AES Message Encryptor

Schifîrneț Petre-Iustin

Petroleum-Gas University of Ploiești
Faculty of Mechanical and Electrical Engineering – Applied Informatics and Automation

Introduction

This project, titled AES Message Encryptor, is a command-line application written in Python that allows users to encrypt and decrypt text messages using the AES (Advanced Encryption Standard) symmetric encryption algorithm in CBC (Cipher Block Chaining) mode. The goal of this tool is to provide a simple and educational experience for understanding basic cryptographic operations, key derivation, and secure text handling, while maintaining compatibility across operating systems.

Although not intended for critical or production-grade cryptographic systems, the application demonstrates secure design choices such as dynamic IV (initialization vector) generation and key derivation using SHA-256 hashing, following best practices for educational use cases.

Functional Overview

The tool provides two main functionalities:

- Encrypt a message: Convert readable text (plaintext) into an unreadable format (ciphertext) using a derived AES key.
- Decrypt a message: Reverse the process and recover the original text from the encrypted data using the correct key.

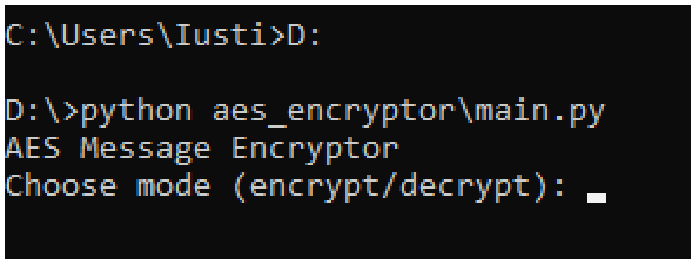
To ensure secure key handling, the user is not required to manually input a 128-bit key. Instead, they provide a simple passphrase, which is internally processed using the SHA-256 hashing algorithm**,** from which the first 16 bytes (128 bits) are extracted and used as the AES key. This makes the tool much more user-friendly while still maintaining strong security.

The application is fully written in Python and uses only standard or lightweight libraries:

- hashlib for SHA-256 hashing
- pycryptodome for AES encryption (an actively maintained alternative to PyCrypto)

Encryption Process – Step by Step

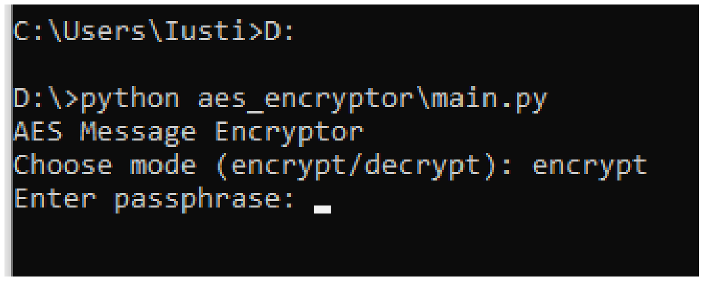
1. User Input: The user selects the encrypt mode and inputs a message along with a custom passphrase.
2. Key Derivation: The passphrase is converted into bytes and hashed with SHA-256. The first 16 bytes of the result are extracted to create a 128-bit AES key.
3. IV Generation: A random 16-byte IV is generated automatically.
4. Encryption: The plaintext is padded to match AES block size and encrypted using AES-CBC with the generated key and IV.
5. Output Format: The IV and ciphertext are both converted to hexadecimal and concatenated into a single string for easy sharing or storage.



```
C:\Users\Iusti>D:

D:\>python aes_encryptor\main.py
AES Message Encryptor
Choose mode (encrypt/decrypt): _
```

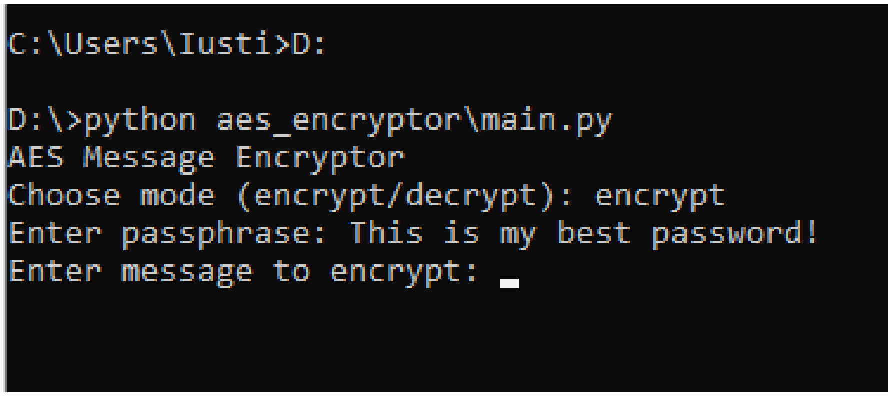
Fig.1, Encrypt mode



```
C:\Users\Iusti>D:

D:\>python aes_encryptor\main.py
AES Message Encryptor
Choose mode (encrypt/decrypt): encrypt
Enter passphrase: _
```

