

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 vgxyz hmrty vbynh ytmq ixur qyhvurn
vlvhpq yhme ytn gvrnrh bnniq imsn v uxuvrnvuhmvu yxx

ytn vlvhpq hvan lvq gxssnupnp gd ytn pncmqn xb tvhfdn lnuqynmu vy myq xzyqny
vup ytn veevhuuy mceixqmxu xb tmq bmic axceved vy ytn nup vup my lvq qtvenp gd
ytn ncnhrruan xb cnyxx ymcnq ze givasrxtlu eximymaq vhcavupd vaymfmqc vup
v uyvmxuvi axufnhqvymxu vq ghmnv vup cvp vq v bnfnd phnvc vgxyz ltntynh ytnhn
xzrty yx gn v ehngmpnuy lmubhnd ytn gnqxu pmpuy ozqy qnnc nkyhv ixur my lvq
nkyhv ixur gnazqn ytn xqavhq lnhn cxfnp yx ytn bmqy lnnsnup mu cvhat yx
vfxmp axubimaymur lmyt ytn aixqmur anhnxcud xb ytn lmuynh xidcemaq ytvusq
ednxuratvur

xun gmr jznqymxu qzhxzupmur ytmq dnvq vavpncd vlvhpq mq txl xh mb ytn
anhnxcud lmiil vpphnq cnyxx ngenamviid vbynh ytn rxipnu rixgnq ltmat gnacvn
v ozgmivuy axcmurxzy evhyd bxh ymcnq ze ytn cxfncnuy qenvhtnvpnp gd
exlnhbzi txiidlxxp lxcnu ltx tniemp hvmqn cmiimxuq xb pxilvhq yx bmrty qnkzvi
tvhqgcnuy vhxzup ytn axzuyhd

qmrurvumr ytnmh qzeexhy rxipnu rixgnq vynyupnnp qlvytnp ytnqcnifnq mu givas
qexhymp iveni emuq vup qzupnp xbb vgxyz qnkmaq exlnh mcgviuanaq bhxc ytn hnp
aheny vup ytn qyvrn xu ytn vmh n lvq avilnp xzy vgxyz evd munjzmyd vbynh
mqy bxhcnh vuatxh avyy qpindh jzmy xuan qtn invhupn ytyv qtn lvq cvsmur bvq
inqq ytvu v cvin axtxqy vup pzhmur ytn anhnxcud uvyvimm exhyvuy yxxs v gizuy
vup qymqbdmur pmr vy ytn viicvin hxqynh xb uxcmuynp pmhnayxhq txl axzip
ytyv gn yxeenp

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

lxcnu mufxifnp mu ymcnq ze qvmp ytyv viytxzrt ytn rixgnq qmrumbnnp ytn
mumymvymfnq ivzuat ytnnd unfnh muynupnp my yx gn ozqy vu vlvhpq gnqxu
avcevmru xh xun ytyv gnacvn vqxxamvynp xuid lmyt hnpavheny vaymxuq muqynvp
v qexsnqlxcvu qvmp ytn rhxze mq lxhsmur gntmup aixqnp pxhq vup tvq qmuan
vcvqnp cmiimxu bxh mq inrvi pnbuqn bzup ltmat vbynh ytn rixgnq lvq
bixxnpn lmyt ytxzqvupq xb pxuvymxuq xb xh inqg bhxc enxein mu qxcn
axzuyhmq

ux avii yx lnhv givas rxluq lnuuy xzy mu vpfvuan xb ytn xqavhq ytxzrt ytn
cxfncnuy lmiil vixqy anhyvmuid gn hnbhnuanp gnbxhn vup pzhmur ytn anhnxcud
nqenamviid qmuan fxavi cnyxx qzeexhyndq imsn vtind ozpp ivzhv pnhu vup
umaxin smpcvu vhn qatnpzinp ehngnyunhq

vuxytnh bnvyzhn xb ytmq gnqxu ux xun hnviiid suxlt ltx mq rxmur yx lmu gnqy
emayzhn vhrzvgid ytmq tveenug v ixy xb ytn ymcn muvhrzvgid ytn umvigmynh
uyhvymfn xuid qnhfnq ytn vlvhpq tden cvatmun gzy xbynu ytn enxein bxhnavqymur
ytn hvan qxaviinp xqavhixrmqyq avu cvsn xuid npzavynp rznnqng

ytn lvd ytn vavpncd yvgzivynq ytn gmr lmuunh pxnquy tnie mu nfnhd xytnh
aynrxhd ytn uxcmun lmyt ytn cxqy fxynq lmuq gzy mu ytn gnqy emayzhn
aynrxhd fxynhq vhn vqsnp yx imqy ytnmh yxe cxfmq mu ehnbhnuymvi xhpnh mb v
cxfmn rnyq cxhn ytvu enhanuy xb ytn bmqyeivan fxynq my lmuq ltnu ux
cxfmn cvuvrnq ytyv ytn xun lmyt ytn bnlqy bmqyeivan fxynq mq nimcmuynp vup

