SYSTEM AND NETWORK SECURITY

FALL 2024 – Symmetric Key Encryption

Due: 1st September, 2024

Task 1: Frequency Analysis

- Below is an image of the frequency analysis for a typical English plaintext. (Source: https://en.wikipedia.org/wiki/Frequency_analysis)

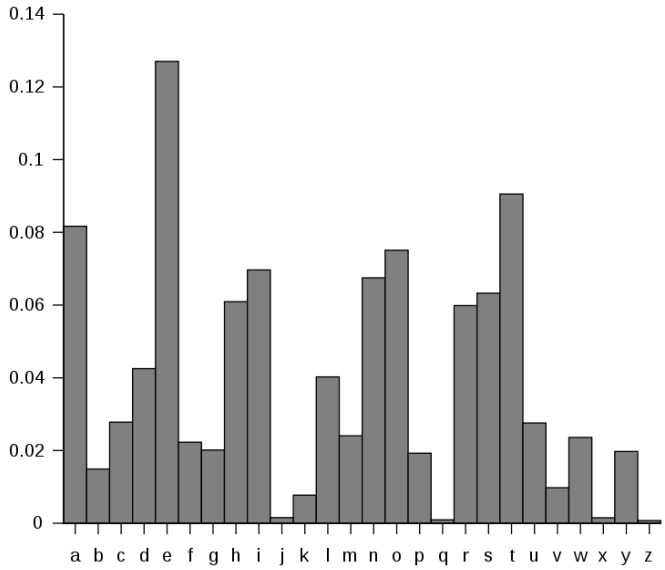


Figure 1

- Using the `freq.py` Python program, we can find the statistics for n-grams, including the single-letter frequencies, bigram frequencies (2-letter sequence), and trigram frequencies (3-letter sequence), etc.

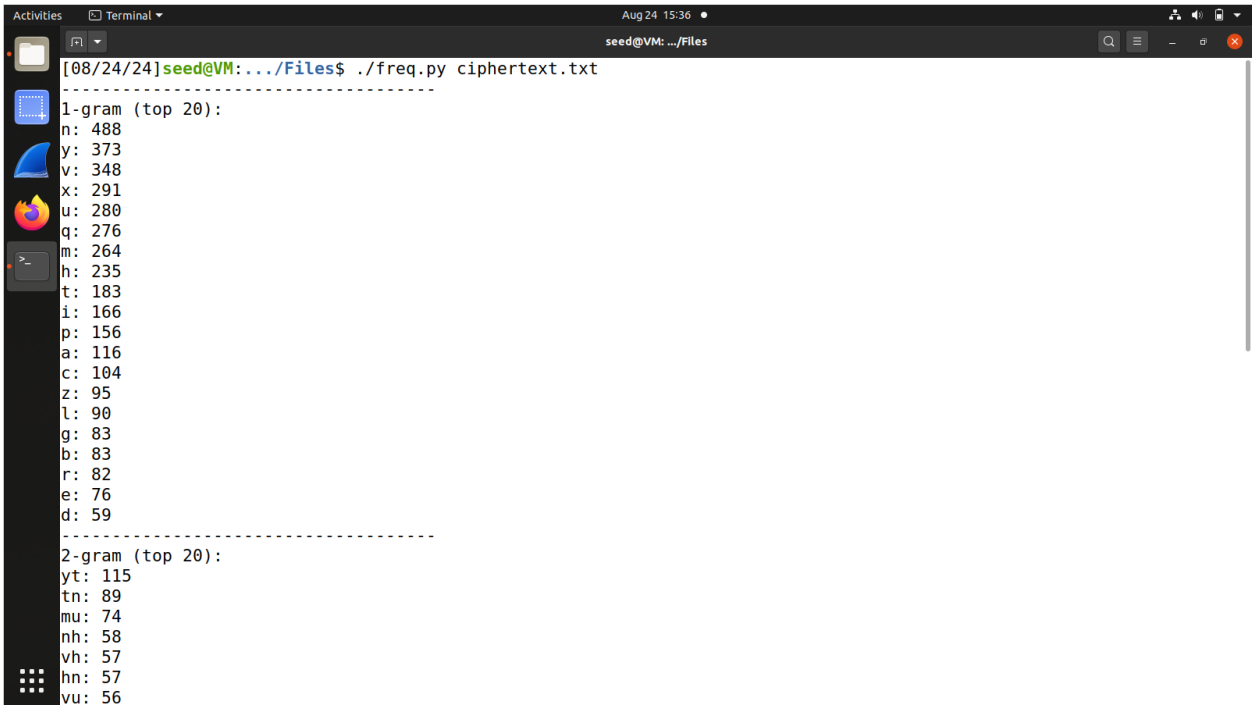
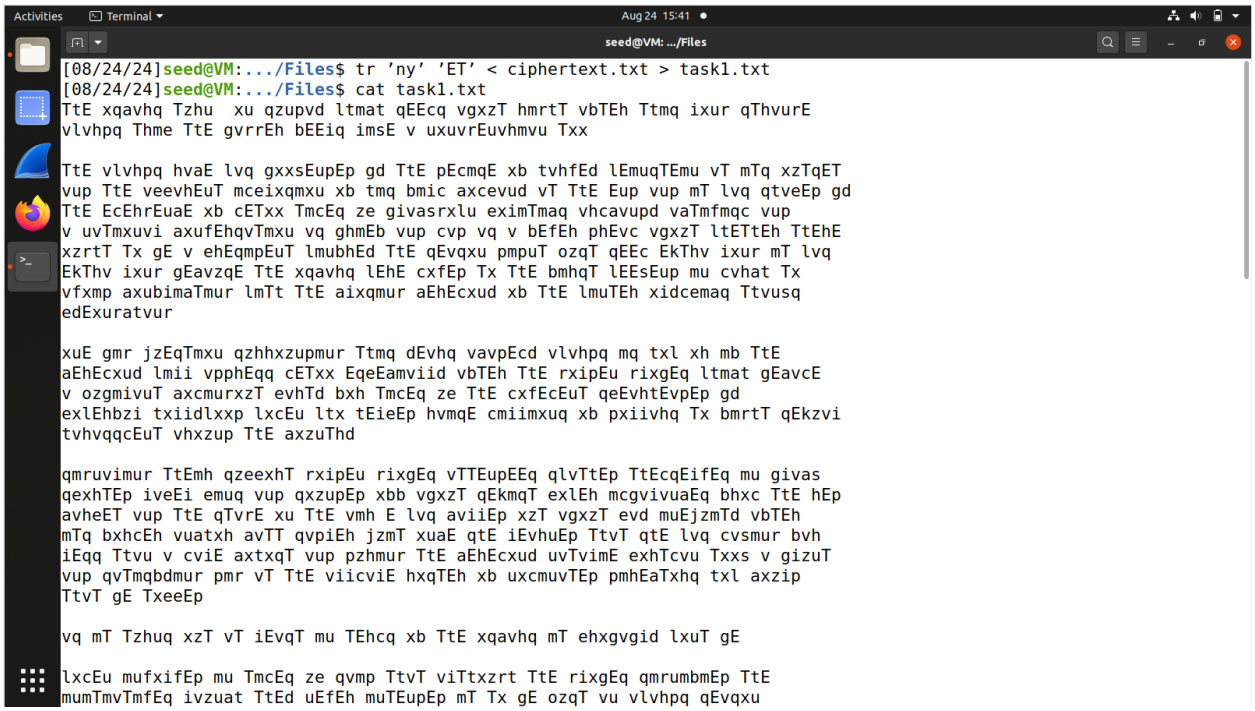
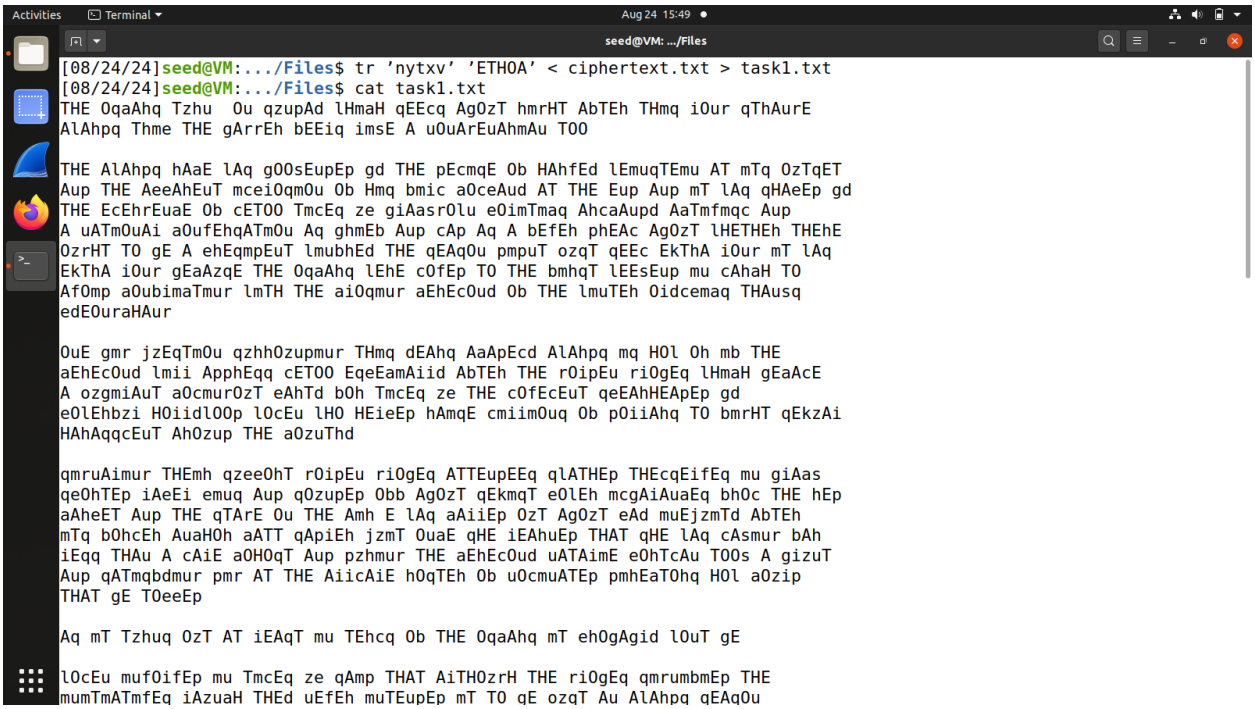


Figure 2

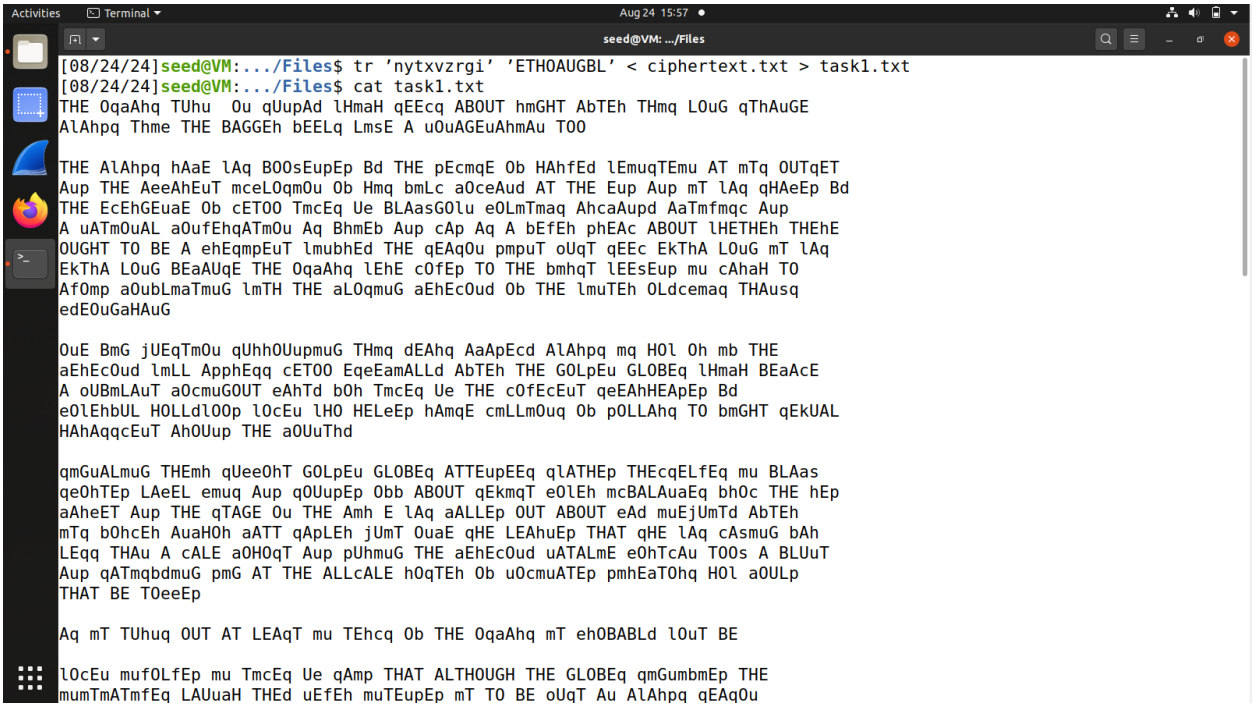
- After getting the above frequencies in the cipher text, we check the frequencies and match them corresponding to Figure 1.
- Using Figure 1 and Figure 2, we replace the letters using the `tr` command. You can use the '`tr`' command to do this.
- Let us replace '`n`' with '`E`' and '`y`' with '`T`' as these are the most occurred letters.



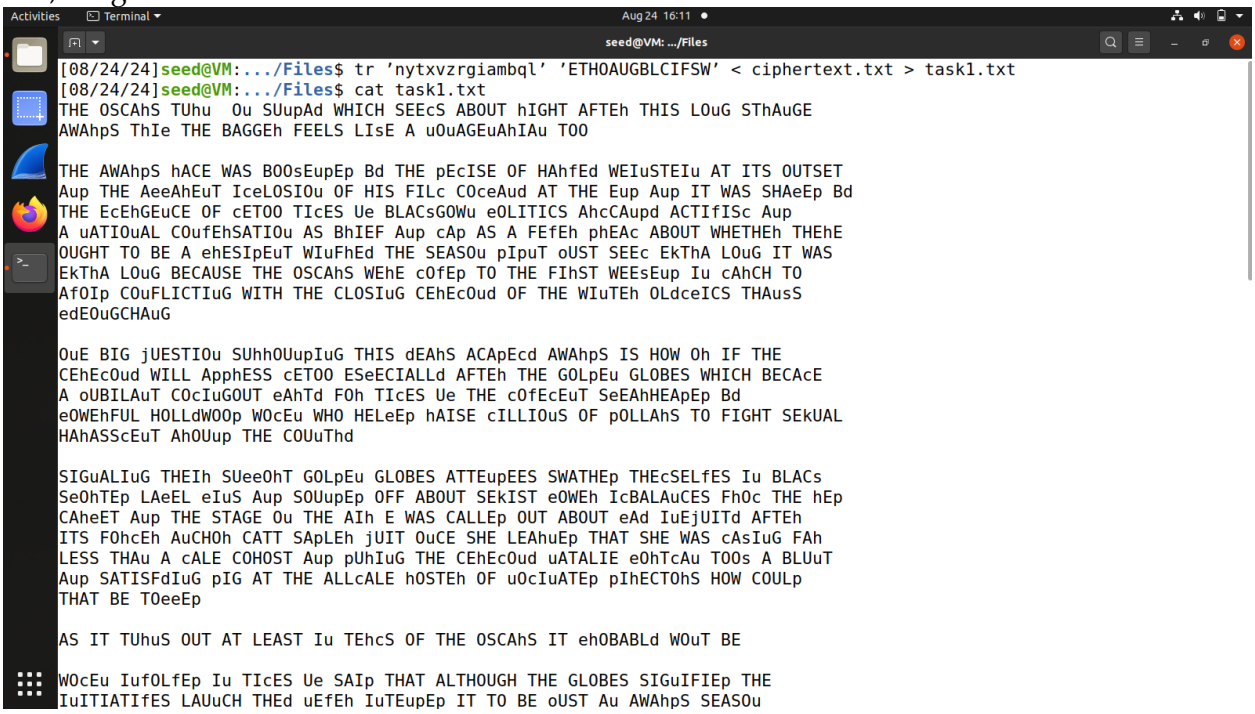
- We can start guessing the letters now. '`t`', '`x`', '`v`' can be replaced by '`H`', '`O`' and '`A`'.



