Project 3 Report

Run the Code:

Open a terminal or command prompt and navigate to the project folder.

To execute the code, run the following command:

python prj3.py

This project focuses on implementing a searchable encryption scheme using inverted index, pseudo-random function (PRF), and Advanced Encryption Standard (AES). The goal is to provide a secure and efficient way to search encrypted data while maintaining privacy. The project includes key generation, encryption, token generation, and search functions.

Key Generation:

Two sets of keys are generated using the generatekey function. One set is used for AES encryption (sk2), and the other set is used for a pseudo-random function (sk1).

The keys are stored in files (skaes.txt and skprf.txt) using the writeKeys function.

File Handling:

The script reads and writes various files, such as plaintext, ciphertext, and key files.

File paths are defined at the beginning of the script (tokenFile, indexFile, etc.).

One-Time Pad (OTP) Encryption:

The script includes a function otp that performs OTP encryption using XOR operations.

AES Encryption:

AES encryption is implemented using the encrypt_aes_cbc function for Cipher Block Chaining (CBC) mode.

An additional function AESEnc is provided to encrypt data using AES.

Pseudo-Random Function (PRF):

The script includes a pseudo-random function (prfEnc) that uses the SHA-256 hash function to generate a hash based on a key and data.

Index Building:

An index of words and associated files is built using the buildEnc_index function.

The index is encrypted using AES and saved to a file (indexFile).

Encryption Function

The encryption function is tasked with creating an encrypted inverted index using a combination of AES for file encryption and PRF for keyword encryption. Additionally, the function encrypts all data files using AES-CBC-256 and writes the ciphertexts to the ciphertextfiles folder.

The inverted index is built by associating each keyword with the list of files in which it appears. Keywords are encrypted using the PRF, and the resulting encrypted index is stored in index.txt. The file encryption is done using AES-CBC-256.

Each file's content is encrypted individually using AES-CBC-256. The encrypted files are stored in the ciphertextfiles folder, and their names are used as file identifiers in the encrypted index.

Token Generation:

A token is generated using a combination of a word (w) and a secret key (sk1).

The token is saved to a file (tokenFile).

Decryption:

The script attempts to decrypt the encrypted index using the provided token and secret key.

The decrypted index is processed to extract file-word relationships.

Functions Encrypt CBC and AES

```python
def buildEnc_index(filesFolder, sk):
    unique = findUnique(filesFolder)
    indexEnc = {}
    for word, files in unique.items():
        wordEnc = keywordEnc(word, sk)
        filesEnc = [f"{i}" for i in files]
        indexEnc[wordEnc] = filesEnc
    return indexEnc

def aes_encrypt_block(block, key): # block and key are bytes
    cipher = AES.new(key, AES.MODE_ECB) # Create a new AES cipher
    encrypted_block = cipher.encrypt(block) # Perform AES encryption on the block
    return encrypted_block # Return the encrypted block

def encrypt_aes_cbc(plaintext, key): # plaintext, key, and iv are bytes
    cipher = AES.new(key, AES.MODE_CBC) # Create a new AES cipher
    block_size = AES.block_size # Get the block size of the cipher
    padded_text = plaintext + (block_size - len(plaintext) % block_size) * chr(block_size - len(plaintext) % block_size) # Pad the plaintex
    ciphertext = cipher.encrypt(padded_text.encode('utf-8')) # Perform AES encryption on the padded plaintext
    return ciphertext
```

SK1

prj3.py skaes.txt

data > skaes.txt

```
1  673a48948fefa6d9519163e327c835d5717069eb9c3ea0a3f63019b3ec573cad99c26d1a9da86387bfbbfdbf954f09801e66fd9c429bf7c79736b8fe01cf355cec54b14649c955
```

SK2

prj3.py skprf.txt

data > skprf.txt

```
1  :a106496b8f59397a7ebea34397cc747c98b1993f855871160f1ef95ffb06d9016392cff282379f03e6e9632d3f024806377685ef238d7171ff291ee8ff8ce7c73aac7ed9e9dd5c
```