mqy fxynq vhn hnpmqyhmgyznp yx ytn cxfmq ytyv rvhunhp ytn nimcmuynp gviiyx
qnxapeivan fxynq vup ytmq axuymuznq zuymi v lmuunh ncnhnq

my mq vii ynhamgid axubzqmur gzy veevhuuyid ytn axuqnuqzq bvfxbmyn axcnq xzy
vtnpv mu ytn nup ytmq cnvuy ytyv nupxbqnvqxu vlvhpq atvynh mufvbmvgid
mufxifnq yxhyznp qenazivymxu vgxyz ltmat bmic lxzip cxqy imsnid gn fxynhq
qnxap xh ytmhp bvfxbmyn vup ytn njzviid yxhyznp axuaizqmxuq vgxyz ltmat
bmic cmrty ehnfvmi

mu my lvq v yxqqze gnylnnu gdxtxp vup ytn nfnuyzvi lmuunh gmhpcvu
mu lmyt ixyq xb nkenhyq gnyymur xu ytn hnfnuvuy xh ytn gmr qtxhy ytn
ehmwn lnuuy yx qexyimrty lvqy dnvh unvhid vii ytn bxhnavqynhq pnaivhnp iv
iv ivup ytn ehngzceymfn lmuunh vup bxh ylx vup v tvib cmuzynq ytnnd lnhn
axhlnay gnbxhn vu nufnixen quvzb lvq hnfnvinp vup ytn hmrtvbi lmuunh
cxxiimrty lvq ahxlunp

ytmq dnvh vlvhpq lvyatnhq vhn zunjzviid pmfmpnp gnylnnu ythnn gmiigxvhpp
xyqmpn nggmr cmqpxzhm ytn bvfxbmyn vup ytn qtven xb lvynd ltmat mq
ytn gvrnrh ehnpmaymxu lmyt v bnl bxhnavqymur v tvmi cvhd lmu bxh rny xzy

gzy vii xb ytxqn bmicq tvfn tmqyxhmavi xqavhfxymur evyynhuq vrvmuqy ytnv ytn
qtven xb lvynd tvq uxcmuynmuq cxhn ytvu vud xytnh bmic vup lvq viqx
ucvnp ytn dnvq gnqy gd ytn ehxpzanhq vup pmhnayxhq rzmiqp dny my lvq uxy
uxcmuynp bxh v qahnnu vayxhq rzmiip vlvhp bxh gnqy nuqncgin vup ux bmic tvq
lxu gnqy emayzhn lmytxzy ehnmfxzqid ivupmur vy invqy ytn vayxhq uxcmuynmuq
qmuan ghvfnfnvhy mu ytmq dnvh ytn gnqy nuqncgin qvr nupnp ze rxmur yx

ythnn gmiigxvhpp ltmat mq qmrumbavuy gnazqn vayxhq cvsn ze ytn vavpncdq
ivhrnqy ghvuat ytyv bmic lmin pmfmqfn viqx lxu ytn gnqy phvcv rxipnu rixgn
vup ytn gbvyv gzy mqy bmiccvsnh cvhymu capxuvrt lvq uxy uxcmuynp bxh gnqy
pmhnayxh vup vevhy bhxc vhrx cxfmq ytyv ivup gnqy emayzhn lmytxzy viqx
nvhumur gnqy pmhnayxh uxcmuynmuq vhn bnl vup bvq gnylnnu
```

[10/02/24]seed@VM:~/.../Files\$ █

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

```
-----  
1-gram (top 20):
```

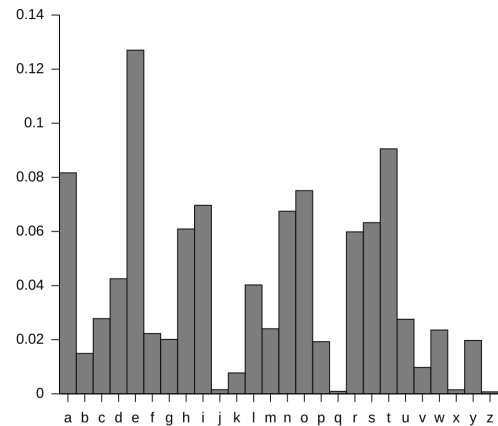
```
n: 488  
y: 373  
v: 348  
x: 291  
u: 280  
q: 276  
m: 264  
h: 235  
t: 183  
i: 166  
p: 156  
a: 116  
c: 104  
z: 95  
l: 90  
g: 83  
b: 83  
r: 82  
e: 76  
d: 59
```

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  
ytn: 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**

