Eric Wong: 501174088
Rohan Manoharan: 501189408
Thylane Rossi: 501340990

# Labs 2 - Secret-Key Encryption Lab

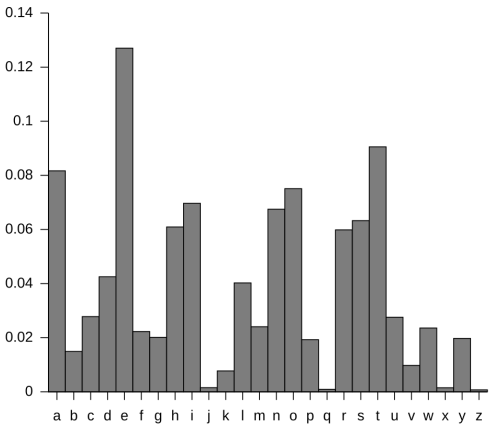
## Task 1 - Frequency Analysis

Here is the text to decrypt:

```
[10/02/24]seed@VM:~/.../Files$ cat ciphertext.txt
ytn xqavhq yzhu  xu qzupvd ltmat qnncq vgxzy hmrty vbynh ytmq ixur qyhvurn
vlvhpq yhme ytn gvrrnh bnniq imsn v uxuvrnuvhmvu yxx

ytn vlvhpq hvan lvq gxxsnupnp gd ytn pncmqn xb tvhfnd lnmuqynmu vy myq xzyqny
vup ytn veevhnuy mceixqmxu xb tmq bmic axcevud vy ytn nup vup my lvq qtvenp gd
ytn ncnhrnuan xb cnyxx ymcnq ze givasrxlu eximymaq vhcavupd vaymfmqc vup
v uvymxuvi axufnhqvymxu vq ghmnb vup cvp vq v bnfnh phnvc vgxzy ltnytnh ytnhn
xzrty yx gn v ehnqmpnuy lmubhnd ytn qnvqxu pmpuy ozqy qnnc nkyhv ixur my lvq
nkyhv ixur gnavzqn ytn xqavhq lnhn cxfnp yx ytn bmhqy lnnsnup mu cvhat yx
vfxmp axubimaymur lmyt ytn aixqmur anhncxud xb ytn lmuynh xidcemaq ytvusq
ednxuratvur

xun gmr jznqymxu qzhhxzupmur ytmq dnvhq vavpncd vlvhpq mq txl xh mb ytn
anhncxud lmii vpphnqq cnyxx nqenamviid vbynh ytn rxipnu rixgnq ltmat gnavcn
v ozgmivuy axcmurxzy evhyd bxh ymcnq ze ytn cxfncnuy qenvhtnvpnp gd
exlnhbzi txiidlxxp lxcnu ltx tnienp hvmqn cmiimxuq xb pxiivhq yx bmrty qnkzvi
tvhvqqcnuy vhxzup ytn axzuyhd

qmruvimur ytnmh qzeexhy rxipnu rixgnq vyynupnnq qlvytnp ytncqnifnq mu givas
qexhynp iveni emuq vup qxzupnp xbb vgxzy qnkmqy exlnh mcgvivuanq bhxc ytn hnp
avheny vup ytn qyvrn xu ytn vmh n lvq aviinp xzy vgxzy evd munjzmyd vbynh
myq bxhcnh vuatxh avyy qvpinh jzmy xuan qtn invhunp ytvy qtn lvq cvsmur bvh
inqq ytvu v cvin axtxqy vup pzhmur ytn anhncxud uvyvimn exhycvu yxxs v gizuy
vup qvymqbdmur pmr vy ytn viicvin hxqynh xb uxcmuvynp pmhnayxhq txl axzip
ytvy gn yxeenp

vq my yzhuq xzy vy invqy mu ynhcq xb ytn xqavhq my ehxgvgid lxuy gn

lxcnu mufxifnp mu ymcnq ze qvmp ytvy viytxzrt ytn rixgnq qmrumbmnp ytn
mumymvymfnq ivzuat ytnd unfnh muynupnp my yx gn ozqy vu vlvhpq qnvqxu
avcevmru xh xun ytvy gnavcn vqqxamvynp xuid lmyt hnpavheny vaymxuq muqynvp
v qexsnqlxcvu qvmp ytn rhxze mq lxhsmur gntmup aixqnp pxxhq vup tvq qmuan
vcvqqnp  cmiimxu bxh myq inrvi pnbnuqn bzup ltmat vbynh ytn rixgnq lvq
bixxpnp lmyt ytxzqvupq xb pxuvymxuq xb  xh inqq bhxc enxein mu qxcn
axzuyhmnq


ux avii yx lnvh givas rxluq lnuy xzy mu vpfvuan xb ytn xqavhq ytxzrt ytn
cxfncnuy lmii vicxqy anhyvmuid gn hnbnhnuanp gnbxhn vup pzhmur ytn anhncxud
nqenamviid qmuan fxavi cnyxx qzeexhynhq imsn vqtind ozpp ivzhv pnhu vup
umaxin smpcvu vhn qatnpzinp ehnqnuynhq

vuxytnh bnvyzhn xb ytmq qnvqxu ux xun hnviid suxlq ltx mq rxmur yx lmu gnqy
emayzhn vhrzvgid ytmq tveenuq v ixy xb ytn ymcn muvhrzvgid ytn uvmigmynh
uvhhvymfn xuid qnhfnq ytn vlvhpq tden cvatmun gzy xbynu ytn enxein bxhnavqymur
ytn hvan qxaviinp xqavhxixrmqyq avu cvsn xuid npzavynp rznqqnq

ytn lvd ytn vavpncd yvgzivynq ytn gmr lmuunh pxnquy tnie mu nfnhd xytnh
avynrxhd ytn uxcmunn lmyt ytn cxqy fxynq lmuq gzy mu ytn gnqy emayzhn
avynrxhd fxynhq vhn vqsnp yx imqy ytnmh yxe cxfmnq mu ehnbnhnuymvi xhpnh mb v
cxfmn rnyq cxhn ytvu  enhanuy xb ytn bmhqyeivan fxynq my lmuq ltnu ux
cxfmn cvuvrnq ytvy ytn xun lmyt ytn bnlnqy bmhqyeivan fxynq mq nimcmuvynp vup

myq fxynq vhn hnpmqyhmgzynp yx ytn cxfmnq ytvy rvhunhnp ytn nimcmuvynp gviixy
qnaxupeivan fxynq vup ytmq axuymuznq zuymi v lmuunh ncnhrnq

my mq vii ynhhmgid axubzqmur gzy veevhnuyid ytn axuqnuqzq bvfxhmyn axcnq xzy
vtnvp mu ytn nup ytmq cnvuq ytvy nupxbqnvqxu vlvhpq atvyynh mufvhmvgid
mufxifnq yxhyzhnp qenazivymxu vgxzy ltmat bmic lxzip cxqy imsnid gn fxynhq
qnaxup xh ytmhp bvfxhmyn vup ytnu njzviid yxhyzhnp axuaizqmxuq vgxzy ltmat
bmic cmrty ehnfvmi

mu  my lvq v yxqqze gnylnnu gxdtxxp vup ytn nfnuyzvi lmuunh gmhpcvu
mu  lmyt ixyq xb nkenhyq gnyymur xu ytn hnfnuvuy xh ytn gmr qtxhy ytn
ehmwn lnuy yx qexyimrty ivqy dnvh unvhid vii ytn bxhnavqynhq pnaivhnp iv
iv ivup ytn ehnqzceymfn lmuunh vup bxh ylx vup v tvib cmuzynq ytnd lnhn
axhhnay gnbxhn vu nufnixen quvbz lvq hnfnvinp vup ytn hmrtybzi lmuunh
cxxuimrty lvq ahxlunp

ytmq dnvh vlvhpq lvyatnhq vhn zunjzviid pmfmpnp gnylnnu ythnn gmiigxvhpq
xzyqmpn nggmur cmqqxzhm ytn bvfxhmyn vup ytn qtven xb lvynh ltmat mq
ytn gvrrnhq ehnpmaymxu lmyt v bnl bxhnavqymur v tvmi cvhd lmu bxh rny xzy

gzy vii xb ytxqn bmicq tvfn tmqyxhmavi xqavhfxymur evyynhuq vrvmuqy ytnc ytn
qtven xb lvynh tvq  uxcmuvymxuq cxhn ytvu vud xytnh bmic vup lvq viqx
uvcnp ytn dnvhq gnqy gd ytn ehxpzanhq vup pmhnayxhq rzmipq dny my lvq uxy
uxcmuvynp bxh v qahnnu vayxhq rzmip vlvhp bxh gnqy nuqncgin vup ux bmic tvq
lxu gnqy emayzhn lmytxzy ehnfmxzqid ivupmur vy invqy ytn vayxhq uxcmuvymxu
qmuan ghvfntnvhy mu  ytmq dnvh ytn gnqy nuqncgin qvr nupnp ze rxmur yx

ythnn gmiigxvhpq ltmat mq qmrumbmavuy gnavzqn vayxhq cvsn ze ytn vavpncdq
ivhrnqy ghvuat ytvy bmic ltmin pmfmqmfn viqx lxu ytn gnqy phvcv rxipnu rixgn
vup ytn gvbyv gzy myq bmiccvsnh cvhymu capxuvrt lvq uxy uxcmuvynp bxh gnqy
pmhnayxh vup vevhy bhxc vhrx cxfmnq ytvy ivup gnqy emayzhn lmytxzy viqx
nvhumur gnqy pmhnayxh uxcmuvymxuq vhn bnl vup bvh gnylnnu



[10/02/24]seed@VM:~/.../Files$
```

```
[10/02/24]seed@VM:~/.../Files$ ./freq.py
--------------------------------------
1-gram (top 20):
n: 488
y: 373
v: 348
x: 291
u: 280
q: 276
m: 264
h: 235
t: 183
i: 166
p: 156
a: 116
c: 104
z: 95
l: 90
g: 83
b: 83
r: 82
e: 76
d: 59
```