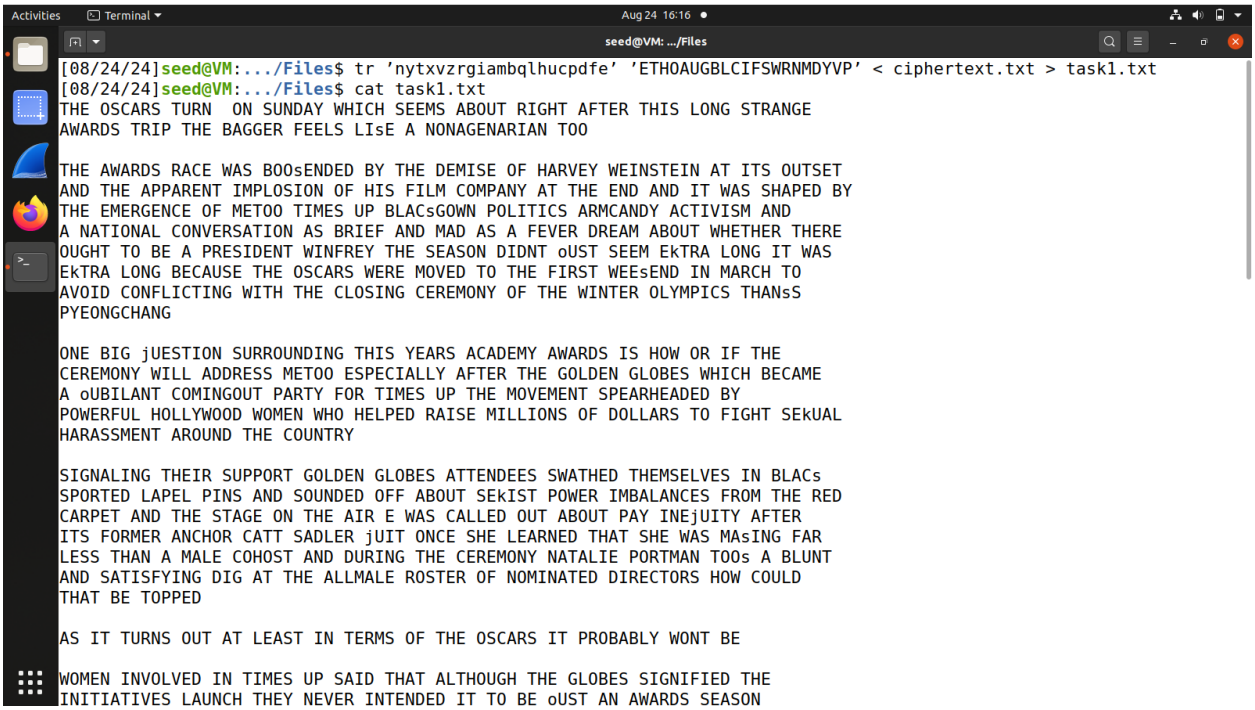
- We can't be sure but we can slightly guess that '`z`' might be '`U`', '`r`' might be '`G`', '`i`' might be '`L`' and '`g`' might be '`B`'. Let's try that and see if it works.



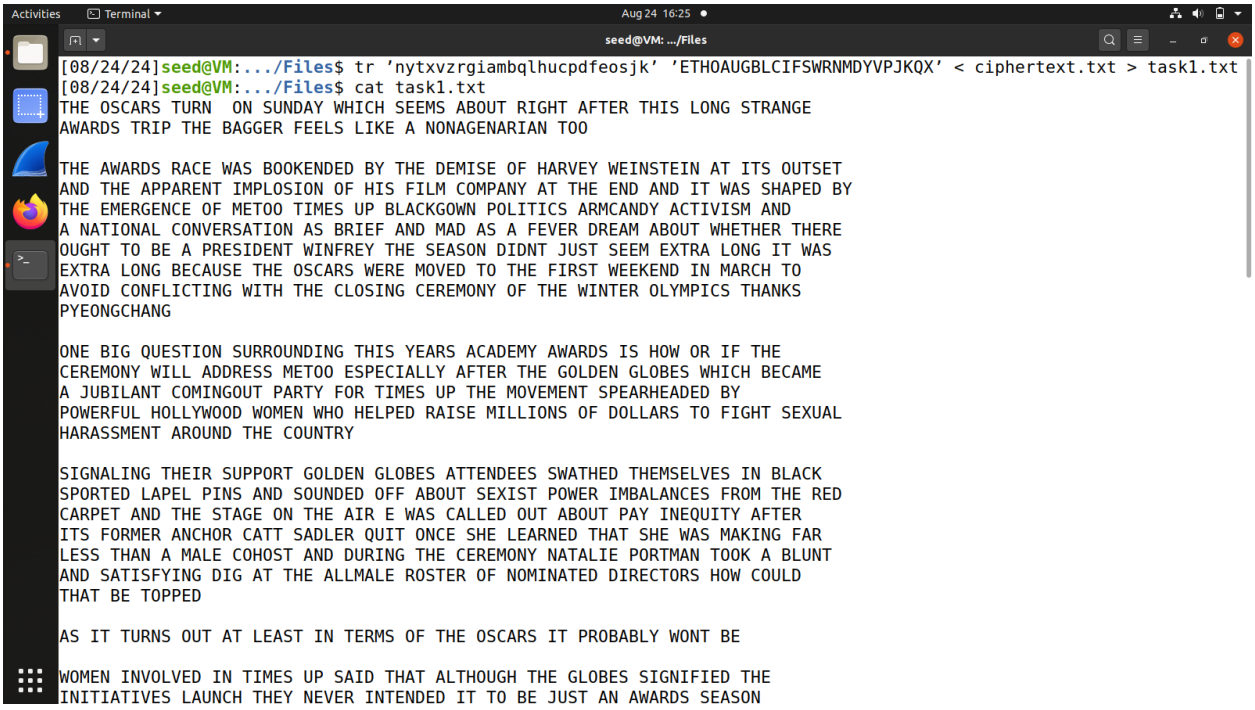
- So we can guess 'a' to be 'C' and 'm' to be 'I'. Also 'b' as 'F', 'l' as 'W' and 'q' as 'S'. If we apply all these, we get the below:



- Now it is getting clear that 'h' → 'R', 'u' → 'N', 'c' → 'M', 'p' → 'D', 'd' → 'Y', 'f' → 'V', 'e' → 'P'. Let us apply this and analyze.



- Now we have 'o' → 'J', 's' → 'K', 'j' → 'Q', 'k' → 'X'