```
seed@VM: ~/.../Files
[10/02/24]seed@VM:~/.../Files$ tr 'ytnxvhqildmurzaegpcsbfkj' 'THE0ARSLWYINGUCPB
DMKFVXJQ' < ciphertext.txt > out.txt
[10/02/24]seed@VM:~/.../Files$ cat out.txt
THE OSCARS TURN ON SUNDAY WHICH SEEMS ABOUT RIGHT AFTER THIS LONG STRANGE
AWARDS TRIP THE BAGGER FEELS LIKE A NONAGENARIAN TOO

THE AWARDS RACE WAS BOOKENDED BY THE DEMISE OF HARVEY WEINSTEIN AT ITS OUTSET
AND THE APPARENT IMPLOSION OF HIS FILM COMPANY AT THE END AND IT WAS SHAPED BY
THE EMERGENCE OF METOO TIMES UP BLACKGOWN POLITICS ARMCANDY ACTIVISM AND
A NATIONAL CONVERSATION AS BRIEF AND MAD AS A FEVER DREAM ABOUT WHETHER THERE
OUGHT TO BE A PRESIDENT WINFREY THE SEASON DIDNT JUST SEEM EXTRA LONG IT WAS
EXTRA LONG BECAUSE THE OSCARS WERE MOVED TO THE FIRST WEEKEND IN MARCH TO
AVOID CONFLICTING WITH THE CLOSING CEREMONY OF THE WINTER OLYMPICS THANKS
PYEONGCHANG

ONE BIG QUESTION SURROUNDING THIS YEARS ACADEMY AWARDS IS HOW OR IF THE
CEREMONY WILL ADDRESS METOO ESPECIALLY AFTER THE GOLDEN GLOBES WHICH BECAME
A JUBILANT COMINGOUT PARTY FOR TIMES UP THE MOVEMENT SPEARHEADED BY
POWERFUL HOLLYWOOD WOMEN WHO HELPED RAISE MILLIONS OF DOLLARS TO FIGHT SEXUAL
HARASSMENT AROUND THE COUNTRY
```

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).

```
seed@VM: ~/.../Files
[10/02/24]seed@VM:~/.../Files$ cat plaintext1
My name is Thylane. I study computer science at TMU.
[10/02/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in plaintext1 -out cipher.bin -K aa2ba79813f19ac270f8cdcc86f542ac -iv d1cfa97860cc34528784e7b79b587667
[10/02/24]seed@VM:~/.../Files$ cat cipher.bin
9[007000#V0000000j      0
>0+00G00mZ.4R000k0
[10/02/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -in cipher.bin -out decrypted.txt -K aa2ba79813f19ac270f8cdcc86f542ac -iv d1cfa97860cc34528784e7b79b587667
[10/02/24]seed@VM:~/.../Files$ cat decrypted.txt
My name is Thylane. I study computer science at TMU.
[10/02/24]seed@VM:~/.../Files$
```

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.

```
seed@VM: ~/.../Files
[10/02/24]seed@VM:~/.../Files$ cat plaintext2
Cryptography is essential for protecting sensitive information.
It helps ensure the confidentiality and integrity of data. By using strong algorithms, we can secure our communications and safeguard our privacy.

[10/02/24]seed@VM:~/.../Files$ openssl enc -bf-cbc -e -in plaintext2 -out ciphertext.bin -K 20f50bc334278ed75e6e4a09bafb5573 -iv 2d39c728d1a9420f
[10/02/24]seed@VM:~/.../Files$ cat ciphertext.bin
0S000S0P00Z000H0yF006y0^g0x:0U;00i?000-0I00G0050w000V0000a00R?zT0
t0: 0000?000+0o[000wZP0H0H0N)00PI.0t700lwT+005月Ks0Ln)kz00M0000000
[10/02/24]seed@VM:~/.../Files$ openssl enc -bf-cbc -d -in ciphertext.bin -out decrypted.txt -K 20f50bc334278ed75e6e4a09bafb5573 -iv 2d39c728d1a9420f
[10/02/24]seed@VM:~/.../Files$ cat decrypted.txt
Cryptography is essential for protecting sensitive information.
It helps ensure the confidentiality and integrity of data. By using strong algorithms, we can secure our communications and safeguard our privacy.