The most frequent letter in the text is 'n'. According to https://en.wikipedia.org/wiki/Frequency_analysis, in English, the most used letter is 'e', followed by 't'.
Thus, we hypothesize that 'y' = 't'. Then, we hypothesize that 'v' = 'a'. Additionally, based on the 2 or 3-letter words:




```
3-gram (top 20):
ytn: 78
vup: 30
mur: 20
ynh: 18
xzy: 16
mxu: 14
gnq: 14
ytv: 13
nqy: 13
vii: 13
bxh: 13
lvq: 12
nuy: 12
vyn: 12
uvy: 11
lmu: 11
nvh: 11
cmu: 11
tmq: 10
vhp: 10
```

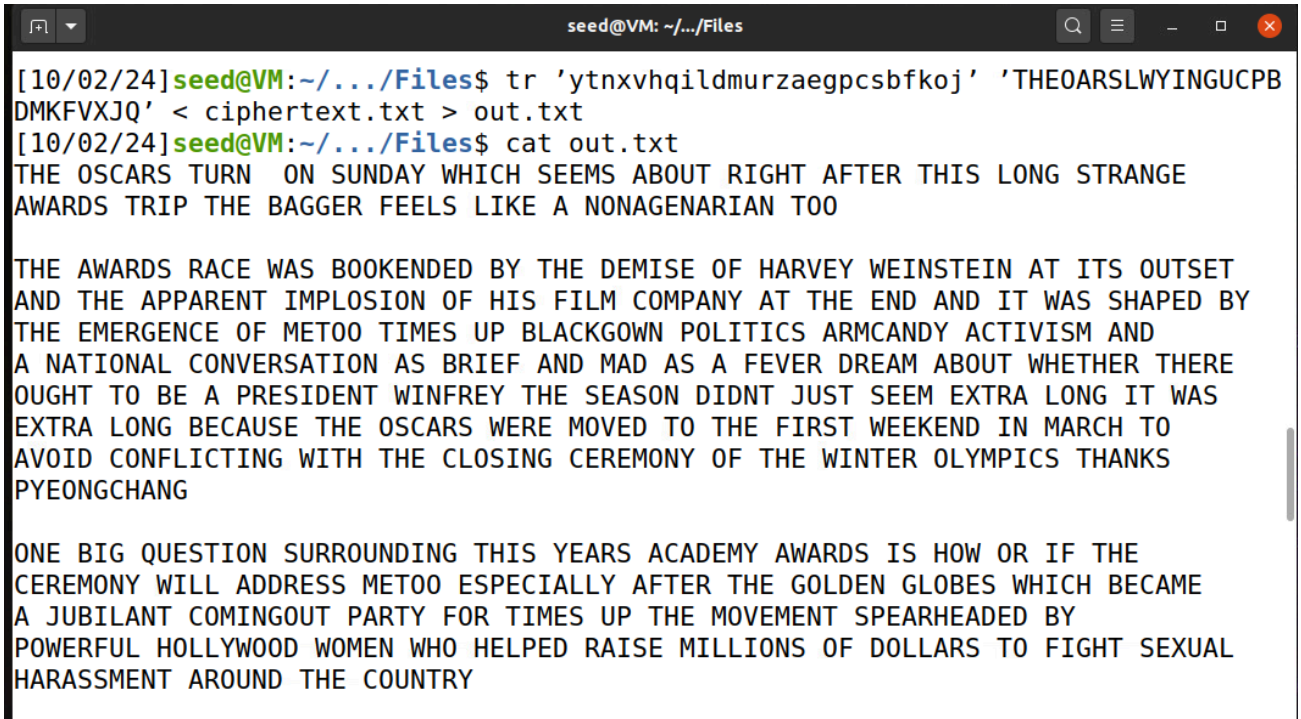
we can easily deduce that 'ytn' = 'the', 'vup' = 'and'...

