|  |  |  |
| --- | --- | --- |
| **ipn** | **INSTITUTO POLITÉCNICO NACIONAL**  **ESCUELA SUPERIOR DE CÓMPUTO** |  |

**Cryptography**

**“Advanced Encryption Standard (AES)”**

Abstract

Implementation in Python of AES Cipher using 5 different modes of operation for two different bmp images, one with just 2 colors and the other with too much variation of color pixel by pixel.

**By:**

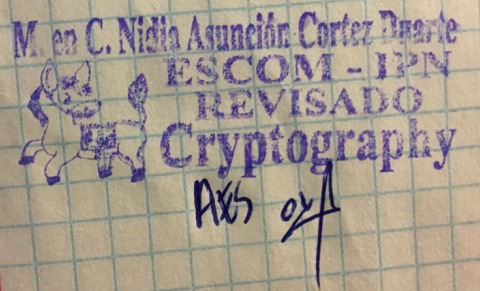
**Garduño Velázquez Alan**

**Romero Gamarra Joel Mauricio**

Professor:

MSc. NIDIA ASUNCIÓN CORTEZ DUARTE

November 2017



**Index**

Contenido

[Introduction: 1](#_Toc499559985)

[Literature review: 3](#_Toc499559986)

[Software (libraries, packages, tools): 8](#_Toc499559987)

[Procedure: 8](#_Toc499559988)

[Results 12](#_Toc499559989)

[Discussion: 12](#_Toc499559990)

[Conclusions: 12](#_Toc499559991)

[References: 12](#_Toc499559992)

[Code 13](#_Toc499559993)

# Introduction:

AES Cipher is a symmetric block cipher capable of encrypt blocks of 128 bits (16 bytes or characters), it has a security of 128 bits because the key to encrypt/decrypt the information has 128 bits of length. This cipher, receive arrays of 16 bytes (it doesn’t matter if we are talking about plaintext, images, wav files, etc.). [1]

It is formed by 4 different functions in regular rounds (9) and 3 of those 4 functions in the last round, on Figure 1 is shown the general process of AES.

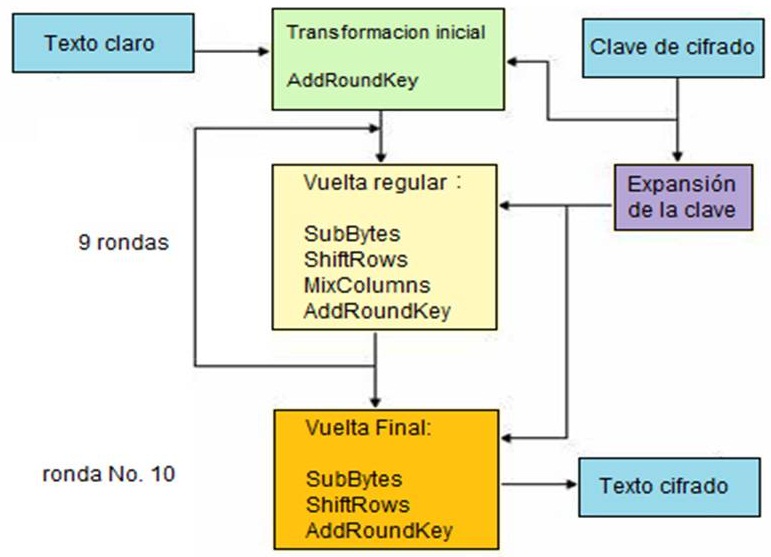


Figure 1. General Diagram of AES

In this cipher, the plaintext is represented by a matrix as such as the key (it has its own process of expansion to select a new key for the next rounds).

As we can observe, the 4 different functions are:

* **AddRoundKey:** XOR operation byte by byte between key and plaintext.
* **SubBytes:** Substitute each byte of the matrix following the S-box.
* **ShiftRows:** Make a left-rotation of n – bytes (with n from 0 to number of rows – 1).
* **MixColumns:** Multiply the matrix obtained by the first 3 functions with the key in GF (28).

Although these 4 functions seems to be very simple, just 3 of them really are, because MixColumns function is very hard if we do it manually, because a multiplication of 2 matrixes in GF (28) is not too simple. This procedure will be explained better on “Literature Review” section.

This cipher works with bytes (I already mentioned that) written in hexadeximal, that’s why GF (28), each byte is composed by 8 bits, but we need only 4 bits to represent from 0 to F and then, we have to have 2 *characters* on each position of the matrix.

On Figure 2, we can observe AddRoundKey function.

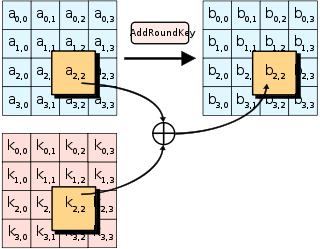


Figure 2. AddRoundKey Function

As you can see, we made and XOR between a2,2 and k2,2 to obtain b2,2. The next function (SubBytes) is shown on Figure 3.

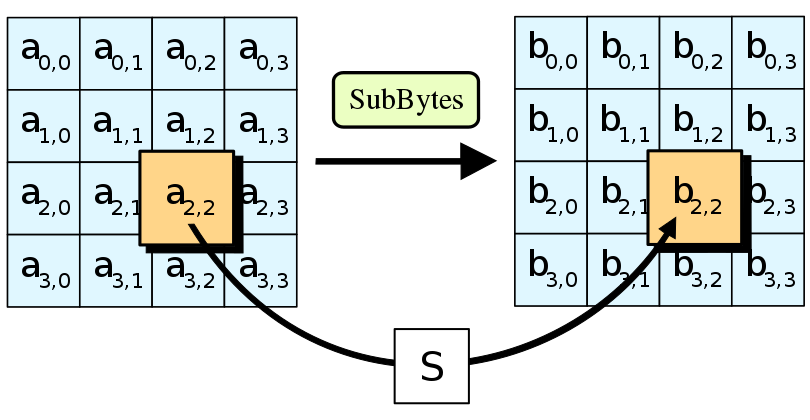


Figure 3. SubBytes Function

On Figure 4, is shown ShiftRows Function.