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

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.

```
[10/02/24]seed@VM: ~/.../Files
[10/02/24]seed@VM:~/.../Files$ cat plaintext3
Hiking is a fantastic way to connect with nature and improve physical health. Spending time outdoors can reduce stress, enhance mood, and boost overall well-being. It provides an opportunity to explore beautiful landscapes, encounter wildlife, and enjoy fresh air. Moreover, hiking can be a great social activity, allowing friends and family to bond over shared experiences. Whether on a challenging mountain trail or a leisurely walk in the park, hiking offers numerous benefits for both the body and mind.
[10/02/24]seed@VM:~/.../Files$ openssl enc -des-cbc -e -in plaintext3 -out ciphertext.bin -K c5f26df3e3ae86b1 -iv 4a2eaf63d43d1ea4
[10/02/24]seed@VM:~/.../Files$ cat ciphertext.bin
0A00m0|0000000dY0W5Lg=0A0)<d0d'.0
0*JU0RqE00000,Y0|000P%00' {5C000606|d0Is0|000D0j#00=0hk0000Y}0?fy"L0}00H<000{ _Blw      00s0}}0{00(0|7000{000}00+0z00"0*U0\00d005J0000t|0000
J<7y0002y0504 0#{0<+0M00c00003D0000Z000c000000
@000010U0"00r0Is*
-0u0N00W0x000000S0lr000000}00h00-00m0\T00|u00X0
診,09,)>000U0Y:p00060Zj$)0@u.0000_#0;HIN"o0#(;00A0ieX00000000
[10/02/24]seed@VM:~/.../Files$ openssl enc -des-cbc -d -in ciphertext.bin -out decrypted.txt -K c5f26df3e3ae86b1 -iv 4a2eaf63d43d1ea4
[10/02/24]seed@VM:~/.../Files$ cat decrypted.txt
Hiking is a fantastic way to connect with nature and improve physical health. Spending time outdoors can reduce stress, enhance mood, and boost overall well-being. It provides an opportunity to explore beautiful landscapes, encounter wildlife, and enjoy fresh air. Moreover, hiking can be a great social activity, allowing friends and family to bond over shared experiences. Whether on a challenging mountain trail or a leisurely walk in the park, hiking offers numerous benefits for both the body and mind.
[10/02/24]seed@VM:~/.../Files$
```

Task 3 - Encryption Mode – ECB vs. CBC

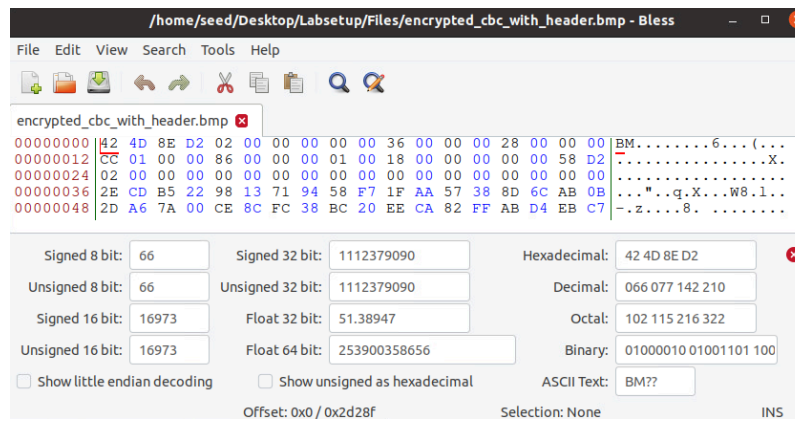
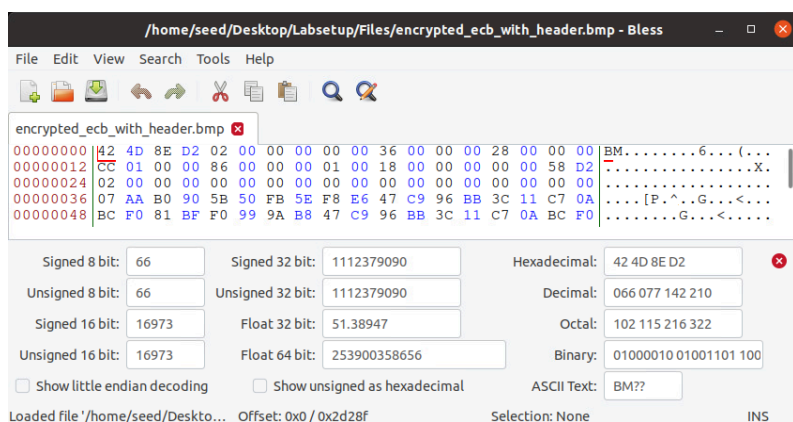
Here is the encrypted image, first with ECB, then with CBC :

```
[10/02/24]seed@VM: ~/.../Files
[10/02/24]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in pic_original.bmp -out encrypted_ecb.bmp -K 55d909cf2694ce9ebcdc4b6580eb5c43
[10/02/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in pic_original.bmp -out encrypted_cbc.bmp -K 687bef4c128d1ab12438f7d3b30a82e0 -iv 370ada72b7e4587edb4c1222ca2991bf
[10/02/24]seed@VM:~/.../Files$
```