On a :
Y = T
T = H
N = E
V = A
U = N
P = D

Then gradually, especially thanks to small words (for example: 'xu', we know that 'u' = 'n', and that 'x' cannot be 'a' because 'v' = 'a', so we deduce that 'x' = 'o'...), we decrypt the entire text and find that:

A = C
B = F
C = M
D = Y
E = P
F = V
G = B
H = R
I = L
J = Q
K = X
L = W
M = I
N = E
O = J
P = D
Q = S
R = G
S = K
T = H
U = N
V = A
W = Z
X = O
Y = T
Z = U

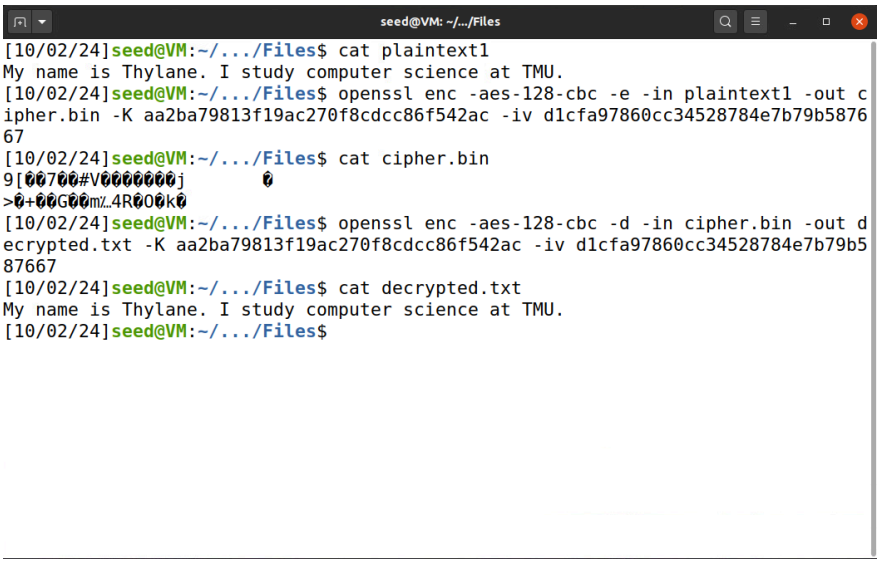
So the key is **CFMYPVBRLQXWIEJDSGKHNAZOTU**



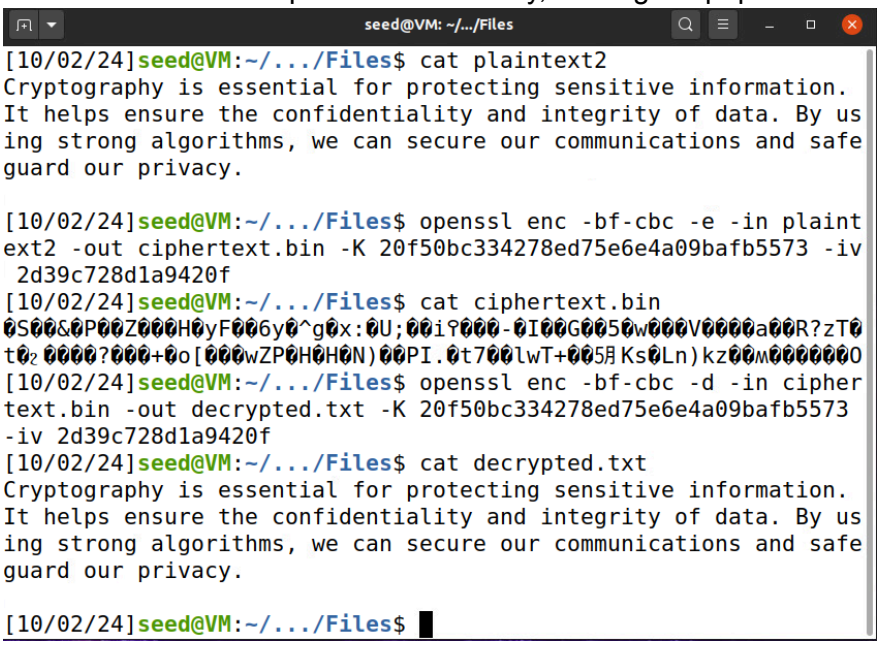
## Task 2 - Encryption using Different Ciphers and Modes

I generate the key and the initialization vector (IV) using the command line: openssl rand -hex 16

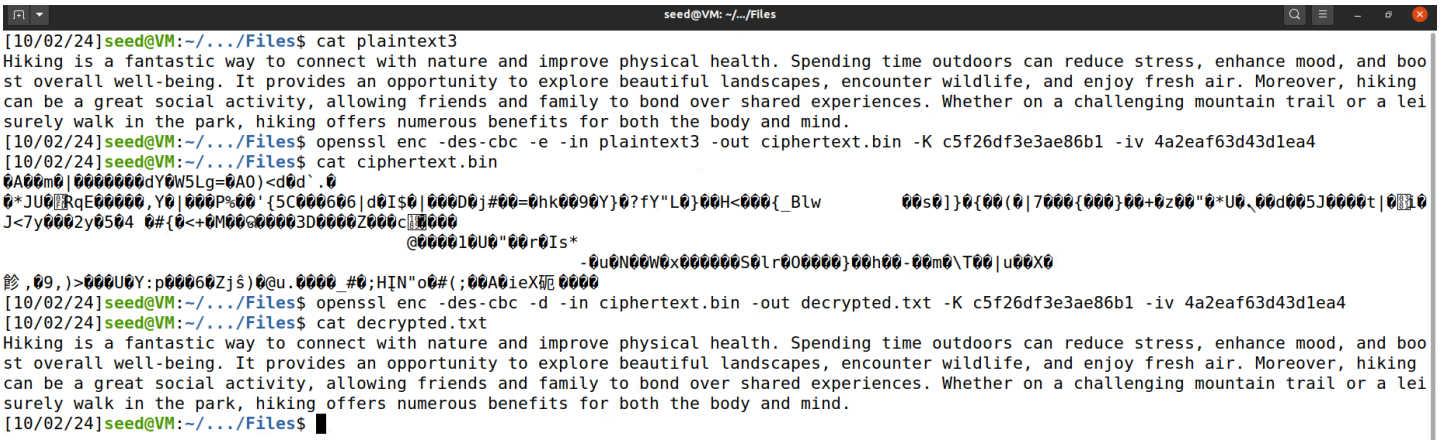
The first example of using OpenSSL and the cipher type -aes-128-cbc is a symmetric encryption algorithm that uses a 128-bit key in Cipher Block Chaining mode, providing good security by combining each block of plaintext with the previous one and requiring a unique initialization vector (IV).