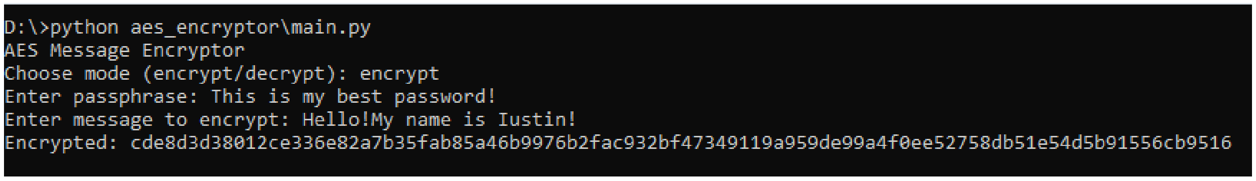
Fig.2, Passphrase (input)



```
C:\Users\Iusti>D:

D:\>python aes_encryptor\main.py
AES Message Encryptor
Choose mode (encrypt/decrypt): encrypt
Enter passphrase: This is my best password!
Enter message to encrypt:
```

Fig.3, Message to encrypt (input)

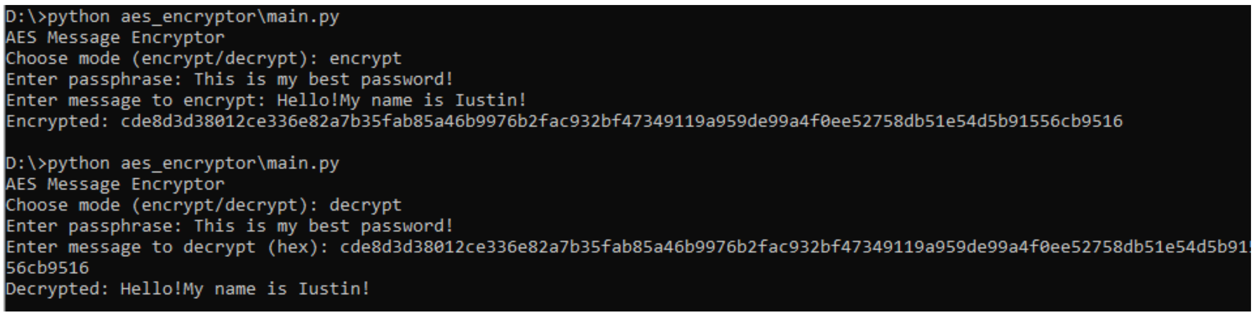


```
D:\>python aes_encryptor\main.py
AES Message Encryptor
Choose mode (encrypt/decrypt): encrypt
Enter passphrase: This is my best password!
Enter message to encrypt: Hello!My name is Iustin!
Encrypted: cde8d3d38012ce336e82a7b35fab85a46b9976b2fac932bf47349119a959de99a4f0ee52758db51e54d5b91556cb9516
```

Fig.4, Encrypted message (output)

Decryption Process – Step by Step

1. User Input: The user selects the `decrypt` mode, enters the ciphertext (in hexadecimal), and re-enters the same passphrase used during encryption.
2. Key Derivation: The passphrase is hashed again using SHA-256, generating the same 128-bit key as before.
3. IV Extraction: The first 32 hex characters (16 bytes) from the ciphertext input represent the IV.
4. Decryption: The ciphertext is decrypted using the extracted IV and derived key.
5. Unpadding and Display: The decrypted result is unpadded and displayed as readable plaintext.



```
D:\>python aes_encryptor\main.py
AES Message Encryptor
Choose mode (encrypt/decrypt): encrypt
Enter passphrase: This is my best password!
Enter message to encrypt: Hello!My name is Iustin!
Encrypted: cde8d3d38012ce336e82a7b35fab85a46b9976b2fac932bf47349119a959de99a4f0ee52758db51e54d5b91556cb9516

D:\>python aes_encryptor\main.py
AES Message Encryptor
Choose mode (encrypt/decrypt): decrypt
Enter passphrase: This is my best password!
Enter message to decrypt (hex): cde8d3d38012ce336e82a7b35fab85a46b9976b2fac932bf47349119a959de99a4f0ee52758db51e54d5b91
56cb9516
Decrypted: Hello!My name is Iustin!
```

Fig.5, Encrypted & Decrypted messages

Limitations & Notes

- This tool handles only text messages, not binary files.
- The passphrase must be remembered exactly; otherwise decryption will fail.
- It does not provide integrity verification (e.g., HMAC), so tampered ciphertext may fail silently or raise exceptions.

Conclusion

The AES Message Encryptor is a lightweight and practical demonstration of secure message encryption and decryption, with a clean and understandable implementation. It serves as a valuable learning resource for students and enthusiasts interested in cryptography, cybersecurity, and Python scripting.

By combining best practices like key derivation, CBC mode, and user-friendly design, this tool strikes a good balance between educational simplicity and real-world relevance.