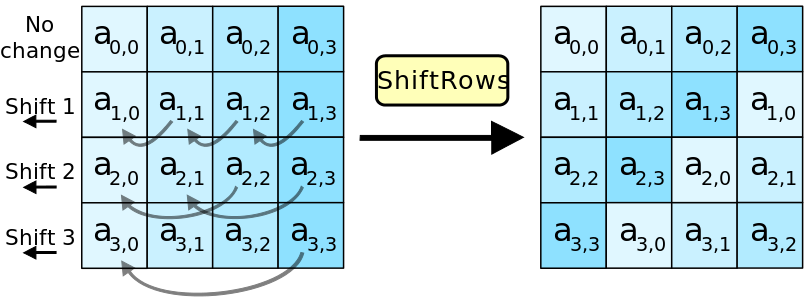


Figure 4. ShiftRows Function

The last function, MixColums, is shown on Figure 5.

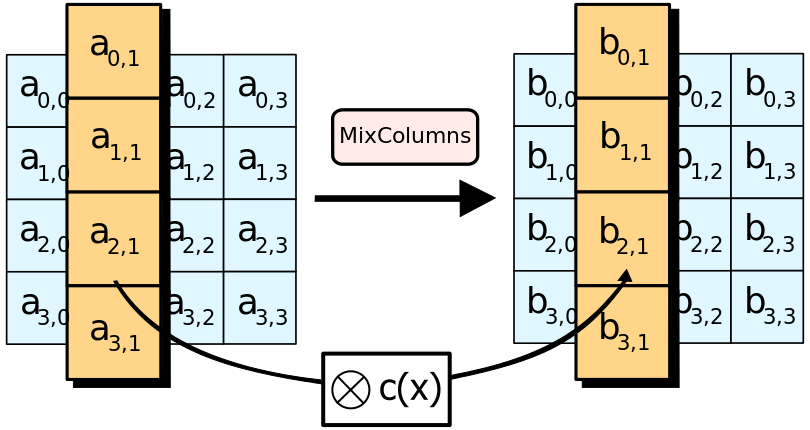


Figure 5. MixColumns Function

On the next section, I will explain the idea of MixColumns function supported by multiplication in GF (28).

# Literature review:

In order to explain what and how MixColumns does, is important to know what is GF (28). GF (28) is known as ***finite binary fields of extension 8***, it means that here, applies the same idea that in the first practice, we are operating from 0 to 28. Every number in this ***finite*** ***field*** could be represented as a polynomial expression as such as the following example of the representation of number 81410:

|  |  |  |
| --- | --- | --- |
| Binary | Hexadecimal | Polynomial expression |
| 1000 1110 | 8E | x7 + x3 + x2 + x1 |

Maybe it is difficult to observe why we obtained that mathematical expression for that number, but is not. As we can already have said it, we need to use 8 bits to represent every number due to the extension of our field. To obtain the polynomial expression we need to name bit by bit every bit on the number, that’s why the most significant bit will be x7, but if we have a zero in that place, it will be the same as 0·x7 = 0. The bit next to the right will be x6 so, successively.

Now, we know what GF (28) means, then, we need to know how to work on it, addition will be the same operation as an XOR, like the following table:

|  |  |  |
| --- | --- | --- |
| A | B | A + B = C |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

As we can observe, is very simple, if we have just 1 bit on, the result will be 1, in other case, the result will be 0. Addition is very simple, now, complexity will increase in multiplication in GF (28). In the following example, the multiplication will be between **1010 1101** \* **1110 1111 = AD \* EF**

x7 + x5 + x3 + x2 + 1  
x7 + x6 + x5 + x3 + x2 + x1 + 1  
x7 + x5 + x3 + x2 + 1  
x8 + x6 + x4 + x3 + x1  
x9 + x7 + x5 + x4 + x2  
x10 + x8 + x6 + x5 + x3  
x12 + x10 + x8 + x7 + x5  
x13 + x11 + x9 + x8 + x6  
x14 + x12 + x10 + x9 + x7  
x14 + x13 + x11 + x10 + x9 + x6 + x4 + x3 + x1 + 1

Now, we can observe that we have exponents bigger than 7 (the biggest possible exponent), so, what we need to do is to make a polynomial reduction with a simple division as the next example with a special polynomial: 1001 1011

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
x7 + x4 + x3 + x1 + 1| x14 + x13 + x11 + x10 + x9 + x6 + x4 + x3 + x1 + 1

The result of that expression is: **x7 + x6 + x5 + x4 + x2 + 1**

MixColumns function works in that way, multiplying 2 matrixes (the corresponding key for each round and the plaintext) in GF (28), that’s why, the important thing here was to understand how GF (28) works and how do we need to reduce a polynomial expression.

Remember that multiplication of matrixes A \* B = C is explained below:

(A1,1 \* B1,1) + (A1,2 \* B2,1) + … + (A1,n \* Bn,1) = C1,1

(A2,1 \* B1,1) + (A2,2 \* B2,1) + … + (A2,n \* Bn,1) = C2,1

As you can observe, the multiplication of 2 matrixes is an addition of a lot of multiplications of 2 numbers (as such as multiplication in GF (28)). On Figure 6, you can see a graphic example of multiplication of 2 matrixes.

