Block Cipher Assignment

Caleb Dennis, Riley Mete,

## Task 1: Modes of Operation

The EBC mode of encryption results in an encrypted BMP with easily discernable patterns. This is because BMPs have lots of similar data, so when the data is broken down into bytes adjacent data is likely to encrypt to the same bytes! In the CBC mode the order bytes are encrypted in effects the final output so even a file like a BMP, with lots of similar data, results in a random looking output.

**Cal Poly Logo Encrypted with EBC method:**

A picture containing text

Description automatically generated

**Cal Poly Logo Encrypted with CBC method:** Background pattern

Description automatically generated with low confidence

## Task 1 Code:

from Crypto.Cipher import AES

import sys

from Crypto.Random import get\_random\_bytes

def main(infile):

#infile = sys.argv[1]

with open(infile, 'rb') as input:

plaintext = input.read()

# Preserve header and get bytes for the image

header = plaintext[:54]

image = plaintext[54:]

# Break up image into 16 byte block

blocks = get\_blocks(image)

key = get\_random\_bytes(16)

iv = get\_random\_bytes(16)

# Encrypt with ebc and cbc method

ebc\_out\_bytes = EBC\_encrypt(blocks)

cbc\_out\_bytes = CBC\_encrypt(blocks, key, iv)

encrypted\_blocks = get\_blocks(cbc\_out\_bytes)

# Decrypt with cbc method

cbc\_decrypt = CBC\_decrypt(encrypted\_blocks, key, iv)

# Write ebc encrypted file

outfile\_ebc = infile.split(".")[0] + "\_ebc.bmp"

write\_file(outfile\_ebc, header, ebc\_out\_bytes)

# Write cbc encrypted file

outfile\_cbc = infile.split(".")[0] + "\_cbc.bmp"

write\_file(outfile\_cbc, header, cbc\_out\_bytes)

# Write cbc decrypted file

outfile\_cbc\_decypted = infile.split(".")[0] + "\_decrypted\_cbc.bmp"

write\_file(outfile\_cbc\_decypted, header, cbc\_decrypt)

def pkcs7\_pad(to\_pad, m):

if len(to\_pad) % m != 0:

pad\_with = (m \* (len(to\_pad) // m + 1)) - len(to\_pad)

to\_pad = to\_pad + bytes([pad\_with] \* pad\_with)

return to\_pad

def get\_blocks(file\_as\_bytes):

file\_as\_bytes = pkcs7\_pad(file\_as\_bytes, 16)

blocks = [bytes(file\_as\_bytes[x - 16: x]) for x in range(16, len(file\_as\_bytes) + 1, 16)]

return blocks

def EBC\_encrypt(blocks):

ciphertext = b""

random\_key = get\_random\_bytes(16)

ecb\_cipher = AES.new(random\_key, AES.MODE\_ECB)

# loop through blocks and encrypt each block independently

for block in blocks:

ciphertext += ecb\_cipher.encrypt(block)

return ciphertext

def xor\_bytes(a, b):

c = bytearray()

for idx in range(16):

c.append(a[idx] ^ b[idx])

return c

def CBC\_encrypt(blocks, key, iv):

cipher = AES.new(key, AES.MODE\_ECB)

encrypted = bytearray()

for bytes\_16 in blocks:

intermediate = xor\_bytes(iv, bytes\_16)

iv = cipher.encrypt(intermediate)

encrypted += iv

return encrypted

def CBC\_decrypt(blocks, key, iv):

cipher = AES.new(key, AES.MODE\_ECB)

decrypted = bytearray()

for bytes\_16 in blocks:

intermediate = cipher.decrypt(bytes\_16)

decrypted += xor\_bytes(iv, intermediate)

iv = bytes\_16

return decrypted

def write\_file(outfile, header, ciphertext):

# Write to output file, for this we need the original bmp header

# and the encrypted ciphertext

with open(outfile, "wb") as output:

output.write(header+ciphertext)

if \_\_name\_\_ == "\_\_main\_\_":

main(sys.argv[1])

## Task 2: Limits of confidentiality

In task 2 we write a program which emulates some of the basic tasks that a web browser performs, decrypting HTTP information for a web page. The CBC mode of operation for AES is a good choice for this task because the HTTP information can be long. However the CBC byte flip attack makes this mode of operation vulnerable. An adversary can modify the encrypted cipher text to produce whatever plaintext they desire. In this task we modified the plaintext to include “;admin=true;”.

