Lab 3 Report

CSL 6010 - Cyber Security

Rahul Barodia

B20CS047

Part 1)

**** For all the four modes of both AES and DES, I have taken the same message as input so that we can compare the ciphertext that are generated by these eight different codes**

For DES, the key should be 8 bytes. But key according to my name and roll no. i.e. B20CS047RAHUL is far greater than 8 bytes. So using SHA-256 Hash function of hashlib library, which takes the input bytes and produces a fixed-length output of 256 bits (32 bytes). Then taking the first 8 bytes using [:8].

Using DES with mode ECB

```
from Crypto.Cipher import DES
import hashlib
import os

# Generate a 64-bit (8-byte) key from given string B20CS047RAHUL
input_str = "B20CS047RAHUL"
input_bytes = input_str.encode()
key = hashlib.sha256(input_bytes).digest()[:8]

# Create a DES cipher object
iv = os.urandom(8)
```

```
cipher = DES.new(key, DES.MODE_CBC, iv)

message = b'My name is Rahul'
ciphertext = cipher.encrypt(message)

decrypted_msg = cipher.decrypt(ciphertext)

print('Plaintext:', message)
print('Ciphertext:', ciphertext)
print('Decrypted:', decrypted_msg)
```

Output:

```
PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

PS C:\Users\ASUS\OneDrive\Desktop\CS Lab> python -u "c:\Users\ASUS\OneDrive\Desktop\CS Lab\des_cfb.py"
Plaintext: b'My name is Rahul'
Ciphertext: b'\xf5iu\xca\xd61\xc0\x03\xdb\x16\x9d\xf7\x15\x92\x02L'
Decrypted: b'My name is Rahul'
PS C:\Users\ASUS\OneDrive\Desktop\CS Lab>
```

Using DES with mode CBC

```
from Crypto.Cipher import DES
import hashlib
import os

# Generate a 64-bit (8-byte) key from given string B20CS047RAHUL
input_str = "B20CS047RAHUL"
input_bytes = input_str.encode()
key = hashlib.sha256(input_bytes).digest()[:8]
```

```python
iv = os.urandom(8)

# Define a function to pad the message with spaces
def pad_message(message):
    while len(message) % 8 != 0:
        message += b' '
    return message

# Define a function to encrypt a message using DES with CBC mode
def encrypt(message):
    padded_message = pad_message(message)
    cipher = DES.new(key, DES.MODE_CBC, iv=iv)
    ciphertext = cipher.encrypt(padded_message)
    return ciphertext

# Define a function to decrypt a message using DES with CBC mode
def decrypt(ciphertext):
    cipher = DES.new(key, DES.MODE_CBC, iv=iv)
    padded_plaintext = cipher.decrypt(ciphertext)
    plaintext = padded_plaintext.rstrip(b' ')
    return plaintext

# Test the encryption and decryption functions with a sample message
message = b'My name is Rahul'
ciphertext = encrypt(message)
print('Ciphertext:', ciphertext.hex())
plaintext = decrypt(ciphertext)
print('Plaintext:', plaintext)
```

Output:

```
PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

PS C:\Users\ASUS\OneDrive\Desktop\CS Lab> python -u "c:\Users\ASUS\OneDrive\Desktop\CS Lab\des_cbc.py"
Ciphertext: 256a06f1523f88e011b342815a579df0
Plaintext: b'My name is Rahul'
PS C:\Users\ASUS\OneDrive\Desktop\CS Lab>
```

Using DES with mode CFB

```python
from Crypto.Cipher import DES
import hashlib
import os

# Generate a 64-bit (8-byte) key from given string B20CS047RAHUL
input_str = "B20CS047RAHUL"
input_bytes = input_str.encode()
key = hashlib.sha256(input_bytes).digest()[:8]

iv = os.urandom(8)

# Define a function to encrypt a message using DES with CFB mode
def encrypt(message):
    cipher = DES.new(key, DES.MODE_CFB, iv=iv)
    ciphertext = cipher.encrypt(message)
    return ciphertext

# Define a function to decrypt a message using DES with CFB mode
def decrypt(ciphertext):
    cipher = DES.new(key, DES.MODE_CFB, iv=iv)
```

```
    plaintext = cipher.decrypt(ciphertext)
    return plaintext

# Test the encryption and decryption functions with a sample message
message = b'My name is Rahul'
ciphertext = encrypt(message)
print('Ciphertext:', ciphertext.hex())
plaintext = decrypt(ciphertext)
print('Plaintext:', plaintext)
```