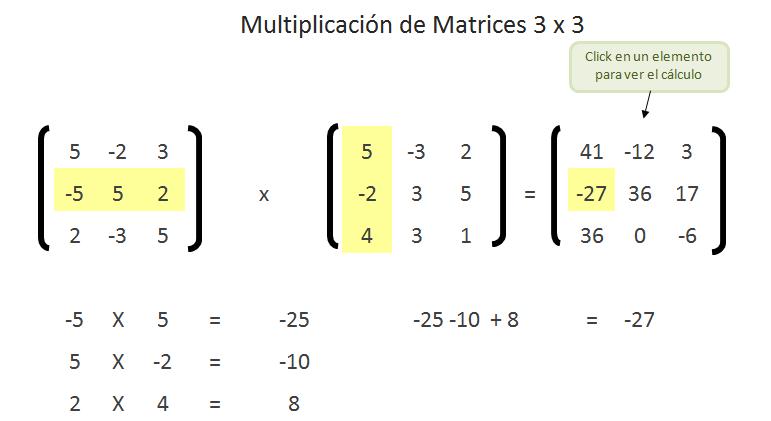


Figure 6. Example of Matrix Multiplication

The next tables represent the differences between AES and DES Cipher, encrypting 2 different images, one of them is Japan’s Flag (only 2 colors), and the other is a landscape (too much variation of color).

|  |  |  |
| --- | --- | --- |
| **DES** | **Japan** | **Paisaje** |
| **Plaintext** |  |  |
| **Electronic Codebook** |  |  |
| **Cipher Block Chaining** |  |  |
| **Output Feedback** |  |  |
| **Cipher Feedback** |  |  |
| **Counter** |  |  |

Table 1. Encryption examples with DES Cipher

The next table represents the same both example images but encrypting now with AES Cipher with its 128 bits of security that is the length of the encryption key.

|  |  |  |
| --- | --- | --- |
| **AES** | **Japan** | **Paisaje** |
| **Plaintext** |  |  |
| **Electronic Codebook** |  |  |
| **Cipher Block Chaining** |  |  |
| **Output Feedback** |  |  |
| **Cipher Feedback** |  |  |
| **Counter** |  |  |

Table 2. Encryption examples with AES Cipher

As we can see on those tables, there’s not a big difference between these two ciphers at least in ECB mode, because the only change is the color and maybe a little distortion in both. For this practice, python facilitates the implementation of the algorithm with 5 different modes of operation and its Crypto Package.

# Software (libraries, packages, tools):

**Packages [2]:**

* Crypto: A collection of cryptographic modules implementing various algorithms and protocols.

**Sub - packages [2]:**

* Cipher: Used for the following objects and/or methods
  + AES [3]: Cipher object
  + Encrypt: Method used to encrypt data with key and parameters set at initialization
  + Decrypt: Method used to decrypt data with key and parameters set at initialization

**Tools:**

* Sublime Text 3 [4]
* Python 3.6 [5]

# Procedure:

The algorithm used for this practice is really simple, it is explained now:

**Algorithm for AES encryption**

* Introduce the name of the original image to encrypt.
* Introduce the name of the encrypted image.
* Select the option “encrypt”.
* Select one of the 5 modes of operation (ECB, CBC, CFB, OFB or CTR).
* For ECB and CTR:
  + Introduce the key of 128 bits (16 characters).
* For CBC, CFB and OFB
  + Introduce the key of 128 bits (16 characters).
  + Introduce the initialization vector (16 characters).

**Algorithm for AES decryption**

* Introduce the name of the encrypted image to decrypt.
* Introduce the name of the decrypted image.
* Select the option “decrypt”.
* Select one of the 5 modes of operation (ECB, CBC, CFB, OFB or CTR).
* For ECB and CTR:
  + Introduce the same key used to encrypt the image.
* For CBC, CFB and OFB
  + Introduce the same key used to encrypt the image.
  + Introduce the same initialization vector used to encrypt the image (16 characters).

On Figure 7, it can be observed the first 2 options for AES.

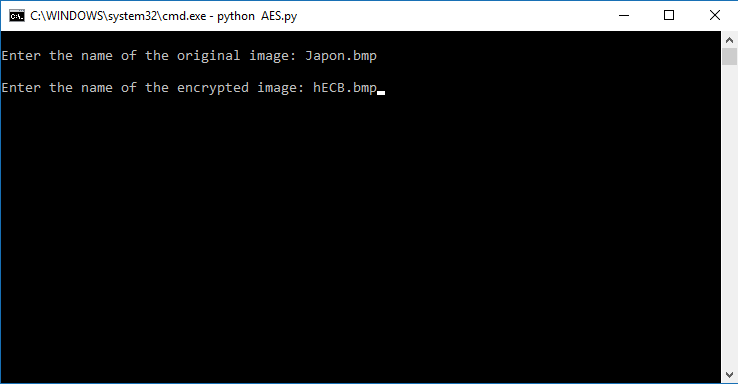


Figure 7. Introducing the name of the original and encrypted/decrypted image

On Figure 8, you can see how the program asks for the option you want (encrypt or decrypt).

