Encryption Lab

CSE644

Internet Security

Chirag Sachdev

680231131

Task 1

Using frequency analysis to determine the key.

I used the following code to calculate the frequency percentage of the occurrences of the characters in the ciphertext.

Code:

```
fp = open("ciphertext.txt","r")
s=fp.read()
fp.close()
count={}
freq={}
sum=0
for i in range(97,123):
    ch=chr(i)
    count[ch]=s.count(ch)
    sum+=count[ch]

for i in range(97,123):
    ch=chr(i)
    freq[ch]=(count[ch]/sum)*100
    print(ch+"\t:\t %.2f"%freq[ch])
```

Output:

a	:	2.95	n	:	12.41
b	:	2.11	o	:	0.10
c	:	2.65	p	:	3.97
d	:	1.50	q	:	7.02
e	:	1.93	r	:	2.09
f	:	1.25	s	:	0.48
g	:	2.11	t	:	4.66
h	:	5.98	u	:	7.12
i	:	4.22	v	:	8.85
j	:	0.13	w	:	0.03
k	:	0.13	x	:	7.40
l	:	2.29	y	:	9.49
m	:	6.72	z	:	2.42

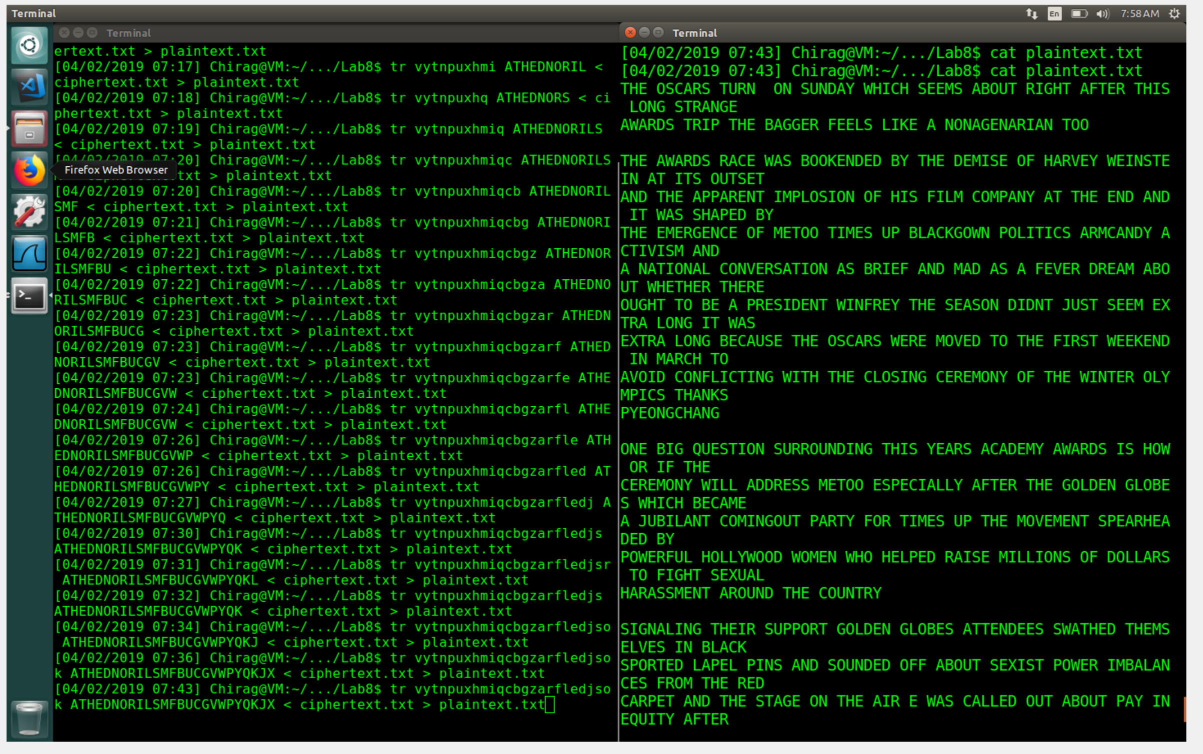
I then started substituting variables based on the occurrence, I started with substituting “v” as A since it was the only single occurring variable.

I kept repeating the process making sense of words.

I used the following command:

```
tr v A < ciphertext > plaintext.txt
```

Output:



Observation:

Repeating the process for all the alphabets, I found the key to be:

vgapnbrtmosicuxejhqyzflkdw

That means the plaintext was encrypted using the command:

tr abcdefghijklmnopqrstuvwxyz vgapnbrtmosicuxejhqyzflkdw < plaintext.txt > ciphertext.txt

Task 2

For this task I referred to the manual from the documentation of the Open SSL website. https://www.openssl.org/docs/man1.1.1/man1/enc.html.

