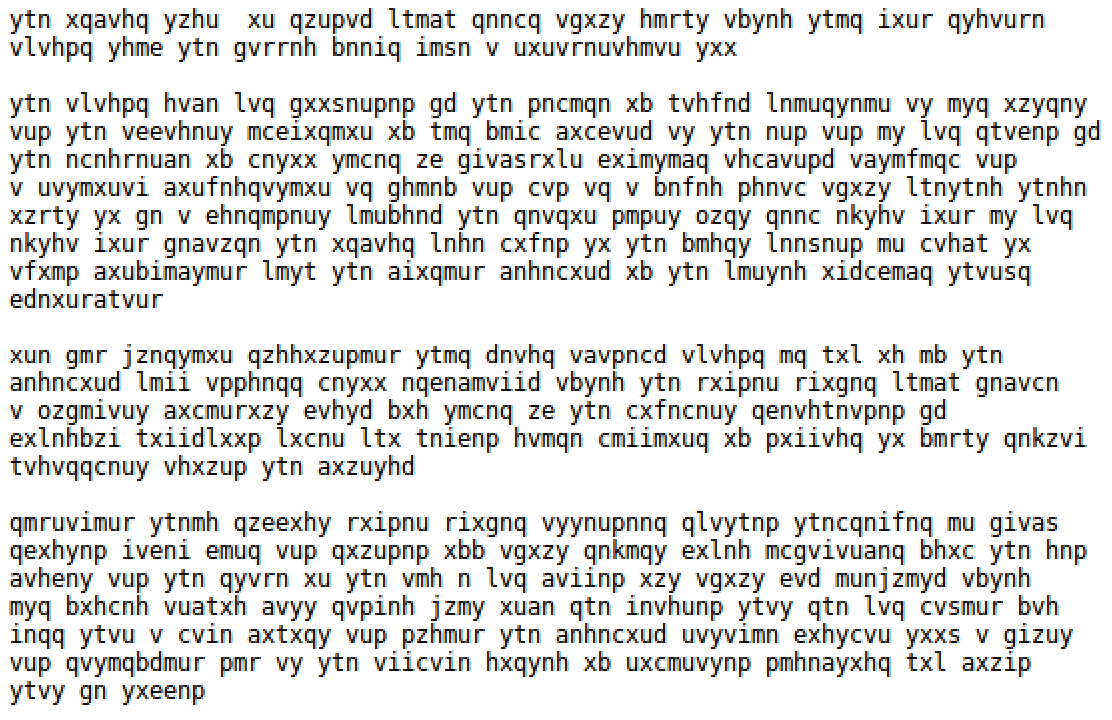
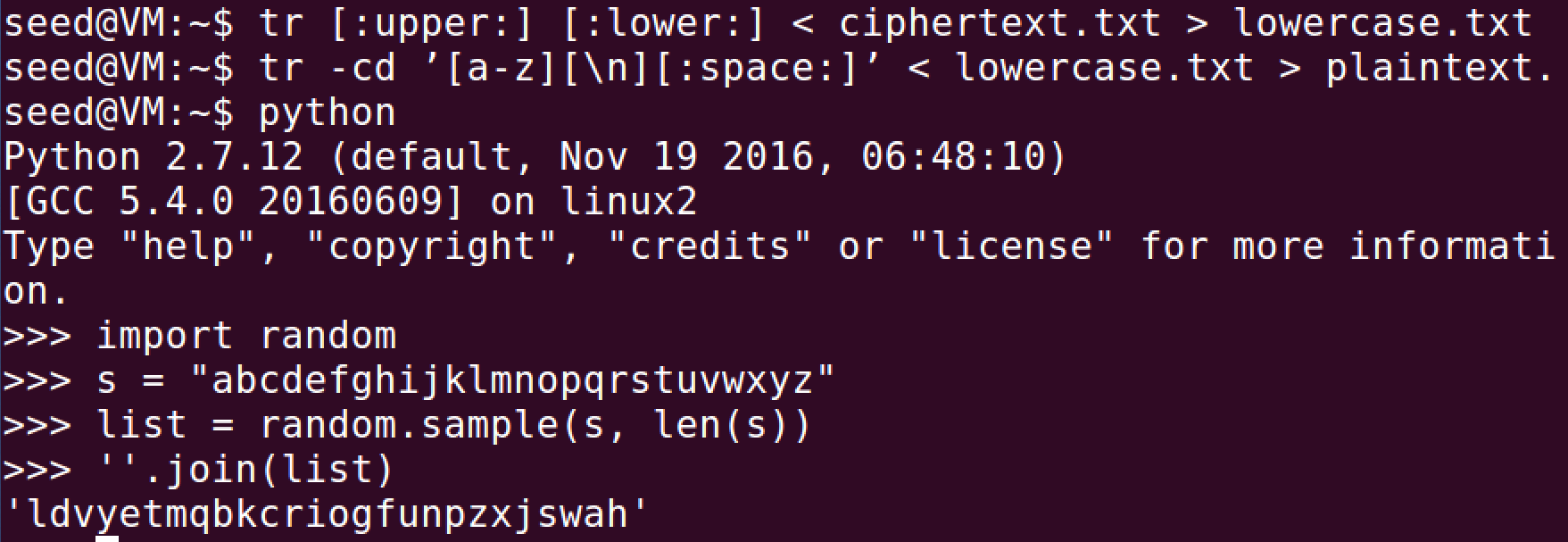