- Lastly we have 'w' → 'Z' as that is the last combination.
- Final Output:

```
[08/24/24]seed@VM:.../Files$ tr 'nytxvzrgiambqlhucpdfeosjk' 'ETHOAUGBLCIFSWRNMDYVPJKQX' < ciphertext.txt >
task1.txt[08/24/24]seed@VM:.../Files$ cat task1.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

SIGNALING THEIR SUPPORT GOLDEN GLOBES ATTENDEES SWATHED THEMSELVES IN BLACK SPORTED LAPEL PINS AND SOUNDED OFF ABOUT SEXIST POWER IMBALANCES FROM THE RED CARPET AND THE STAGE ON THE AIR E WAS CALLED OUT ABOUT PAY INEQUITY AFTER ITS FORMER ANCHOR CATT SADLER QUIT ONCE SHE LEARNED THAT SHE WAS MAKING FAR LESS THAN A MALE COHOST AND DURING THE CEREMONY NATALIE PORTMAN TOOK A BLUNT AND SATISFYING DIG AT THE ALLMALE ROSTER OF NOMINATED DIRECTORS HOW COULD THAT BE TOPPED

AS IT TURNS OUT AT LEAST IN TERMS OF THE OSCARS IT PROBABLY WONT BE

WOMEN INVOLVED IN TIMES UP SAID THAT ALTHOUGH THE GLOBES SIGNIFIED THE INITIATIVES LAUNCH THEY NEVER INTENDED IT TO BE JUST AN AWARDS SEASON CAMPAIGN OR ONE THAT BECAME ASSOCIATED ONLY WITH REDCARPET ACTIONS INSTEAD A SPOKESWOMAN SAID THE GROUP IS WORKING BEHIND CLOSED DOORS AND HAS SINCE AMASSED MILLION FOR ITS LEGAL DEFENSE FUND WHICH AFTER THE GLOBES WAS FLOODED WITH THOUSANDS OF DONATIONS OF OR LESS FROM PEOPLE IN SOME COUNTRIES

NO CALL TO WEAR BLACK GOWNS WENT OUT IN ADVANCE OF THE OSCARS THOUGH THE MOVEMENT WILL ALMOST CERTAINLY BE REFERENCED BEFORE AND DURING THE CEREMONY ESPECIALLY SINCE VOCAL METOO SUPPORTERS LIKE ASHLEY JUDD LAURA DERN AND NICOLE KIDMAN ARE SCHEDULED PRESENTERS

ANOTHER FEATURE OF THIS SEASON NO ONE REALLY KNOWS WHO IS GOING TO WIN BEST PICTURE ARGUABLY THIS HAPPENS A LOT OF THE TIME INARGUABLY THE NAILBITER NARRATIVE ONLY SERVES THE AWARDS HYPE MACHINE BUT OFTEN THE PEOPLE FORECASTING THE RACE SOCALLED OSCAROLOGISTS CAN MAKE ONLY EDUCATED GUESSES

THE WAY THE ACADEMY TABULATES THE BIG WINNER DOESNT HELP IN EVERY OTHER CATEGORY THE NOMINEE WITH THE MOST VOTES WINS BUT IN THE BEST PICTURE CATEGORY VOTERS ARE ASKED TO LIST THEIR TOP MOVIES IN PREFERENTIAL ORDER IF A MOVIE GETS MORE THAN PERCENT OF THE FIRSTPLACE VOTES IT WINS WHEN NO MOVIE MANAGES THAT THE ONE WITH THE FEWEST FIRSTPLACE VOTES IS ELIMINATED AND ITS VOTES ARE REDISTRIBUTED TO THE MOVIES THAT GARNERED THE ELIMINATED BALLOTS SECONDPLACE VOTES AND THIS CONTINUES UNTIL A WINNER EMERGES

IT IS ALL TERRIBLY CONFUSING BUT APPARENTLY THE CONSENSUS FAVORITE COMES OUT AHEAD IN THE END THIS MEANS THAT ENDOFSEASON AWARDS CHATTER INVARIABLY INVOLVES TORTURED SPECULATION ABOUT WHICH FILM WOULD MOST LIKELY BE VOTERS SECOND OR THIRD FAVORITE AND THEN EQUALLY TORTURED CONCLUSIONS ABOUT WHICH FILM MIGHT PREVAIL

IN IT WAS A TOSSUP BETWEEN BOYHOOD AND THE EVENTUAL WINNER BIRDMAN
IN WITH LOTS OF EXPERTS BETTING ON THE REVENANT OR THE BIG SHORT THE PRIZE WENT TO SPOTLIGHT LAST YEAR NEARLY ALL THE FORECASTERS DECLARED LA LA LAND THE PRESUMPTIVE WINNER AND FOR TWO AND A HALF MINUTES THEY WERE CORRECT BEFORE AN ENVELOPE SNAFU WAS REVEALED AND THE RIGHTFUL WINNER MOONLIGHT WAS CROWNED

THIS YEAR AWARDS WATCHERS ARE UNEQUALLY DIVIDED BETWEEN THREE BILLBOARDS OUTSIDE EBBING MISSOURI THE FAVORITE AND THE SHAPE OF WATER WHICH IS THE BAGGERS PREDICTION WITH A FEW FORECASTING A HAIL MARY WIN FOR GET OUT