Output:

```
PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

PS C:\Users\ASUS\OneDrive\Desktop\CS Lab> python -u "c:\Users\ASUS\OneDrive\Desktop\CS Lab\des_cfb.py"
Ciphertext: c6772ed9e5e33317ab1cbb67945b976b
Plaintext: b'My name is Rahul'
PS C:\Users\ASUS\OneDrive\Desktop\CS Lab>
```

Using DES with mode OFB

```
from Crypto.Cipher import DES
import hashlib
import os

# Generate a 64-bit (8-byte) key from given string B20CS047RAHUL
input_str = "B20CS047RAHUL"
input_bytes = input_str.encode()
key = hashlib.sha256(input_bytes).digest()[:8]

iv = os.urandom(8)
```

```python
# Define a function to encrypt a message using DES with OFB mode
def encrypt(message):
    cipher = DES.new(key, DES.MODE_OFB, iv=iv)
    ciphertext = cipher.encrypt(message)
    return ciphertext

# Define a function to decrypt a message using DES with OFB mode
def decrypt(ciphertext):
    cipher = DES.new(key, DES.MODE_OFB, iv=iv)
    plaintext = cipher.decrypt(ciphertext)
    return plaintext

# Test the encryption and decryption functions with a sample message
message = b'My name is Rahul'
ciphertext = encrypt(message)
print('Ciphertext:', ciphertext.hex())
plaintext = decrypt(ciphertext)
print('Plaintext:', plaintext)
```

Output:

```
PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

PS C:\Users\ASUS\OneDrive\Desktop\CS Lab> python -u "c:\Users\ASUS\OneDrive\Desktop\CS Lab\des_ofb.py"
Ciphertext: 05a38d7871f2c4f42eb0e0f249c599cd
Plaintext: b'My name is Rahul'
PS C:\Users\ASUS\OneDrive\Desktop\CS Lab>
```

**My key B20CS047RAHUL (14 bytes) becomes B20CS047RAHUL00 (16 bytes) for AES. I searched on the internet and found a function which converts any key into 16 bytes(or even larger) by padding with 0's at end. So we do not need to look for the size of our key maually.

Using AES with mode ECB

```
from Crypto.Cipher import AES
import os

# Convert the input string to bytes
input_str = "B20CS047RAHUL"
input_bytes = input_str.encode()

# Pad the input bytes with zeroes to the right until it reaches 16 bytes
key = input_bytes.ljust(16, b'\x00')


# Define a function to pad the message to a multiple of 16 bytes
def pad_message(message):
    padding_length = 16 - (len(message) % 16)
    padding = bytes([padding_length] * padding_length)
    return message + padding

# Define a function to unpad the message after decryption
def unpad_message(message):
    padding_length = message[-1]
    return message[:-padding_length]
```

```python
# Define a function to encrypt a message using AES with ECB mode
def encrypt(message):
    padded_message = pad_message(message)
    cipher = AES.new(key, AES.MODE_ECB)
    ciphertext = cipher.encrypt(padded_message)
    return ciphertext

# Define a function to decrypt a message using AES with ECB mode
def decrypt(ciphertext):
    cipher = AES.new(key, AES.MODE_ECB)
    padded_message = cipher.decrypt(ciphertext)
    plaintext = unpad_message(padded_message)
    return plaintext

# Test the encryption and decryption functions with a sample message
message = b'My name is Rahul'
ciphertext = encrypt(message)
print('Ciphertext:', ciphertext.hex())
plaintext = decrypt(ciphertext)
print('Plaintext:', plaintext)
```

Output:

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS C:\Users\ASUS\OneDrive\Desktop\CS Lab> python -u "c:\Users\ASUS\OneDrive\Desktop\CS Lab\aes_ecb.py"
Ciphertext: fbe3ac0385bc90a1a09b7e436c9f14849d28721f40b66063a777877c23f32b35
Plaintext: b'My name is Rahul'
PS C:\Users\ASUS\OneDrive\Desktop\CS Lab>
```

Using AES with mode CBC

```
from Crypto.Cipher import AES
import os
from Crypto.Util.Padding import pad, unpad

# Convert the input string to bytes
input_str = "B20CS047RAHUL"
input_bytes = input_str.encode()

# Pad the input bytes with zeroes to the right until it reaches 16 bytes
key = input_bytes.ljust(16, b'\x00')


iv = os.urandom(16)