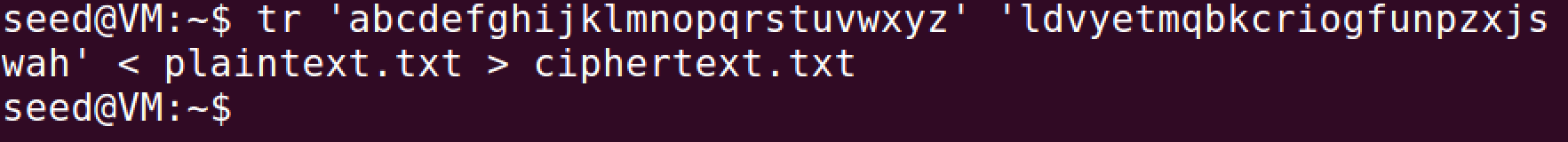
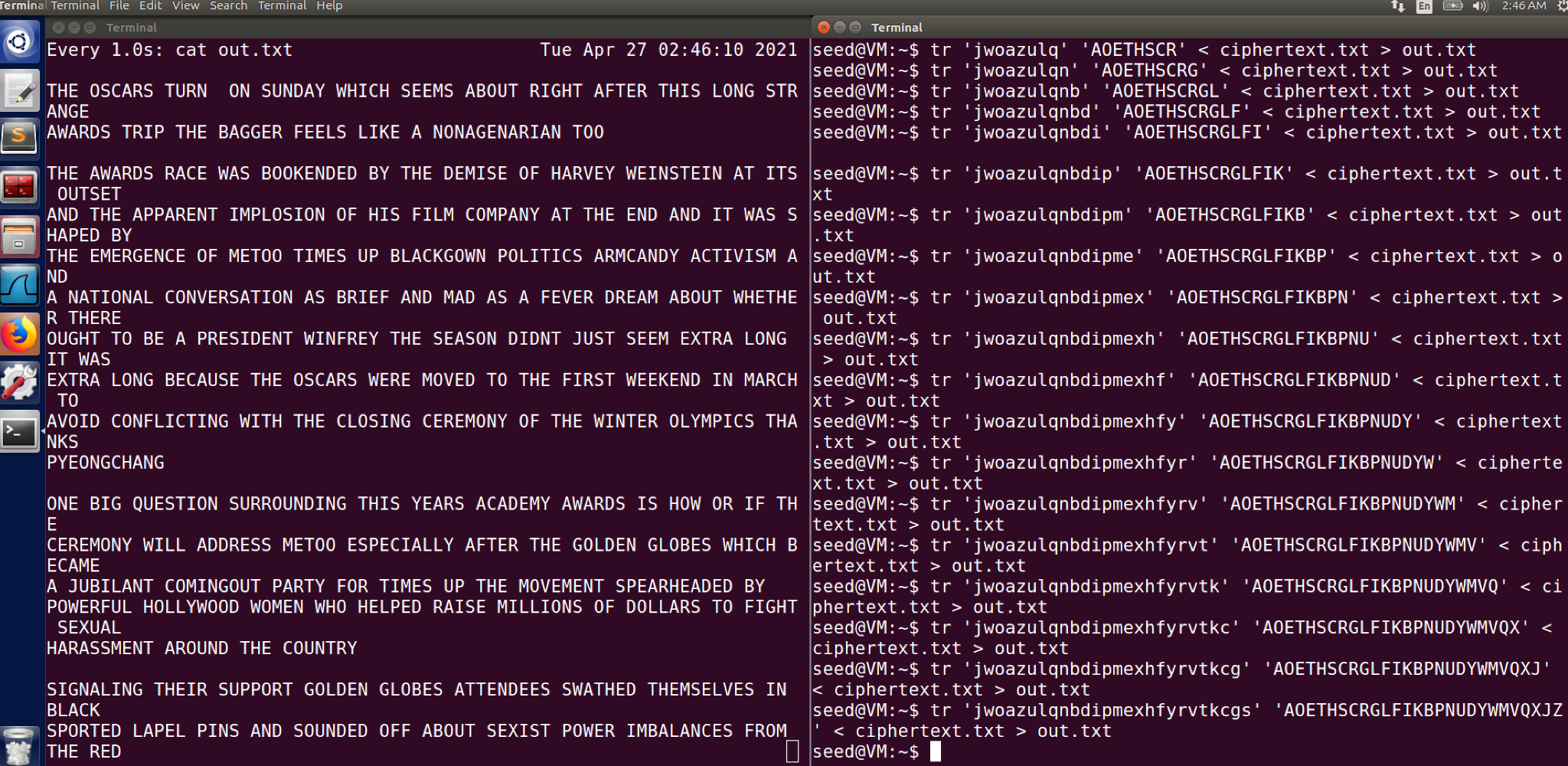
Lab 10