BUT ALL OF THOSE FILMS HAVE HISTORICAL OSCARVOTING PATTERNS AGAINST THEM THE SHAPE OF WATER HAS NOMINATIONS MORE THAN ANY OTHER FILM AND WAS ALSO NAMED THE YEARS BEST BY THE PRODUCERS AND DIRECTORS GUILDS YET IT WAS NOT NOMINATED FOR A SCREEN ACTORS GUILD AWARD FOR BEST ENSEMBLE AND NO FILM HAS WON BEST PICTURE WITHOUT PREVIOUSLY LANDING AT LEAST THE ACTORS NOMINATION SINCE BRAVEHEART IN THIS YEAR THE BEST ENSEMBLE SAG ENDED UP GOING TO THREE BILLBOARDS WHICH IS SIGNIFICANT BECAUSE ACTORS MAKE UP THE ACADEMYS LARGEST BRANCH THAT FILM WHILE DIVISIVE ALSO WON THE BEST DRAMA GOLDEN GLOBE AND THE BAFTA BUT ITS FILMMAKER MARTIN MCDONAGH WAS NOT NOMINATED FOR BEST DIRECTOR AND APART FROM ARGO MOVIES THAT LAND BEST PICTURE WITHOUT ALSO EARNING BEST DIRECTOR NOMINATIONS ARE FEW AND FAR BETWEEN

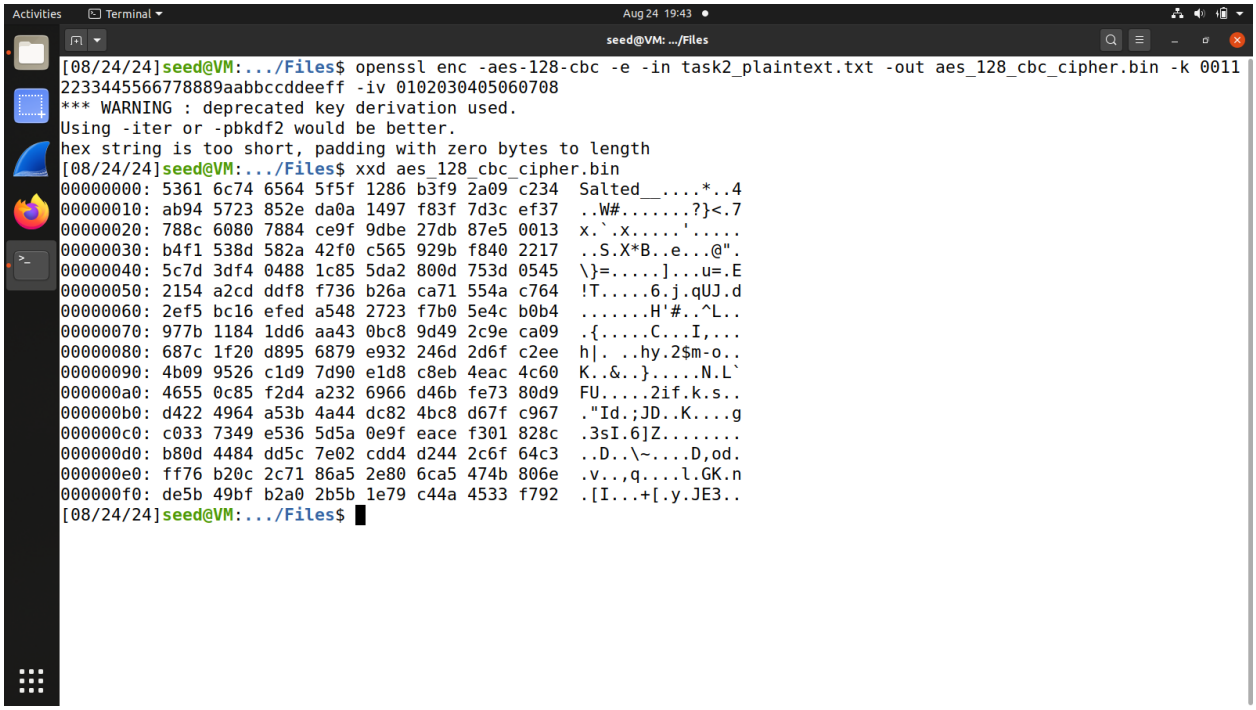
- So we have our final key as below

Cipher text	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Plain text	C	F	M	Y	P	V	B	R	L	Q	X	W	I	E	J	D	S	G	K	H	N	A	Z	O	T	U

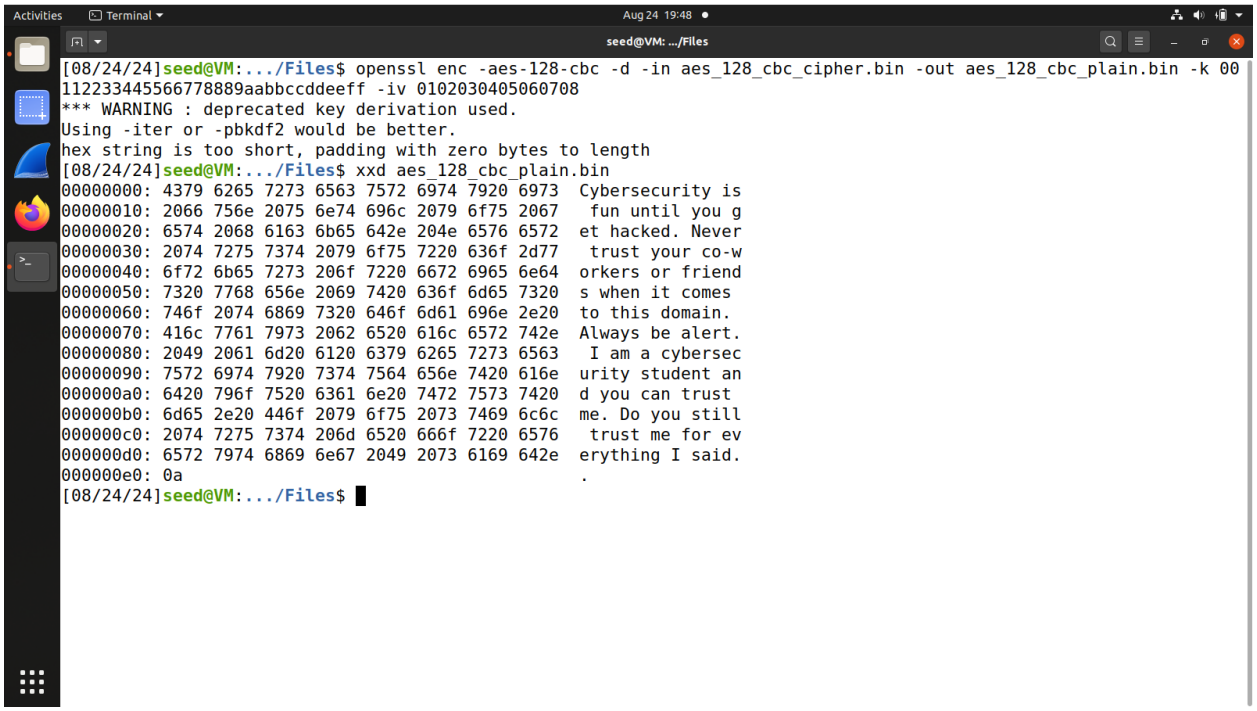
Task 2: Encryption using Different Ciphers and Modes

- Out of all the cipher types, we will be using 3 different cipher types for this task.
  - `-aes-128-cbc`
  - `-des-cfb`
  - `-bf-ofb`

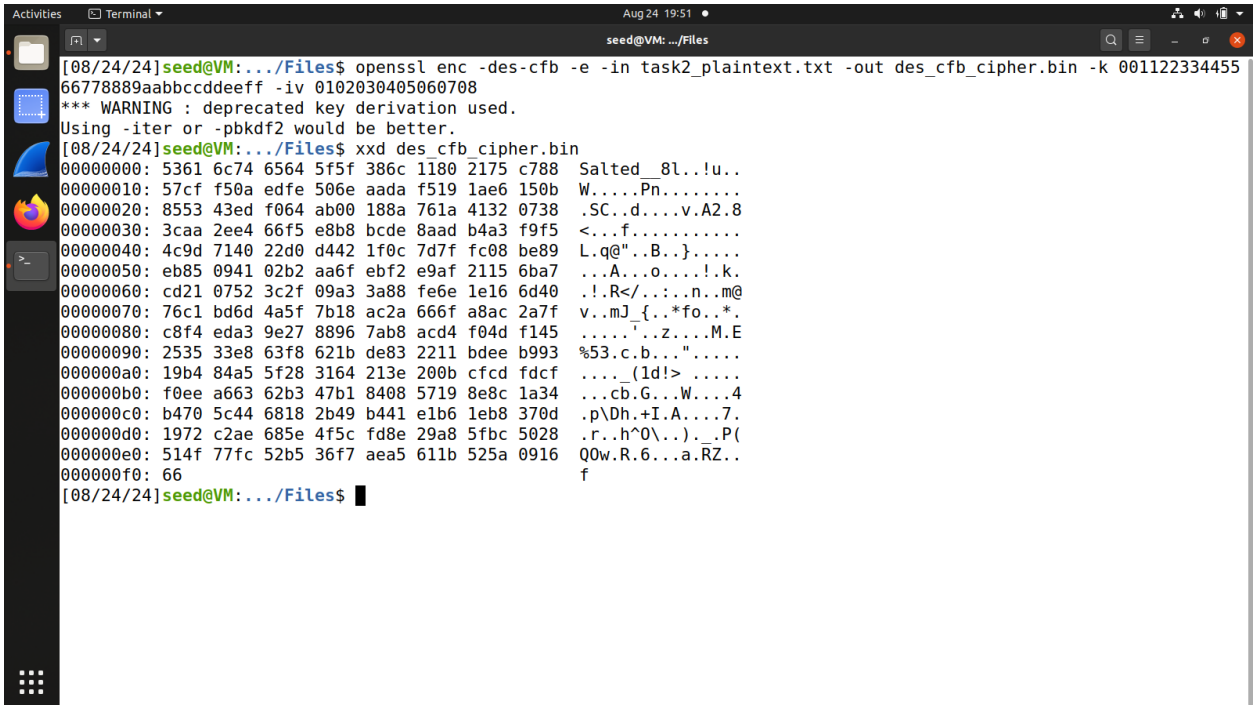
1. `-aes-128-cbc`
   - Run the command to encrypt the `task2_plaintext.txt` file using the below command `$ openssl enc -aes-128-cbc -e -in task2_plaintext.txt -out aes_128_cbc_cipher.bin -k 00112233445566778889aabbccddeeff -iv 0102030405060708`
   - We can view the encrypted file using the command `$cat aes_128_cbc_cipher.bin` but to make it more meaningful, run the `$xxd aes_128_cbc_cipher.bin` and we can view in hexadecimal format.



   - For decrypting, use the command `$ openssl enc -aes-128-cbc -d -in aes_128_cbc_cipher.bin -out aes_128_cbc_plain.bin -k 00112233445566778889aabbccddeeff -iv 0102030405060708`
   - For viewing the decrypted content, just use the `cat` or `xxd` command followed by the file name.

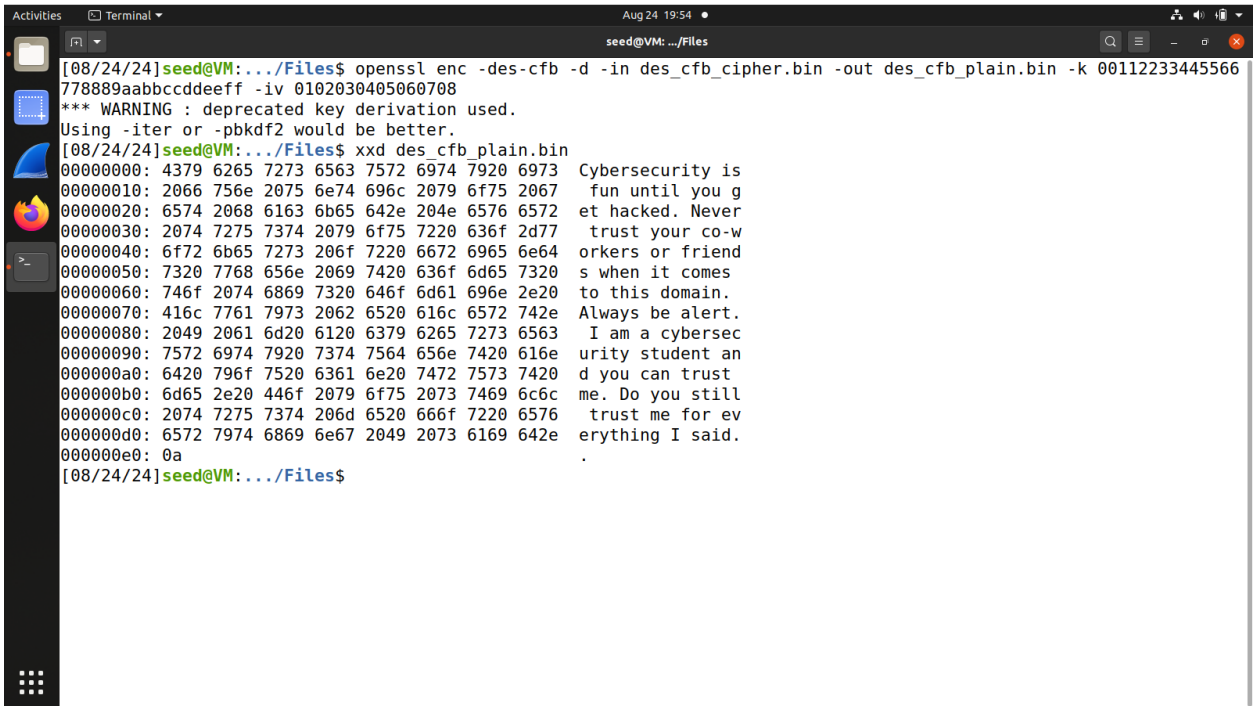


2. `-des-cfb`
   - Run the command to encrypt the `task2_plaintext.txt` file using the below command `$openssl enc -des-cfb -e -in task2_plaintext.txt -out des_cfb_cipher.bin -k 00112233445566778889aabbccddeeff -iv 0102030405060708`
   - We can view the encrypted file using the command `$cat des_cfb_cipher.bin` but to make it more meaningful, run the `$xxd des_cfb_cipher.bin` and we can view in hexadecimal format.

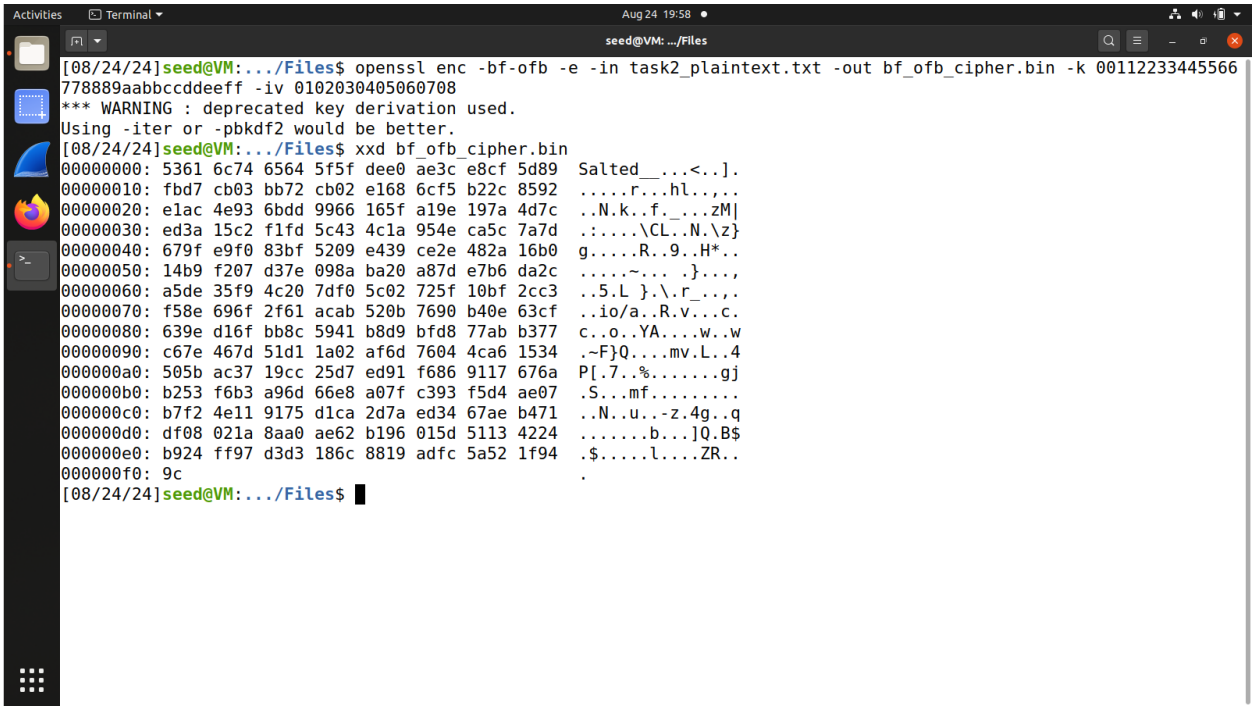


   - For decrypting, use the command `$openssl enc -des-cfb -d -in des_cfb_cipher.bin -out des_cfb_plain.bin -k 00112233445566778889aabbccddeeff -iv 0102030405060708`

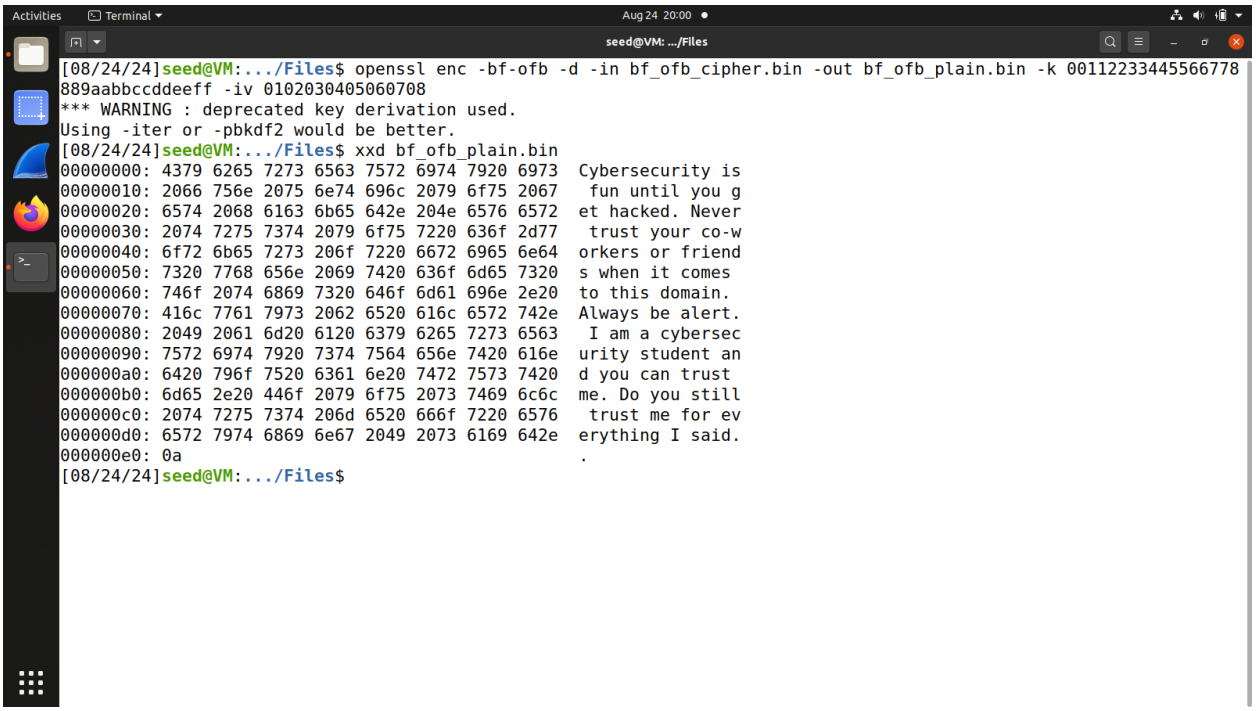
- For viewing the decrypted content, just use the `cat` or `xxd` command followed by the file name.



3. `-bf-ofb`
   - Run the command to encrypt the `task2_plaintext.txt` file using the below command `$ openssl enc -bf-ofb -e -in task2_plaintext.txt -out bf_ofb_cipher.bin -k 00112233445566778889aabbccddeeff -iv 0102030405060708`
   - We can view the encrypted file using the command `$cat bf_ofb_cipher.bin` but to make it more meaningful, run the `$xxd bf_ofb_cipher.bin` and we can view in hexadecimal format.



- For decrypting, use the command `$ openssl enc -bf-ofb -d -in bf_ofb_cipher.bin -out bf_ofb_plain.bin -k 00112233445566778889aabbccddeeff -iv 0102030405060708`
- For viewing the decrypted content, just use the `cat` or `xxd` command followed by the file name.



Task 3: Encryption Mode – ECB vs. CBC