Here is the second example, this time, the cipher type used is -bf-cbc, blowfish in CBC (Cipher Block Chaining) mode is a symmetric encryption algorithm that uses a key length between 32 and 448 bits. In this mode, each plaintext block is XORed with the previous ciphertext block, enhancing security. It requires a unique initialization vector (IV) for each encryption, preventing replay attacks. Blowfish is favored for its speed and efficiency, making it a popular choice for encryption tasks.

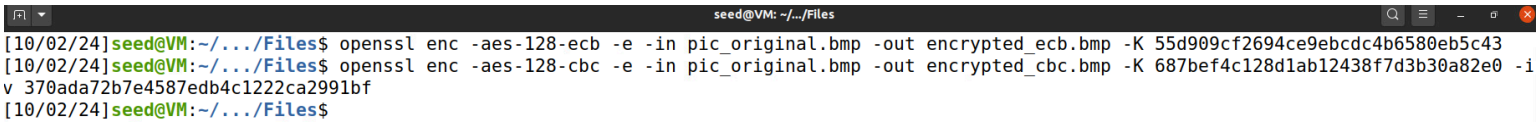


Here is the third example that uses the -des-cbc cipher type, the Data Encryption Standard (DES) in Cipher Block Chaining (CBC) mode is a symmetric encryption algorithm that encrypts 64-bit blocks. Each plaintext block is combined with the previous ciphertext block, enhancing security. Although DES uses a 56-bit key, it is considered outdated and vulnerable to attacks.




## Task 3 - Encryption Mode – ECB vs. CBC

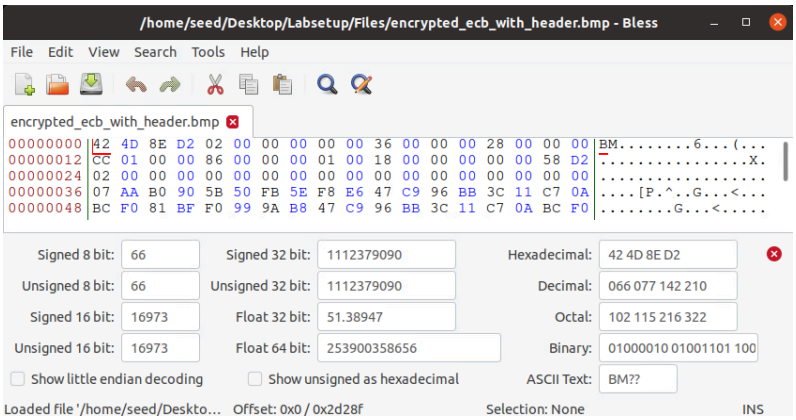
Here is the encrypted image, first with ECB, then with CBC :

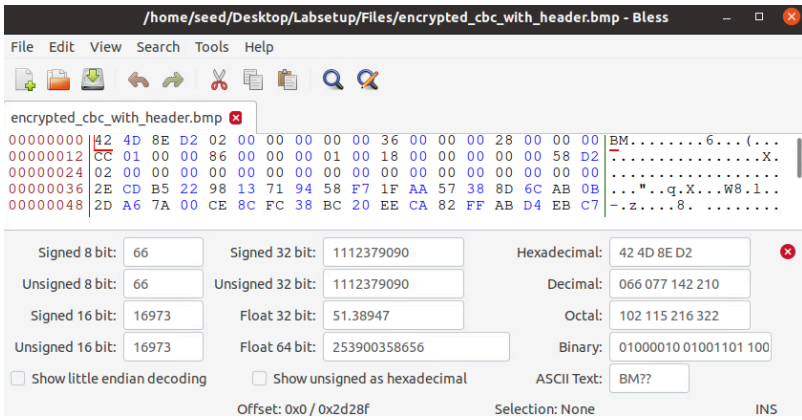



Modification of the image header :

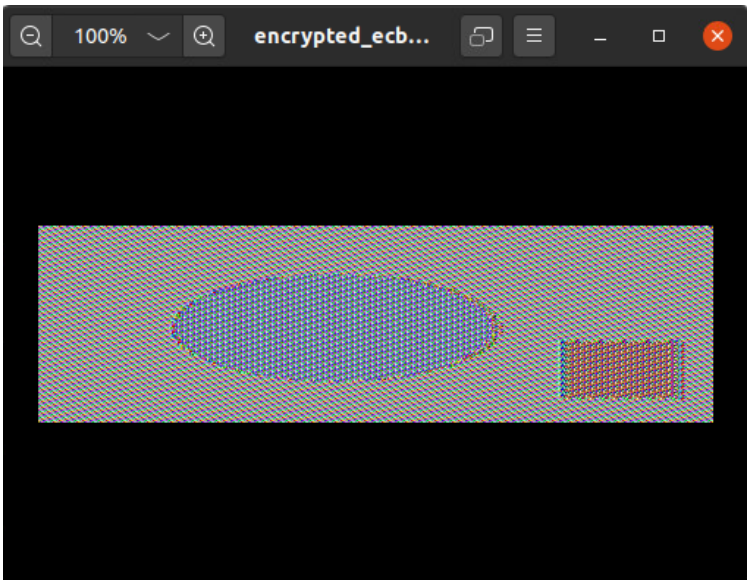
```
[10/02/24]seed@VM:~/.../Files$ head -c 54 pic_original.bmp > header
[10/02/24]seed@VM:~/.../Files$ tail -c +55 encrypted_ecb.bmp > body_ecb
[10/02/24]seed@VM:~/.../Files$ cat header body_ecb > encrypted_ecb_with_header.b
mp
[10/02/24]seed@VM:~/.../Files$ tail -c +55 encrypted_cbc.bmp > body_cbc
[10/02/24]seed@VM:~/.../Files$ cat header body_cbc > encrypted_cbc_with_header.b
mp
```

Displaying images with bless :