# Define a function to encrypt a message using AES with CBC mode
def encrypt(message):
    cipher = AES.new(key, AES.MODE_CBC, iv=iv)
    ciphertext = cipher.encrypt(pad(message, AES.block_size))
    return ciphertext

# Define a function to decrypt a message using AES with CBC mode
def decrypt(ciphertext):
    cipher = AES.new(key, AES.MODE_CBC, iv=iv)
    plaintext = unpad(cipher.decrypt(ciphertext), AES.block_size)
    return plaintext
```

```python
# Test the encryption and decryption functions with a sample message
message = b'My name is Rahul'
ciphertext = encrypt(message)
print('Ciphertext:', ciphertext.hex())
plaintext = decrypt(ciphertext)
print('Plaintext:', plaintext)
```

Output:

```
PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

PS C:\Users\ASUS\OneDrive\Desktop\CS Lab> python -u "c:\Users\ASUS\OneDrive\Desktop\CS Lab\aes_cbc.py"
Ciphertext: 4f5518d734a49828f00a22713ee1b94bd097e70e20d2b86efb34a477548ba5c0
Plaintext: b'My name is Rahul'
PS C:\Users\ASUS\OneDrive\Desktop\CS Lab>
```

Using AES with mode CFB

```python
from Crypto.Cipher import AES
import os

# Convert the input string to bytes
input_str = "B20CS047RAHUL"
input_bytes = input_str.encode()

# Pad the input bytes with zeroes to the right until it reaches 16 bytes
key = input_bytes.ljust(16, b'\x00')
```

```python
iv = os.urandom(16)

# Define a function to encrypt a message using AES with CFB mode
def encrypt(message):
    cipher = AES.new(key, AES.MODE_CFB, iv=iv)
    ciphertext = cipher.encrypt(message)
    return ciphertext

# Define a function to decrypt a message using AES with CFB mode
def decrypt(ciphertext):
    cipher = AES.new(key, AES.MODE_CFB, iv=iv)
    plaintext = cipher.decrypt(ciphertext)
    return plaintext

# Test the encryption and decryption functions with a sample message
message = b'My name is Rahul'
ciphertext = encrypt(message)
print('Ciphertext:', ciphertext.hex())
plaintext = decrypt(ciphertext)
print('Plaintext:', plaintext)
```

Output:

```
PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

PS C:\Users\ASUS\OneDrive\Desktop\CS Lab> python -u "c:\Users\ASUS\OneDrive\Desktop\CS Lab\aes_cfb.py"
Ciphertext: 9418a4430bc1ddea88de99ea6250d04c
Plaintext: b'My name is Rahul'
PS C:\Users\ASUS\OneDrive\Desktop\CS Lab>
```

Using AES with mode OFB

```
from Crypto.Cipher import AES

import os


# Convert the input string to bytes

input_str = "B20CS047RAHUL"

input_bytes = input_str.encode()


# Pad the input bytes with zeroes to the right until it reaches 16 bytes

key = input_bytes.ljust(16, b'\x00')



iv = os.urandom(16)


# Define a function to encrypt a message using AES with OFB mode

def encrypt(message):

    cipher = AES.new(key, AES.MODE_OFB, iv=iv)

    ciphertext = cipher.encrypt(message)

    return ciphertext


# Define a function to decrypt a message using AES with OFB mode

def decrypt(ciphertext):

    cipher = AES.new(key, AES.MODE_OFB, iv=iv)

    plaintext = cipher.decrypt(ciphertext)

    return plaintext


# Test the encryption and decryption functions with a sample message

message = b'My name is Rahul'
```

```
ciphertext = encrypt(message)

print('Ciphertext:', ciphertext.hex())

plaintext = decrypt(ciphertext)

print('Plaintext:', plaintext)
```

Output:

```
PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

PS C:\Users\ASUS\OneDrive\Desktop\CS Lab> python -u "c:\Users\ASUS\OneDrive\Desktop\CS Lab\aes_ofb.py"
Ciphertext: 57a1f99323fb5831ed2f6610f5e1e5a0
Plaintext: b'My name is Rahul'
PS C:\Users\ASUS\OneDrive\Desktop\CS Lab>
```

Part 2)

In Diffie-Hellman key exchange, both the client and server decides two values. One is a large prime number, and another one is its primitive root. For example, 42809 and 3 , etc.

Then both server and client generate secret key through mathematical calculations, which results in the exact same secret key, which can be used further for encryption and decryption.

server.py