1. Electronic Code Block (ECB)

Figure 4. `pic_original.bmp`

- Figure 4 is the original image. We encrypt it using the AES-128-ECB cipher type by using the command `$ openssl enc -aes-128-ecb -e -in pic_original.bmp -out aes_128_ecb_pic.bmp -k 00112233445566778889aabbccddeeff -iv 0102030405060708`
- NOTE: Once we run this command, we get a warning stating that initial value isn't used by this cipher.

- Using the Bless tool, we can use the following commands
    - `$head -c 54 pic_original.bmp > header`
    - `$tail -c +55 aes_128_ecb_pic.bmp > body_ecb`
    - `$cat header body_ecb > aes_128_ecb_pic.bmp`

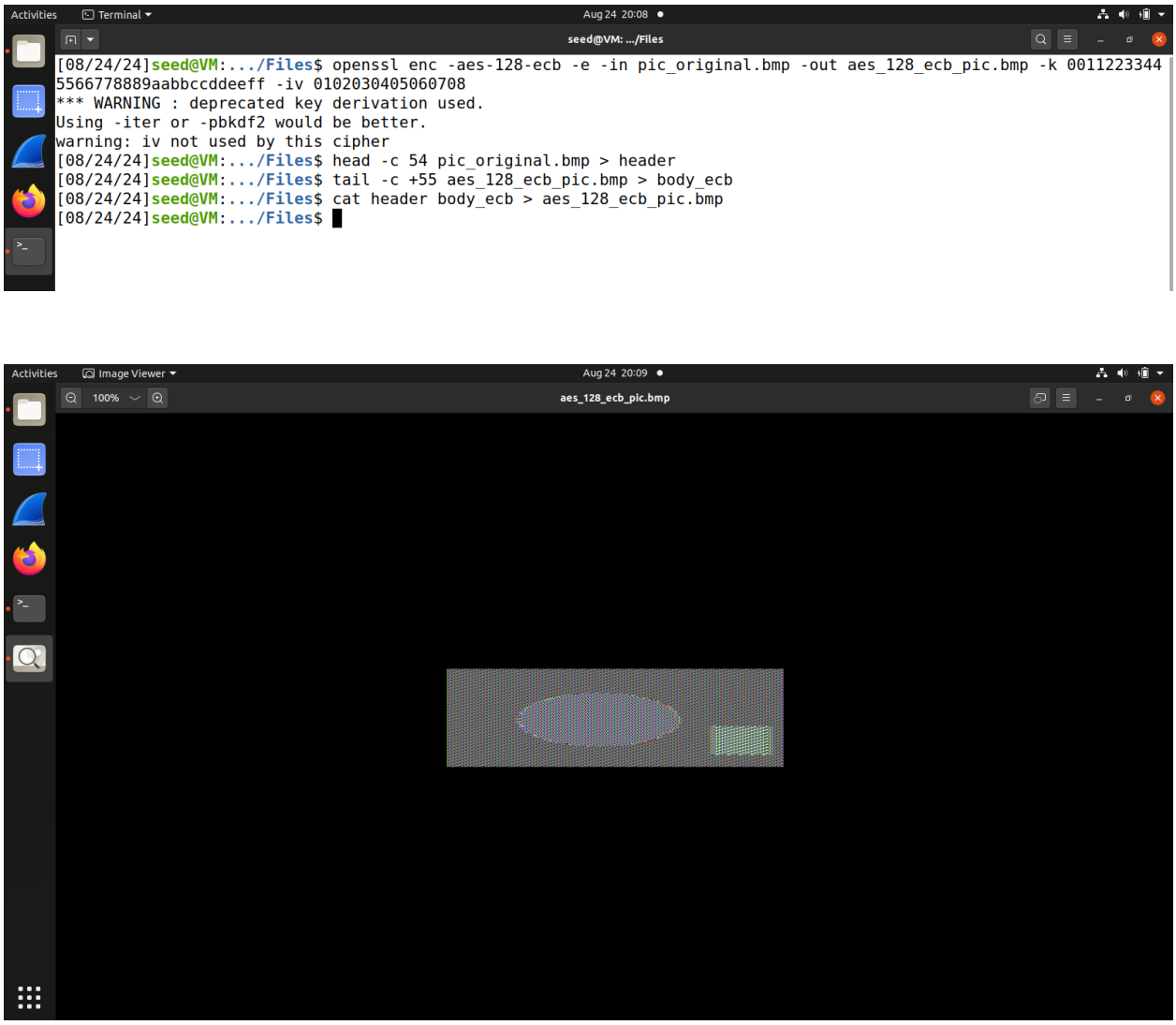


Figure 5. `aes_128_ecb_pic.bmp`

2. Code Block Cipher (CBC)
    - Again we use the same image (Figure 3). This time, we encrypt it using the AES-128-CBC cipher type:
      `$openssl enc -aes-128-cbc -e -in pic_original.bmp -out aes_128_cbc_pic.bmp -k 00112233445566778889aabbccddeeff -iv 0102030405060708`
    - NOTE: Once we run this command, we get a warning stating that initial value isn't used by this cipher.
        - Using the Bless tool, we can use the following commands. Note here that we didn't run `$head -c 54 pic_original.bmp > header` as we already have this header stored and there is no point in running this command again.
        - `$tail -c +55 aes_128_cbc_pic.bmp > body_cbc`
        - `$cat header body_cbc > aes_128_cbc_pic.bmp`

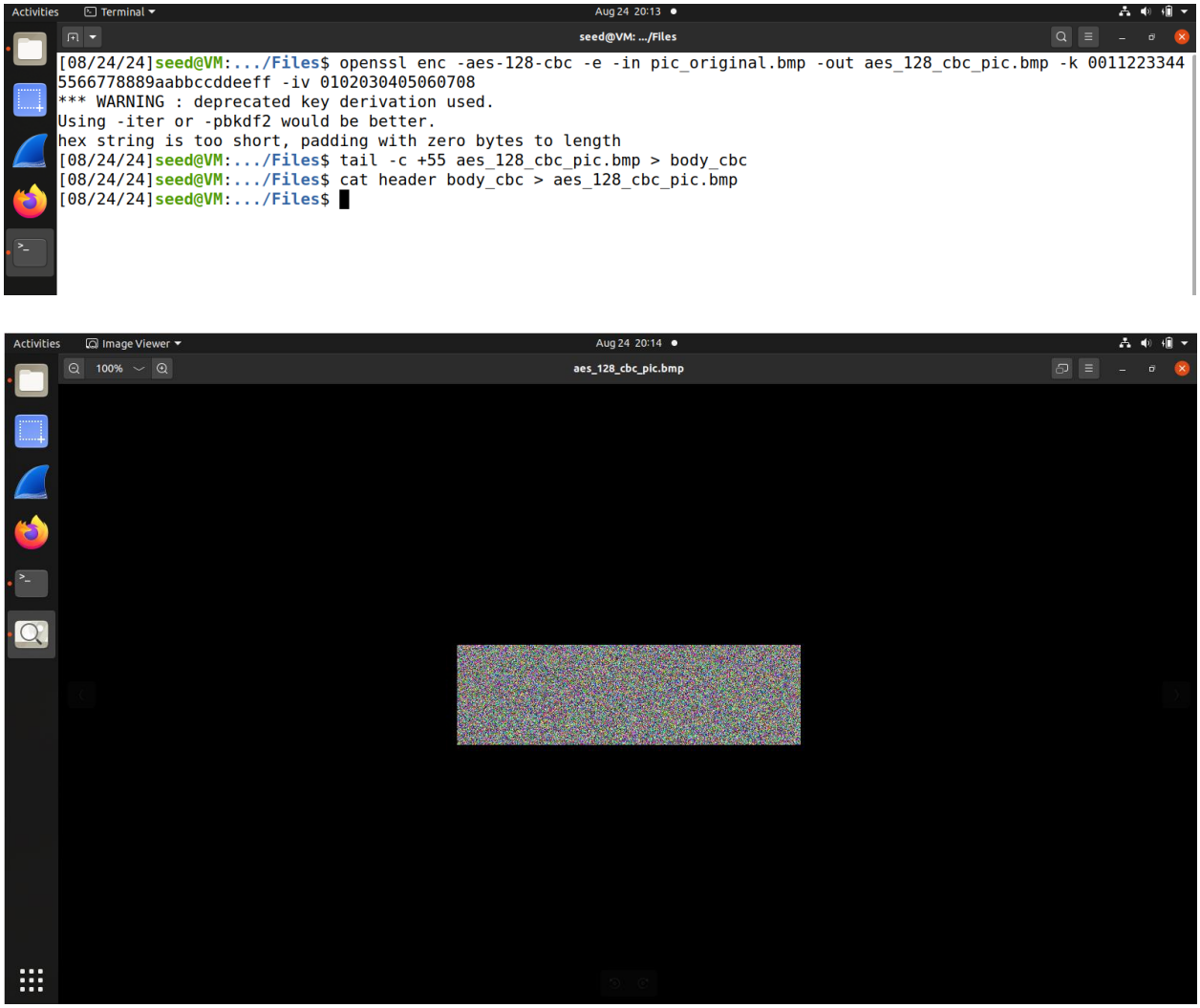


Figure 6. `aes_128_cbc_pic.bmp`

- Figure 5 and Figure 6 are the resulting encrypted images using EBC and CBC respectively. If we observe, we see that using EBC, we might not be able to identify the actual image perfectly but can easily get an idea of the content present in the image. This is because some contents of the image can still be estimated like the oval shape, the circle and the rectangle. On the other hand, we have the results of the encryption using CBC. There is barely any data or clue that an attacker can extract from Figure 6.

- Now let us examine another image.

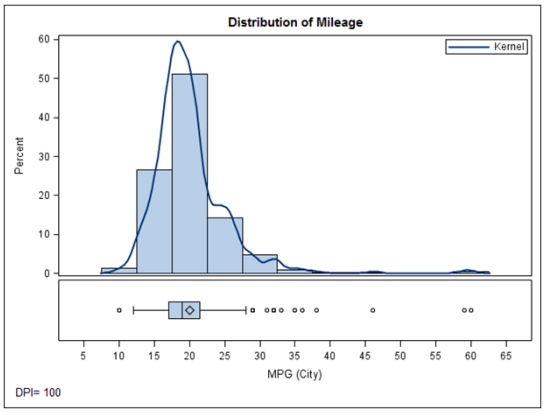


Figure 7. `my_image.bmp`

- Commands to run.
    - `$ openssl enc -aes-128-ecb -e -in my_image.bmp -out my_image_ecb.bmp -k 00112233445566778889aabbccddeeff`
    - `$ openssl enc -aes-128-cbc -e -in my_image.bmp -out my_image_cbc.bmp -k 00112233445566778889aabbccddeeff -iv 0102030405060708`
    - `$head -c 54 my_image.bmp > my_image_header`
    - `$tail -c +55 my_image_ecb.bmp > my_image_body_ecb`
    - `$tail -c +55 my_image_cbc.bmp > my_image_body_cbc`
    - `$cat my_image_header my_image_body_ecb > my_image_ecb.bmp`
    - `$cat my_image_header my_image_body_cbc > my_image_cbc.bmp`

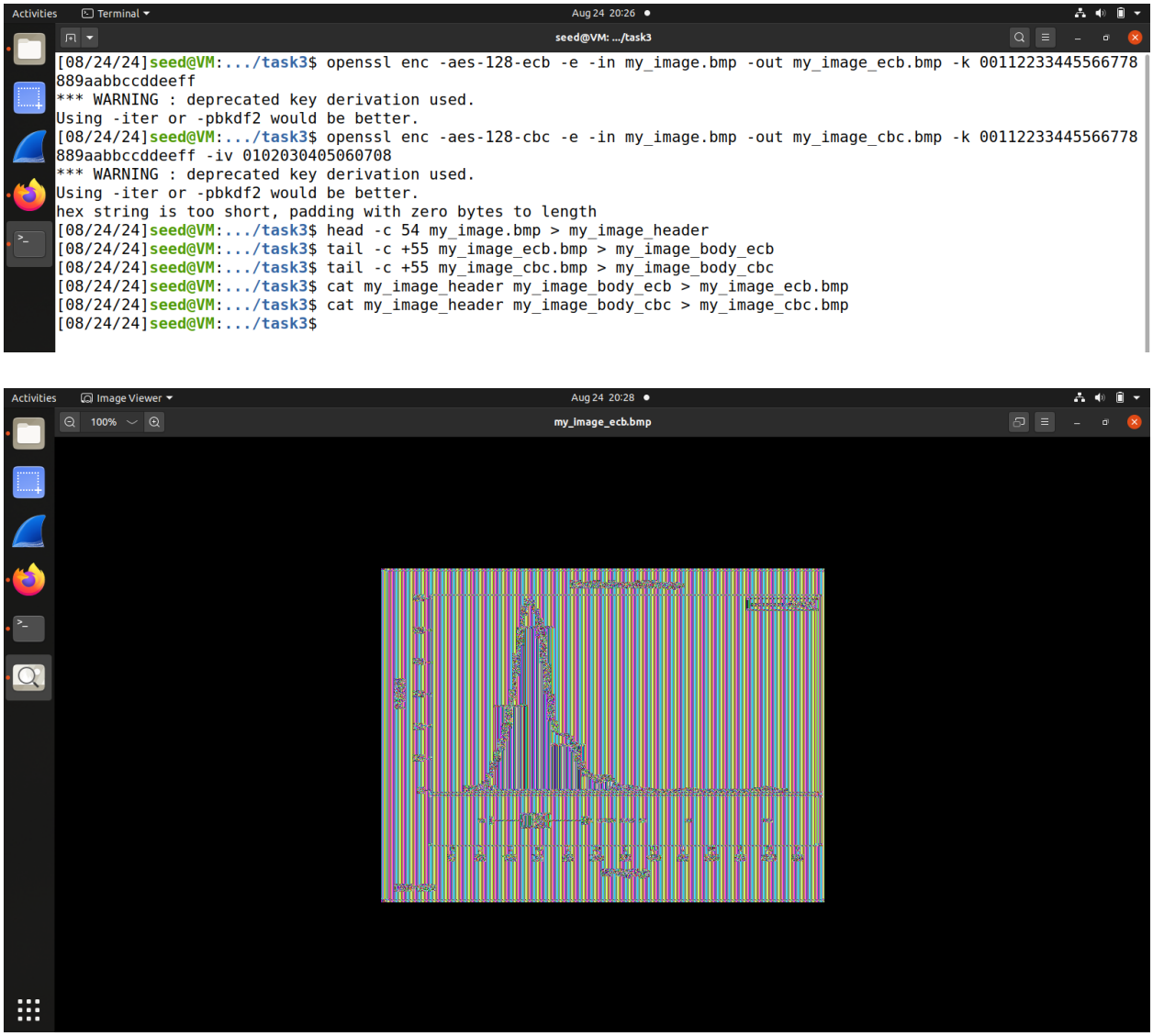


Figure 8. `my_image_ecb.bmp`

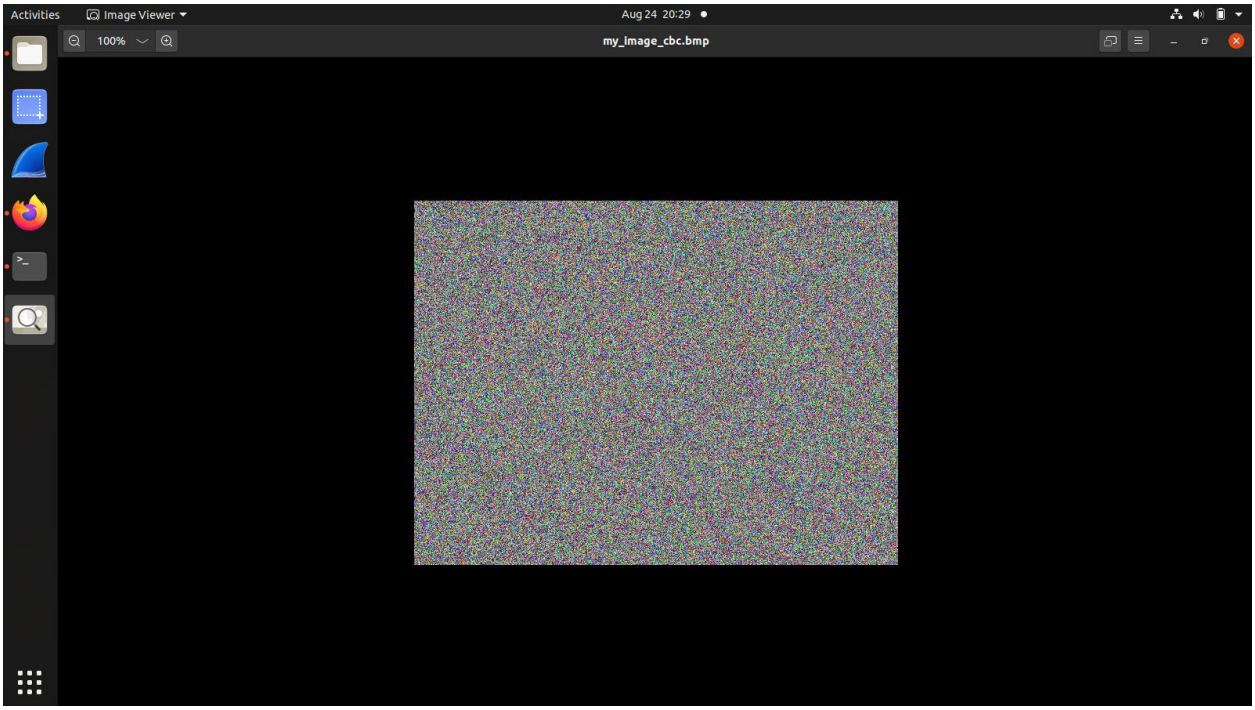


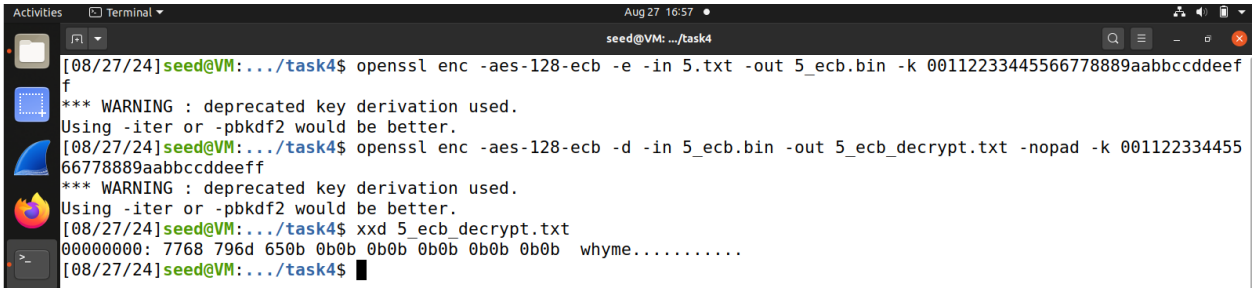
Figure 9. `my_image_cbc.bmp`

Task 4: Padding

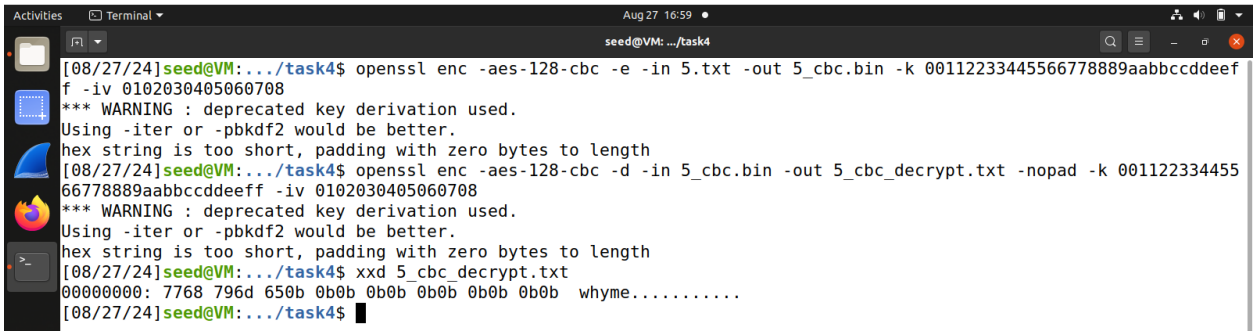
- We will be encrypting 3 files of lengths 5 bytes, 10 bytes and 16 bytes respectively using ECB, CBC, CFB and OFB modes of encrypting a file.
- We will be using the below ciphers
  - **`aes-128-ecb`** (Block cipher)(Doesn't require IV for encryption)
  - **`aes-128-cbc`** (Block cipher)
  - **`cast5-cfb`** (Converts block cipher to stream cipher)
  - **`cast5-ofb`** (Converts block cipher to stream cipher)
- Similarly for 10 bytes and 16 bytes file.
- Let's create files of sizes 5, 10 and 16 bytes.