For plaintext, I have used the Lab description stored in a file "plaintext.txt".

The contents of plaintext.txt are,

"The learning objective of this lab is for students to get familiar with the concepts in the secret-key encryption. After finishing the lab, students should be able to gain a first-hand experience on encryption algorithms, encryption modes, paddings, and initial vector. Moreover, students will be able to use tools and write programs to encrypt or decrypt messages."

To generate random keys, I used the python program to give a 8 character string for the DES Key(64 bits) and a 16 character string for a AES key(128 bits).

Code:

```
import random as rd
s="abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890"
des_lst=rd.sample(s,8)
des_key=''.join(des_lst)
aes_lst=rd.sample(s,16)
aes_key=''.join(aes_lst)
print("DES Key:", des_key)
print("AES Key:", aes_key)
```

Output:
The keys generated are:
DES Key: 34WVF7hI
AES Key: c4aP93MboJVQWZrX

I used the following initial vector for DES(64 bit) and converted it to HEX using an online ASCII to hex convertor:
ASCII: initialv
HEX: 696e697469616c76

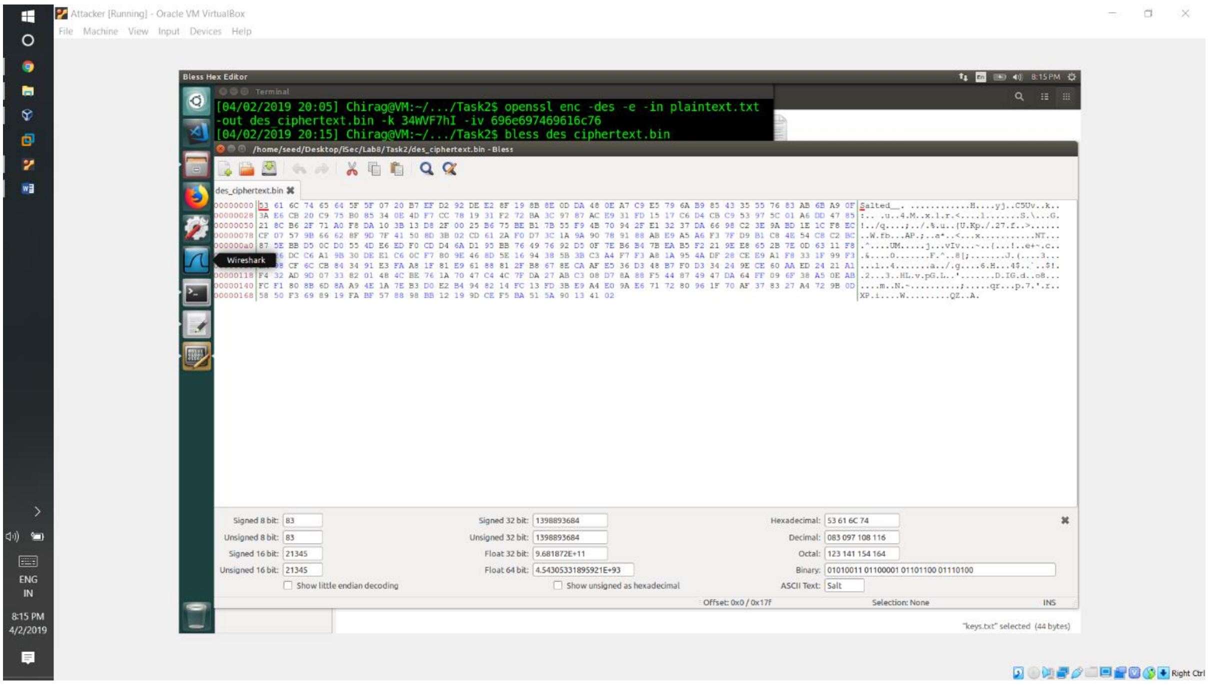
I used the following initial vector for AES(128 bit) and converted it to HEX using an online ASCII to HEX convertor:
ASCII: initialvectoraes
HEX: 696e697469616c766563746f72616573

ACII to HEX convertor:
https://www.rapidtables.com/convert/number/ascii-to-hex.html

1. DES:

Command: openssl enc -des -e -in plaintext.txt -out des_ciphertext.bin -k 34WVF7hI -iv 696e697469616c76

Output:

