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.

## 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 Comparison