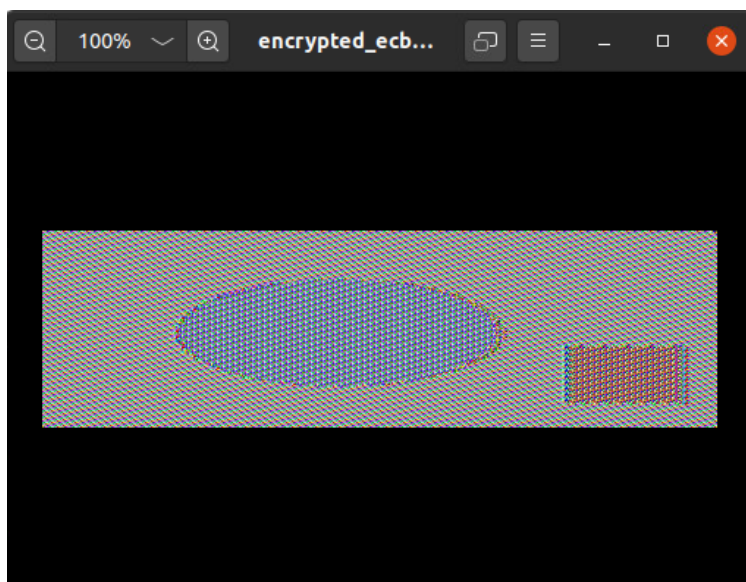
Modification of the image header :

```
[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.bmp
[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.bmp
```

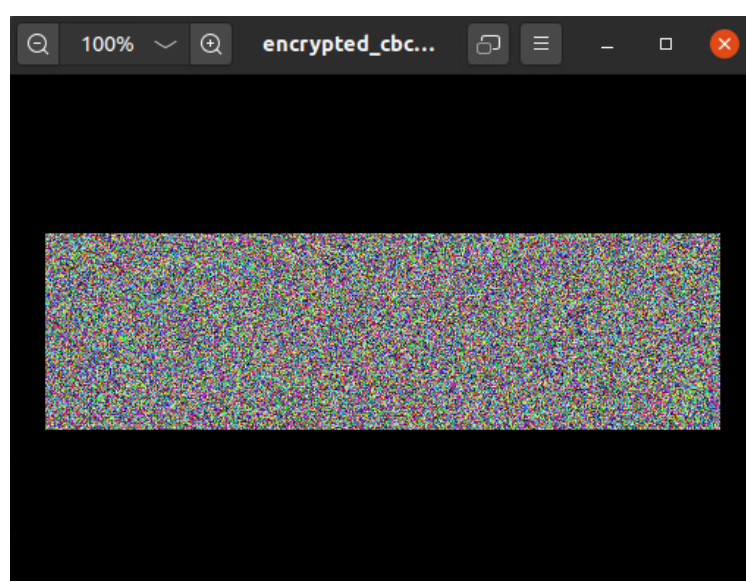

Displaying images with bless :



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

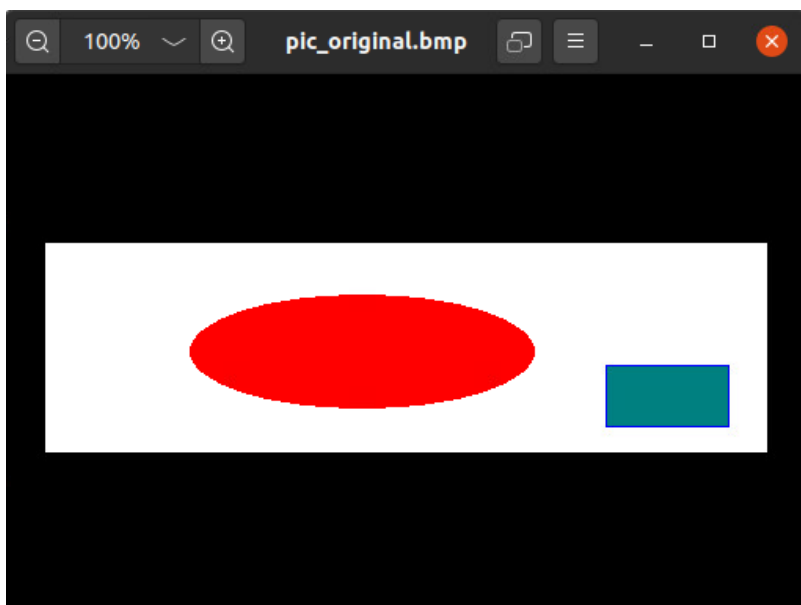


eog encrypted_ecb_with_header.bmp



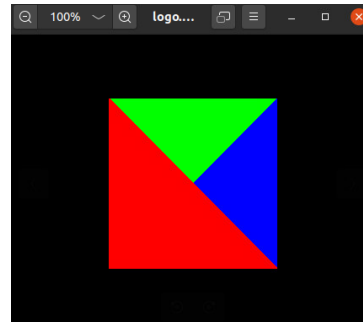
eog encrypted_cbc_with_header.bmp

Original image :