1. 5 bytes file

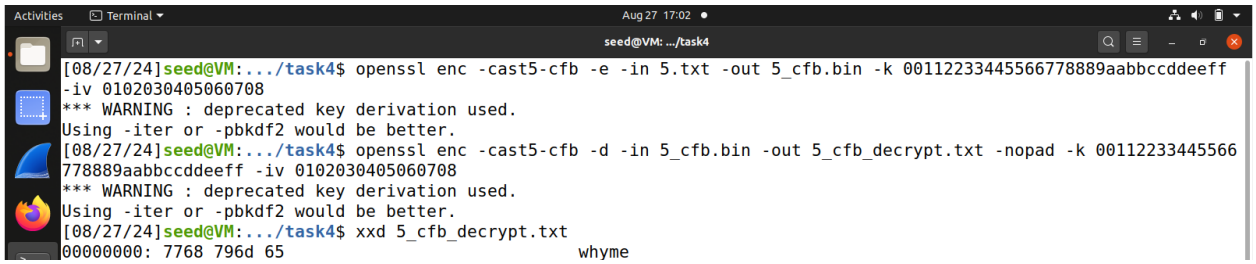
- ECB



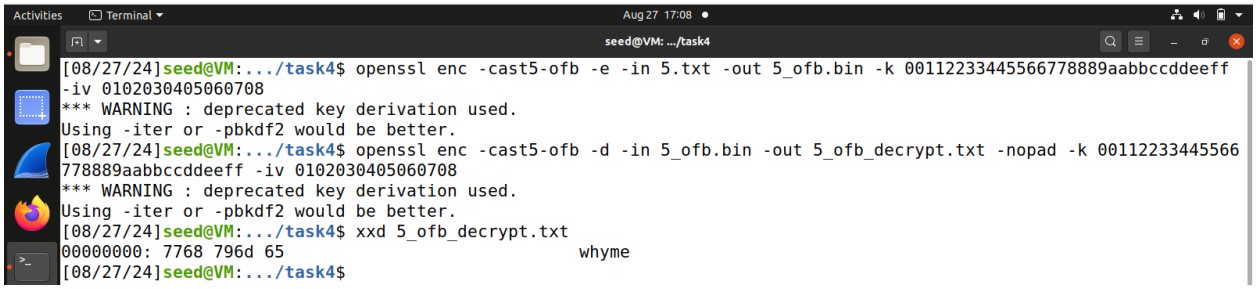
- CBC



- CFB

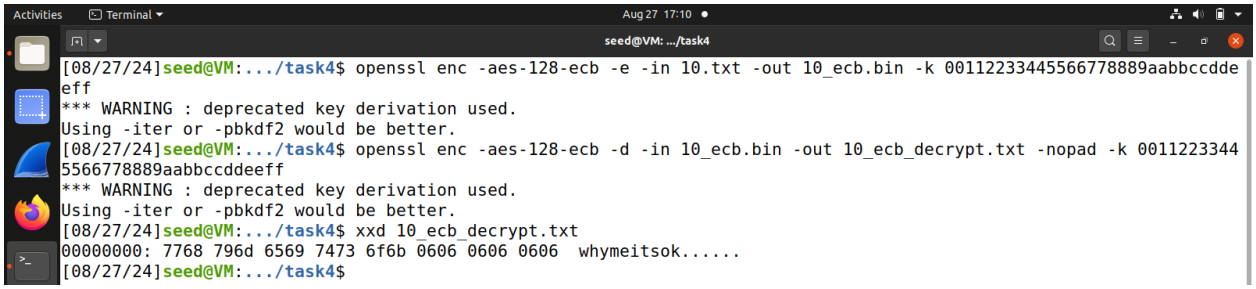


- OFB

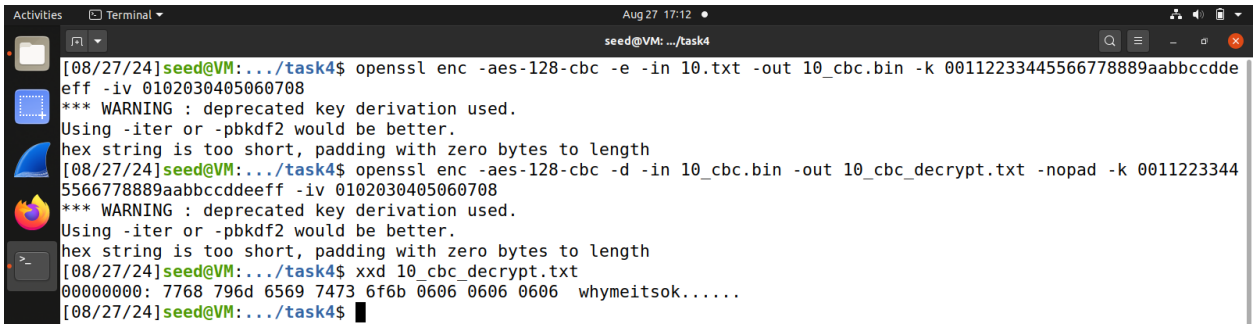


2. 10 bytes file

- ECB

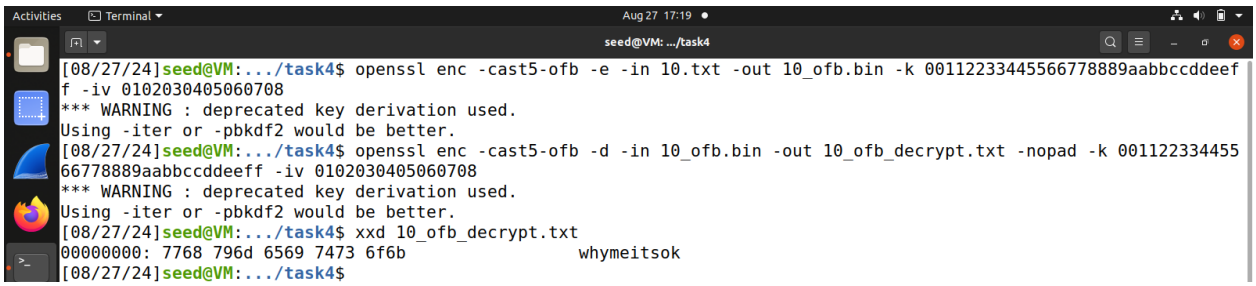


- CBC



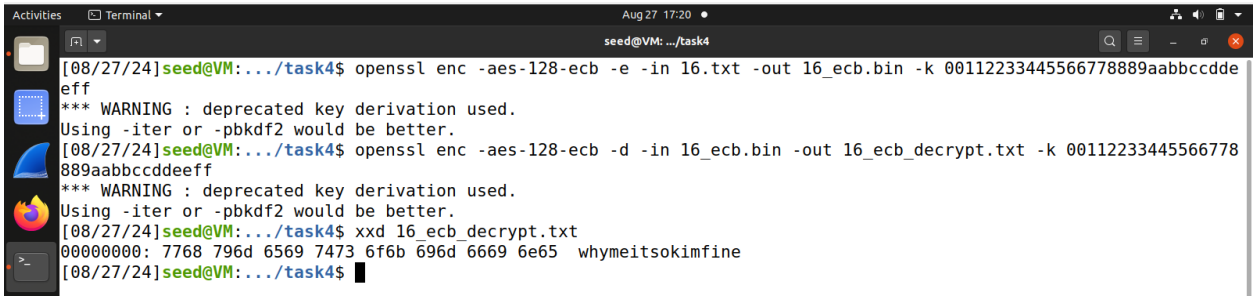
- CFB

- OFB

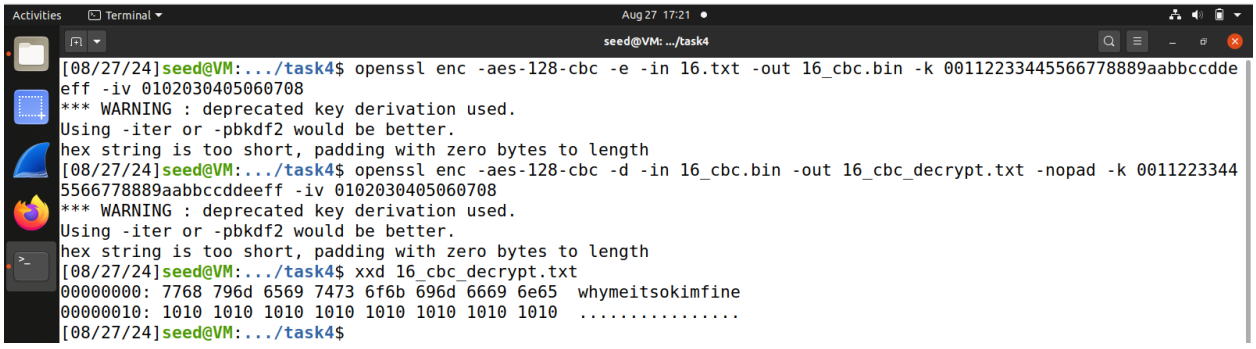


3. 16 bytes file

- ECB

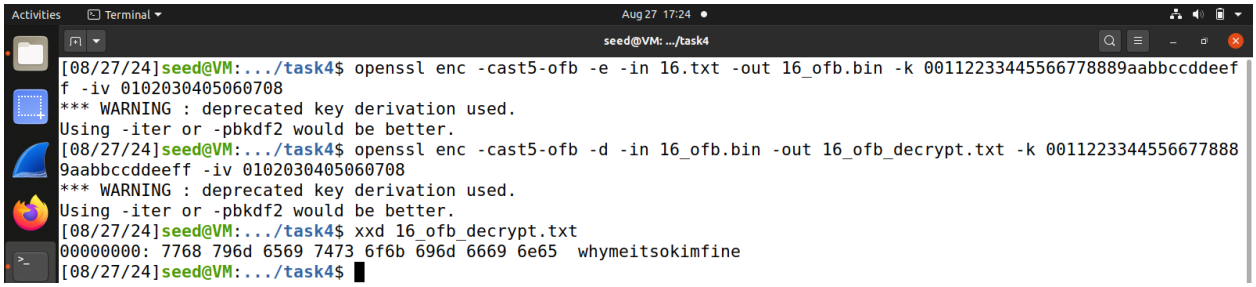


- CBC



- CFB

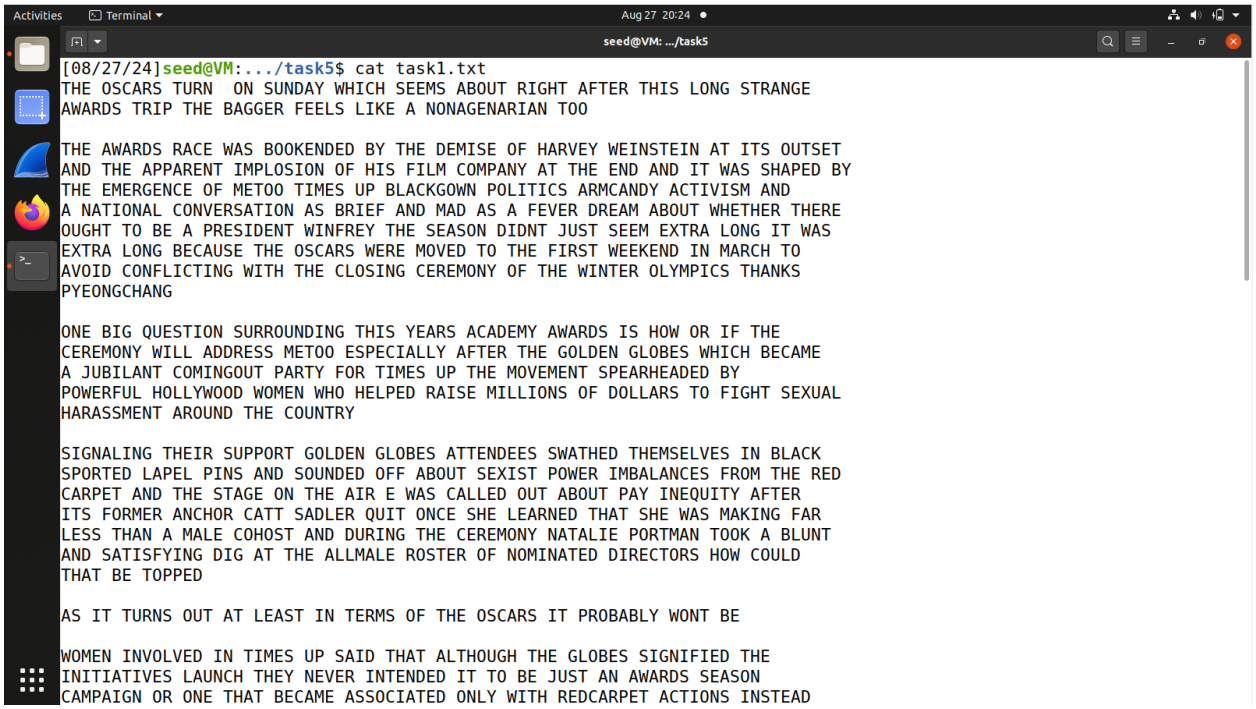
- OFB



- ECB and CBC mode of cipher performs padding to the plaintext while encrypting. The point is that these types of encryption modes require the size of the plaintext to be equal to the block size and hence it pads to the plaintext in order to perform encryption efficiently.
- Unlike the modes aforementioned, CFB and OFB don't perform any padding while encryption. This is because these modes convert the plaintext from blocks to stream cipher.
- For all the files are padded with the data that is based on the sizes of the files. Let's take an example where the size of the file is 5 bytes. For making the length equal to the key size of 128 bits or 16 bytes, we need 11 more bytes. In hexadecimal notation, it is 0x0b. And this is the data padded 11 times at the end of the 5 bytes file. Similarly, for the 10 bytes file, we need 0x06 data repeated for 6 times.

Task 5: Error Propagation – Corrupted Cipher Text

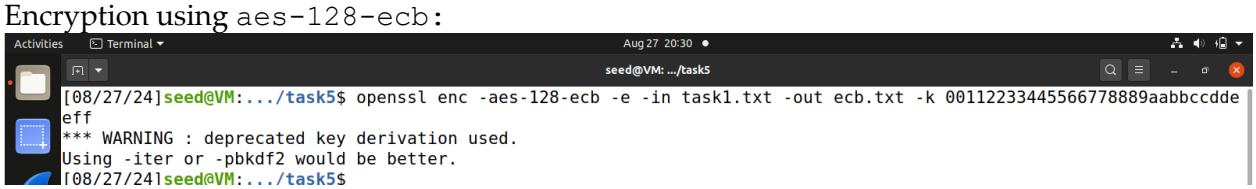
- I'll create a new folder `task5` and copied `task1.txt` file (the decrypted file of task1) in order to keep everything transparent and clear. I'll perform encryption of task1.txt using aes-128-ecb, aes-128-cbc, cast5-cfb and cast5-ofb modes of encryption. Once I encrypt, using bless hex editor, I'll make change to the 1st bit of the 55th byte of the encrypted file and then decrypt it.



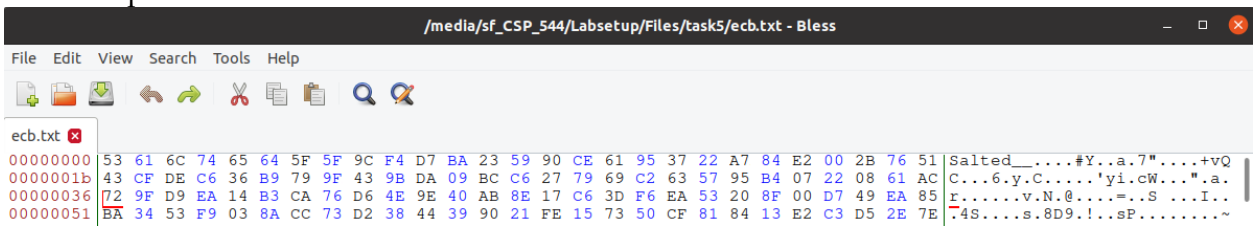
- **ECB:** Using ECB mode, only the block that has the corrupted cipher byte can't be retrieved and the rest of the file should be good as this mode is independent of other ciphers while decrypting. (Also called parallel encryption/decryption)
- **CBC:** During the decryption process, we first decrypt the cipher block with a correct key and then perform XOR operation with the IV. This will give us a plaintext of the first block. For the next block, we perform the same method but this time we take the previous block's cipher text instead of the IV. While decrypting the 55th byte containing block, we get corrupted plaintext due to the corrupted cipher block and then as we changed one bit in the cipher text, it gets propagated to the next block's plaintext resulting in change of one bit of the block.
- **CFB:** In this case, the bytes in the corrupted block can't be fetched along with the next couple of blocks or may be one or two as we take bits from the plaintext and use them in the shift register for decrypting the next block.
- **OFB:** The decryption of corrupted cipher text would yield us the plaintext except the byte that is corrupted as here we won't be passing the cipher text to the next steam of bits during encryption/decryption; instead we pass the 8 selected bits to the shift register.

1. **ECB**

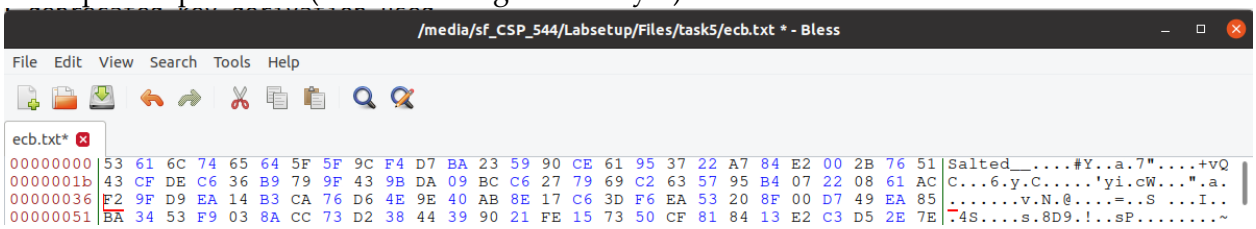
- Encryption using `aes-128-ecb`:



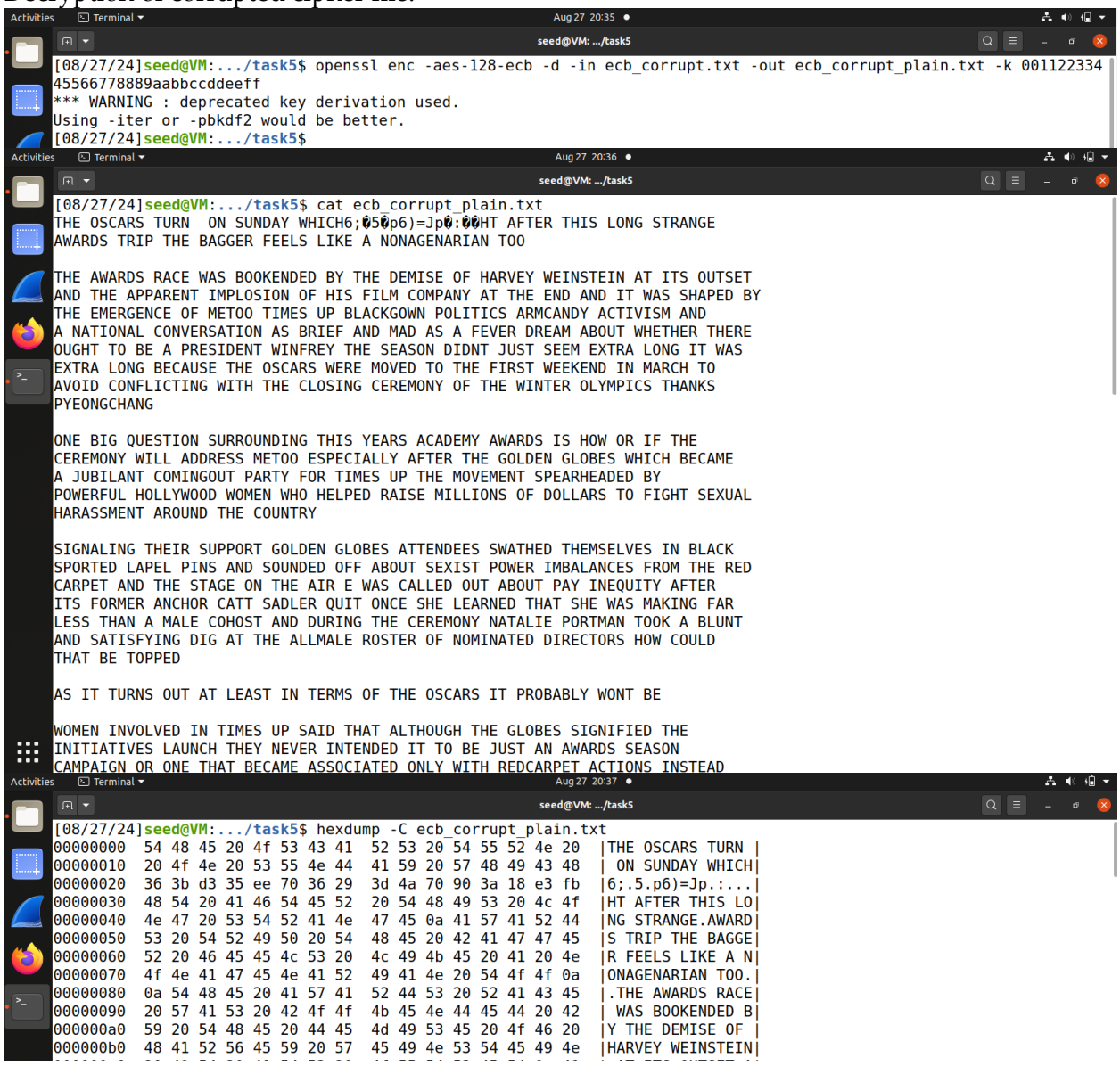
