Comparing CBC and ECB

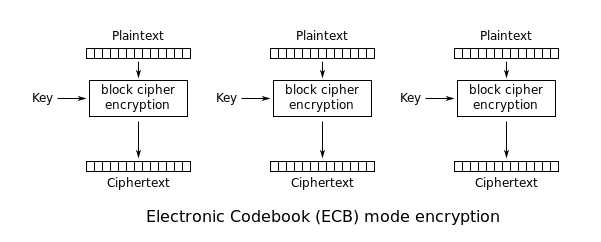
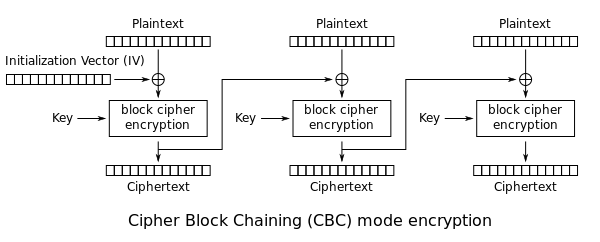
Mark Nakamae

Cal Poly SLO

August 2020

Description

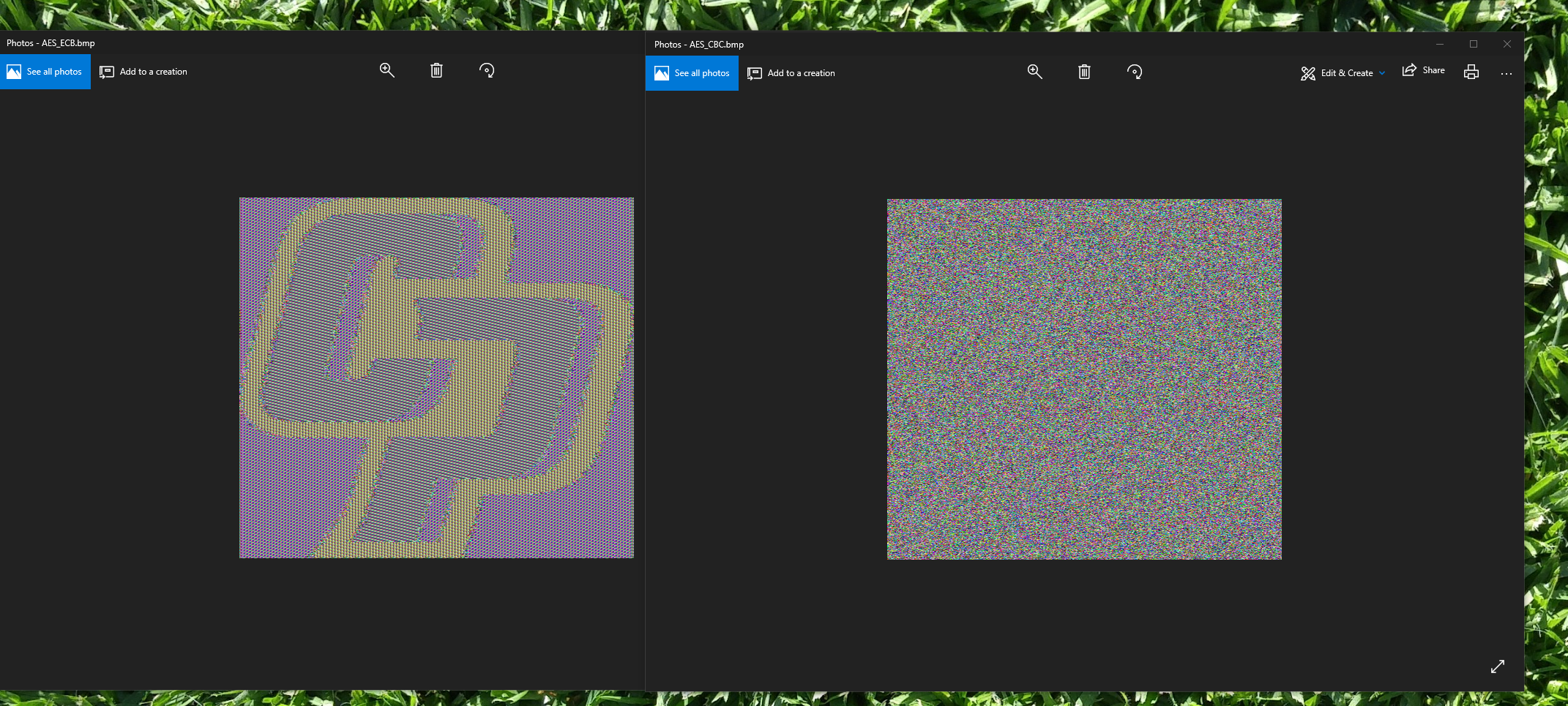
The goal of this project was to further understand symmetric key cryptography by further exploring the Electronic Codebook (ECB) and Cipher Block Chaining (CBC) methods described [here](https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation). The results showed that ECB does not handle encrypting patterns well since all blocks are encrypted with the same key. For this reason, the output .bmp file still had defining features similar to the input .bmp file. On the other hand, CBC solves this issue by utilizing the previous block’s encryption result as the key for the next block’s encryption. As a result, no discernible patterns can be seen in the output .bmp file.