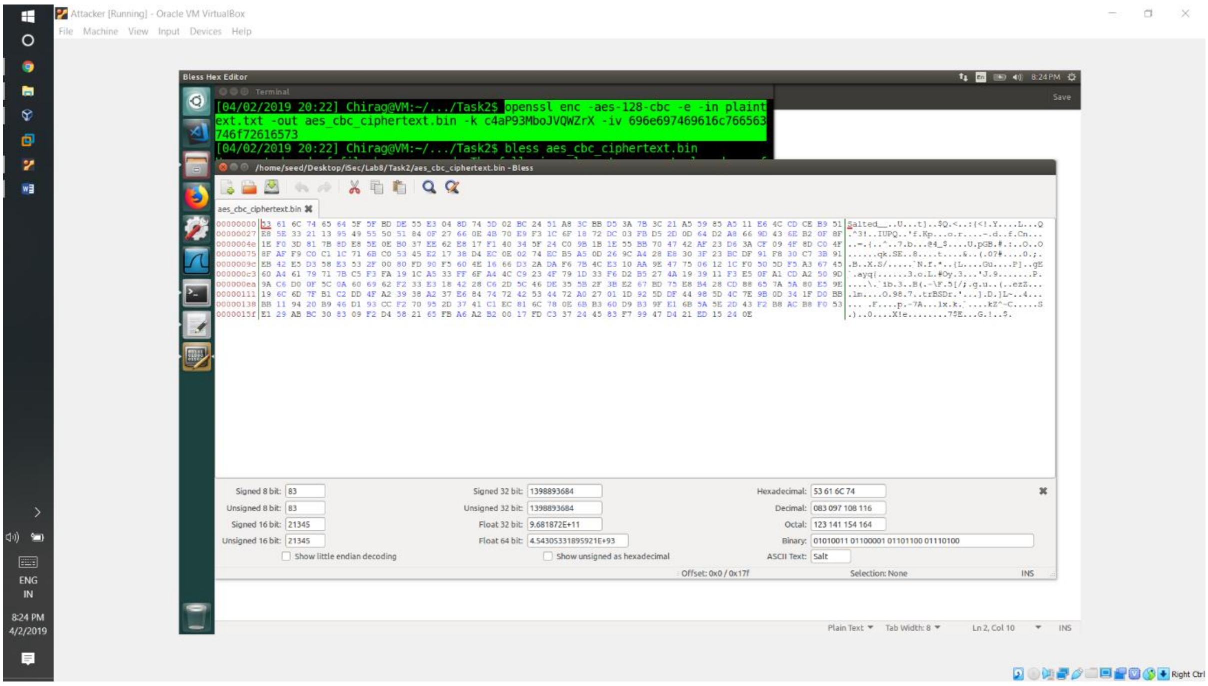


Observation: The output is stored in a binary file, to see the file we use the bless tool.

2. AES - Cipher block chaining:

Command: openssl enc -aes-128-cbc -e -in plaintext.txt -out aes_cbc_ciphertext.bin -k c4aP93MboJVQWZrX -iv 696e697469616c766563746f72616573

Output:



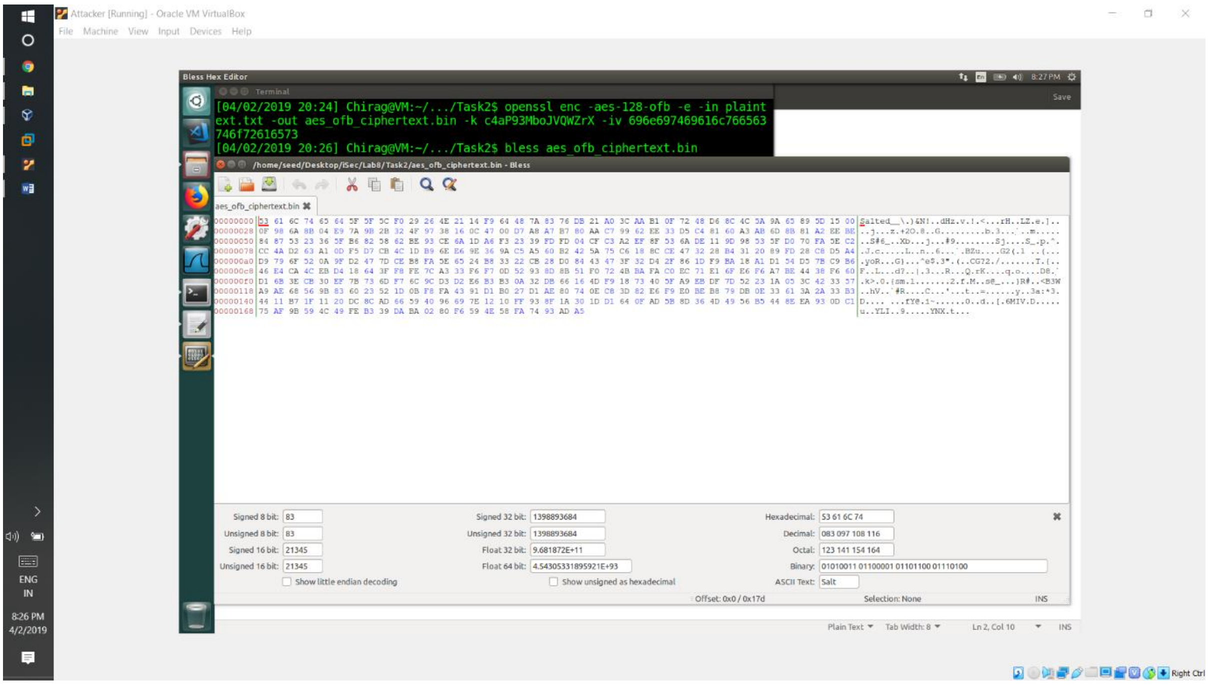
Observation:

The binary file can be seen using the bless tool.