Task 1:





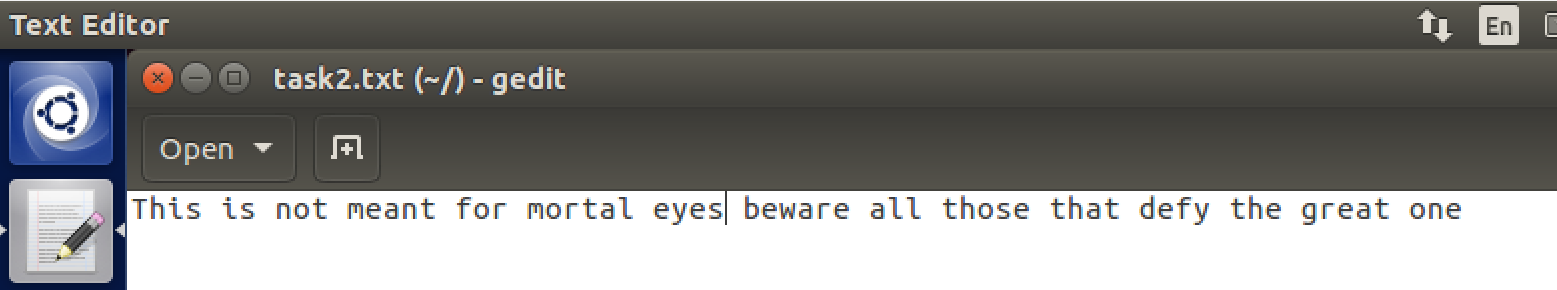


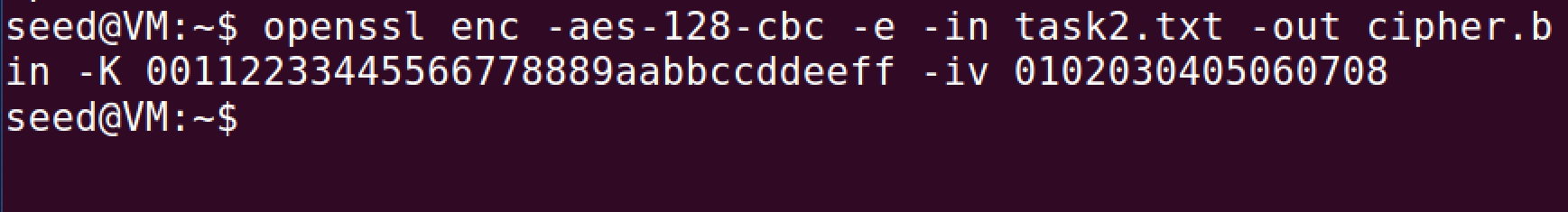


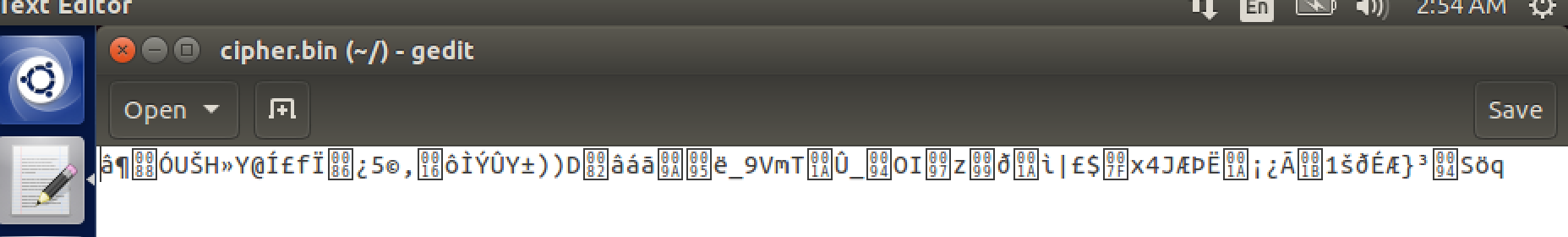
Comments:

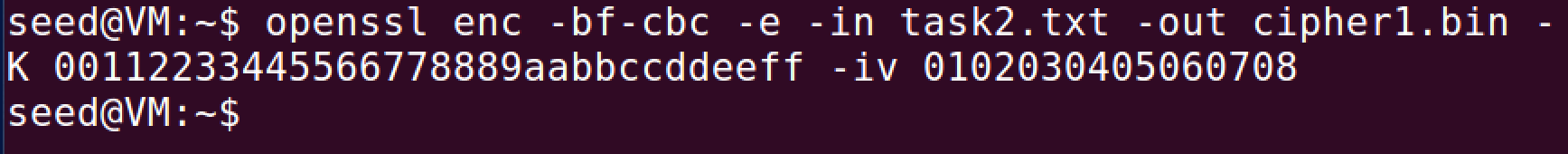
For the encrypted file we do some simplification and convert all uppercase to lower cases and then remove all the punctuations and numbers. Next is to generate the encryption key. Then we use the tr command to do the encryption while leaving the spaces and return characters alone. Then I just use estimated guesses to try and decode the encrypted text.

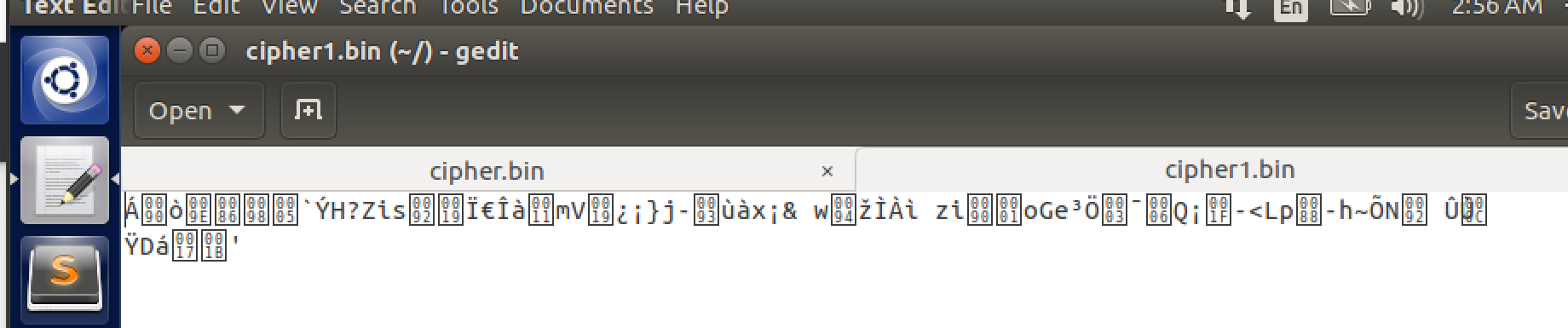
Task 2:

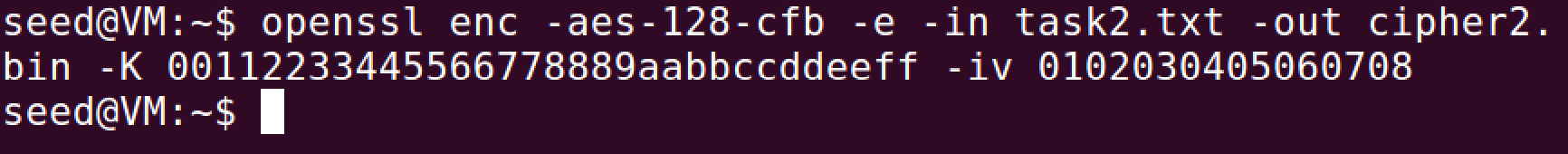


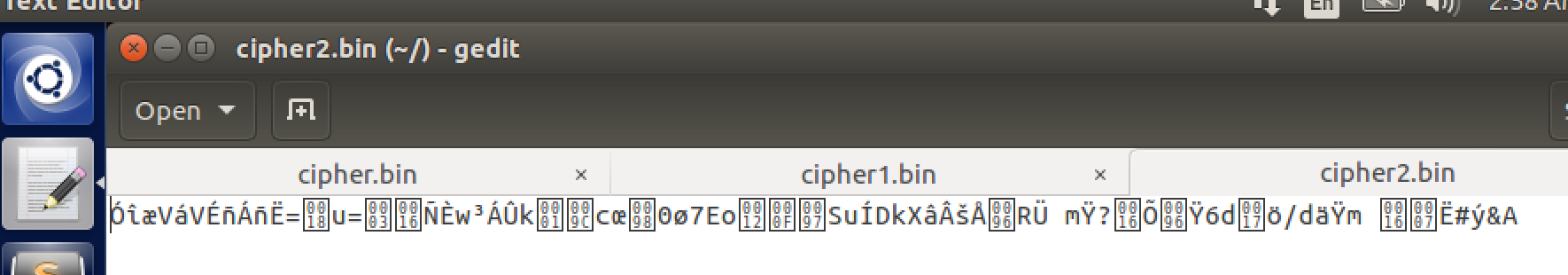










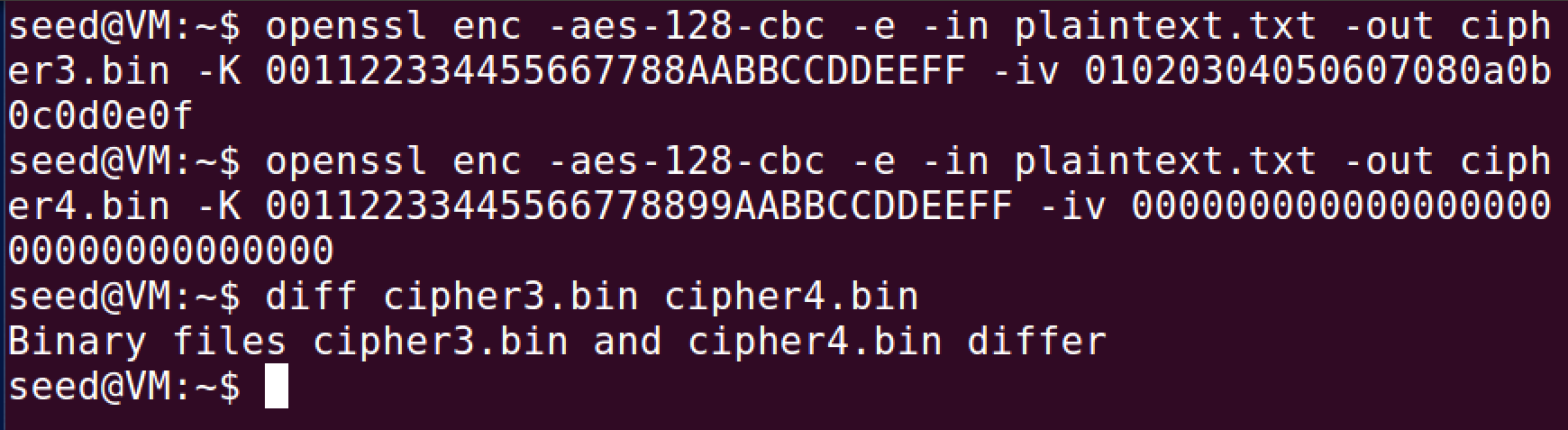


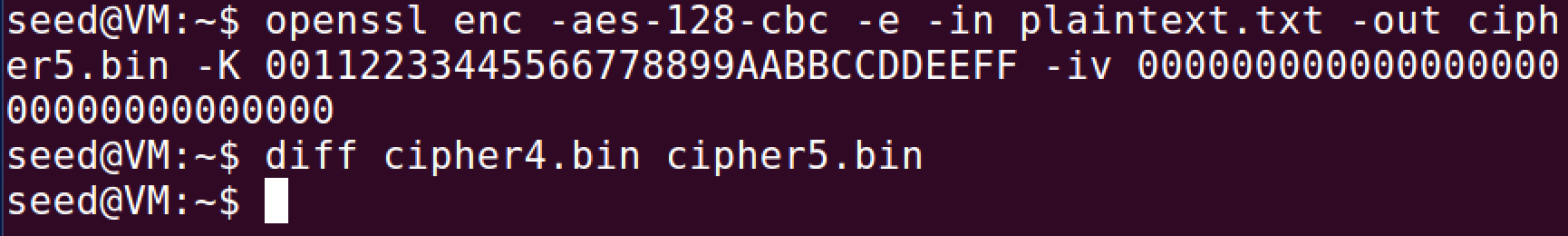
Comments:

So the three cipher types I used were -aes-128-cbc, -bf-cbc, and -aes-128-cfb. As shown each of them lead to each cipher.bin being encrypted differently.