****

Modes of Operation

*Once again, view the resulting ciphertexts. What do you observe? Are you able to derive any useful information about from either of the encrypted images? What are the causes for what you observe?*

We observe that the image that is encrypted with ECB is still very recognizable. The other method CBC is much better at obscuring the original image. Some useful information can be derived from the image encrypted with AES128 ECB mode encryption like the shape of the original image and color boundaries/outlines. The causes for what we observe are simply that the ECB method of encryption re-uses the same key over and over again, whereas the CBC method utilized the previously encrypted data as the new key for the next block of data to be encrypted, increasing randomness.



ECB left and CBC right

Helper Functions

# Generic wrapper to f.write

**def** writeToFile(fileName, transactionString):

f = open(fileName, "a") # Open File

f.write(transactionString)

f.close()

# Generic wrapper to read file contents

**def** readFile(fileNameString):

**with** open(fileNameString, encoding="utf8") **as** fp:

**return** fp.read()

# Generic wrapper to read file contents as binary

**def** readFileBinary(fileNameString):

**with** open(fileNameString, 'rb') **as** fp:

**return** fp.read()

# Generic wrapper to read specified number of bytes as binary

**def** readFileBytes(fileNameString, numBytes):

**with** open (fileNameString, 'rb') **as** fp:

**return** fp.read(numBytes)

# Generic wrapper to write bytes to a file

**def** writeToFileBytes(fileName, payload):

f = open(fileName, "wb") # Open File

f.write(payload)

f.close()

# Returns a random number

**def** getRandomNumber():

**return** os.urandom(1)

# Returns a random value of specified size in bytes

**def** getRandomBytes(size):

randomBytes = os.urandom(size)

**return** randomBytes

# Generic wrapper for AES-128 ECB encrypting

**def** ecbEncrypt(fileContents, key = None):

**global** globalKeyVariable

fileContentsBytes = bytes(fileContents, 'utf-8')

lengthFileContents = len(fileContentsBytes)

**if** key == None:

AES\_KEY = getRandomBytes(16)

**else**:

AES\_KEY = key

ECB\_CIPHER = AES.new(AES\_KEY, AES.MODE\_ECB)

encrypt = ECB\_CIPHER.encrypt(fileContentsBytes)

encrypt = encrypt[:lengthFileContents] # remove padded bytes before writing to file

**return** encrypt

# Generic wrapper for AES-128 CBC encrypting

**def** cbcEncrypt(fileContents, key = None):

**global** globalKeyVariable

fileContentsBytes = bytes(fileContents, 'utf-8')

lengthFileContents = len(fileContentsBytes)

**if** key == None:

AES\_KEY = getRandomBytes(16)

**else**:

AES\_KEY = key

CBC\_CIPHER = AES.new(AES\_KEY, AES.MODE\_CBC)

encrypt = CBC\_CIPHER.encrypt(fileContentsBytes)

encrypt = encrypt[:lengthFileContents] # remove padded bytes before writing to file

**return** encrypt

ECB Main

**def** main():

AES\_KEY = getRandomBytes(16)

ECB\_CIPHER = AES.new(AES\_KEY, AES.MODE\_ECB)

# read file contents

fileToBeEncrypted = "cp-logo.bmp"

fileContents = readFileBinary(fileToBeEncrypted)

bmpFileHeader = readFileBytes(fileToBeEncrypted, 54)

lengthFileContents = len(fileContents)

# pad bytes before encrypting

fileContents += getRandomBytes(16 - (lengthFileContents % 16))

# change this to AES stuffs

encrypt = ECB\_CIPHER.encrypt(fileContents)

# remove padded bytes before writing to file

encrypt = encrypt[:lengthFileContents]

encryptedFileName = fileToBeEncrypted + "\_ECB.bmp"

writeToFileBytes(encryptedFileName, encrypt)

# now replace .bmp header

f = open(encryptedFileName, 'rb')

s = f.read()

f.close()

bmpFileHeaderEncrypted = readFileBytes(encryptedFileName, 54)

s = s.replace(bmpFileHeaderEncrypted, bmpFileHeader)

writeToFileBytes(encryptedFileName, s)

CBC Main

**def** main():

AES\_KEY = getRandomBytes(16)

CBC\_CIPHER = AES.new(AES\_KEY, AES.MODE\_CBC)

# read file contents

fileToBeEncrypted = "cp-logo.bmp"

fileContents = readFileBinary(fileToBeEncrypted)

bmpFileHeader = readFileBytes(fileToBeEncrypted, 54)

lengthFileContents = len(fileContents)

# pad bytes before encrypting

fileContents += getRandomBytes(16 - (lengthFileContents % 16))

# encrypt padded file contents

encrypt = CBC\_CIPHER.encrypt(fileContents)

# remove padded bytes before writing to file

encrypt = encrypt[:lengthFileContents]

encryptedFileName = fileToBeEncrypted + "\_CBC.bmp"

writeToFileBytes(encryptedFileName, encrypt)

# now replace .bmp header

f = open(encryptedFileName, 'rb')

s = f.read()

f.close()

bmpFileHeaderEncrypted = readFileBytes(encryptedFileName, 54)

s = s.replace(bmpFileHeaderEncrypted, bmpFileHeader)

writeToFileBytes(encryptedFileName, s)