2 Assignment – Block Ciphers

Naomi Rojo

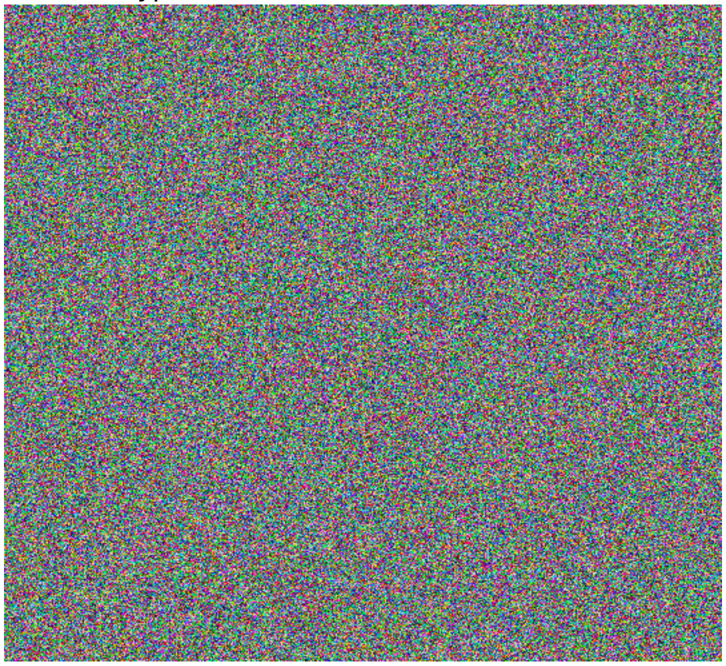
Task 1 – Modes of Operation:

For Task 1, I implemented an Electronic Codebook Mode (ECB) and Cipher Block Chaining Mode (CBC) encryption to encrypt BMP-1 (cp-logo.bmp). The encryption process involved generating a random key (and IV for CBC), preserving and re-appending BMP headers, and applying PKCS#7 packing. The encryption process consisted of reading in the BMP image file and extracting the plaintext, generating a random key and IV for both ECB and CBC modes, padding the plaintext using PKCS#7 padding, encrypting the padded plaintext using custom ECB and CBC implementations, preserving, and concatenating the BMP image header with the encrypted data, and then writing the resulting ciphertext to separate files for ECB and CBC. The ECB Ciphertext resembles the original image but with changed colors. The CBC Ciphertext appears as a randomized pattern of colors (bunch of colored pixels).

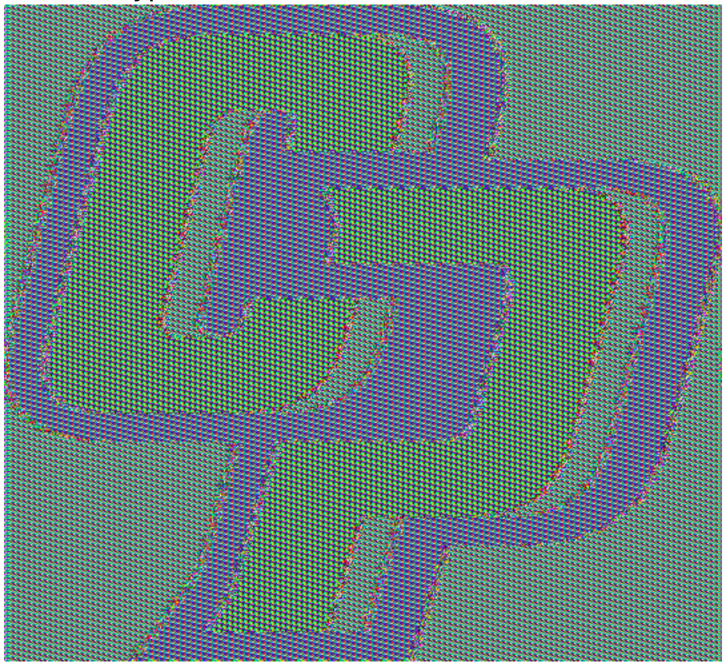
Code:

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

def pad_pkcs7(data, block_size):
    padding_size = block_size - len(data) % block_size
    padding = bytes([padding_size]) * padding_size
    return data + padding

def generate_key_and_iv():
    key = get_random_bytes(16)
    iv = get_random_bytes(16)
    return key, iv

def encrypt_ecb(data, key):
    cipher = AES.new(key, AES.MODE_ECB)
    ciphertext = b""

    for i in range(0, len(data), 16):
        block = data[i:i+16]
        encrypted_block = cipher.encrypt(block)
        ciphertext += encrypted_block

    return ciphertext

def encrypt_cbc(data, key, iv):
    cipher = AES.new(key, AES.MODE_CBC, iv)
    ciphertext = b""

    for i in range(0, len(data), 16):
        block = data[i:i+16]
        encrypted_block = cipher.encrypt(block)
        ciphertext += encrypted_block