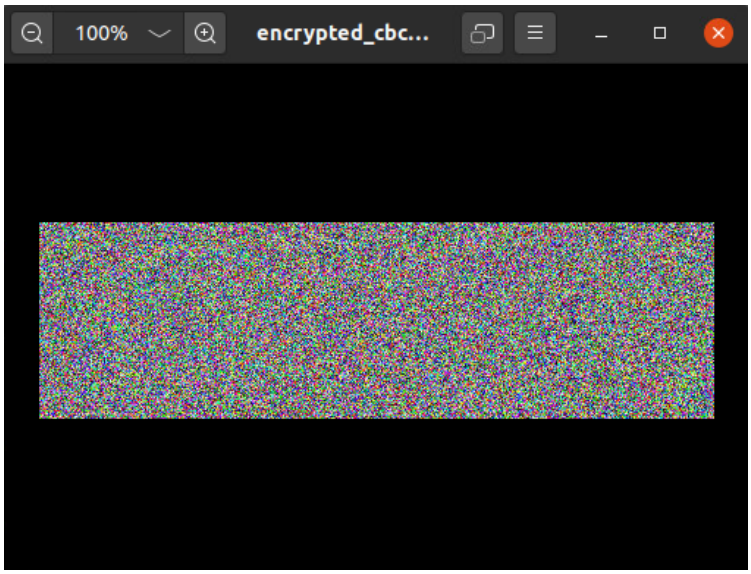



Displaying images in eog. On the left, ECB encryption; on the right, CBC

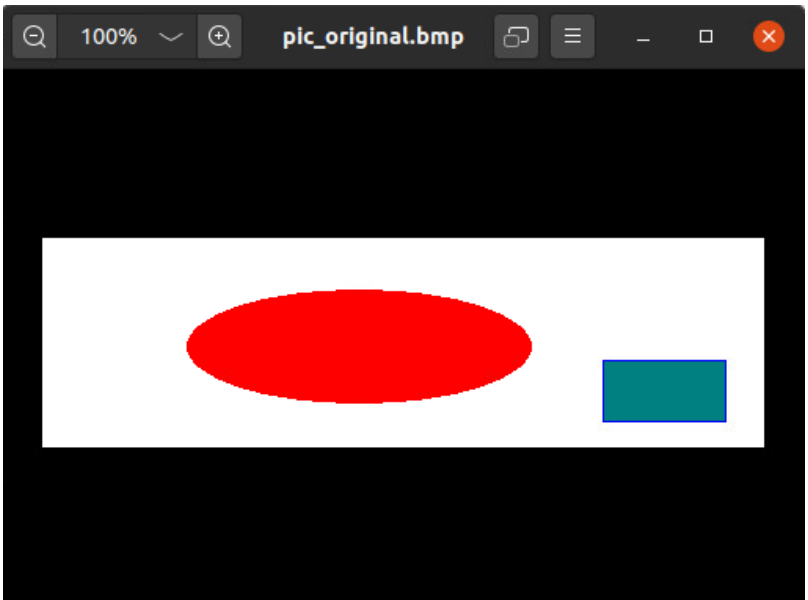



`eog encrypted_ecb_with_header.bmp`



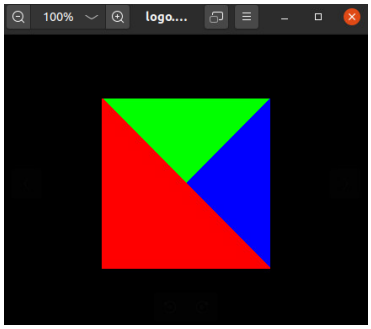

`eog encrypted_cbc_with_header.bmp`

Original image :

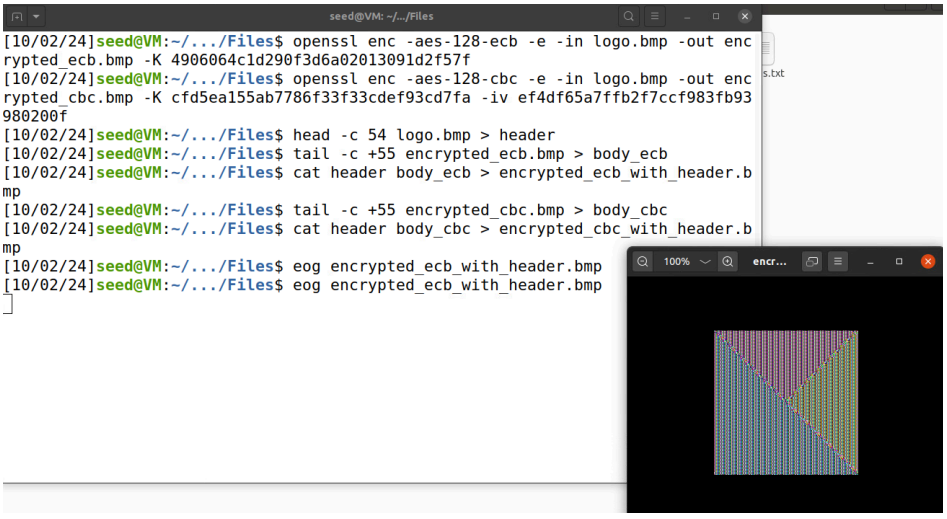


With ECB encryption, it is very easy to distinguish the elements present in the image. The encryption is poor because the pixels of the same color are all coded in the same color, as if there were just a filter. On the other hand, with CBC encryption, the image has nothing to do with the original; nothing can be distinguished.

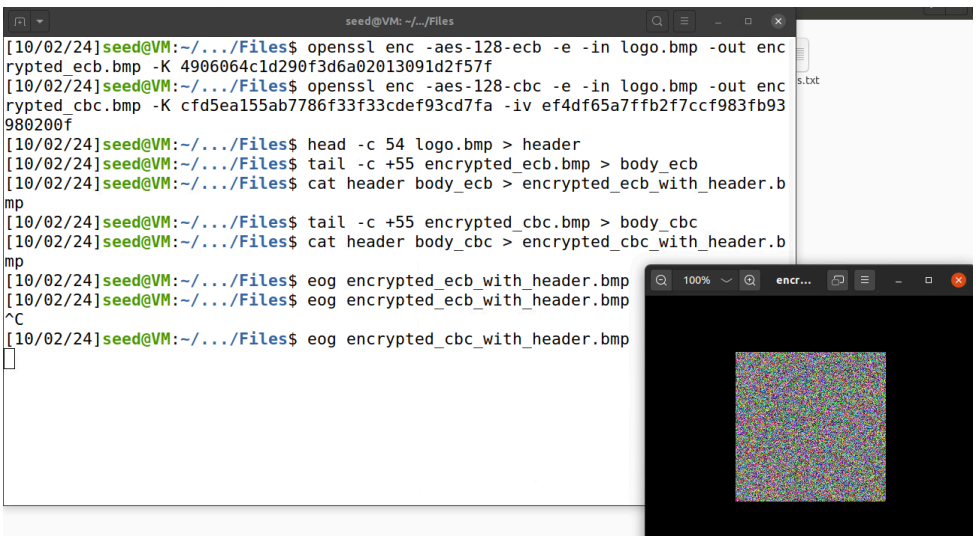
Here is another example with this original image




