization Vector (IV)

cipher = Cipher(algorithms.AES(key), modes.CFB(iv), backend=default\_backend())

encryptor = cipher.encryptor()

ciphertext = encryptor.update(plaintext.encode()) + encryptor.finalize()

return iv + ciphertext

# AES Decryption Function

def aes\_decrypt(ciphertext, key):

iv = ciphertext[:16]

actual\_ciphertext = ciphertext[16:]

cipher = Cipher(algorithms.AES(key), modes.CFB(iv), backend=default\_backend())

decryptor = cipher.decryptor()

decrypted\_text = decryptor.update(actual\_ciphertext) + decryptor.finalize()

return decrypted\_text.decode()

# Key generation (256-bit key)

key = os.urandom(32)

plaintext = "This is a secret message."

# Encrypt the message

ciphertext = aes\_encrypt(plaintext, key)

encoded\_ciphertext = base64.b64encode(ciphertext).decode()

print("Ciphertext (Base64):", encoded\_ciphertext)

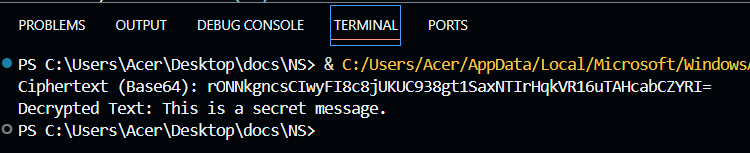
# Decrypt the message

decoded\_ciphertext = base64.b64decode(encoded\_ciphertext)

decrypted\_text = aes\_decrypt(decoded\_ciphertext, key)

print("Decrypted Text:", decrypted\_text)

**4. Output**



**5. Explanation of Implementation**

1️) Key and IV Generation

* The 256-bit AES key is randomly generated (os.urandom(32)).
* A 16-byte IV is also randomly generated (os.urandom(16)) to ensure unique encryption each time.

2️) Encryption Process

* The plaintext is first converted into bytes.
* AES encryption is performed using CFB mode, which turns AES into a stream cipher (suitable for variable-length messages).
* The IV is prepended to the ciphertext to ensure proper decryption.
* The final output is Base64 encoded for readability.

3️) Decryption Process

* The IV is extracted from the first 16 bytes of the ciphertext.
* AES decryption is performed using the same key and IV.
* The decrypted bytes are converted back into a readable string.

**6. Security Considerations**

-> Why AES-256?

* AES-256 provides high security with a 256-bit key, making brute-force attacks impractical.

-> Why CFB Mode?

* Cipher Feedback (CFB) mode turns AES into a stream cipher, useful for encrypting text of any length without requiring padding.

-> Why Base64 Encoding?

* AES output is binary data, which is not readable. Base64 encoding makes it easier to store and transmit.

Key Security

* The encryption key must remain secret. If exposed, an attacker can decrypt the message.
* Instead of hardcoding, keys should be stored securely (e.g., environment variables, key vaults).

**7. Conclusion**

This implementation demonstrates AES-256 encryption and decryption in Python using a secure, efficient approach. By leveraging the cryptography library, we ensure correctness while following best security practices.

This method can be extended for encrypting files, messages, and sensitive data in real-world applications.