    return ciphertext

header_size = 54
text_file = "cp-logo.bmp"
with open(text_file, "rb") as file:
    header = file.read(header_size)
    plaintext = file.read()

key, iv = generate_key_and_iv()

ecb_ciphertext = encrypt_ecb(pad_pkcs7(plaintext, 16), key)

cbc_ciphertext = encrypt_cbc(pad_pkcs7(plaintext, 16), key, iv)

with open("ecb_ciphertext.bmp", "wb") as file:
    file.write(header + ecb_ciphertext)

with open("cbc_ciphertext.bmp", "wb") as file:
    file.write(header + cbc_ciphertext)
```

CBC Encrypted BMP:



ECB Encrypted BMP:



Task 2 – Limits of Confidentiality:

In Task 2, I implemented a padding oracle attack against a system that uses AES-CBC encryption with PKCS#7 padding. I modified the Initialization Vector (IV) and observed the padding validity.

Code:

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
import urllib.parse

key, iv = get_random_bytes(16), get_random_bytes(16)

def pad_pkcs7(data, block_size):
    padding_size = block_size - len(data) % block_size
    padding = bytes([padding_size]) * padding_size
    return data + padding

def submit(user_input, modified_iv):
    data = f"userid=456;userdata={user_input};session-id=31337"
    data = urllib.parse.quote(data)
    padded_data = pad_pkcs7(data.encode('utf-8'), 16)
    cipher = AES.new(key, AES.MODE_CBC, modified_iv)
    ciphertext = cipher.encrypt(padded_data)
    return modified_iv + ciphertext
```

```python
def verify(ciphertext, modified_iv):
    cipher = AES.new(key, AES.MODE_CBC, modified_iv)
    decrypted_data = cipher.decrypt(ciphertext)
    #print("Decrypted Data (Hex):", decrypted_data.hex())

    try:
        decrypted_data_utf8 = decrypted_data.decode('utf-8')
        print("Decrypted Data (UTF-8):", decrypted_data_utf8)
    except UnicodeDecodeError as e:
        print("Error decoding:", e)
        return False

    if ";admin=true;" in decrypted_data_utf8:
        return True
    else:
        return False

#example
user_input = "You're the man now, dog"
modified_iv = bytearray(iv)
modified_iv[15] ^= 1

ciphertext = submit(user_input, modified_iv)

#print("Original Ciphertext:", ciphertext)

result = verify(ciphertext[16:], modified_iv)
print("Verification Result:", result)
```

Task 3 – Performance Comparisons:

For Task 3, I measured the performance difference between public and symmetric key algorithms using OpenSSL's sped test. I then plotted these results using the matplotlib library.