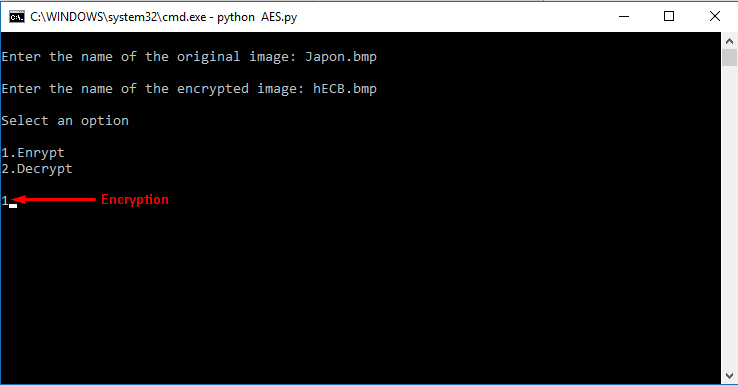


Figure 8. Introducing the option (encrypt/decrypt)

After we select the option (in this case, encryption), the program will ask us for the mode of operation that we want to use as shown in the following Figure 9:

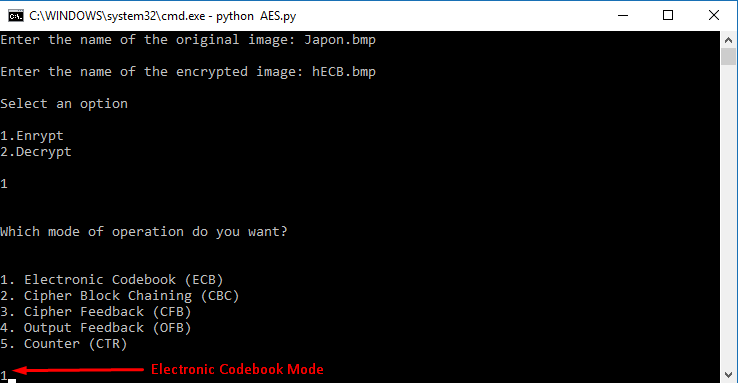


Figure 9. Selecting the mode of operation

After we finish with the mode of operation (ECB in this case) we need to introduce the key (and depending on the mode of operation, an initialization vector) as shown in the following in the Figure 10:

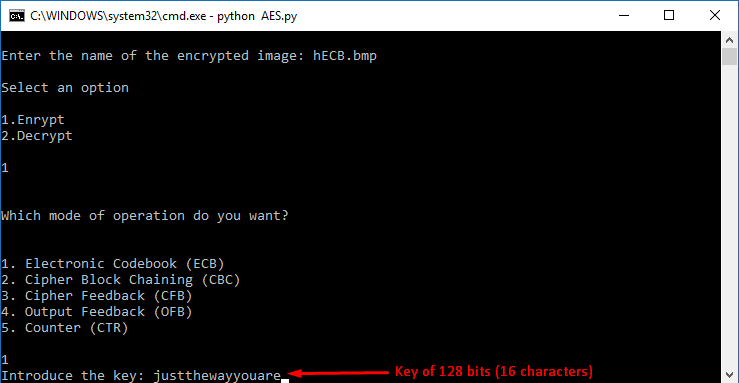


Figure 10. Introducing the key of 128 bits length

After the program finishes, it will appear a message to the user indicating everything was correct (Figure 11).

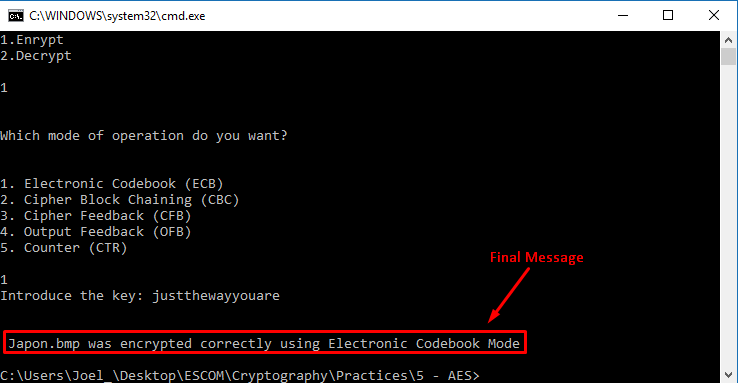


Figure 11. Final message to the user indicating everything was ok

In the next section, I will present the original images and the encryption with all the modes of operation (the same images presented on Table 2).

# Results

We previously show how to initialize the program, in this section it will be just presented the original images and the result of its encryption with all the implemented modes of operation.

First of all, Figure 12 shows the original image (just 2 colors).

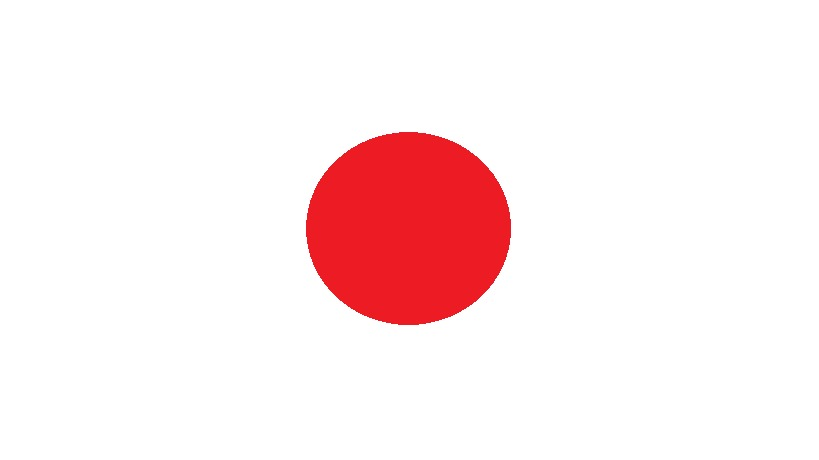


Figure 12. Original image (Japon.bmp)

On Figure 13, it is shown the encrypted image with ECB Mode of Operation.