3. AES - output feedback :

<u>Command</u>: openssl enc -aes-128-ofb -e -in plaintext.txt -out aes_ofb_ciphertext.bin -k c4aP93MboJVQWZrX -iv 696e697469616c766563746f72616573

Output:



Observation:

The output can be seen using the bless tool.

Task 3

Comparing ECB vs CBC for pic_original.bmp

<u>Key used</u>: c4aP93MboJVQWZrX

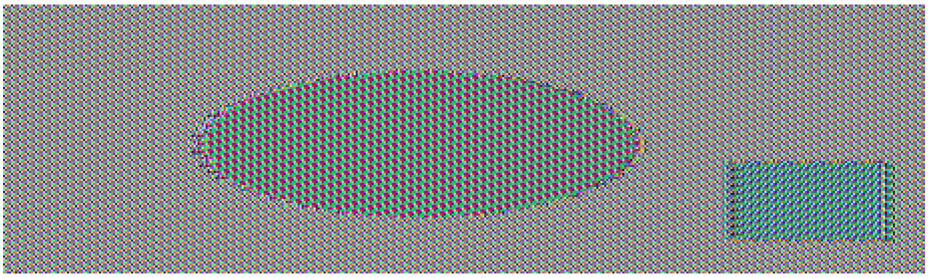
<u>IV used</u>: 696e697469616c766563746f72616573

a. Electronic code Block:

Command:

openssl enc -aes-128-ecb -e -in pic_original.bmp -out pic_ecb.bmp -k c4aP93MboJVQWZrX

Output:



Observation:

The image cannot be seen as the header is encrypted. We use the following commands to combine the encrypted data with the original header:

head -c 54 pic_original.bmp > header

tail -c +55 pic_ecb.bmp > body_ecb

cat header body_ecb > new_ecb.bmp

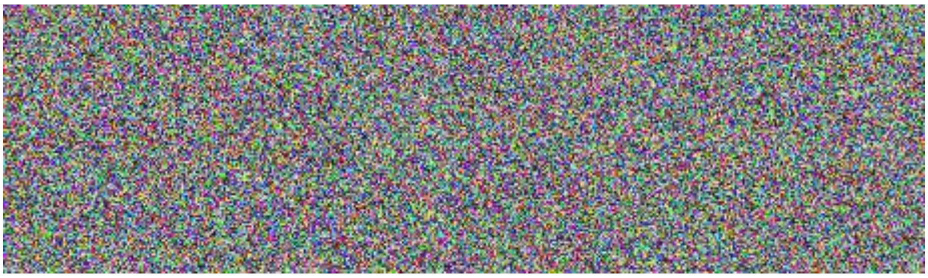
The picture can be distinguished into 3 sections, the oval, the square and the background.

b. Cipher block chaining:

Command

openssl enc -aes-128-cbc -e -in pic_original.bmp -out pic_cbc.bmp -k c4aP93MboJVQWZrX -iv 696e697469616c766563746f72616573

Output:



Observation:

The image cannot be seen as the header is encrypted. We use the following commands to combine the encrypted data with the original header:

head -c 54 pic_original.bmp

tail -c +55 > body_cbc

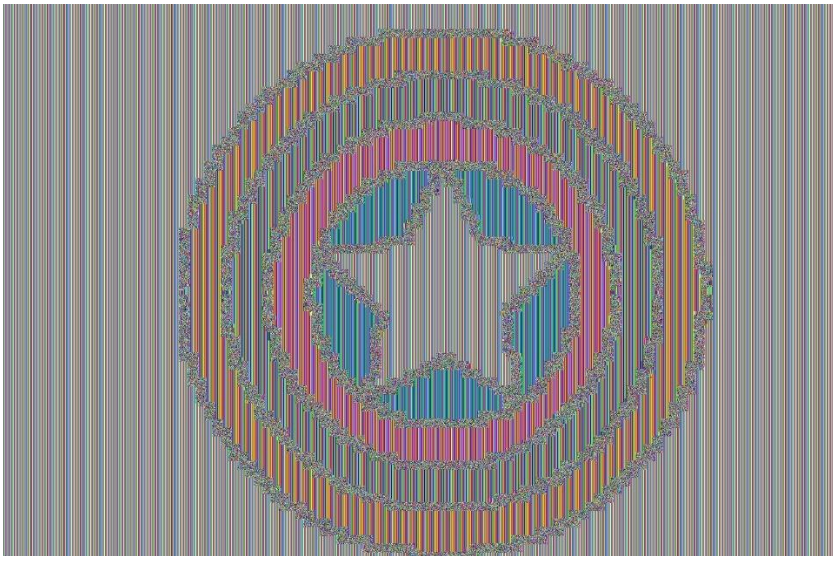
cat header body_cbc > new_cbc.bmp

The image is completely noise and cant be figured out

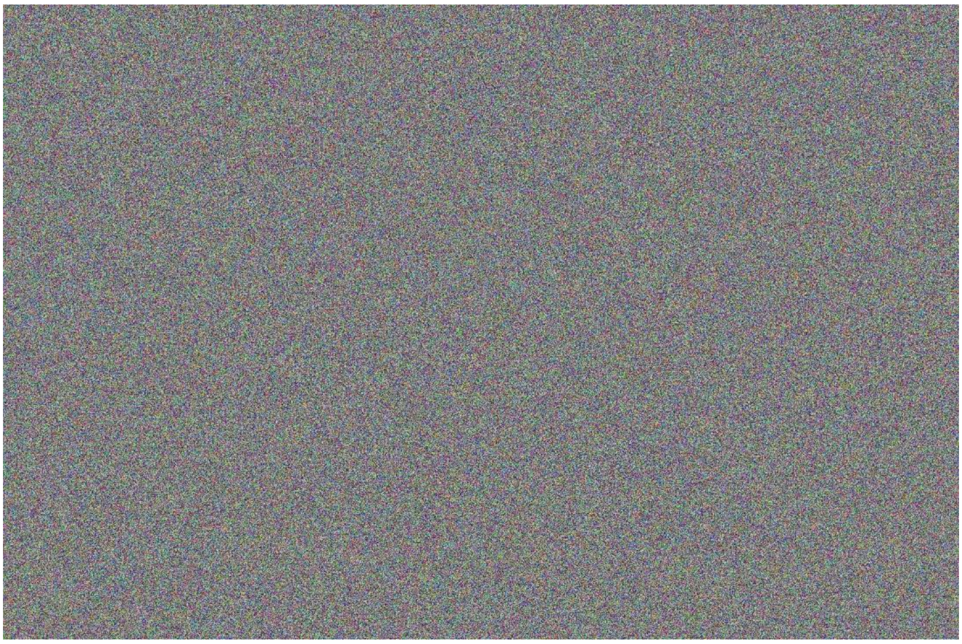
c. Image:

I encrypted the image and made it into a readable file using the steps as before:
openssl enc -aes-128-ecb -e -in captain-america-logo.bmp -out ca_ecb.bmp -k c4aP93MboJVQWZrX
openssl enc -aes-128-cbc -e -in captain-america-logo.bmp -out ca_cbc.bmp -k c4aP93MboJVQWZrX -iv 696e697469616c766563746f72616573
head -c 54 captain-america-logo.bmp > header_ca
tail -c +55 ca_ecb.bmp > body_ca_ecb
tail -c +55 ca_cbc.bmp > body_ca_cbc
cat header_ca body_ca_ecb > ca_ecb.bmp
cat header_ca body_ca_cbc > ca_cbc.bmp

Output:
ECB



CBC:



Observation:
The ECB mode of encryption encrypts the image but we can still make sense of the image whereas the CBC mode encrypts the image completely and we cannot make any sense of what it was.

Task 4

a. Plaintext used: “This is a sample plaintext”
AES Key: c4aP93MboJVQWZrX

Commands used for encryption:
ECB: *openssl enc -aes-128-ecb -e -in plaintext.txt -out ciphertext_ecb.txt -k c4aP93MboJVQWZrX*
CBC: *openssl enc -aes-128-cbc -e -in plaintext.txt -out ciphertext_cbc.txt -k c4aP93MboJVQWZrX*
CFB: *openssl enc -aes-128-cfb -e -in plaintext.txt -out ciphertext_cfb.txt -k c4aP93MboJVQWZrX*
OFB: *openssl enc -aes-128-ofb -e -in plaintext.txt -out ciphertext_ofb.txt -k c4aP93MboJVQWZrX*