Task 6:

6.1:

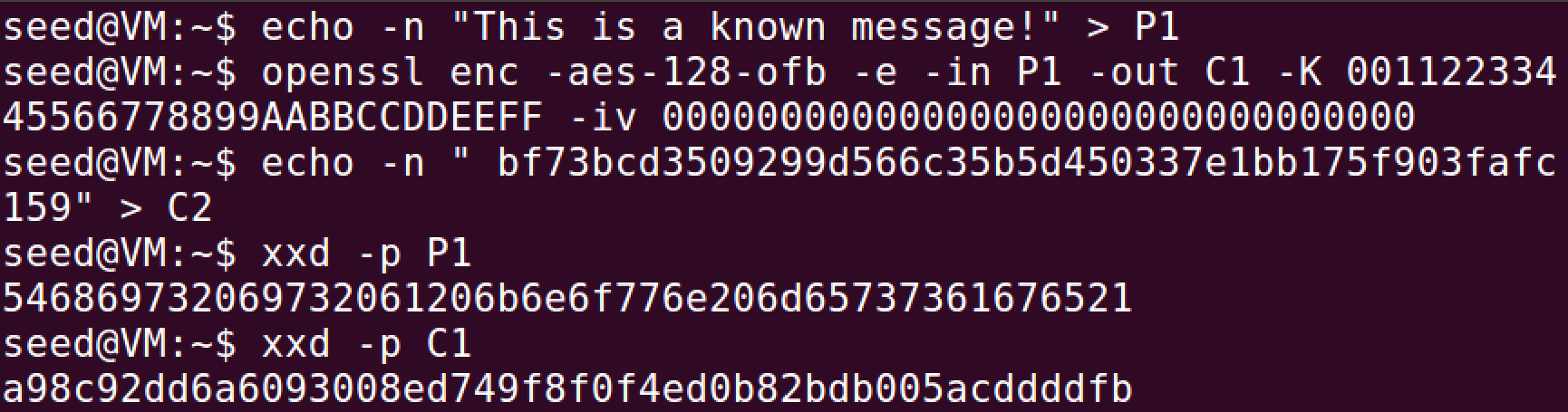


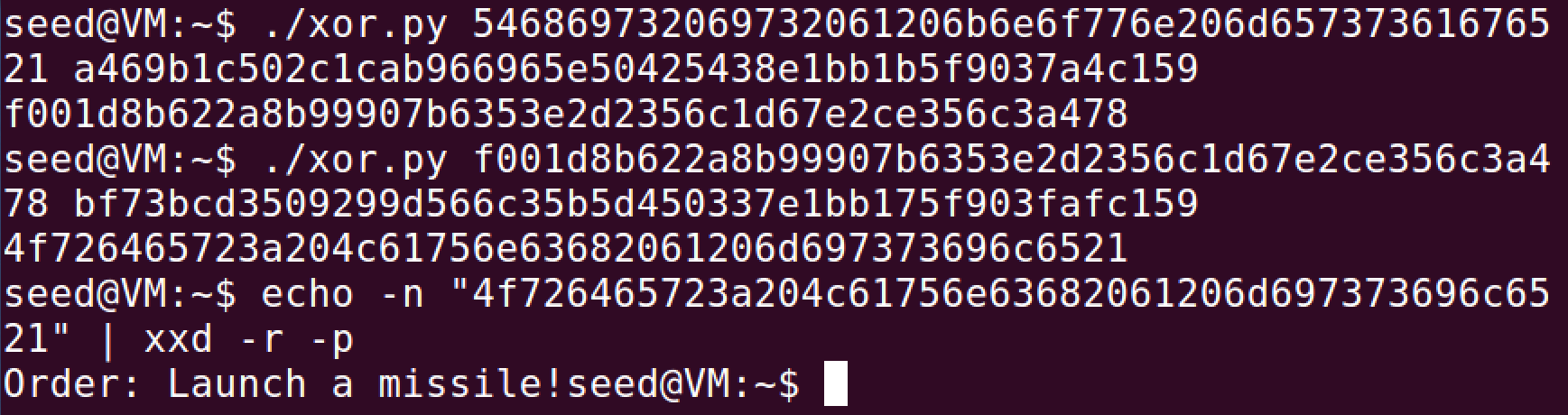


Comments:

When the ivs are different and unique the cipher.bin files are different but when the ivs are the same there is no difference in them. It is important that ivs be unique otherwise they can be easily decoded and makes for bad encryption.