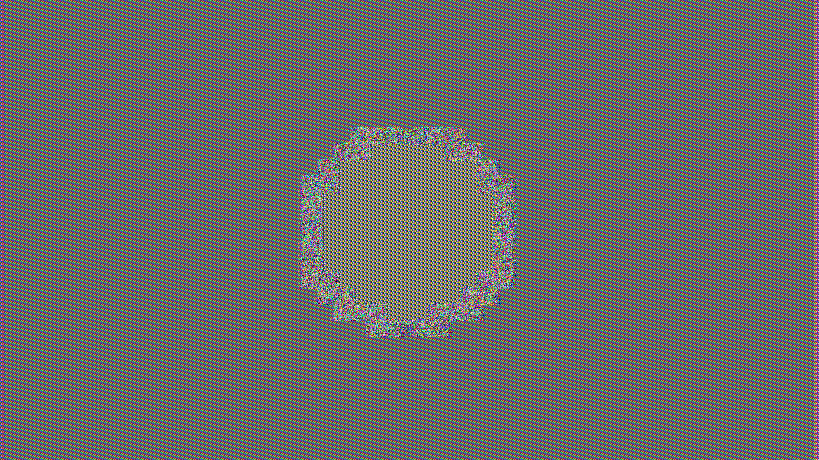


Figure 13. Encrypted image with ECB (hECB)

On Figure 14, it is shown the encrypted image with CBC Mode of Operation.



Figure 14. Encrypted image with CBC (hCBC)

On Figure 15, it is shown the encrypted image with CFB Mode of Operation.



Figure 15. Encrypted image with CFB (hCFB)

On Figure 16, it is shown the encrypted image with OFB Mode of Operation.



Figure 16. Encrypted image with OFB (hOFB)

On Figure 17, it is shown the encrypted image with CTR Mode of Operation.



Figure 17. Encrypted image with CTR (hCTR)

For all of these tests, the password was **justthewayyouare** and the initialization vector (in the mode of operation that corresponds) was 8765432112345678. For decryption, I will show how to introduce the name of the images (now, the original is the encrypted image, as we already explained on the procedure).

On Figure 18, you can see how to introduce the name of the images and selecting mode to obtain the decrypted image.

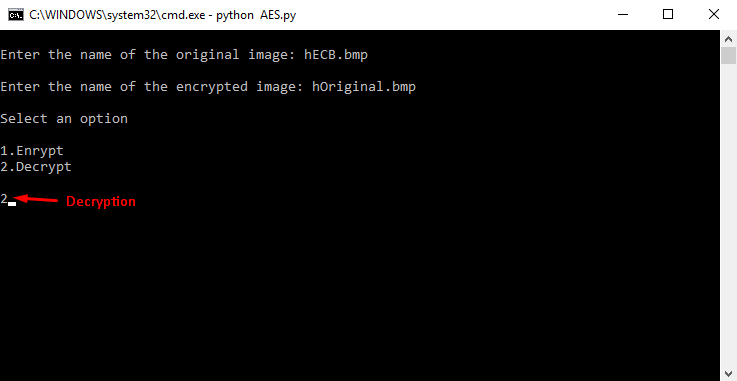


Figure 18. Introducing the name of images and selecting decryption option

Now, we need to introduce the same key that we used to encrypt (justthewayyouare), but, just for testing, I will introduce another key to observe what happen with the image.



Figure 19. Introducing a different key

As you can see, the key is different, the original key is *justthewayyouare* and this is *justthewayyouarr*, with just 1 character different. On Figure 20, you can see the result of giving the program a different key.

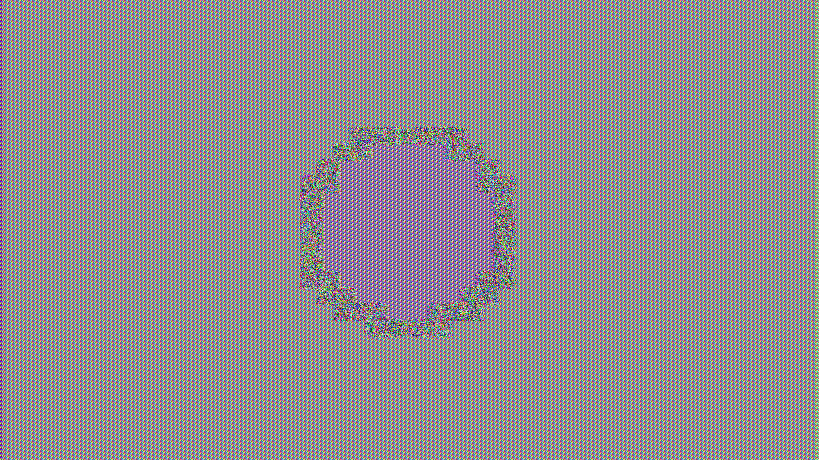


Figure 20. Decryption of Japon.bmp with different key

Now, on Figure 21 you can see the correct key used to encrypt the image.

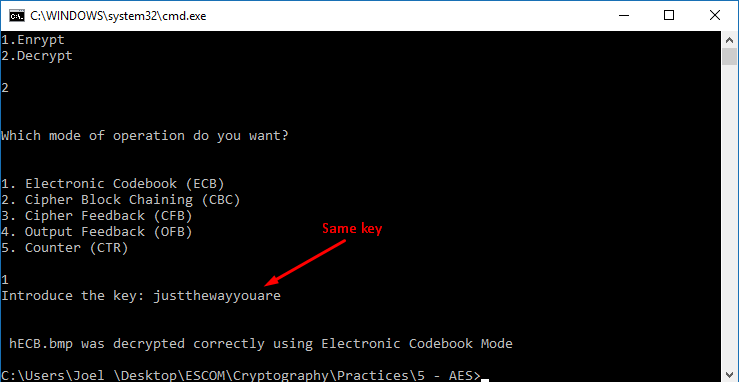


Figure 21. Introducing the same key

On Figure 22, you can see the result of giving it the same key.