Commands used for decryption:
ECB: *openssl enc -aes-128-ecb -d -in ciphertext_ecb.txt -out plaintext_ecb.txt -k c4aP93MboJVQWZrX -nopad*
CBC: *openssl enc -aes-128-cbc -d -in ciphertext_cbc.txt -out plaintext_cbc.txt -k c4aP93MboJVQWZrX -nopad*
CFB: *openssl enc -aes-128-cfb -d -in ciphertext_cfb.txt -out plaintext_cfb.txt -k c4aP93MboJVQWZrX -nopad*
OFB: *openssl enc -aes-128-ofb -d -in ciphertext_ofb.txt -out plaintext_ofb.txt -k c4aP93MboJVQWZrX -nopad*

Command to view plaintext.txt
xxd plaintext.txt

Output:

```
[04/04/2019 12:02] Chirag@VM:~/.../Task4$ xxd plaintext.txt
00000000: 5468 6973 2069 7320 6120 7361 6d70 6c65  This is a sample
00000010: 2070 6c61 696e 7465 7874 0a               plaintext.
[04/04/2019 12:16] Chirag@VM:~/.../Task4$ xxd plaintext
plaintext_cbc.txt  plaintext_ecb.txt  plaintext.txt
plaintext_cfb.txt  plaintext_ofb.txt
[04/04/2019 12:16] Chirag@VM:~/.../Task4$ xxd plaintext_ecb.txt
00000000: 5468 6973 2069 7320 6120 7361 6d70 6c65  This is a sample
00000010: 2070 6c61 696e 7465 7874 0a05 0505 0505   plaintext......
[04/04/2019 12:17] Chirag@VM:~/.../Task4$ xxd plaintext_cbc.txt
00000000: 5468 6973 2069 7320 6120 7361 6d70 6c65  This is a sample
00000010: 2070 6c61 696e 7465 7874 0a05 0505 0505   plaintext......
[04/04/2019 12:17] Chirag@VM:~/.../Task4$ xxd plaintext_cfb.txt
00000000: 5468 6973 2069 7320 6120 7361 6d70 6c65  This is a sample
00000010: 2070 6c61 696e 7465 7874 0a               plaintext.
[04/04/2019 12:17] Chirag@VM:~/.../Task4$ xxd plaintext_ofb.txt
00000000: 5468 6973 2069 7320 6120 7361 6d70 6c65  This is a sample
00000010: 2070 6c61 696e 7465 7874 0a               plaintext.
[04/04/2019 12:17] Chirag@VM:~/.../Task4$
```

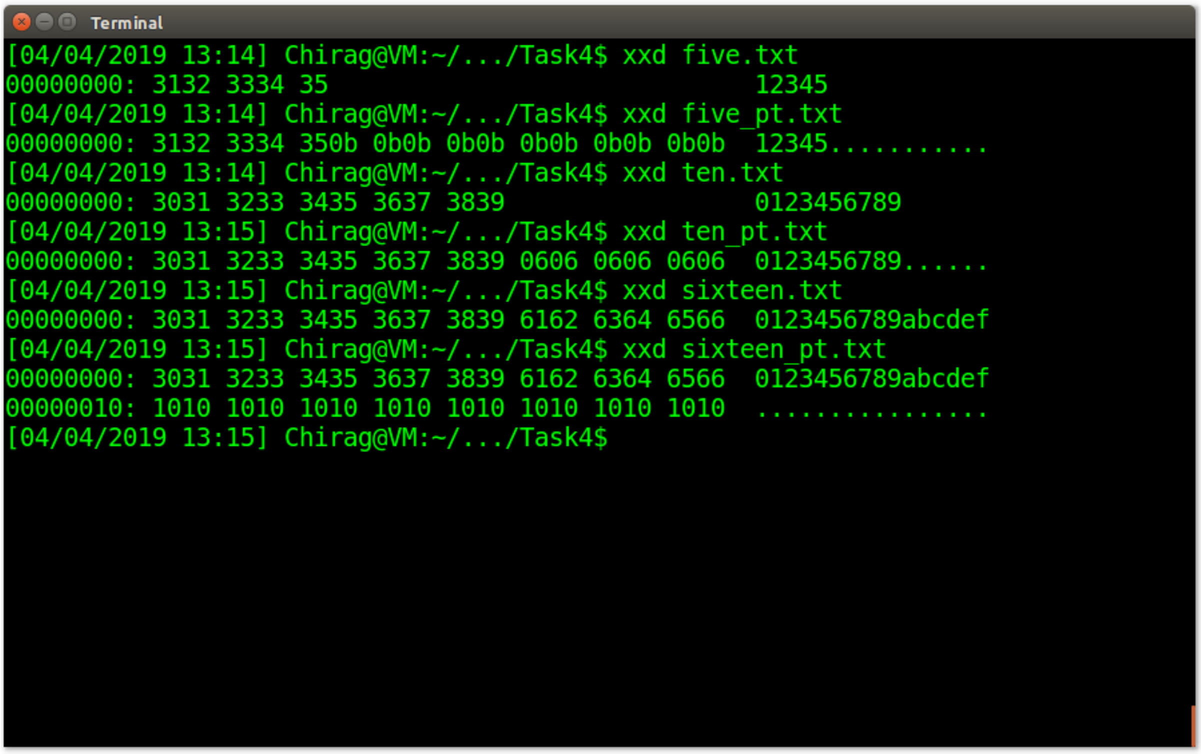
Observation:
The ECB and CBC modes require padding whereas the CFB and OFB modes do not require any padding.