Here is the result of the encryption with ECB




And the result of the encryption with CBC

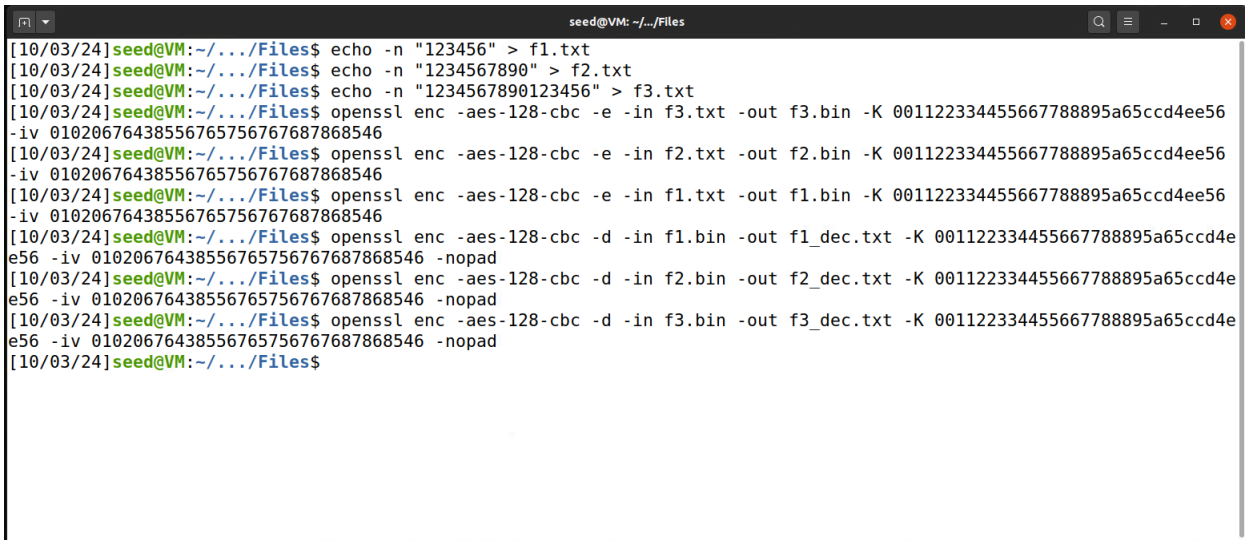


Once again, there is better encryption with the CBC cipher type than with ECB.

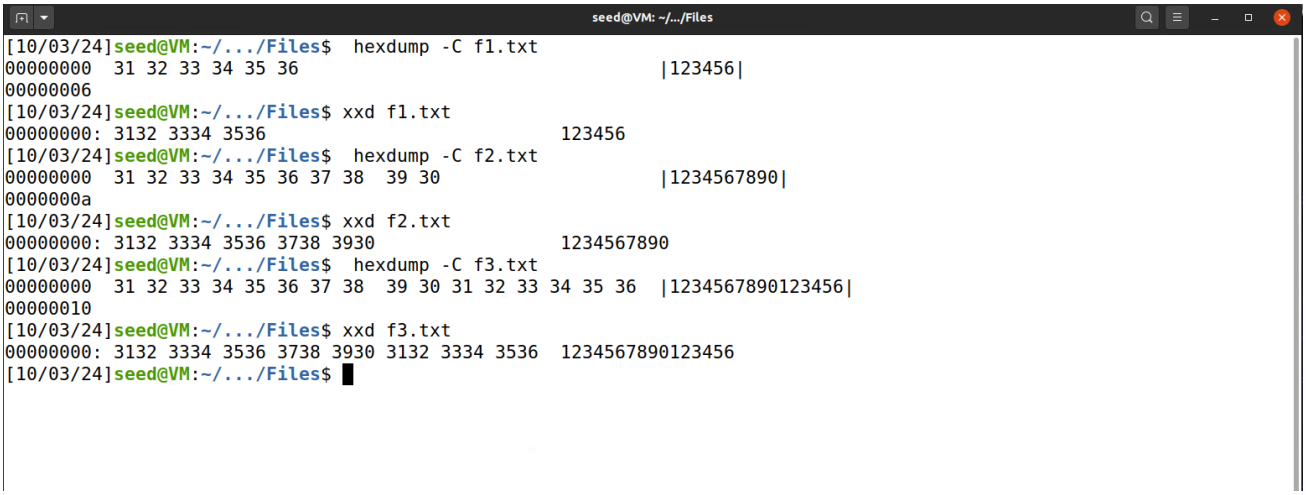
## Task 4 - Padding

### 4.1 - Cipher Block Chaining (CBC)

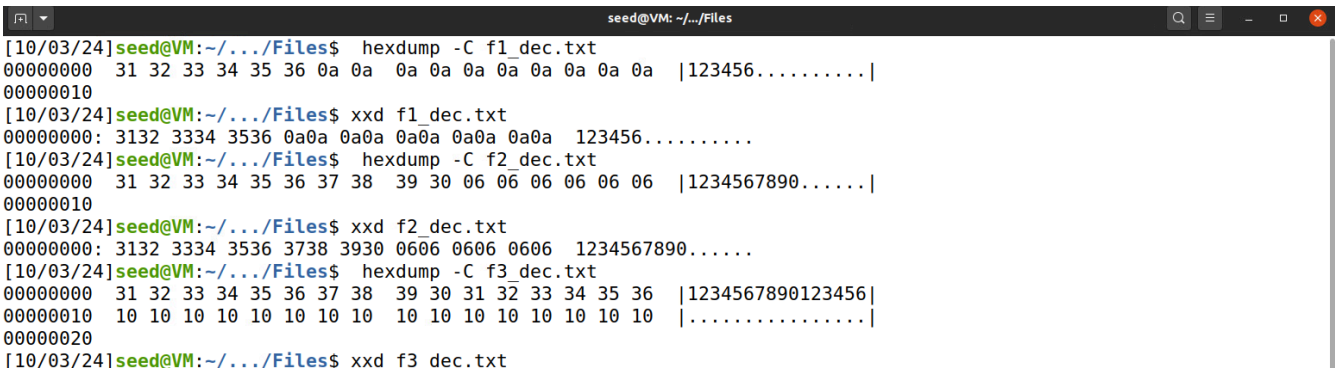
Creation of the files f1 (6 bytes), f2 (10 bytes), and f3 (16 bytes), encryption and decryption with the -nopad option to observe the padding added, the cipher type used here is **Cipher Block Chaining** (**CBC**).