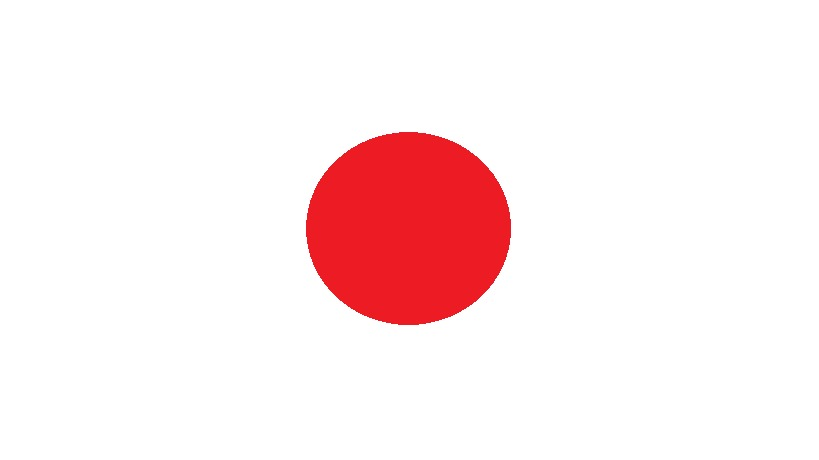


Figure 22. Decrypted image (hOriginal.bmp)

As you can see, we obtain again the original image by introducing the correct key. On Figures 23 - 24 you can see the process of giving the correct key but incorrect initialization vector for CBC, CFB and OFB.

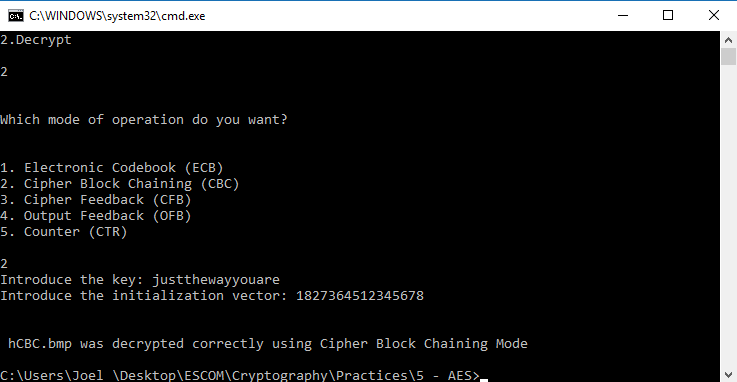


Figure 23. Introducing correct key but initialization vector

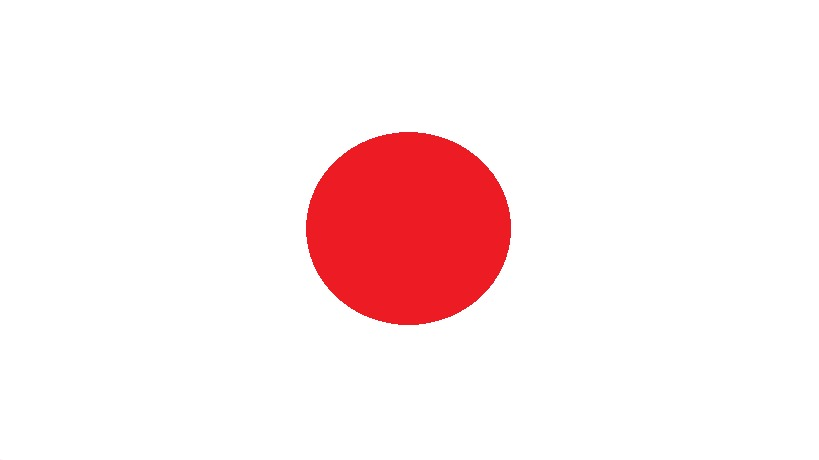


Figure 24. Decrypted image (hOriginal.bmp)

As you can see, the image looks alike the original one, because IV only affects 2 pixels of the image indistinguishable for us. But in Figure 25 (OFB mode), you can see the change with a correct key but different IV.

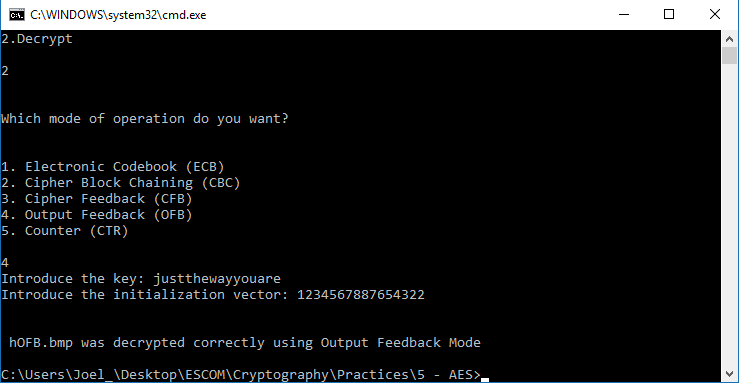


Figure 25. Introducing different IV in OFB Mode

On Figure 26, you can see the decrypted image with different IV.