b. I created the five byte file using the command:
echo -n 12345 > five.txt
I created the five byte file using the command
echo -n 0123456789 > ten.txt
I created the five byte file using the command
echo -n 0123456789abcdef > sixteen.txt

The table of sizes in bytes before and after encryption is shown below

	Plaintext	Ciphertext	Plaintext with padding
Five byte	5	32	16
Ten byte	10	32	16
Sixteen byte	16	48	16

Output:



```
Terminal
[04/04/2019 13:14] Chirag@VM:~/.../Task4$ xxd five.txt
00000000: 3132 3334 35                             12345
[04/04/2019 13:14] Chirag@VM:~/.../Task4$ xxd five_pt.txt
00000000: 3132 3334 350b 0b0b 0b0b 0b0b 0b0b 0b0b  12345...........
[04/04/2019 13:14] Chirag@VM:~/.../Task4$ xxd ten.txt
00000000: 3031 3233 3435 3637 3839                 0123456789
[04/04/2019 13:15] Chirag@VM:~/.../Task4$ xxd ten_pt.txt
00000000: 3031 3233 3435 3637 3839 0606 0606 0606  0123456789......
[04/04/2019 13:15] Chirag@VM:~/.../Task4$ xxd sixteen.txt
00000000: 3031 3233 3435 3637 3839 6162 6364 6566  0123456789abcdef
[04/04/2019 13:15] Chirag@VM:~/.../Task4$ xxd sixteen_pt.txt
00000000: 3031 3233 3435 3637 3839 6162 6364 6566  0123456789abcdef
00000010: 1010 1010 1010 1010 1010 1010 1010 1010  ................
[04/04/2019 13:15] Chirag@VM:~/.../Task4$
```

Observation:

The five byte file was padded with 11 bytes of “0b”

The ten byte file was padded with 6 bytes of “06”

The sixteen byte file was padded with 16 bytes of “10”

Task 5

For this task, I used the plaintext from Task 1.
Key: c4aP93MboJVQWZrX

Encryption:
openssl enc -aes-128-ecb -e -in plaintext.txt -out ciphertext_ecb.txt -k c4aP93MboJVQWZrX
openssl enc -aes-128-cbc -e -in plaintext.txt -out ciphertext_cbc.txt -k c4aP93MboJVQWZrX
openssl enc -aes-128-cfb -e -in plaintext.txt -out ciphertext_cfb.txt -k c4aP93MboJVQWZrX
openssl enc -aes-128-ofb -e -in plaintext.txt -out ciphertext_ofb.txt -k c4aP93MboJVQWZrX

Decryption:
openssl enc -aes-128-ecb -d -in ciphertext_ecb.txt -out plaintext_ecb.txt -k c4aP93MboJVQWZrX
openssl enc -aes-128-cbc -d -in ciphertext_cbc.txt -out plaintext_cbc.txt -k c4aP93MboJVQWZrX
openssl enc -aes-128-cfb -d -in ciphertext_cfb.txt -out plaintext_cfb.txt -k c4aP93MboJVQWZrX
openssl enc -aes-128-ofb -d -in ciphertext_ofb.txt -out plaintext_ofb.txt -k c4aP93MboJVQWZrX

I replaced the 55^{th} byte in all encrypted files with "$" to corrupt the cipher.

I feel that in the:
ECB mode, 1 block of data would be corrupted
CBC mode, 2 blocks of data would be corrupted
OFB mode, 1 block of data would be corrupted
CFB mode, the entire data after the corrupt block should be corrupted

Observation:
In all cases, only 2 blocks of data were corrupted.