Here are the hexadecimal analyses of the original files:



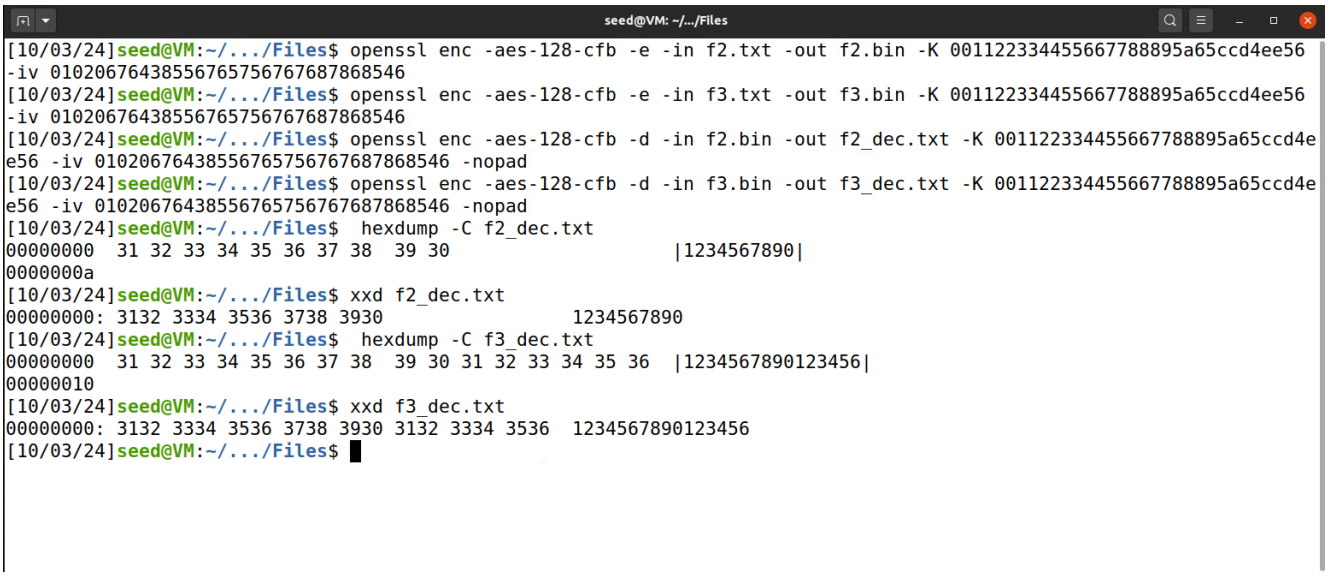

Here is the analysis of the files after decryption:



In conclusion, we notice that the file *f1.txt*, which was composed of 6 bytes, was extended with 10 blocks of '0a' to reach the nearest multiple of 16 bytes (in this case 16). Similarly, for the file *f2.txt*, which originally had 10 bytes, 6 blocks of '06' were added to make the file 16 bytes. However, the file *f3.txt* did not need any blocks since it was already 16 bytes. Therefore, we can confirm that the Cipher Block Chaining (CBC) cipher type requires padding.

### 4.2 - Cipher Feedback (CFB)

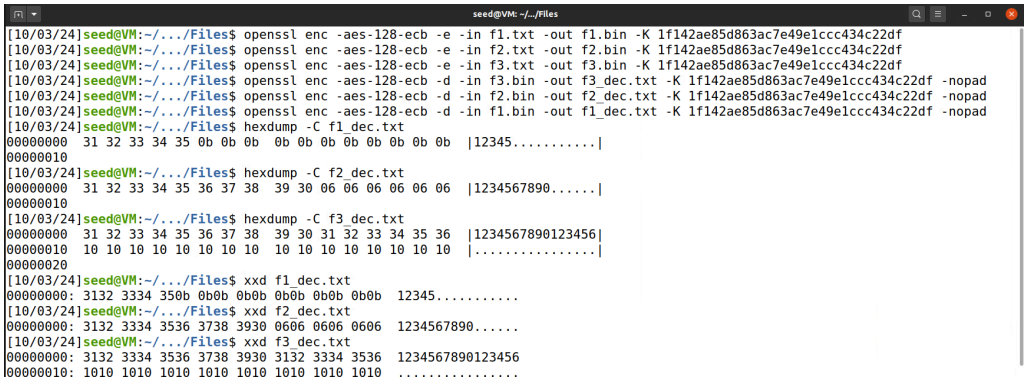
Let's test with the **Cipher Feedback** (**CFB**) encryption mode.




We kept the same files *f2.txt* and *f3.txt*, and we notice that even though *f2* is 10 bytes, its last block did not require padding. Similarly, *f3*, which is 16 bytes, remained the same.
We can conclude that the Cipher Feedback encryption/decryption mode does not require the use of padding.

### 4.3 - Electronic Codebook (ECB)

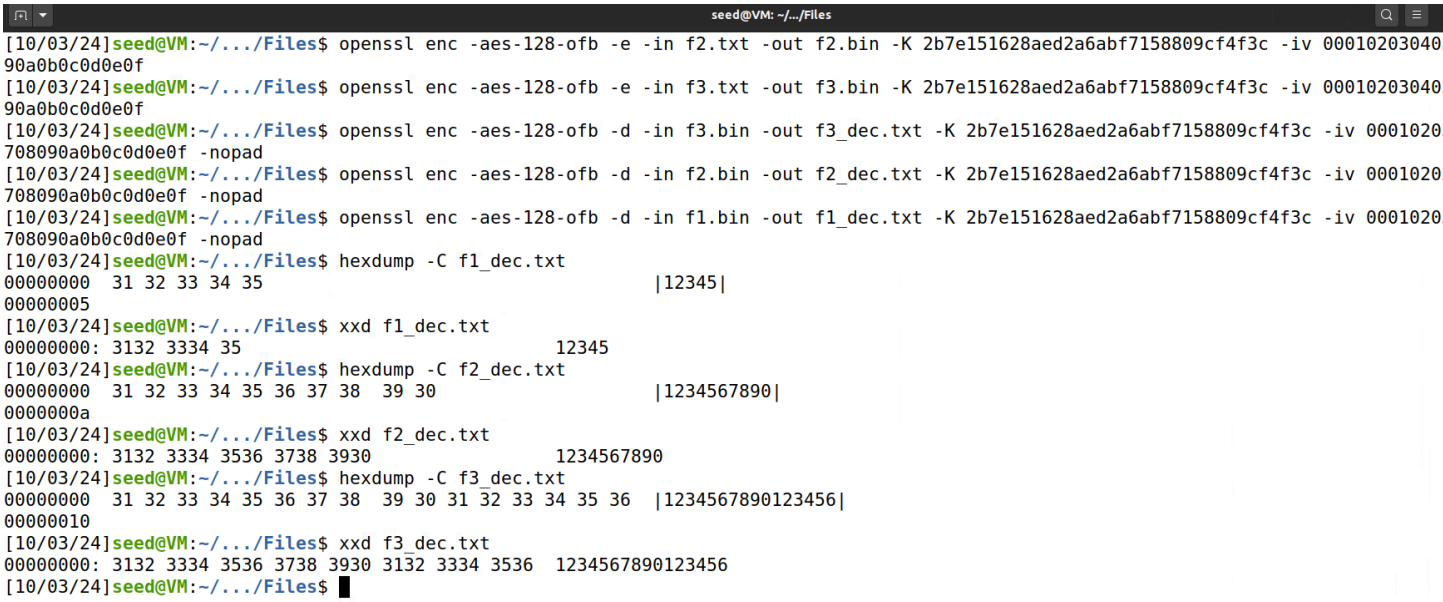
Here is the test of the **Electronic Codebook** (**ECB**) cipher type.



