Project 2: 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 prj_2.py

The code will execute the following steps and print relevant information along the way:

- Generate a random secret key and store it in data/key.txt.
- Read the plaintext from data/plaintext.txt.
- Generate a random Initialization Vector (IV) and store it in data/iv.txt.
- Encrypt the plaintext using AES-CBC mode and the secret key.
- Write the ciphertext to data/ciphertext.txt.
- Decrypt the ciphertext using the same secret key and IV.
- Display the decrypted plaintext and write it to data/result.txt.

aes_encrypt_block(block, key):

This function takes a block of data and an AES encryption key as input, both in bytes format.

It creates an AES cipher in ECB (Electronic Codebook) mode using the provided key.

The function encrypts the input block using AES encryption and returns the encrypted block as bytes.

aes_decrypt_block(block, key):

Similar to aes_encrypt_block, this function takes an encrypted block and an AES decryption key as input.

It creates an AES cipher in ECB mode with the provided key.

The function decrypts the input block using AES decryption and returns the decrypted block as bytes.

```python
# Function to perform AES encryption on a single block
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

# Function to perform AES decryption on a single block
def aes_decrypt_block(block, key): # block and key are bytes
    cipher = AES.new(key, AES.MODE_ECB) # Create a new AES cipher
    decrypted_block = cipher.decrypt(block) # Perform AES decryption on the block
    return decrypted_block  # Return the decrypted block
```

encrypt_aes_cbc(plaintext, key, iv):

This function performs AES encryption in CBC (Cipher Block Chaining) mode.

It takes plaintext (bytes), an encryption key (bytes), and an initialization vector (bytes) as input.

The function pads the plaintext to ensure it is a multiple of the block size (16 bytes for AES).

It creates an AES cipher in CBC mode using the provided key and IV.

The plaintext is encoded in UTF-8, encrypted, and the ciphertext is returned as bytes.

decrypt_aes_cbc(ciphertext, key, iv):

This function performs AES decryption in CBC mode.

It takes the ciphertext (bytes), an encryption key (bytes), and an initialization vector (bytes) as input.

The function decrypts the ciphertext using AES decryption in CBC mode and returns the plaintext as a UTF-8 encoded string.

```python
# Function to perform AES encryption in CBC mode
def encrypt_aes_cbc(plaintext, key, iv): # plaintext, key, and iv are bytes
    cipher = AES.new(key, AES.MODE_CBC, iv) # 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 plaintext
    ciphertext = cipher.encrypt(padded_text.encode('utf-8')) # Perform AES encryption on the padded plaintext
    return ciphertext

# Function to perform AES decryption in CBC mode
def decrypt_aes_cbc(ciphertext, key, iv): # ciphertext, key, and iv are bytes
    cipher = AES.new(key, AES.MODE_CBC, iv) # Create a new AES cipher
    plaintext = cipher.decrypt(ciphertext).decode('utf-8') # Perform AES decryption on the ciphertext
    padding_length = ord(plaintext[-1]) # Get the length of the padding
    return plaintext[:-padding_length] # Return the unpadded plaintext
```

gen_iv():

This function generates a random Initialization Vector (IV) for AES encryption.

It returns a 16-byte random value as bytes.

```
# Function to generate a random initialization vector (IV)
def gen_iv(): # Returns a 16-byte random value
    return get_random_bytes(16) # Return a 16-byte random value
```

writeKey(key):

This function takes an AES encryption key (bytes) as input and writes it to a file named data/key.txt in hexadecimal format.

getKey():

This function reads the AES encryption key from the data/key.txt file in hexadecimal format, converts it to bytes, and returns it.

If the file is not found, it prints an error message and returns None.

printKey(sk):

This function prints the hexadecimal representation of the AES encryption key provided as input (bytes).

```
def writeKey(key):
    key_file='data/key.txt' # Set the key file path
    hexKey = key.hex() # Convert the key to a hexadecimal string
    with open(key_file, 'w') as file: # Open the key file
        file.write(hexKey) # Write the hexadecimal key to the key file


def getKey(): # Returns the key as bytes
    key_file='data/key.txt' # Set the key file path
    try: # Try to open the key file
        with open(key_file, 'r') as file: # Open the key file
            hexKey = file.read().strip()  # Read the hexadecimal key as a string
            key = bytes.fromhex(hexKey)  # Convert the hexadecimal string to bytes
        return key # Return the key as bytes
    except FileNotFoundError: # If the key file is not found
        print(f"File '{key_file}' not found.") # Print an error message
        return None # Return None
def printKey(sk):
    if sk:
        hexValues = ' '.join([format(byte, '02X') for byte in sk]) # Convert the key to a hexadecimal string
        print(hexValues)
```

getPlaintext(plaintext_file='data/plaintext.txt'):

This function reads the plaintext from the specified file (default: data/plaintext.txt) and returns it as a string.