```
import socket


prime = 42809

r = 3


sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)


ser_add = ('localhost', 5829)

#print(f"Starting up on {ser_add[0]} port {ser_add[1]}")

sock.bind(ser_add)


sock.listen(1)


while True:

   print("Waiting for a connection...")

   conc, client_address = sock.accept()


   try:

       data = conc.recv(1024)

       M = int(data.decode())
```

```
        #print(f"Received A = {A} from the client")


        random = 12345


        B = pow(r, random, prime)

        #print(f"Sending B = {B} to the client")

        conc.sendall(str(B).encode())


        s = pow(M, random, prime)

        print(f"Shared secret key: {s}")


        data = conc.recv(1024)

        enc_msg = data.decode()

        print(f"The received encrypted message is: {enc_msg}")


        msg = ""

        for char in enc_msg:

            decrypted_char = chr(ord(char) ^ s)   # XOR the character with the shared
secret key

            msg += decrypted_char

        print(f"The decrypted message is : {msg}")


    finally:

        conc.close()
```

client.py

```
import socket
import random

prime = 42809
r = 3

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server_address = ('localhost', 5829)
print(f"Connecting to {server_address[0]} port {server_address[1]}")
sock.connect(server_address)

random = random.randint(1, prime-1)

M = pow(r, random, prime)
#print(f"Sending A = {A} to the server")
sock.sendall(str(M).encode())

data = sock.recv(1024)
B = int(data.decode())
#print(f"Received B = {B} from the server")

s = pow(B, random, prime)
print(f"Shared secret key: {s}")

message = "My name is Rahul"
encrypted_message = ""
```

```python
for char in message:
    encrypted_char = chr(ord(char) ^ s)  # XOR the character with the shared secret key
    encrypted_message += encrypted_char
print(f"Encrypted message: {encrypted_message}")


sock.sendall(encrypted_message.encode())


sock.close()
```

Output :

```
PROBLEMS  1    OUTPUT    TERMINAL    DEBUG CONSOLE

○ rahulbarodia@Rahuls-MacBook-Air Lab3 % python3 q2_server.py
  Waiting for a connection...
  █
```

```
PROBLEMS  1    OUTPUT    TERMINAL    DEBUG CONSOLE

● rahulbarodia@Rahuls-MacBook-Air Lab3 % python3 q2_client.py
  Connecting to localhost port 5829
  Shared secret key: 19722
  Encrypted message: 䵂䵯䵦䵦䵥䴦䴪䵹䵯䵸䵼䵯䵸䴫
○ rahulbarodia@Rahuls-MacBook-Air Lab3 % █
```

We can see that the shared secret key generated is 19722. The plain text “ My name is Rahul ” is encrypted into some cipher text 軁軵躬転軭軡軩躬軥軿躬軞軭軤軹軠

```
PROBLEMS  1    OUTPUT    TERMINAL    DEBUG CONSOLE

○ rahulbarodia@Rahuls-MacBook-Air Lab3 % python3 q2_server.py
  Waiting for a connection...
  Shared secret key: 36492
  The received encrypted message is: 軁軵躬転軭軡軩躬軥軿躬軞軭軤軹軠
  The decrypted message is : My name is Rahul
  Waiting for a connection...
  ▯
```

The cipher text is decrypted back in the last using the shared secret key.

Part 3)

Using ECB mode of AES

```
pip install pycryptodome
pip install numpy

from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from PIL import Image
import numpy as np

# Define the AES key and block size
key = b'secret_key16byte'
block_size = 16

# Open the image and convert it to a numpy array
img = Image.open('Lab3_image.jpg')
img_array = np.array(img)

# Convert the image array to bytes
img_bytes = img_array.tobytes()

# Pad the image bytes to match the block size
img_bytes_padded = pad(img_bytes, block_size)

# Create an AES cipher object and encrypt the padded image bytes
```

```
cipher = AES.new(key, AES.MODE_ECB)
encrypted_bytes = cipher.encrypt(img_bytes_padded)

# Save the encrypted image to a file
encrypted_img = Image.frombytes(img.mode, img.size, encrypted_bytes)
encrypted_img.save('encrypted_image.jpg')

# Decrypt the encrypted image bytes
decrypted_bytes = cipher.decrypt(encrypted_bytes)

# Unpad the decrypted image bytes and convert them back to a numpy array
decrypted_bytes_unpadded = unpad(decrypted_bytes, block_size)
decrypted_img_array = np.frombuffer(decrypted_bytes_unpadded,
dtype=np.uint8).reshape(img_array.shape)