Code:

```python
import matplotlib.pyplot as plt

aes_key_sizes = [128, 192, 256]
aes_throughput = [323230.52, 277175.00, 240674.21]

rsa_key_sizes = [512, 1024, 2048, 4096]
rsa_sign_throughput = [7143.9, 2174.3, 540.9, 101.1]
rsa_verify_throughput = [235224.4, 111199.9, 38364.1, 10074.7]
```

```
plt.figure(figsize=(8, 5))
plt.plot(aes_key_sizes, aes_throughput, marker='o', label='AES Throughput')
plt.title('AES Throughput vs. Key Size')
plt.xlabel('AES Key Size (bits)')
plt.ylabel('Throughput (KB per second)')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(8, 5))
plt.plot(rsa_key_sizes, rsa_sign_throughput, marker='o', label='RSA Signing
Throughput')
plt.title('RSA Signing Throughput vs. Key Size')
plt.xlabel('RSA Key Size (bits)')
plt.ylabel('Throughput (operations per second)')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(8, 5))
plt.plot(rsa_key_sizes, rsa_verify_throughput, marker='o', label='RSA Verification
Throughput')
plt.title('RSA Verification Throughput vs. Key Size')
plt.xlabel('RSA Key Size (bits)')
plt.ylabel('Throughput (operations per second)')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 1:

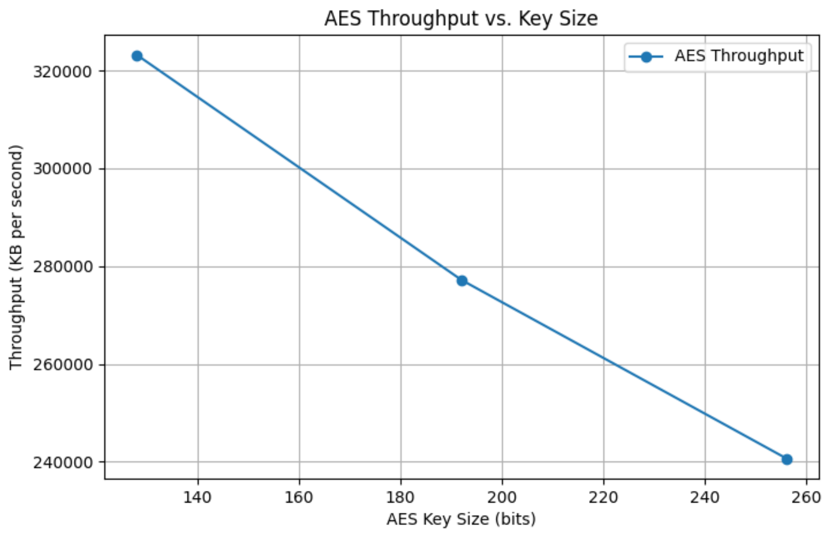


Figure 2:

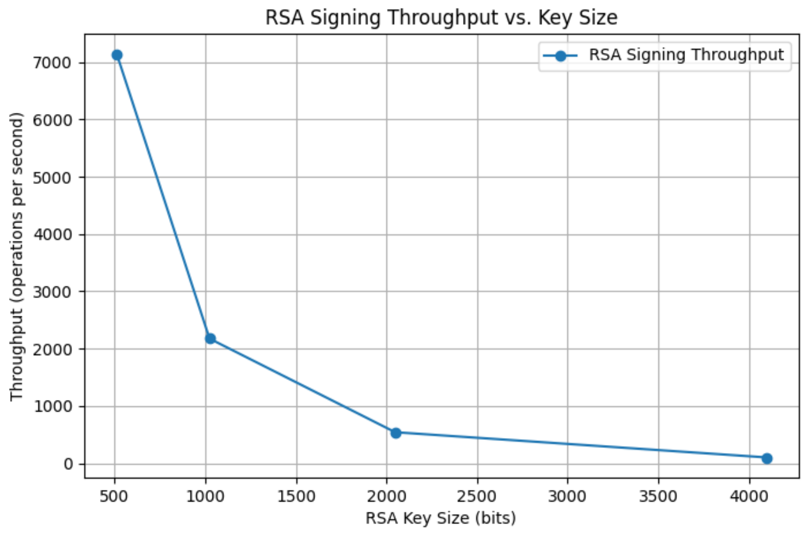
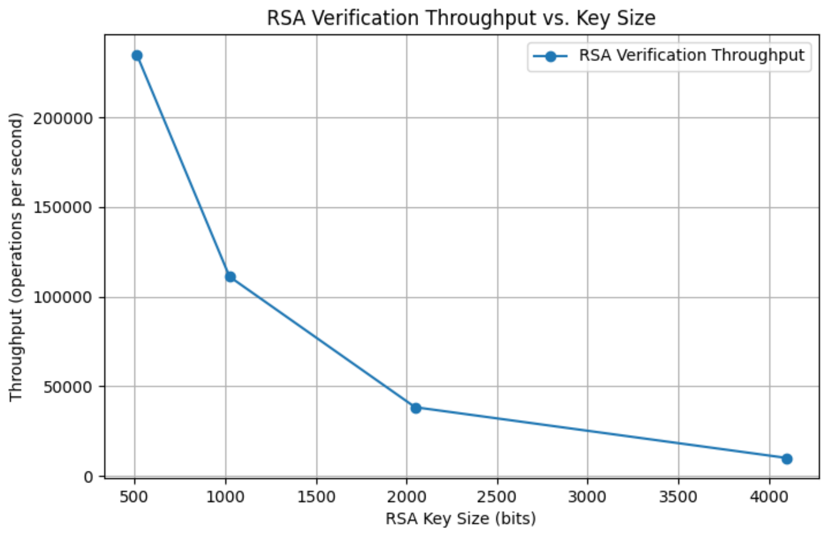


Figure 3:



Questions:

1: Looking at the two ciphertexts that I have made it is obvious that the CBC mode encrypted the BMP better than the ECB. Looking at the CBC mode the BMP is just a bunch of randomized colored pixels while the ECB mode was the exact same logo, but the pixels were different colors. The ECB mode BMP kind of looked like it was stripped of the first few layers but still intact. The ECB encrypts each block independently, leading to a deterministic pattern. The CBC XORs each block with the previous ciphertext block before encryption, providing better security compared to ECB.

2: This attack is possible because of the nature of CBC mode. By modifying a bit in the ciphertext block, it affects the corresponding plaintext block and the next one. This attack involves manipulating the ciphertext to make the verify() function true for a crafted input. To prevent these attacks, schemes need to use authenticated encryption modes or message authentication codes (MACs) to ensure the integrity of the ciphertext.

3: Looking at the results of AES and RSA we can see that all three graphs have an exponential decrease. From the AES graph we can see that the larger key sizes generally result in lower throughput, but the trade-off is increased security. From the RSA graphs we can see that the throughput decreases as the key size increases, reflecting the computational cost associated with larger key sizes. AES is generally faster than RSA, especially for larger key sizes.