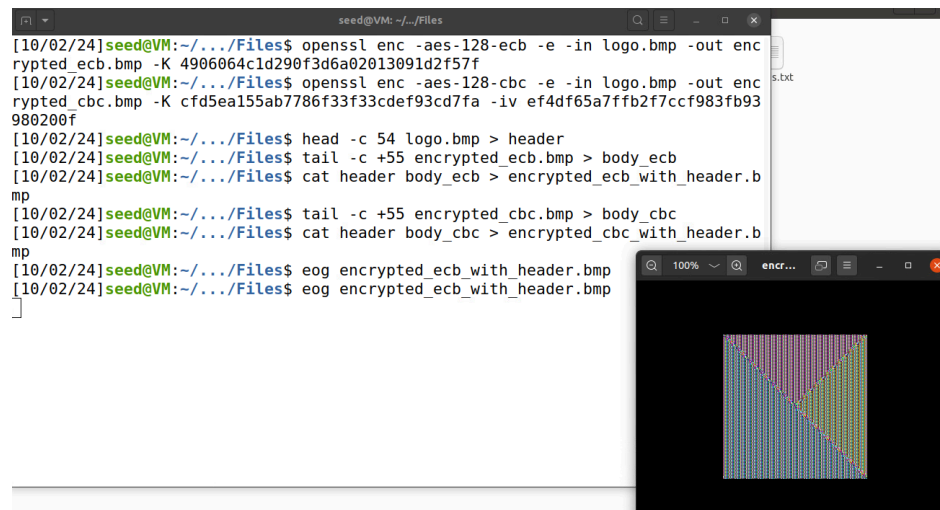


With ECB encryption, it is very easy to distinguish the elements present in the image. The encryption is poor because the pixels of the same color are all coded in the same color, as if there were just a filter. On the other hand, with CBC encryption, the image has nothing to do with the original; nothing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

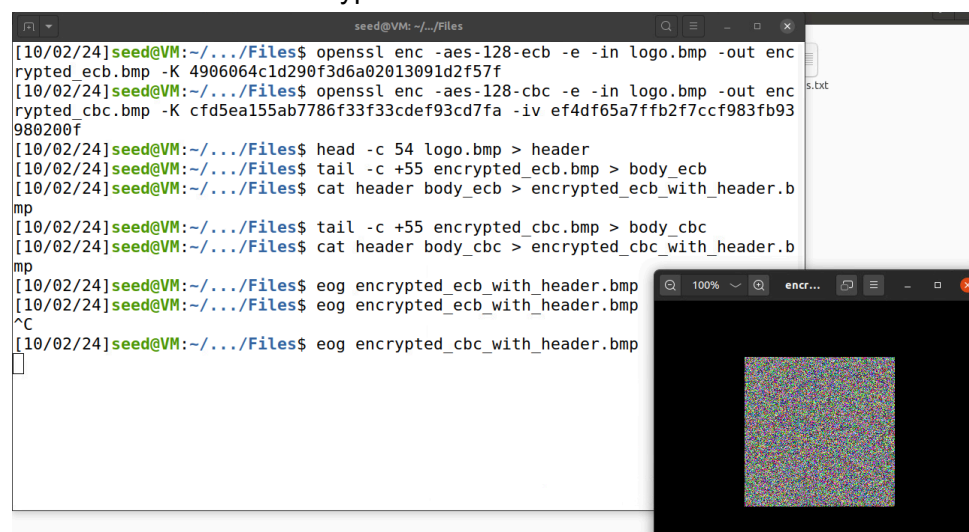
Here is another example with this original image



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)**.

```
seed@VM: ~/.../Files
[10/03/24]seed@VM:~/.../Files$ echo -n "123456" > f1.txt
[10/03/24]seed@VM:~/.../Files$ echo -n "1234567890" > f2.txt
[10/03/24]seed@VM:~/.../Files$ echo -n "1234567890123456" > f3.txt
[10/03/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in f3.txt -out f3.bin -K 001122334455667788895a65ccd4ee56 -iv 01020676438556765756767687868546
[10/03/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in f2.txt -out f2.bin -K 001122334455667788895a65ccd4ee56 -iv 01020676438556765756767687868546
[10/03/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in f1.txt -out f1.bin -K 001122334455667788895a65ccd4ee56 -iv 01020676438556765756767687868546
[10/03/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -in f1.bin -out f1_dec.txt -K 001122334455667788895a65ccd4ee56 -iv 01020676438556765756767687868546 -nopad
[10/03/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -in f2.bin -out f2_dec.txt -K 001122334455667788895a65ccd4ee56 -iv 01020676438556765756767687868546 -nopad
[10/03/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -in f3.bin -out f3_dec.txt -K 001122334455667788895a65ccd4ee56 -iv 01020676438556765756767687868546 -nopad
[10/03/24]seed@VM:~/.../Files$
```