# Save the decrypted image to a file
decrypted_img = Image.fromarray(decrypted_img_array)
decrypted_img.save('decrypted_image.jpg')

# Compare the original and decrypted image arrays
if np.array_equal(img_array, decrypted_img_array):
   print('The encrypted and decrypted images match')
else:
   print('The encrypted and decrypted images do not match')

# Print the encrypted and decrypted images to the console
print('Encrypted image:')
encrypted_img.show()
```

```
print('Decrypted image:')

decrypted_img.show()
```

Original Image:

Encrypted Image:

Decrypted Image :

Using CBF mode of AES

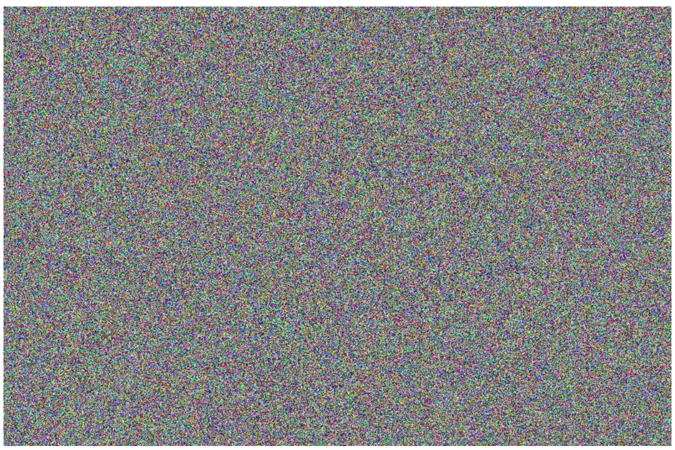
```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np

# Define the AES key and block size
key = b'secretkey_16byte'
block_size = 16

# Open the image and convert it to a numpy array
img = Image.open('/content/Lab3_image.jpg')
img_array = np.array(img)

# Convert the image array to bytes
img_bytes = img_array.tobytes()

# Pad the image bytes to match the block size
img_bytes_padded = pad(img_bytes, block_size)

# Generate an initialization vector (IV)
iv = b'ThisisanIVvector'

# Create an AES cipher object and encrypt the padded image bytes using CFB
mode
```

```
cipher = AES.new(key, AES.MODE_CFB, iv)
encrypted_bytes = cipher.encrypt(img_bytes_padded)

# Save the encrypted image to a file
encrypted_img = Image.frombytes(img.mode, img.size, encrypted_bytes)
encrypted_img.save('encrypted_image_cfb.jpg')

# Decrypt the encrypted image bytes using CFB mode
cipher = AES.new(key, AES.MODE_CFB, iv)
decrypted_bytes = cipher.decrypt(encrypted_bytes)

# Unpad the decrypted image bytes and convert them back to a numpy array
decrypted_bytes_unpadded = unpad(decrypted_bytes, block_size)
decrypted_img_array = np.frombuffer(decrypted_bytes_unpadded,
dtype=np.uint8).reshape(img_array.shape)

# Save the decrypted image to a file
decrypted_img = Image.fromarray(decrypted_img_array)
decrypted_img.save('decrypted_image_cfb.jpg')

# Compare the original and decrypted image arrays
if np.array_equal(img_array, decrypted_img_array):
   print('The encrypted and decrypted images match')
else:
   print('The encrypted and decrypted images do not match')

# Display the original, encrypted, and decrypted images
plt.imshow(encrypted_img)
```

```
plt.title('Encrypted image')


plt.show()

plt.imshow(decrypted_img)

plt.title('Decrypted image')


plt.show()
```

Output :

Encrypted Image:



Decrypted Image:

Now comparing the above two encrypted images. I have used **MSE (Mean squared error)** for image comparison. MSE measures the average squared difference between the pixels of two images.

Here is the code for MSE

```
import cv2
import numpy as np

# Load the images
img1 = cv2.imread('encrypted_image.jpg')
img2 = cv2.imread('encrypted_image_cfb.jpg')

# Convert to grayscale
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

# Calculate MSE
mse = np.mean((gray1 - gray2) ** 2)

print("MSE:", mse)
```

Output :

```
MSE: 105.45710400763359
```

The higher the MSE , the greater the difference between the two images.

Here the pixel value range of both the encrypted images is 0-255. So an MSE of 100.058 may be considered low, as the maximum possible MSE between two images in this range is 65,025.