If the file is not found, it prints an error message and returns None.

```
def getPlaintext(plaintext_file='data/plaintext.txt'):
    try:
        with open(plaintext_file, 'r') as file: # Open the plaintext file
            text = file.read()
        return text
    except FileNotFoundError:
        print(f"File '{plaintext_file}' not found.")
        return None

# Function to write data to a file in hexadecimal format
```

writeHex(data, filename):

This function writes binary data as a hexadecimal string to the specified file.

getHex(filename):

This function reads a hexadecimal string from the specified file and converts it back to bytes.

```
# Function to write data to a file in hexadecimal format
def writeHex(data, filename):
    hex_data = b64encode(data).decode('utf-8') # Convert the data to a hexadecimal string
    with open(filename, 'w') as file:
        file.write(hex_data)


# Function to read data from a file in hexadecimal format
def getHex(filename):
    with open(filename, 'r') as file:
        hex_data = file.read()
    return b64decode(hex_data) # Convert the hexadecimal string to bytes
```

generateKey():

This function generates a random 256-bit AES encryption key (32 bytes) and returns it as bytes.

```
def generateKey():
    # Generate a random 256-bit key
    key = get_random_bytes(32) # Generate a 32-byte (256-bit) random value
    return key
```

writeResult(decrypted_plaintext):

This function takes the decrypted plaintext as input (string) and writes it to a file named data/result.txt.

```python
# Function to write the writeResult to a file
def writeResult(decrypted_plaintext): # decrypted_plaintext is a string
    result_file = 'data/result.txt' # Set the writeResult file path
    with open(result_file, 'w') as file: # Open the writeResult file
        file.write(decrypted_plaintext) # Write the decrypted plaintext to the writeResult file
```

Encryption():

This function represents the encryption process:

Reads the AES encryption key from the key file.

Reads the plaintext from the data/plaintext.txt file.

Generates a random IV.

Encrypts the plaintext using AES-CBC mode.

Writes the ciphertext and IV to their respective files.

```python
# Function to encrypt plaintext
def Encryption():
    print("*ENCRYPTION*")
    secret_key = getKey()
    print("SECRET KEY: ")
    printKey(secret_key)

    f = getPlaintext()
    print("PLAINTEXT: ", f)

    iv = gen_iv()
    print("IV: ", iv)

    enc_cbc = encrypt_aes_cbc(f, secret_key, iv)
    print("CIPHERTEXT: ", enc_cbc)

    # Write ciphertext to file
    writeHex(enc_cbc, ciphertext_file)

    # Write IV to file
    writeHex(iv, iv_file)
```

Decryption():

This function represents the decryption process:

Reads the AES encryption key from the key file.

Reads the ciphertext and IV from their respective files.

Decrypts the ciphertext using AES-CBC mode.

Prints the decrypted plaintext and writes it to data/result.txt.

```
# Function to decrypt ciphertext
def Decryption():
    print("*DECRYPTION*")
    secret_key = getKey()
    print("SECRET KEY: ")
    printKey(secret_key)

    c = getHex(ciphertext_file)
    iv = getHex(iv_file)

    decrypted_plaintext = decrypt_aes_cbc(c, secret_key, iv)
    print("DECRYPTED TEXT: ", decrypted_plaintext)
    writeResult(decrypted_plaintext)

# Function to generate secret key, encrypt, and decrypt
def genKey():
    secret_key = generateKey()
    printKey(secret_key)
    writeKey(secret_key)
```

genKey():

This function generates a random AES encryption key, prints it in hexadecimal format, and writes it to the key file.

```
# Function to generate secret key, encrypt, and decrypt
def genKey():
    secret_key = generateKey()
    printKey(secret_key)
    writeKey(secret_key)
```

OUTPUT:

```
PS C:\Users\muhib\OneDrive - University of Cincinnati\Desktop\Fall 2023\Data Security\aes_m14056283> & C:/Users/muhib/AppData/Local/Programs/Python/Python31
ython.exe "c:/Users/muhib/OneDrive - University of Cincinnati/Desktop/Fall 2023/Data Security/aes_m14056283/src/prj_2.py"
35 5A A4 18 5C EA DC F3 33 E9 E7 02 A5 E0 7C 7F 1E 49 96 50 15 2B AE 31 FA 1A B3 A3 43 C9 4F 03
*ENCRYPTION*
SECRET KEY:
35 5A A4 18 5C EA DC F3 33 E9 E7 02 A5 E0 7C 7F 1E 49 96 50 15 2B AE 31 FA 1A B3 A3 43 C9 4F 03
PLAINTEXT:  Welcome to data security and privacy 2023.
IV:  b'R\xf6\x88<x3\x01\xd6("\xf5]/\xff\xcc\xf0'
CIPHERTEXT:  b'\r\x90T\xdc/\xb1\xc1.\xa3\xbd\x9c\x94V7$\x96\x10f<P)T\xdf\xa6Z\xb1\x19|^\xe1\x18\xd9\x99\x19S\x03S/\x86\xf9v\x93\n\x0b\xc8\x03\x1et'
*DECRYPTION*
SECRET KEY:
35 5A A4 18 5C EA DC F3 33 E9 E7 02 A5 E0 7C 7F 1E 49 96 50 15 2B AE 31 FA 1A B3 A3 43 C9 4F 03
DECRYPTED TEXT:  Welcome to data security and privacy 2023.
```

ciphertext.txt

ciphertext.txt

data > ciphertext.txt

```
1   DZBU3C+xwS6jvZyUVjcklhBmPFApVN+mWrEZfF7hGNmZGVMDUy+G+XaTCgvIAx50
```

iv.txt

iv.txt

data > iv.txt

```
1   UvaIPHgzAdYoIvVdL//M8A==
```

key.txt

key.txt

data > key.txt

```
1   355aa4185ceadcf333e9e702a5e07c7f1e499650152bae31fa1ab3a343c94f03
```

plaintext.txt

result.txt