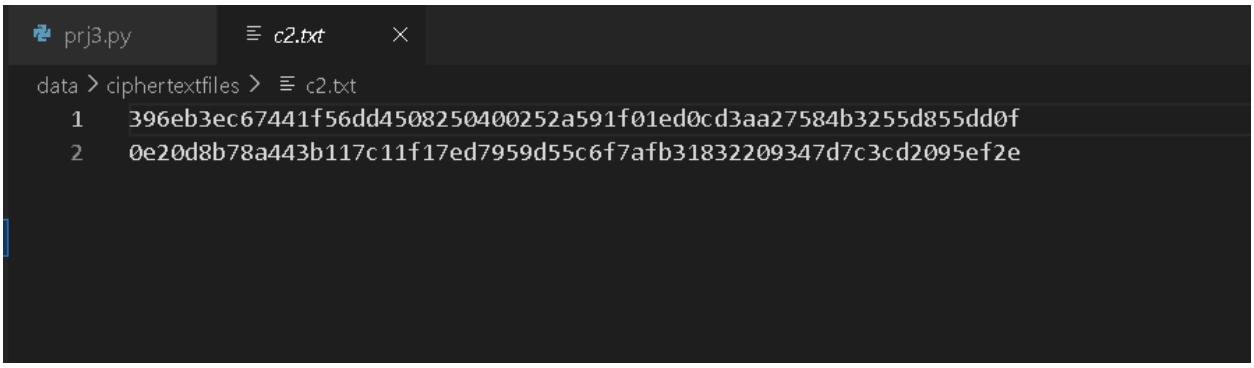
c1

prj3.py c1.txt

data > ciphertextfiles > c1.txt

```
1  18e0809e14a24aa9a3fb4abe7f20e5328e15859cdc94f18c233df81f4f328b22
2  da49c9ee3872064034126754212f112bd9d1c7ab3702c1f399e175a4532f9d21
3  396eb3ec67441f56dd4508250400252a591f01ed0cd3aa27584b3255d855dd0f
```

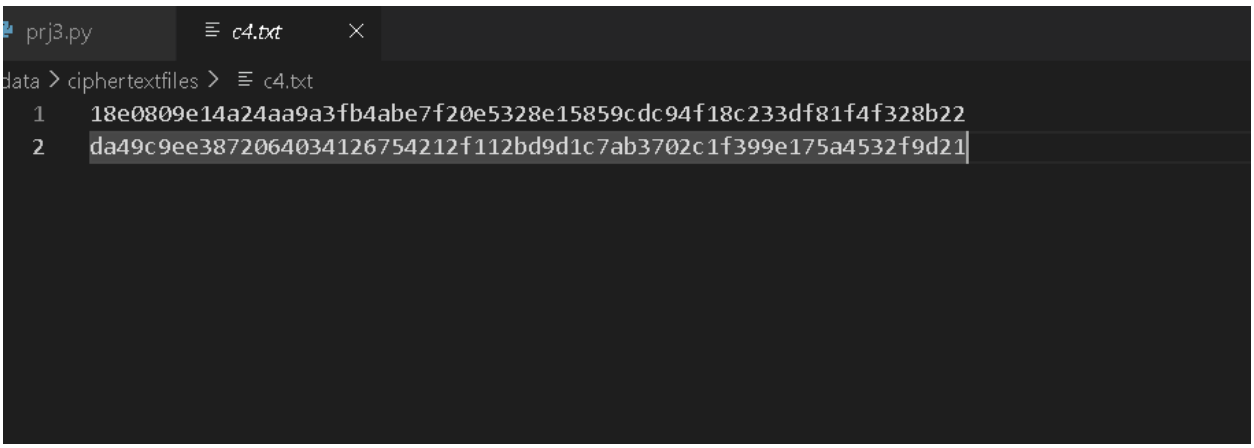
c2


prj3.py
c2.txt
data > ciphertextfiles > c2.txt
1 396eb3ec67441f56dd4508250400252a591f01ed0cd3aa27584b3255d855dd0f
2 0e20d8b78a443b117c11f17ed7959d55c6f7afb31832209347d7c3cd2095ef2e

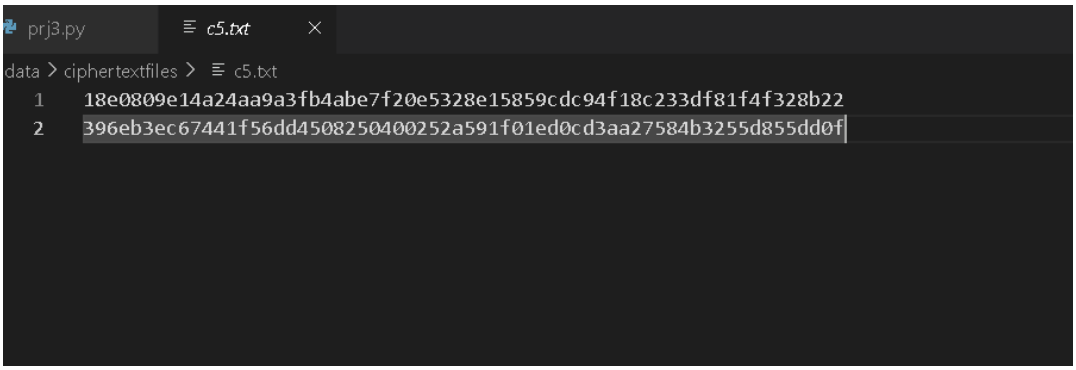
c3


data > ciphertextfiles > c3.txt
1 396eb3ec67441f56dd4508250400252a591f01ed0cd3aa27584b3255d855dd0f

