Computer Security Report

Group Member : Alex Tang

Introduction

In today's digital age, where technological advancements have revolutionized the way we communicate and exchange information, the security and privacy of sensitive data have become paramount concerns. The widespread use of Internet has introduced new challenges, as cyber threats continue to evolve and proliferate. The need for robust cryptographic solutions to safeguard data from unauthorized access and interception has never been more critical. This project embarks on the development of a secure system, aimed at enabling users to encrypt or decrypt messages securely over potentially insecure channels. By leveraging cryptographic primitives, the system ensures the confidentiality and integrity of transmitted data, thereby addressing the inherent vulnerabilities associated with digital communication.

Problem Analysis and Solution

I propose my own project, it is based on cryptography and uses cryptographic primitives. The project is about generating a random key then verifying it then process decryption and encryption plaintext.

Solution:

Key Generation:

The key generation function first generates a random salt using salt = os.urandom(16). Then derive a key using PBKDF2(Password Based Key Derivation Function 2). This function follows a secure key generation process to ensure that the resulting key is strong and suitable for cryptographic operations. Then I store the key that is generated into a local file called “key.txt”. And we could use the key to verify the process to continue the next step.

Get Key Input:

The input string is assumed to represent a hexadecimal-encoded key. The function then uses bytes.fromhex() to convert the hexadecimal string into bytes. The function returns the byte representation of the input key.

Get Plaintext Input:

The plaintext input by the user is assumed to be in string format. The function then uses the encode() method to convert the plaintext string into bytes, The function returns the byte representation of the plaintext entered by the user.

Decryption:

A Cipher object is created using the AES algorithm in CBC mode. The AES algorithm requires a key and an IV for decryption. The key and IV are provided as arguments to the function. This decryptor is used to perform the decryption operation on the ciphertext. The ciphertext is passed to the decryptor's update method to decrypt the data. The result of the decryption operation is obtained by calling the finalize method on the decryptor. Since the plaintext was originally padded before encryption, it needs to be unpadded after decryption to retrieve the original plaintext. This is accomplished by creating an un-padder object with PKCS7 padding, and then applying it to the decrypted plaintext. Then the function returns a plaintext.

Encryption:

A random IV is generated using os.urandom(16). A Cipher object is created using the AES algorithm in CBC mode with the provided key and the randomly generated IV. An encryptor object is created using the initialized Cipher object. This encryptor is used to perform the encryption operation on the plaintext. The plaintext is padded to a multiple of 16 bytes using PKCS7 padding. This ensures that the plaintext is of the correct length for AES encryption. The padded plaintext is passed to the encryptor's update method to encrypt the data. The result of the encryption operation is obtained by calling the finalize method on the encryptor. The function returns a ciphertext.

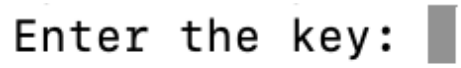
Main Function:

First the function call generate() function to generate a key then store in the local file “key.txt”. Then ask the user to enter the key using the getInput() function. The user-input key is compared with the generated key. If they match, the program proceeds with encryption and decryption. Otherwise, the program will not continue. After passing the key matching part, the user asked to enter plaintext using the getPlaintext() function. The plaintext is then encrypted using the encrypt() function. Next encrypted ciphertext and IV are passed to the decrypt() function along with the generated key to decrypt the ciphertext and obtain the original plaintext. Lastly, the console print out the encryption and decryption message.

Implementation Result

Below will be the process of the implementation. The project use python3 to run.

It ask the user to enter key first.



The key will write to file call “key.txt”

Copy the key enter into console, after pass the verification, ask user input plaintext.

```
Enter the key: df1e78f6d21b49a19ce5604c2633e76d74bd576bd6d0d49457c409ed4a0c8b10
Input Key: df1e78f6d21b49a19ce5604c2633e76d74bd576bd6d0d49457c409ed4a0c8b10
Key matched. Enter message to do encryption or decryption.
Enter the plaintext:
```

Encrypt and Decrypt result.

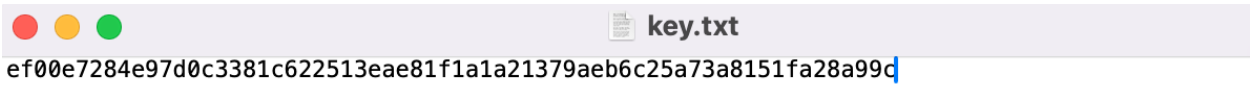
```
Enter the key: df1e78f6d21b49a19ce5604c2633e76d74bd576bd6d0d49457c409ed4a0c8b10
Input Key: df1e78f6d21b49a19ce5604c2633e76d74bd576bd6d0d49457c409ed4a0c8b10
Key matched. Enter message to do encryption or decryption.
Enter the plaintext: Hello
Encrypted ciphertext: 84b9dc1ee3a7289106c8c6308a5bdef3
Decrypted plaintext: Hello
```

Below will be some other results.

Key did not match.

```
Enter the key: cad6e3370dac4ef5d0248d77fb13de00c0c5539fd29fc42315266d9e49e0f6d3
Input Key: cad6e3370dac4ef5d0248d77fb13de00c0c5539fd29fc42315266d9e49e0f6d3
Key did not match.
```


```
Enter the key: a6485beacc2a24e36bf16ecd9a977f267cdc5368d898e162ce270c4f67f53d35
Input Key: a6485beacc2a24e36bf16ecd9a977f267cdc5368d898e162ce270c4f67f53d35
Key matched. Enter message to do encryption or decryption.
Enter the plaintext: 12345678
Encrypted ciphertext: e30bda95d4bb829a1718abc0b3d0bded
Decrypted plaintext: 12345678
```




```
Enter the key: ef00e7284e97d0c3381c622513eae81f1a1a21379aeb6c25a73a8151fa28a99c
Input Key: ef00e7284e97d0c3381c622513eae81f1a1a21379aeb6c25a73a8151fa28a99c
Key matched. Enter message to do encryption or decryption.
Enter the plaintext: Good Morning!
Encrypted ciphertext: d21243be3df03c727e503039095640c3
Decrypted plaintext: Good Morning!
```

Analysis Result

After running several times of the project, I could find that every time the generated key were different, ensuring that subsequent executions produce different keys. When the user input key matches the generated key, the program proceeds with encryption and decryption. If the keys do not match, the program indicates that the keys did not match and terminates further processing. The encrypted ciphertext appears as a hexadecimal string, which is the expected output of the encryption process. The decrypted plaintext matches the original input plaintext, indicating successful encryption and decryption operations.

Conclusion

As a result of implementing decryption and encryption using cryptographic primitives such as random number generation, Advanced Encryption Standard (AES), SHA-256, Password-Based Key Derivation Function 2 (PBKDF2), and Cipher Block Chaining (CBC), several things have been achieved.
Confidentiality: The utilization of AES encryption in combination with CBC mode ensures that plaintext messages are transformed into ciphertext in a secure way.
Key Management: The implementation of PBKDF2 facilitates the secure generation of cryptographic keys from passwords. This enhances key management practices by deriving unique keys from user-provided passwords, reducing the risk of key compromise

Resistance to Attacks: By leveraging a combination of cryptographic primitives, including AES, SHA-256, PBKDF2, and CBC mode, the system enhances its resilience against various cryptographic attacks.