Figure 26. Decrypted image (hOriginal.bmp)

As you can see the image is unrecognizable because this mode of operation changes all pixels if the IV is wrong.

# Discussion:

Your discussion section has two fundamental aims:

to interpret and explain the results of your study,

to explore the significance of your study’s findings. [qualify and explore](https://unilearning.uow.edu.au/report/2bvi.html) the theoretical importance/significance of your results.

The discussion is also the place in a report where any qualifications or reservations you have about the research should be aired.

# Conclusions:

\* List one thing you learned and describe how it applies to a real-life situation.  
\*Discuss possible errors that could have occurred in the collection of the data (experimental errors)

\*How generally do your results apply?

\*Were there any defects in your experimental design or procedure?

# References:

**[1]** Federal Information Processing Standards, “Advanced Encryption Standard (AES)”, November 2001. [Online]. Available: [https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/ fips-197.pdf](https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/%20fips-197.pdf)

**[2]** EpyDoc, “API Documentation - Package Crypto”, May 2012. [Online]. Available: https://www. dlitz.net/software/pycrypto/api/2.6/

**[3]** EpyDoc, “API Documentation – Class AESCipher”, May 2012. [Online]. Available: <https://www.dlitz.net/software/pycrypto/api/2.6/Crypto.Cipher.AES.AESCipher-class.html>

**[4]** Sublime HQ Pty Ltd, “Sublime Text – Download”, November 2017. [Online]. Available: <https://www.sublimetext.com/3>

**[5]** Python Software Foundation, “python”, October 2017. [Online]. Available: https://www.python. org/downloads/

# Code

**AES.py**

**import** os

**from** Crypto.Cipher **import** AES

**from** Crypto.Util **import** Counter

**def** main ():

os.system ("cls")

#We define the functions (modes of operation) inside a dictionary

modos\_operacion = {1: ECB, 2: CBC, 3: CFB, 4: OFB, 5: CTR}

#Menu to the user

original = input ("\nEnter the name of the original image: ")

cipher = input ("\nEnter the name of the encrypted image: ")

option = int (input ("\nSelect an option\n\n1.Enrypt\n2.Decrypt\n\n"))

**print** ("\n\nWhich mode of operation do you want?\n\n")

**print** ("1. Electronic Codebook (ECB)")

**print** ("2. Cipher Block Chaining (CBC)")

**print** ("3. Cipher Feedback (CFB)")

**print** ("4. Output Feedback (OFB)")

**print** ("5. Counter (CTR)")

mode = int (input ("\n"))

#Calling the selected mode of operation

modos\_operacion [mode] (original, cipher, option)

#Electronic Codebook

**def** ECB (original, ciphered, option):

#Asking for the key to the user (16 bytes)

key = bytes (input ('Introduce the key: '), 'utf-8')

#Creating a new AES cipher

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

#Opening both files

original\_file = open (original, "rb")

encrypted\_file = open (ciphered, "wb")

#We copy the entire head of the image

data = original\_file.read (54)

encrypted\_file.write (data)

#Obtaining the size of the image

original\_file.seek (34)

size = int.from\_bytes (original\_file.read (4), byteorder = 'little')

#We move to the start of the real image to encrypt it

original\_file.seek (54)

i = 0

#Encrypt

**if** option == 1:

**while** (i < size):

#Reading 16 bytes to encrypt it using AES cipher

pixels = original\_file.read (16)

#Encrypting 16 bytes readed

encrypted\_pixels = cipher.encrypt (pixels)

#Writing encrypted pixels

encrypted\_file.write (encrypted\_pixels)

#Updating the counter

i = i + 16

**print** ("\n\n", original, "was encrypted correctly using Electronic Codebook Mode")

#Decrypt

**elif** option == 2:

**while** (i < size):

#Reading 16 bytes to decrypt it using AES cipher

pixels = original\_file.read (16)

#Encrypting 16 bytes readed

encrypted\_pixels = cipher.decrypt (pixels)

#Writing decrypted pixels

encrypted\_file.write (encrypted\_pixels)

#Updating the counter

i = i + 16

**print** ("\n\n", original, "was decrypted correctly using Electronic Codebook Mode")

original\_file.close ()

encrypted\_file.close ()

#Cipher Block Chaining

**def** CBC (original, ciphered, option):

#Asking for the key to the user (16 bytes)

key = bytes (input ('Introduce the key: '), 'utf-8')

#Asking for the initialization vector to the user (16 bytes)

IV = bytes (input ('Introduce the initialization vector: '), 'utf-8')

#Creating a new AES cipher

cipher = AES.new (key, AES.MODE\_CBC, IV)

#Opening both files

original\_file = open (original, "rb")

encrypted\_file = open (ciphered, "wb")

#We copy the entire head of the image

data = original\_file.read (54)

encrypted\_file.write (data)

#Obtaining the size of the image

original\_file.seek (34)

size = int.from\_bytes (original\_file.read (4), byteorder = 'little')

#We move to the start of the real image to encrypt it

original\_file.seek (54)

i = 0

#Encrypt

**if** option == 1:

**while** (i < size):

#Reading 16 bytes to encrypt it using AES cipher

pixels = original\_file.read (16)

#Encrypting 16 bytes readed

encrypted\_pixels = cipher.encrypt (pixels)

#Writing encrypted pixels

encrypted\_file.write (encrypted\_pixels)

#Updating the counter

i = i + 16

**print** ("\n\n", original, "was encrypted correctly using Cipher Block Chaining Mode")

#Decrypt

**elif** option == 2:

**while** (i < size):

#Reading 16 bytes to decrypt it using AES cipher

pixels = original\_file.read (16)