6.2:



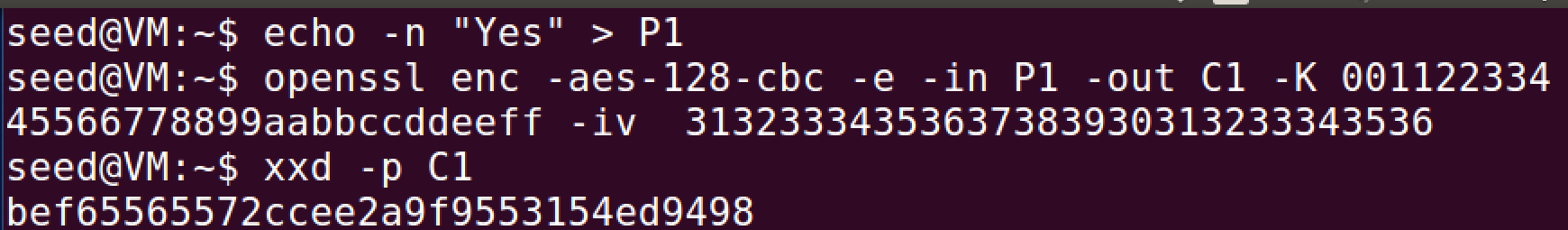


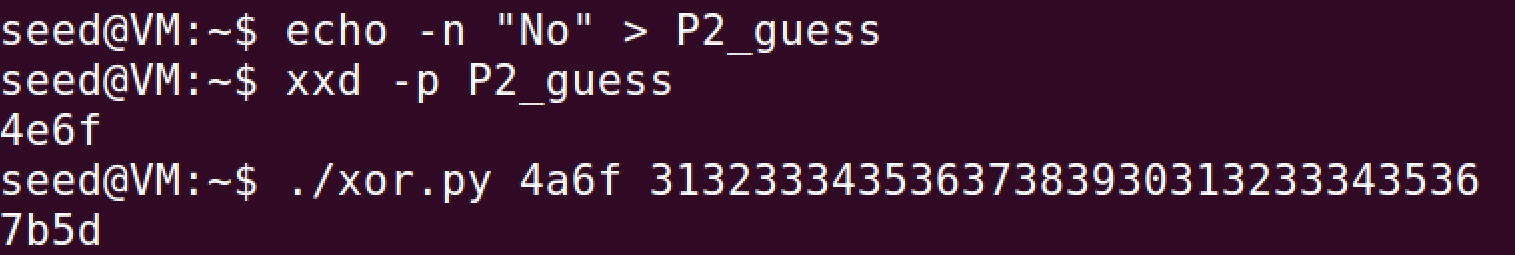
Comments:

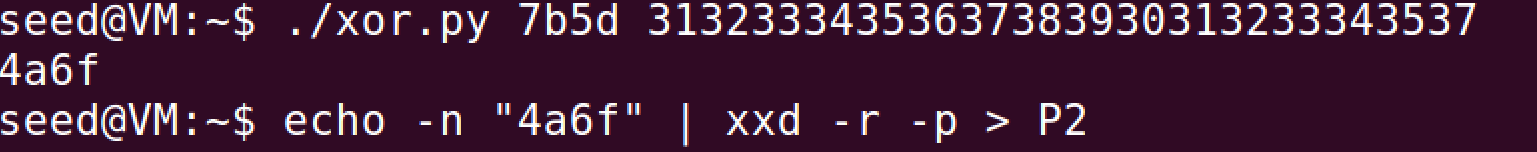
We start with the known message in P1 and encrypt it into C1. We set the Cipher text into C2. Then we use xxd -p to convert them to hex strings. Then we use xor on P1, C1, and the answer from those two we use xor along with C2. After that we convert the answer from a hex string to ascii string and we get the answer.