Here are the hexadecimal analyses of the original files:

```
seed@VM: ~/.../Files
[10/03/24]seed@VM:~/.../Files$ hexdump -C f1.txt
00000000 31 32 33 34 35 36                                     |123456|
00000006
[10/03/24]seed@VM:~/.../Files$ xxd f1.txt
00000000: 3132 3334 3536                                     123456
[10/03/24]seed@VM:~/.../Files$ hexdump -C f2.txt
00000000 31 32 33 34 35 36 37 38 39 30                         |1234567890|
0000000a
[10/03/24]seed@VM:~/.../Files$ xxd f2.txt
00000000: 3132 3334 3536 3738 3930                         1234567890
[10/03/24]seed@VM:~/.../Files$ hexdump -C f3.txt
00000000 31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36         |1234567890123456|
00000010
[10/03/24]seed@VM:~/.../Files$ xxd f3.txt
00000000: 3132 3334 3536 3738 3930 3132 3334 3536         1234567890123456
[10/03/24]seed@VM:~/.../Files$
```

Here is the analysis of the files after decryption:

```
seed@VM: ~/.../Files
[10/03/24]seed@VM:~/.../Files$ hexdump -C f1_dec.txt
00000000 31 32 33 34 35 36 0a 0a 0a 0a 0a 0a 0a 0a 0a 0a         |123456.....|
00000010
[10/03/24]seed@VM:~/.../Files$ xxd f1_dec.txt
00000000: 3132 3334 3536 0a0a 0a0a 0a0a 0a0a 0a0a         123456.....
[10/03/24]seed@VM:~/.../Files$ hexdump -C f2_dec.txt
00000000 31 32 33 34 35 36 37 38 39 30 06 06 06 06 06 06         |1234567890.....|
00000010
[10/03/24]seed@VM:~/.../Files$ xxd f2_dec.txt
00000000: 3132 3334 3536 3738 3930 0606 0606 0606         1234567890.....
[10/03/24]seed@VM:~/.../Files$ hexdump -C f3_dec.txt
00000000 31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36         |1234567890123456|
00000010 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10         |.....|
00000020
[10/03/24]seed@VM:~/.../Files$ xxd f3 dec.txt
```


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.

```
seed@VM: ~/Files
[10/03/24]seed@VM:~/Files$ openssl enc -aes-128-cfb -e -in f2.txt -out f2.bin -K 001122334455667788895a65ccd4ee56 -iv 01020676438556765756767687868546
[10/03/24]seed@VM:~/Files$ openssl enc -aes-128-cfb -e -in f3.txt -out f3.bin -K 001122334455667788895a65ccd4ee56 -iv 01020676438556765756767687868546
[10/03/24]seed@VM:~/Files$ openssl enc -aes-128-cfb -d -in f2.bin -out f2_dec.txt -K 001122334455667788895a65ccd4ee56 -iv 01020676438556765756767687868546 -nopad
[10/03/24]seed@VM:~/Files$ openssl enc -aes-128-cfb -d -in f3.bin -out f3_dec.txt -K 001122334455667788895a65ccd4ee56 -iv 01020676438556765756767687868546 -nopad
[10/03/24]seed@VM:~/Files$ hexdump -C f2_dec.txt
00000000 31 32 33 34 35 36 37 38 39 30 |1234567890|
0000000a
[10/03/24]seed@VM:~/Files$ xxd f2_dec.txt
00000000: 3132 3334 3536 3738 3930 1234567890
[10/03/24]seed@VM:~/Files$ hexdump -C f3_dec.txt
00000000 31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36 |1234567890123456|
00000010
[10/03/24]seed@VM:~/Files$ xxd f3_dec.txt
00000000: 3132 3334 3536 3738 3930 3132 3334 3536 1234567890123456
[10/03/24]seed@VM:~/Files$
```