#Encrypting 16 bytes readed

encrypted\_pixels = cipher.decrypt (pixels)

#Writing decrypted pixels

encrypted\_file.write (encrypted\_pixels)

#Updating the counter

i = i + 16

**print** ("\n\n", original, "was decrypted correctly using Cipher Block Chaining Mode")

original\_file.close ()

encrypted\_file.close ()

#Cipher Feedback

**def** CFB (original, ciphered, option):

#Asking for the key to the user (16 bytes)

key = bytes (input ('Introduce the key: '), 'utf-8')

#Asking for the initialization vector to the user (16 bytes)

IV = bytes (input ('Introduce the initialization vector: '), 'utf-8')

#Creating a new AES cipher

cipher = AES.new (key, AES.MODE\_CFB, IV)

#Opening both files

original\_file = open (original, "rb")

encrypted\_file = open (ciphered, "wb")

#We copy the entire head of the image

data = original\_file.read (54)

encrypted\_file.write (data)

#Obtaining the size of the image

original\_file.seek (34)

size = int.from\_bytes (original\_file.read (4), byteorder = 'little')

#We move to the start of the real image to encrypt it

original\_file.seek (54)

i = 0

#Encrypt

**if** option == 1:

**while** (i < size):

#Reading 16 bytes to encrypt it using AES cipher

pixels = original\_file.read (16)

#Encrypting 16 bytes readed

encrypted\_pixels = cipher.encrypt (pixels)

#Writing encrypted pixels

encrypted\_file.write (encrypted\_pixels)

#Updating the counter

i = i + 16

**print** ("\n\n", original, "was encrypted correctly using Cipher Feedback Mode")

#Decrypt

**elif** option == 2:

**while** (i < size):

#Reading 16 bytes to decrypt it using AES cipher

pixels = original\_file.read (16)

#Encrypting 16 bytes readed

encrypted\_pixels = cipher.decrypt (pixels)

#Writing decrypted pixels

encrypted\_file.write (encrypted\_pixels)

#Updating the counter

i = i + 16

**print** ("\n\n", original, "was decrypted correctly using Cipher Feedback Mode")

original\_file.close ()

encrypted\_file.close ()

#Output Feedback

**def** OFB(original,ciphered,option):

#Asking for the key to the user (16 bytes)

key = bytes (input ('Introduce the key: '), 'utf-8')

#Asking for the initialization vector to the user (16 bytes)

IV = bytes (input ('Introduce the initialization vector: '), 'utf-8')

#Creating a new AES cipher

cipher = AES.new (key, AES.MODE\_OFB, IV)

#Opening both files

original\_file = open (original, "rb")

encrypted\_file = open (ciphered, "wb")

#We copy the entire head of the image

data = original\_file.read (54)

encrypted\_file.write (data)

#Obtaining the size of the image

original\_file.seek (34)

size = int.from\_bytes (original\_file.read (4), byteorder = 'little')

#We move to the start of the real image to encrypt it

original\_file.seek (54)

i = 0

#Encrypt

**if** option == 1:

**while** (i < size):

#Reading 16 bytes to encrypt it using AES cipher

pixels = original\_file.read (16)

#Encrypting 16 bytes readed

encrypted\_pixels = cipher.encrypt (pixels)

#Writing encrypted pixels

encrypted\_file.write (encrypted\_pixels)

#Updating the counter

i = i + 16

**print** ("\n\n", original, "was encrypted correctly using Output Feedback Mode")

#Decrypt

**elif** option == 2:

**while** (i < size):

#Reading 16 bytes to decrypt it using AES cipher

pixels = original\_file.read (16)

#Encrypting 16 bytes readed

encrypted\_pixels = cipher.decrypt (pixels)

#Writing decrypted pixels

encrypted\_file.write (encrypted\_pixels)

#Updating the counter

i = i + 16

**print** ("\n\n", original, "was decrypted correctly using Output Feedback Mode")

original\_file.close ()

encrypted\_file.close ()

#Counter

**def** CTR(original,ciphered,option):

#Asking for the key to the user (16 bytes)

key = bytes (input ('Introduce the key: '), 'utf-8')

#Creating a new AES cipher

ctr = Counter.new (128)

cipher = AES.new (key, AES.MODE\_CTR, counter = ctr)

#Opening both files

original\_file = open (original, "rb")

encrypted\_file = open (ciphered, "wb")

#We copy the entire head of the image

data = original\_file.read (54)

encrypted\_file.write (data)

#Obtaining the size of the image

original\_file.seek (34)

size = int.from\_bytes (original\_file.read (4), byteorder = 'little')

#We move to the start of the real image to encrypt it

original\_file.seek (54)

i = 0

#Encrypt

**if** option == 1:

**while** (i < size):

#Reading 16 bytes to encrypt it using AES cipher

pixels = original\_file.read (16)

#Encrypting 16 bytes readed

encrypted\_pixels = cipher.encrypt (pixels)

#Writing encrypted pixels

encrypted\_file.write (encrypted\_pixels)

#Updating the counter

i = i + 16

**print** ("\n\n", original, "was encrypted correctly using Counter Mode")

#Decrypt

**elif** option == 2:

**while** (i < size):

#Reading 16 bytes to decrypt it using AES cipher

pixels = original\_file.read (16)

#Encrypting 16 bytes readed

encrypted\_pixels = cipher.decrypt (pixels)

#Writing decrypted pixels

encrypted\_file.write (encrypted\_pixels)

#Updating the counter

i = i + 16

**print** ("\n\n", original, "was decrypted correctly using Counter Mode")

original\_file.close ()

encrypted\_file.close ()

#Main function

main ()