- Actual Cipher Text:



- Corrupted Cipher Text (note the change in 55th byte):

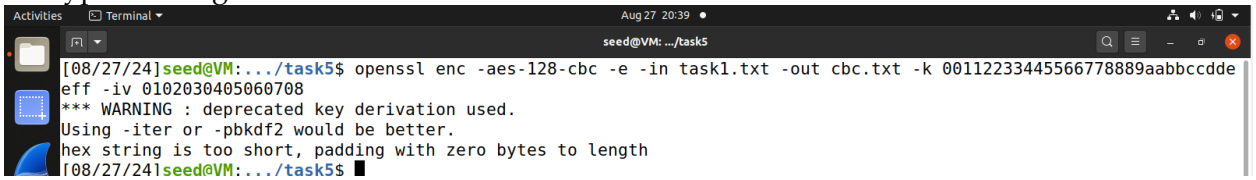


- Decryption of corrupted cipher file:

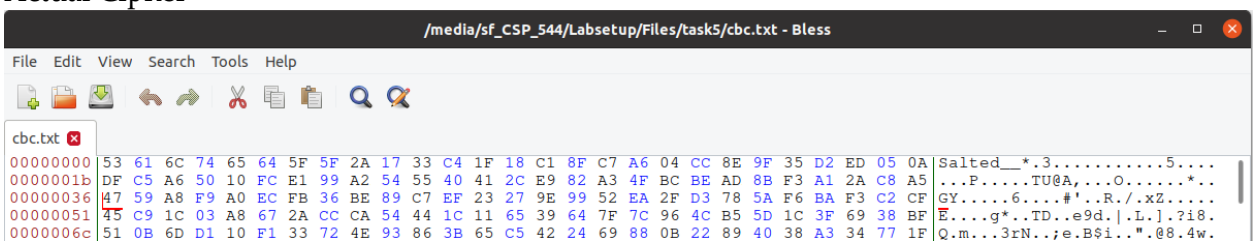


2. CBC

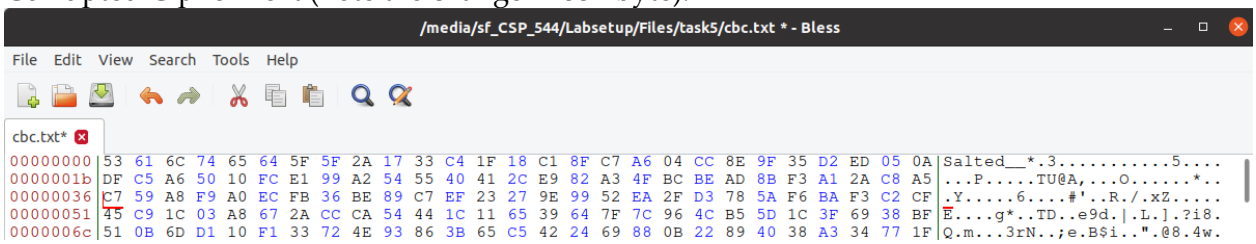
- Encryption using `aes-128-cbc`:



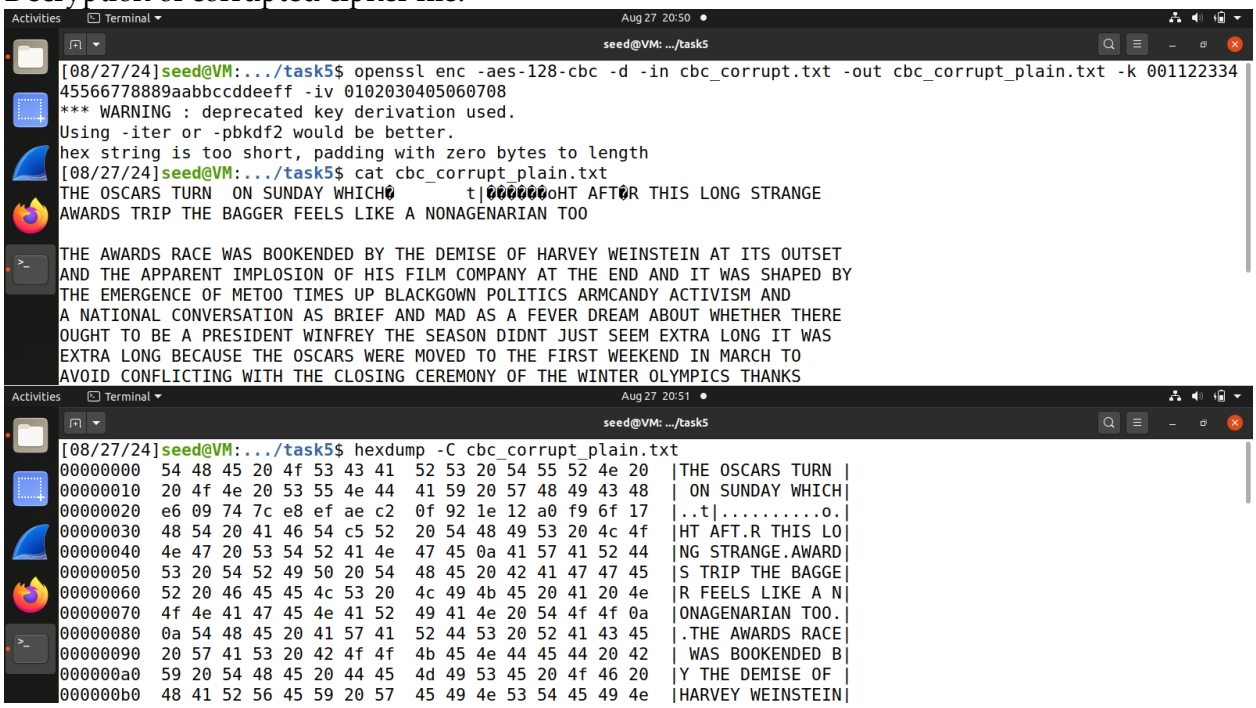
- Actual Cipher



- Corrupted Cipher Text (note the change in 55th byte):

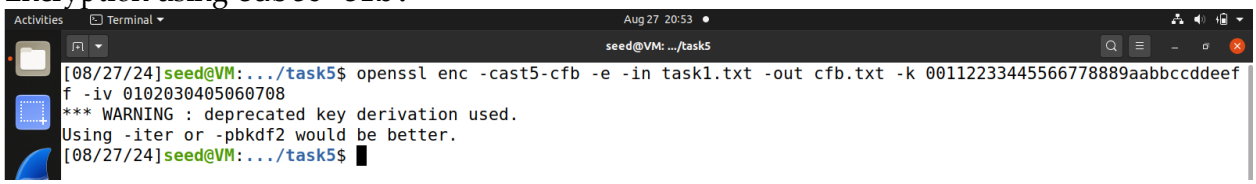


- Decryption of corrupted cipher file:

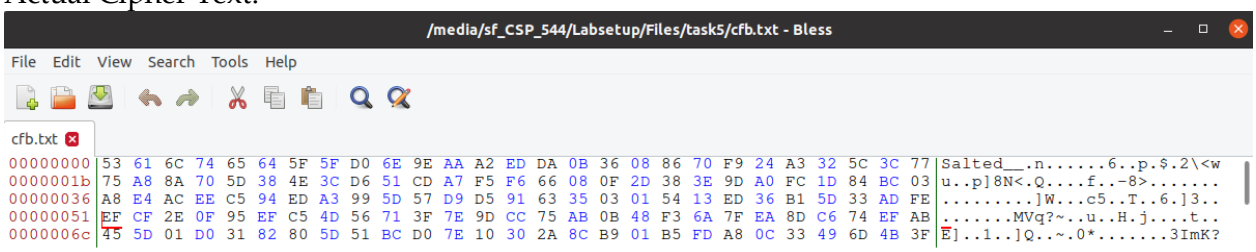


3. CFB

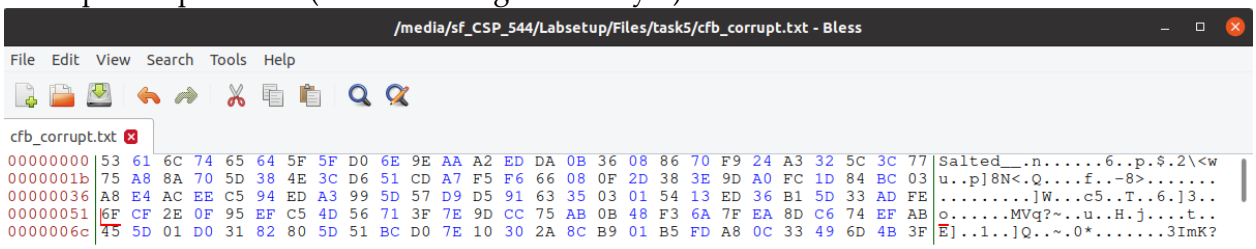
- Encryption using `cast5-cfb`:



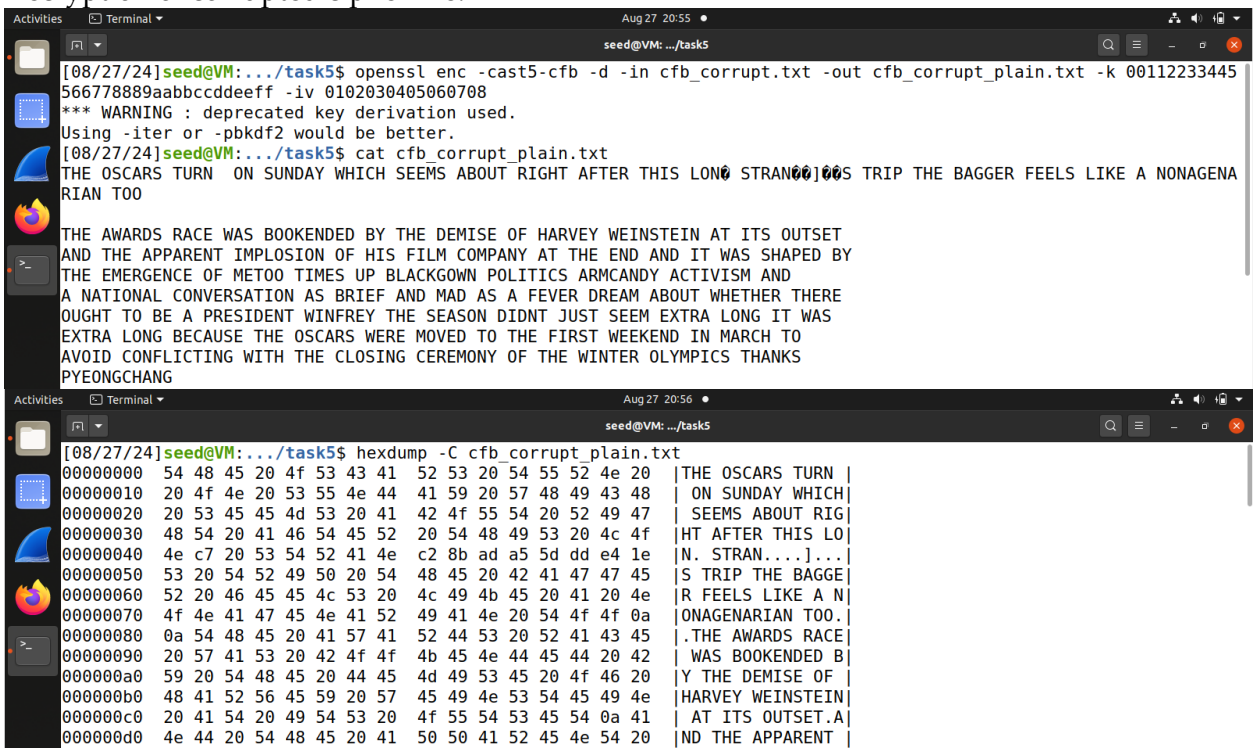
- Actual Cipher Text:



- Corrupted Cipher Text (note the change in 55th byte):

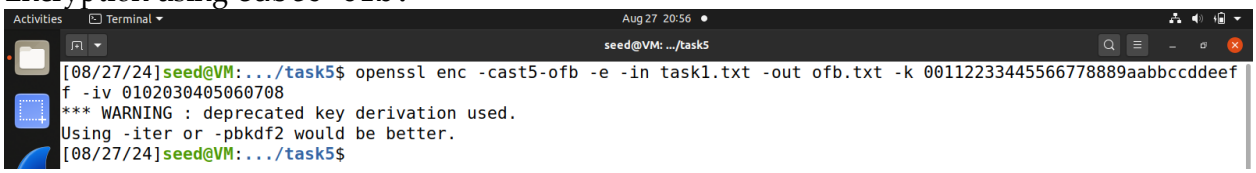


- Decryption of corrupted cipher file:

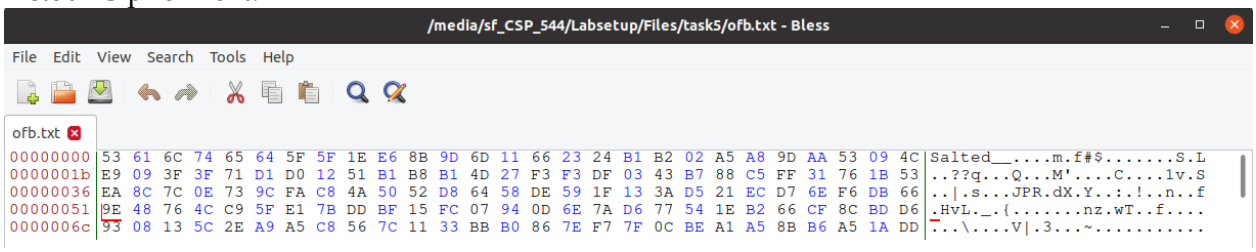


4. OFB

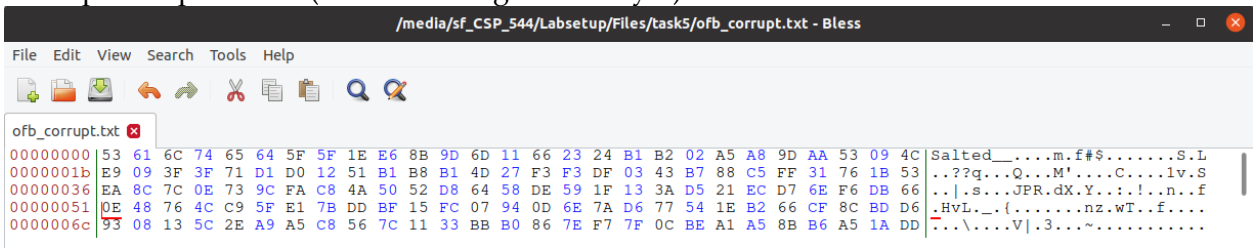
- Encryption using `cast5-ofb`:



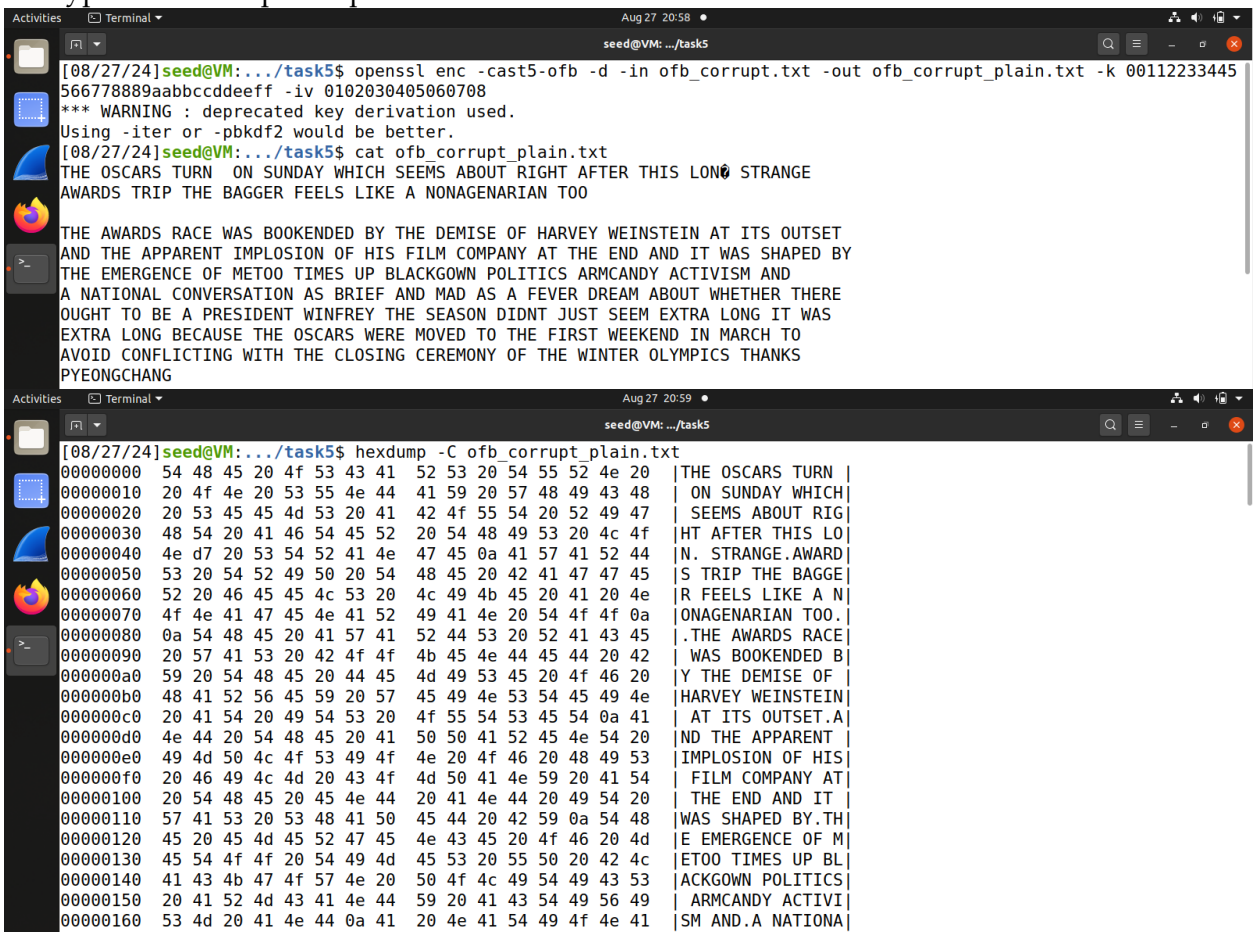
- Actual Cipher Text:



- Corrupted Cipher Text (note the change in 55th byte):



- Decryption of corrupted cipher file:



Out of the 4 expected observations, 3 are correct. CFB looks wrong; the cause might be the fact that the bits used in the shift register weren't a part of the corrupted byte.

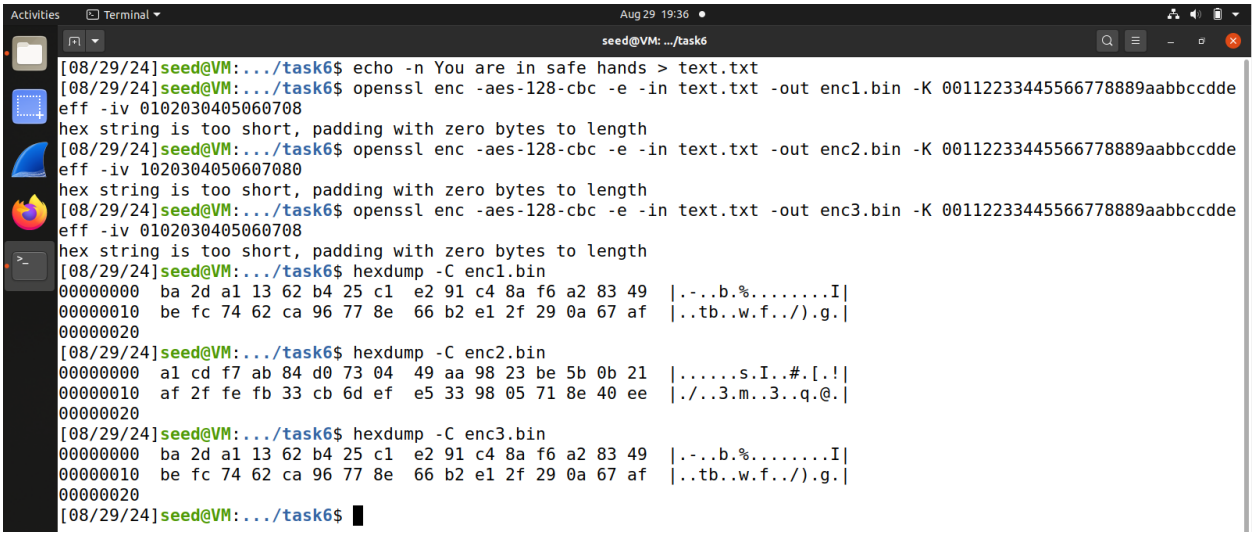
Task 6: Initial Vector (IV) and Common Mistakes

Task 6.1. IV Experiment

We create a file `text.txt` which has the string "`Hello, I love Cybersecurity`" as input for encryption and the encryption mode here is `aes-128-cbc`. For the first 2 encryptions, I've used different initial vectors and the last command used the same initial vector.

- Output cipher file for 1st encryption using IV1: `enc1.bin`
- Output cipher file for 2nd encryption using IV2: `enc2.bin`
- Output cipher file for 3rd encryption using IV1: `enc3.bin`

Let us compare 1st and 2nd cipher files and then 1st and 3rd cipher files using the `$hexdump -C <file_name>`.

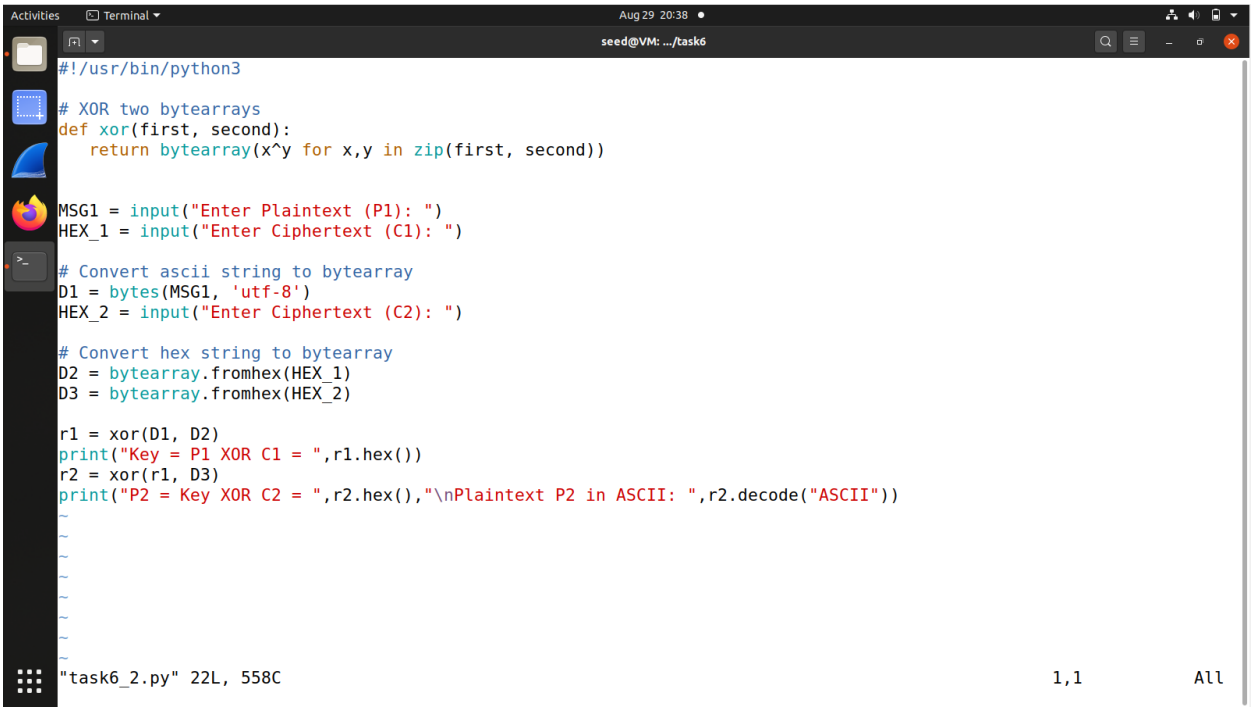


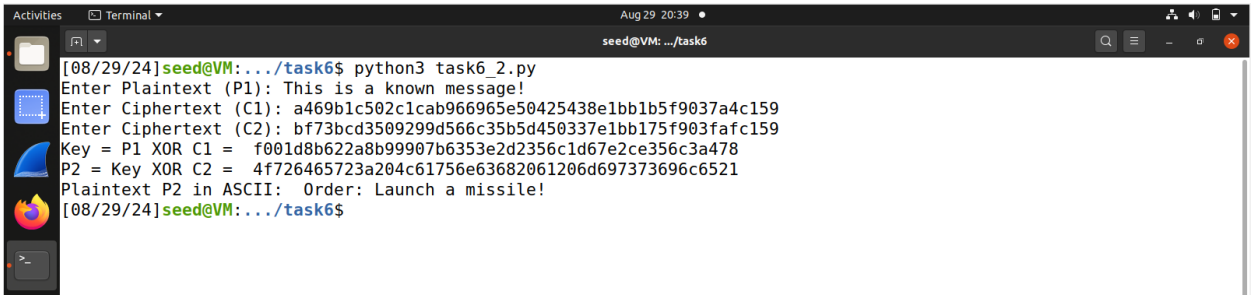
- If we observe carefully, we see that using the same IV for a key and encrypt a file; we obtain the same cipher text. This signifies the importance of having unique initial vector. We should be selecting different IV for a key in order to make it difficult for an adversary to predict the plaintext.
- `enc1.bin` and `enc3.bin` have the same information stored and `enc2.bin` is different because of the change in IV while encrypting.

Task 6.2. Common Mistake: Use the Same IV

- For this task, I've developed a script `task6_2.py` which performs operations similar to `sample.py`. Below is `task6_2.py`. The script is self-explanatory for any python developer as it's a basic code that coverts ASCII to HEX and then to BYTE ARRAY and finally back from HEX to ASCII.
- This script performs XOR operations to find plaintexts and keys. It starts by defining a function to XOR two byte arrays. The user is prompted to enter the first plaintext (P1) and two cipher texts in hexadecimal format (C1 and C2). The script then converts these inputs into byte arrays. Using XOR, it calculates the key by XOR-ing the plaintext with the first cipher text and prints the key in hexadecimal format. It then calculates the second plaintext by XOR-ing the key with the second cipher text and prints the result both in hexadecimal and ASCII formats.