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.

```
seed@VM: ~/Files
[10/03/24]seed@VM:~/Files$ openssl enc -aes-128-ecb -e -in f1.txt -out f1.bin -K 1f142ae85d863ac7e49e1ccc434c22df
[10/03/24]seed@VM:~/Files$ openssl enc -aes-128-ecb -e -in f2.txt -out f2.bin -K 1f142ae85d863ac7e49e1ccc434c22df
[10/03/24]seed@VM:~/Files$ openssl enc -aes-128-ecb -e -in f3.txt -out f3.bin -K 1f142ae85d863ac7e49e1ccc434c22df
[10/03/24]seed@VM:~/Files$ openssl enc -aes-128-ecb -d -in f3.bin -out f3_dec.txt -K 1f142ae85d863ac7e49e1ccc434c22df -nopad
[10/03/24]seed@VM:~/Files$ openssl enc -aes-128-ecb -d -in f2.bin -out f2_dec.txt -K 1f142ae85d863ac7e49e1ccc434c22df -nopad
[10/03/24]seed@VM:~/Files$ openssl enc -aes-128-ecb -d -in f1.bin -out f1_dec.txt -K 1f142ae85d863ac7e49e1ccc434c22df -nopad
[10/03/24]seed@VM:~/Files$ hexdump -C f1_dec.txt
00000000 31 32 33 34 35 0b 0b 0b 0b 0b 0b 0b 0b 0b 0b 0b |12345.....|
00000010
[10/03/24]seed@VM:~/Files$ hexdump -C f2_dec.txt
00000000 31 32 33 34 35 36 37 38 39 30 06 06 06 06 06 06 |1234567890.....|
00000010
[10/03/24]seed@VM:~/Files$ hexdump -C f3_dec.txt
00000000 31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36 |1234567890123456|
00000010 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 |.....|
00000020
[10/03/24]seed@VM:~/Files$ xxd f1_dec.txt
00000000: 3132 3334 350b 0b0b 0b0b 0b0b 0b0b 0b0b 12345.....
[10/03/24]seed@VM:~/Files$ xxd f2_dec.txt
00000000: 3132 3334 3536 3738 3930 0606 0606 0606 1234567890.....
[10/03/24]seed@VM:~/Files$ xxd f3_dec.txt
00000000: 3132 3334 3536 3738 3930 3132 3334 3536 1234567890123456
00000010: 1010 1010 1010 1010 1010 1010 1010 1010 .....
```

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.

```
seed@VM: ~/.../Files
[10/03/24]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -e -in f2.txt -out f2.bin -K 2b7e151628aed2a6abf7158809cf4f3c -iv 0001020304090a0b0c0d0e0f
[10/03/24]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -e -in f3.txt -out f3.bin -K 2b7e151628aed2a6abf7158809cf4f3c -iv 0001020304090a0b0c0d0e0f
[10/03/24]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -d -in f3.bin -out f3_dec.txt -K 2b7e151628aed2a6abf7158809cf4f3c -iv 0001020304090a0b0c0d0e0f -nopad
[10/03/24]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -d -in f2.bin -out f2_dec.txt -K 2b7e151628aed2a6abf7158809cf4f3c -iv 0001020304090a0b0c0d0e0f -nopad
[10/03/24]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -d -in f1.bin -out f1_dec.txt -K 2b7e151628aed2a6abf7158809cf4f3c -iv 0001020304090a0b0c0d0e0f -nopad
[10/03/24]seed@VM:~/.../Files$ hexdump -C f1_dec.txt
00000000 31 32 33 34 35                                     |12345|
00000005
[10/03/24]seed@VM:~/.../Files$ xxd f1_dec.txt
00000000: 3132 3334 35                                     12345
[10/03/24]seed@VM:~/.../Files$ hexdump -C f2_dec.txt
00000000 31 32 33 34 35 36 37 38 39 30                     |1234567890|
0000000a
[10/03/24]seed@VM:~/.../Files$ xxd f2_dec.txt
00000000: 3132 3334 3536 3738 3930                     1234567890
[10/03/24]seed@VM:~/.../Files$ hexdump -C f3_dec.txt
00000000 31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36 |1234567890123456|
00000010
[10/03/24]seed@VM:~/.../Files$ xxd f3_dec.txt
00000000: 3132 3334 3536 3738 3930 3132 3334 3536 1234567890123456
[10/03/24]seed@VM:~/.../Files$
```

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.

Original File:

Encrypting the file the **ECB** (file has also been encrypted with **CBC**, **CFB**, **OFB** but not shown)

Encrypted files after 55th bite is corrupted

ECB

```
> aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa00020Z60-"}/I0aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

CBC

```
< aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa0000$, _00}1Iaaaaaaaaaaaaaaagaaaaaaaaaaaaaaaaaaaaaa
```

CFB

[illegible]

OFB

```
< aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaqaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

In conclusion, encrypting files in OFB mode will recover the most data after a file gets corrupted.

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;0000!f6000
\ No newline at end of file
---
> ;00000~0/00
\ 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 outputted 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 ciphertext: 0ff487afacdf254dc99b6e83e04a24c1
The IV used      : 40e3adace0e312b852f928138ef8e945

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

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

Next IV          : b29dc02ce1e312b852f928138ef8e945
Your plaintext   : 
```

- 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#####