We notice that the decrypted files *f1* and *f2* require padding.

### 4.4 - Output Feedback (OFB)

Here is the test of the **Output Feedback (OFB)** cipher type.




We notice here that for the files *f1* and *f2*, no padding is added to their last block.

In conclusion of task 4, the **CBC** and **ECB** modes require padding, and the size of their decrypted files must be multiples of 16 bytes.
However, the **OFB** and **CFB** modes did not require padding.

**Task 5**

Original File:

```
< aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaa
```

## Task 6

### 6.1 - IV Experiment

```
[10/03/24]seed@VM:~/.../Labsetup$ echo -n "hello12345" > task.txt
[10/03/24]seed@VM:~/.../Labsetup$ openssl enc  -aes-128-cbc  -e -in task.txt -out task_same1.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[10/03/24]seed@VM:~/.../Labsetup$ openssl enc  -aes-128-cbc  -e -in task.txt -out task_same2.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[10/03/24]seed@VM:~/.../Labsetup$ diff task_same1.txt task_same2.txt
[10/03/24]seed@VM:~/.../Labsetup$ openssl enc  -aes-128-cbc  -e -in task.txt -out task_dif1.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060701
hex string is too short, padding with zero bytes to length
[10/03/24]seed@VM:~/.../Labsetup$ openssl enc  -aes-128-cbc  -e -in task.txt -out task_dif2.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060702
hex string is too short, padding with zero bytes to length
[10/03/24]seed@VM:~/.../Labsetup$ diff task_dif1.txt task_dif2.txt
1c1
B8;▯▯▯▯!f6▯N▯
\ No newline at end of file
---
> ;▯▯▯▯▯^▯w▯▯
\ No newline at end of file
[10/03/24]seed@VM:~/.../Labsetup$
```

- Encrypting the plaintext file with the same IV results in the same cipher file, as evident with the diff command not outputting anything different between the two encrypted files
- In contrast, encrypting the plaintext file with a different IV makes a difference in the cipher files, evident with the diff command showing an output to those encrypted files

### 6.2 - Using the Same IV

```
[10/03/24]seed@VM:~/.../Files$ python3 sample_code.py
bytearray(b'\xf0\x01\xd8\xb6"\xa8\xb9\x99\x07\xb65>-#V\xc1\xd6~,\xe3V\xc3\xa4x')
Order: Launch a missile!
```

- By initially using the xor() function on p1 and c1,  it returned the buffer value (which emulates an IV value) shown on the first outputtec line
- Applying the xor() function to said buffer and c2 and decrypting it results in the message, being “Order: Launch a missile!”

This code was used to find the message

```
p1 = "This is a known message!"
c1 = "a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159"
p2 = ""
c2 = "bf73bcd3509299d566c35b5d450337e1bb175f903fafc159"


pb1 = bytes(p1, 'utf-8')
pb2 = bytes()
cb1 = bytearray.fromhex(c1)
cb2 = bytearray.fromhex(c2)

buffer = xor(pb1, cb1)
pb2 = xor(buffer, cb2)

p2 = pb2.decode('utf-8')

print (buffer)
print (p2)
```

### 6.3 - Using a Predictable IV

```
[10/03/24]seed@VM:~/.../Files$  nc 10.9.0.80 3000
Bob's secret message is either "Yes" or "No", without quotations.
Bob's ciphertex: 0ff487afacdf254dc99b6e83e04a24c1
The IV used    : 40e3adace0e312b852f928138ef8e945

Next IV        : e079f0fbe0e312b852f928138ef8e945
Your plaintext : f9ff2e
Your ciphertext: caa3b6ec2f0b079b92ac12834908717e

Next IV        : b29dc02ce1e312b852f928138ef8e945
Your plaintext :
```

```
[10/03/24]seed@VM:~/.../Files$ python3 sample_code.py
f9ff2e
```

- Using a slightly modified version of the code in 6.2 (pb2 is just decoded into hex vs plain text format), the inputs of the guess “Yes”, the IV used for the ciphertext and the IV of the next cipher text were inputted to be xor’d
- The output (seen on the screenshot on the top right) outputs a hexadecimal string. Inputting the string as the “Your plaintext” input will output a corresponding ciphertext.
- If the outputted ciphertext matches that of Bob’s ciphertext, that means the guess of “Yes” would be correct.
    - As seen on the screenshot above, it is not, meaning that the guess was wrong and the secret message was actually “No” in this instance

The below screenshot is the modified code used

```
p1 = "Yes"
c1 = "40e3adace0e312b852f928138ef8e945"
p2 = ""
c2 = "e079f0fbe0e312b852f928138ef8e945"


pb1 = bytes(p1, 'utf-8')
pb2 = bytes()
cb1 = bytearray.fromhex(c1)
cb2 = bytearray.fromhex(c2)

buffer = xor(pb1, cb1)
pb2 = xor(buffer, cb2)
print(pb2.hex())
```

**Task 7**

Program getKey.py

```
#!/usr/bin/python3
from sys import argv
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad

_, first, second, third = argv

# Ensure the input is the correct length
if len(first) != 21:
    raise ValueError("The first argument must be 21 characters long.")

# Convert inputs to the appropriate byte representations
data = bytearray(first, encoding='utf-8')
ciphertext = bytearray.fromhex(second)
iv = bytearray.fromhex(third)

# Open the file and read all the keys
with open('./words.txt', 'r') as f:
    keys = f.read().splitlines()

# Loop over each key
for k in keys:
    # Adjust key length to ensure it is exactly 16 characters long
    if len(k) <= 16:
        key = (k + '#' * 16)[:16]
    else:
        continue

    # Create a new AES cipher using the key and IV
    cipher = AES.new(bytearray(key, encoding='utf-8'), AES.MODE_CBC, iv=iv)
```

```
    # Encrypt the data and compare it to the ciphertext
    encrypted_guess = cipher.encrypt(pad(data, 16))

    if encrypted_guess == ciphertext:
        print(f"Found the key: {key}")
        exit(0)
```

Running the program:

```
[10/03/24]seed@VM:~/.../Files$ findKey.py "This is a top secret." 764aa26b55a4da
654df6b19e4bce00f4ed05e09346fb0e762583cb7da2ac93a2 aabbccddeeff00998877665544332
211
```

Result:

```
Syracuse########
```