Task 6

Task 6.1

Plaintext1: “This is a sample plaintext”
Plaintext2: “This sample is a plaintext”
IV1: 7a6b51795a6e4f327041425056594d57
IV2: 4c754552614b704f6b364e3172477362
Key: c4aP93MboJVQWZrX

Plaintext 1 encrypted using IV1:

```
[04/04/2019 18:08] Chirag@VM:~/.../Task6$ xxd ciphertext1-1.txt
00000000: 5361 6c74 6564 5f5f bc3a 1419 2f2e 8517  Salted__.:../...
00000010: 1fe1 8b2b cfa6 915d 8a8f 102c 49d0 98e7  ...+...]...,I...
00000020: 4d90 ac03 9424 8bb2 e5b9 5ba7 01c7 74af  M....$....[...t.
[04/04/2019 18:08] Chirag@VM:~/.../Task6$
```

Plaintext 2 encrypted using IV1:

```
[04/04/2019 18:08] Chirag@VM:~/.../Task6$ xxd ciphertext2-1.txt
00000000: 5361 6c74 6564 5f5f 0cc0 568c f80d 9cf7  Salted__..V.....
00000010: 4c78 2d43 c849 e5aa a16c a9ce 5f89 3364  Lx-C.I...l.._.3d
00000020: ebc9 c174 2cfb ff21 8679 b019 2fda 54f5  ...t,..!.y../.T.
[04/04/2019 18:08] Chirag@VM:~/.../Task6$
```

Plaintext 1 encrypted using IV2:

```
[04/04/2019 18:08] Chirag@VM:~/.../Task6$ xxd ciphertext1-2.txt
00000000: 5361 6c74 6564 5f5f 9d60 7c2a 43ce 0b40  Salted__.`|*C..@
00000010: f95c bf86 f609 ee43 a2f3 dded 6543 4b70  .\.....C....eCKp
00000020: 4424 fd91 4ad8 5af6 62f5 adc3 e711 b5d5  D$..J.Z.b.......
[04/04/2019 18:09] Chirag@VM:~/.../Task6$
```

Task 6.2

Code:

```
temp=""
plaintext=b"This is a known message!"
pt1=plaintext.hex()
ct1="a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159"
pt1list=bytearray.fromhex(pt1)
ct1list=bytearray.fromhex(ct1)

templist=bytearray((x^y for x,y in zip(pt1list,ct1list)))
# print(templist.hex())

ct2="bf73bcd3509299d566c35b5d450337e1bb175f903fafc159"
ct2list=bytearray.fromhex(ct2)

pt2list=bytearray((x^y for x,y in zip(templist,ct2list)))
pt2=bytes.fromhex(pt2list.hex())
print(pt2.decode())
```

Output:
Order: Launch a missile!

In the OFB mode, it is essential to change the IV for every plaintext.

If the encryption mode was changed to CFB, only 1 block of the ciphertext would be recovered since the ciphertext would then be passed as the IV for the second block and would still require the key to actually decrypt it.

Task 6.3

Code to generate new input:

```
from Crypto.Util import Padding
temp=""

plaintext=(Padding.pad("Yes",16)).encode("ascii")
pt1=plaintext.hex()
iv1="31323334353637383930313233343536"
iv2="31323334353637383930313233343537"
pt1list=bytearray.fromhex(pt1)
iv1list=bytearray.fromhex(iv1)
iv2list=bytearray.fromhex(iv2)

templist=bytearray((x^y for x,y in zip(pt1list,iv1list)))
ip2list=bytearray((x^y for x,y in zip(templist,iv2list)))

ip2 = bytes.fromhex(ip2list.hex())
print(ip2.decode())
```

The generated input is passed as plaintext to the encryption.

If the new ciphertext matches the old ciphertext then the old plaintext was “Yes”, if it is different then the old plaintext was “No”.

Task 7

Dictionary based attack to find key when the plaintext, ciphertext and IV is known.
English wordlist from the lab website.

Code:

```
from Crypto.Cipher import AES
from Crypto.Util import Padding

# defining known values and converting them from hex to binary strings
plaintext=b"This is a top secret."
iv_hex='aabbccddeeff00998877665544332211'
ciphertext_hex="764aa26b55a4da654df6b19e4bce00f4ed05e09346fb0e762583cb7da2ac93a2"
iv=bytes.fromhex(iv_hex)
ciphertext=bytes.fromhex(ciphertext_hex)
key=""

# Reading the english wordlist into a list
fp=open("words.txt","r")
wordlist_ip=fp.readlines()
fp.close()

# removing the whitespaces and \n from the words and storing the words in a new dictionary
wordlist=[]
for word in wordlist_ip:
    word=word.replace("\n","")
    word=word.strip()
    wordlist.append(word)

# applying bruteforce on the plaintext with keys from the english wordlist
for word in wordlist:
    # padding the key with pound size to make it 128 bits
    # # is 0x23 in hex
    if len(word) <= 16:
        n=16-len(word)
        key_bin=word.encode("ascii")+b"\x23"*n
    # create instance of the object of the AES cipher
    cipher=AES.new(key_bin,AES.MODE_CBC,iv)
    # encrypt the plaintext with padding with block size set as 16 bytes
    ciphertext_new=cipher.encrypt(Padding.pad(plaintext,16))
    if ciphertext ==ciphertext_new:
        key=word
        break
print("The key is :",key)
```

Observation:
On running the program, the key was found to be “Syracuse”