This attack was only possible because the encrypted text could be read and modified. Therefore, to prevent this type of attack secure connections must be established such that the receiver knows the information they are receiving has not been modified.

### User input and Terminal Output

Input: "some non-malicious user input, that has at least 256 bytes"

Output:

bytearray(b'NMR\x14Q\xafVL\x7fD\x1a\xb5p\x9a\xcf\x85;admin=true;\_\_\_\_hat has at least 256 bytes\x06\x06\x06\x06\x06\x06')

True

## Task 2 code:

import AES\_modes

from Crypto.Random import get\_random\_bytes

import sys

def main(*user\_input*):

    key = get\_random\_bytes(16)

    iv = get\_random\_bytes(16)

    # when the input is too short there is only one block, so the attack isn't possible

    user\_input = "some non-malicious user input, that has at least 256 bytes"

    user\_input\_bytes = bytes(user\_input, "ascii")

    encrypted\_bytes = submit(user\_input, key, iv)

    #

    #CBC byte flipping attack

    #

    we\_want = bytes(";admin=true;\_\_\_\_", "ascii")

    decoded\_not\_xored = AES\_modes.xor\_bytes(encrypted\_bytes[:16], user\_input\_bytes[16:32]) #second block so we can modify the first

    malicious\_cipher\_text = AES\_modes.xor\_bytes(decoded\_not\_xored, we\_want)

    #do modification

    encrypted\_bytes = malicious\_cipher\_text + encrypted\_bytes[16:]

    print(verify(encrypted\_bytes, key, iv))

def submit(*usr\_input*, *key*, *iv*):

    colon = bytes("%3b", "ascii")

    equals = bytes("%3d", "ascii")

    input\_bytes = bytearray()

    for char\_byte in usr\_input:

        if char\_byte == ';':

            input\_bytes += colon

        elif char\_byte == '=':

            input\_bytes += equals

        else:

            input\_bytes += bytes(char\_byte, "ascii")

    input\_blocks = AES\_modes.get\_blocks(input\_bytes)

    encrypted = AES\_modes.CBC\_encrypt(input\_blocks, key, iv)

    return encrypted

def verify(*input\_bytes*, *key*, *iv*):

    input\_blocks = AES\_modes.get\_blocks(input\_bytes)

    decrypted = AES\_modes.CBC\_decrypt(input\_blocks, key, iv)

    print(decrypted)

    look\_for = bytes(";admin=true;", "ascii")

    for idx in range(len(look\_for) - 1, len(decrypted)):

        if decrypted[idx - len(look\_for):idx] == look\_for:

            return True

    return False

if \_\_name\_\_ == "\_\_main\_\_":

    main(sys.argv[1])

## Task 3: Performance ComparisonChart Description automatically generatedA picture containing timeline Description automatically generated

Chart, line chart

Description automatically generated

**Questions:**

1. After viewing the resulting ciphertexts, it is obvious to see that EBC method is a lot less efficient of an encryption compared to the CBC method. In the resulting image using the EBC encryption method, we can get a general idea of what the actual plaintext image is. This is because the EBC method encrypts each block separately and the encryptions are not chained together. In the resulting image using the CBC encryption method, the image is well encrypted, and we are not able to tell what the actual plaintext image is supposed to be. This is because the CBC encryption method chains together each encrypted block resulting in well encrypted cyphertext image file as shown above.
2. This attack was only possible because the encrypted text could be read and modified. Therefore, to prevent this type of attack secure connections must be established such that the receiver knows the information they are receiving has not been modified.
3. For the AES block size vs throughput, the graphs show us that the overall throughput will increase greatly when we increase the initial size of the block. For example, a block size of 1024 will have a much better throughput than a block size of 16. However, after the block size reaches 1024 the throughput rate appears to remain stagnant and a block size of 1024 vs 8096 will result in a similar throughput. For the RSA block size vs throughput, the graphs show us that the throughput drastically diminishes whenever the block size is increased.