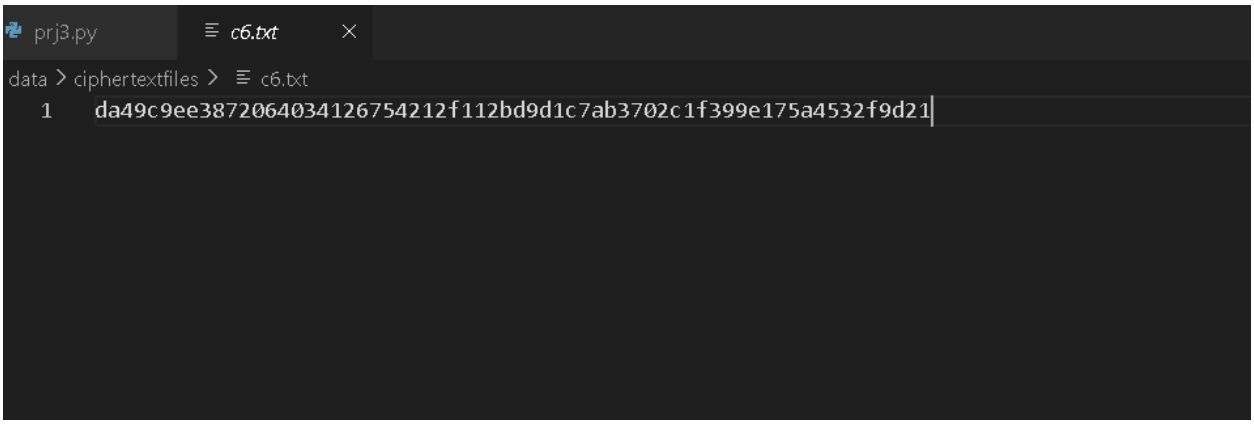
c4


prj3.py
c4.txt
data > ciphertextfiles > c4.txt
1 18e0809e14a24aa9a3fb4abe7f20e5328e15859cdc94f18c233df81f4f328b22
2 da49c9ee3872064034126754212f112bd9d1c7ab3702c1f399e175a4532f9d21

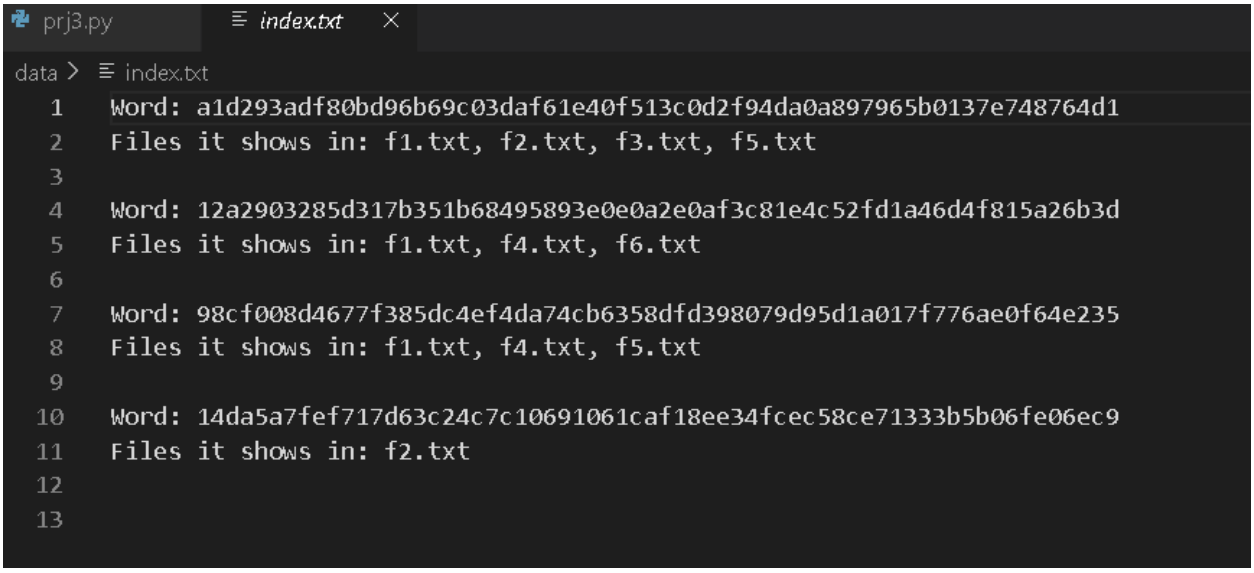
c5


prj3.py
c5.txt
data > ciphertextfiles > c5.txt
1 18e0809e14a24aa9a3fb4abe7f20e5328e15859cdc94f18c233df81f4f328b22
2 396eb3ec67441f56dd4508250400252a591f01ed0cd3aa27584b3255d855dd0f

c6



Index / Search



Token Generator