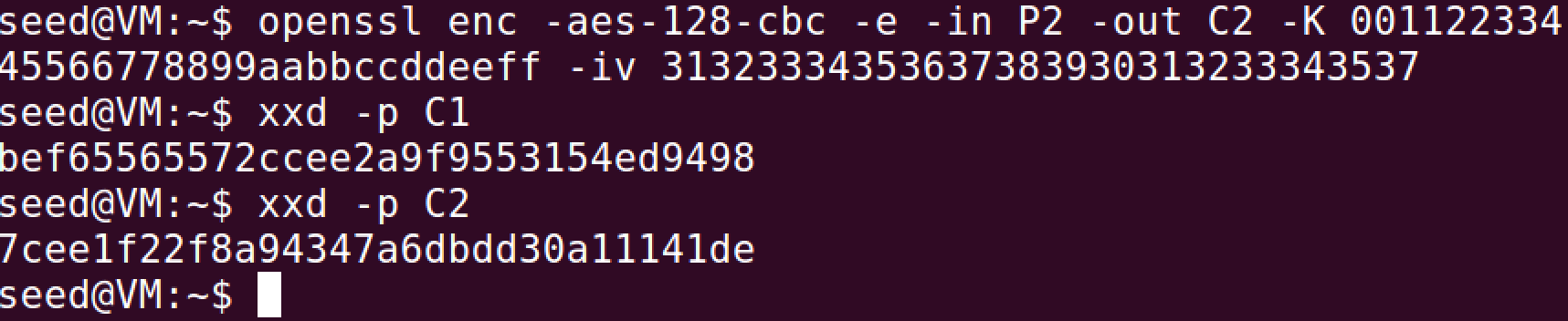
Since CFB requires padding for the last block and OFB does not require it, it won’t be able to decrypt the final message.

6.3:





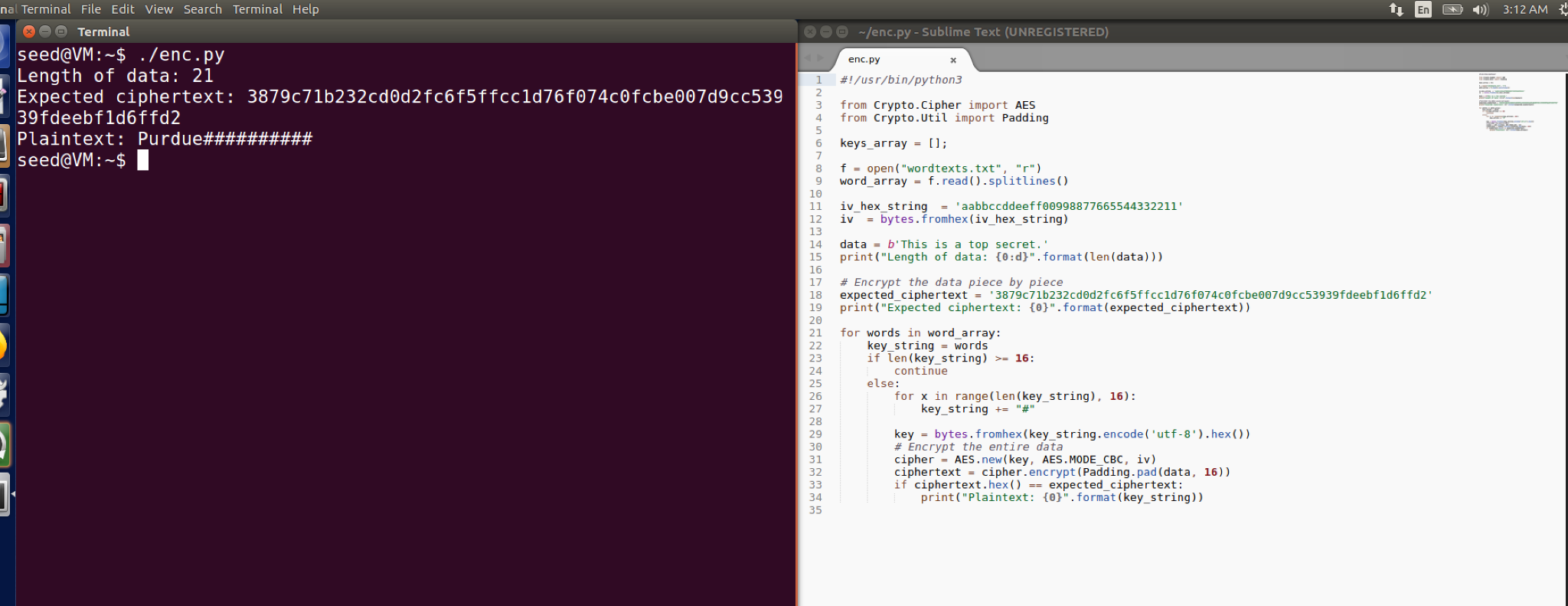




Comments:

For this task, I found out which answer he had and it turned out to be Yes. I then put in my guess No as a file for him to encrypt and then once I had that I checked and compared the cipher hex strings and saw that they were different and did not match. Since the only options are yes and no then if it wasn’t no, then it must have been yes.

Task 7:



Comments:

This code loops through the word file and makes checks if the word is 16 bytes and if it isn’t then it pads it with # symbols. It takes the ciphertext and finds the plaintext associated with it. In this case it is Purdue.