```
#!/usr/bin/python3

# XOR two bytearrays
def xor(first, second):
   return bytearray(x^y for x,y in zip(first, second))


MSG1 = input("Enter Plaintext (P1): ")
HEX_1 = input("Enter Ciphertext (C1): ")

# Convert ascii string to bytearray
D1 = bytes(MSG1, 'utf-8')
HEX_2 = input("Enter Ciphertext (C2): ")

# Convert hex string to bytearray
D2 = bytearray.fromhex(HEX_1)
D3 = bytearray.fromhex(HEX_2)

r1 = xor(D1, D2)
print("Key = P1 XOR C1 = ",r1.hex())
r2 = xor(r1, D3)
print("P2 = Key XOR C2 = ",r2.hex(),"\nPlaintext P2 in ASCII: ",r2.decode("ASCII"))
```

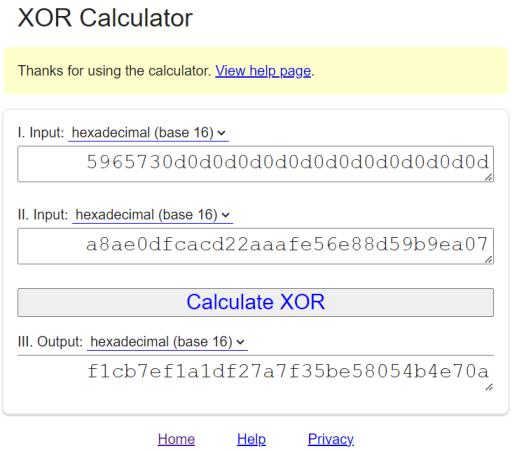




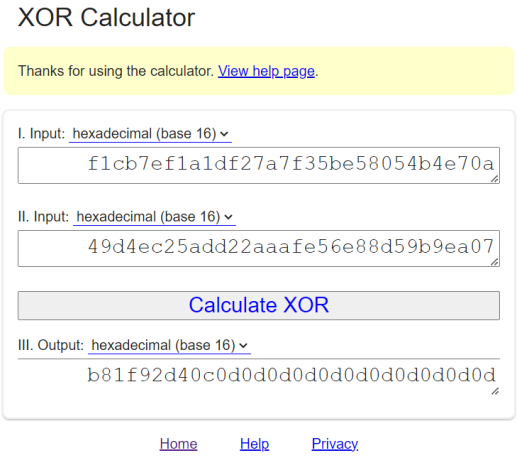
- If OFB is replaced with CFB, then only the first block can be revealed from the plaintext. If the IV is reused, the whole key stream will be reproduced again, since the keystream is done by successive encryption of the IV, i.e., $K_0 = IV$, $K_1 = E_k(C_0)$, $K_2 = E_k(C_1)$, ...

Task 6.3. Common Mistake: Use a Predictable IV

- I guess the plaintext (`P1`) to be "`Yes`". So, I construct `P2` such that `P2 = P1 XOR IV XOR Next_IV` where `IV` is the IV used to generate C1 by Bob and `Next_IV` is the predictable IV used to encrypt the next plaintext input. "`Yes`" in hex is 596573. I padded it to make sure it is of 16 bytes or 128 bites.
- To perform XOR, I used xor.pw/#.
- `temp = P1 XOR IV`

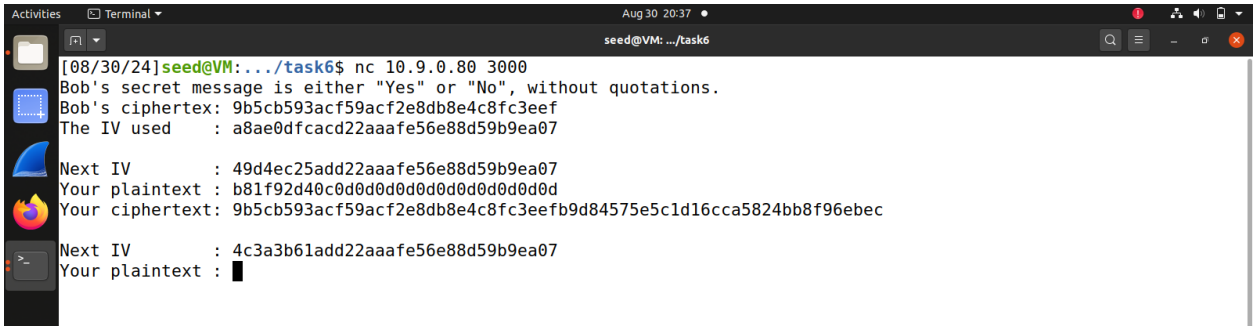


- `temp XOR Next_IV`



- Below is the data.

```
Bob's secret message is either "Yes" or "No", without quotations.
Bob's ciphertex: 9b5cb593acf59acf2e8db8e4c8fc3eef
The IV used    : a8ae0dfcacd22aaafe56e88d59b9ea07

Next IV        : 49d4ec25add22aaafe56e88d59b9ea07
Your plaintext : b81f92d40c0d0d0d0d0d0d0d0d0d0d0d
Your ciphertext: 9b5cb593acf59acf2e8db8e4c8fc3eefb9d84575e5c1d16cca5824bb8f96ebec

Next IV        : 4c3a3b61add22aaafe56e88d59b9ea07
Your plaintext :
```



By exploiting the predictability of Bob's IVs in AES-CBC mode, I constructed a specific input to the oracle to compare the resulting cipher texts. " This demonstrates the vulnerability of using predictable IVs, even with strong encryption like AES.

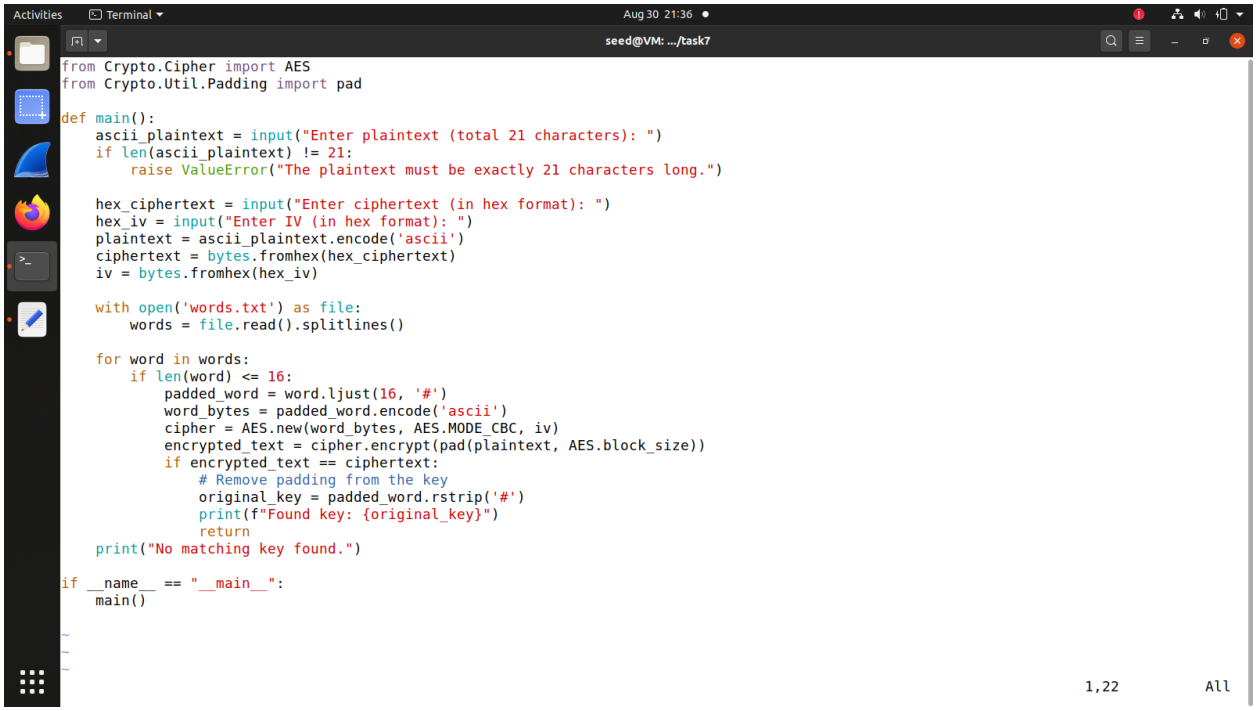
Task 7: Programming using the Crypto Library

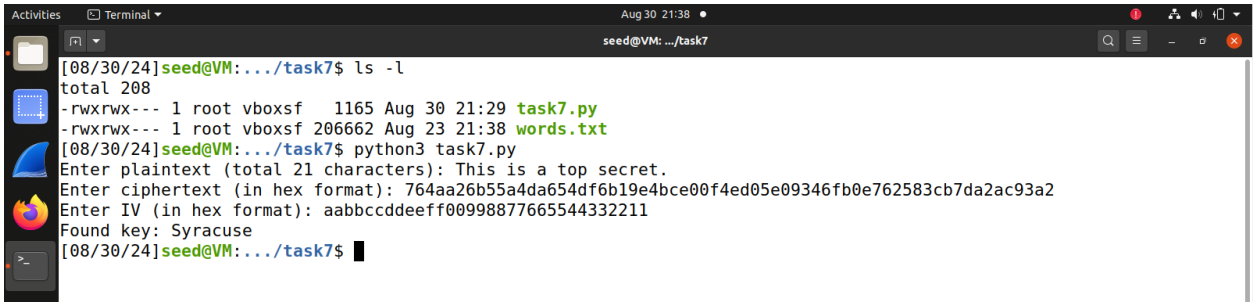
- The following Python code looks for the correct key from a list `words.txt`, padding each word to 16 bytes if necessary with a `#`, and checks if it matches the cipher text when used with `AES-128-CBC` encryption.
- The code begins by accepting an input in ASCII format, which must be exactly 21 characters in length. If the input length deviates from this requirement, an error is raised. Subsequently, the code requests the ciphertext and Initialization Vector (IV) used for encryption. All inputs are then converted into the appropriate byte array formats.

- The code proceeds to open a file containing a list of potential keys. For each key in the file, the code performs AES-128 encryption in Cipher Block Chaining (CBC) mode using the given IV. It then compares the resulting ciphertext with the provided ciphertext. If a match is found, the key is identified as correct; otherwise, the search continues.

- Before encryption, the code ensures that each key is precisely 16 bytes in length. If a key is shorter than 16 bytes, it is padded with # characters to achieve the required length.

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

def main():
    ascii_plaintext = input("Enter plaintext (total 21 characters): ")
    if len(ascii_plaintext) != 21:
        raise ValueError("The plaintext must be exactly 21 characters long.")

    hex_ciphertext = input("Enter ciphertext (in hex format): ")
    hex_iv = input("Enter IV (in hex format): ")
    plaintext = ascii_plaintext.encode('ascii')
    ciphertext = bytes.fromhex(hex_ciphertext)
    iv = bytes.fromhex(hex_iv)

    with open('words.txt') as file:
        words = file.read().splitlines()

    for word in words:
        if len(word) <= 16:
            padded_word = word.ljust(16, '#')
            word_bytes = padded_word.encode('ascii')
            cipher = AES.new(word_bytes, AES.MODE_CBC, iv)
            encrypted_text = cipher.encrypt(pad(plaintext, AES.block_size))
            if encrypted_text == ciphertext:
                # Remove padding from the key
                original_key = padded_word.rstrip('#')
                print(f"Found key: {original_key}")
                return
    print("No matching key found.")

if __name__ == "__main__":
    main()
```


Activities
Terminal
Aug 30 21:36
seed@VM: .../task7
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad
def main():
ascii_plaintext = input("Enter plaintext (total 21 characters): ")
if len(ascii_plaintext) != 21:
raise ValueError("The plaintext must be exactly 21 characters long.")
hex_ciphertext = input("Enter ciphertext (in hex format): ")
hex_iv = input("Enter IV (in hex format): ")
plaintext = ascii_plaintext.encode('ascii')
ciphertext = bytes.fromhex(hex_ciphertext)
iv = bytes.fromhex(hex_iv)
with open('words.txt') as file:
words = file.read().splitlines()
for word in words:
if len(word) <= 16:
padded_word = word.ljust(16, '#')
word_bytes = padded_word.encode('ascii')
cipher = AES.new(word_bytes, AES.MODE_CBC, iv)
encrypted_text = cipher.encrypt(pad(plaintext, AES.block_size))
if encrypted_text == ciphertext:
# Remove padding from the key
original_key = padded_word.rstrip('#')
print(f"Found key: {original_key}")
return
print("No matching key found.")
if __name__ == "__main__":
main()
1,22
All


Activities
Terminal
Aug 30 21:38
seed@VM: .../task7
[08/30/24]seed@VM:.../task7$ ls -l
total 208
-rwxrwx--- 1 root vboxsf 1165 Aug 30 21:29 task7.py
-rwxrwx--- 1 root vboxsf 206662 Aug 23 21:38 words.txt
[08/30/24]seed@VM:.../task7$ python3 task7.py
Enter plaintext (total 21 characters): This is a top secret.
Enter ciphertext (in hex format): 764aa26b55a4da654df6b19e4bce00f4ed05e09346fb0e762583cb7da2ac93a2
Enter IV (in hex format): aabbccddeeff00998877665544332211
Found key: Syracuse
[08/30/24]seed@VM:.../task7$