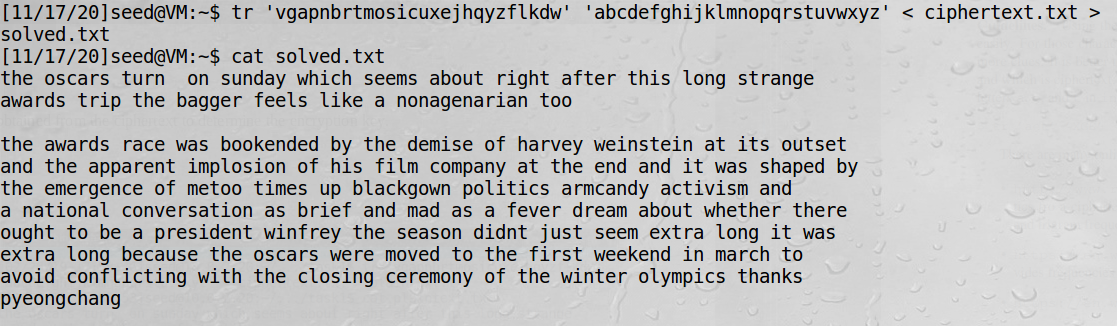
Lab 5 – Kevin Martin

**Task 1 – Frequency Analysis**

First, I downloaded the **ciphertext.txt** file from the seedlabs website. I saved it in the home directory as a simple text file. Then, I went to the suggested resource of <https://www.guballa.de/substitution-solver> I entered in the entire text. It gave the following suggestion as a way to decrpyt this text:



Finally, I exectued the **tr** command on the **ciphertext.txt** file, with output being **solved.txt**. The result:

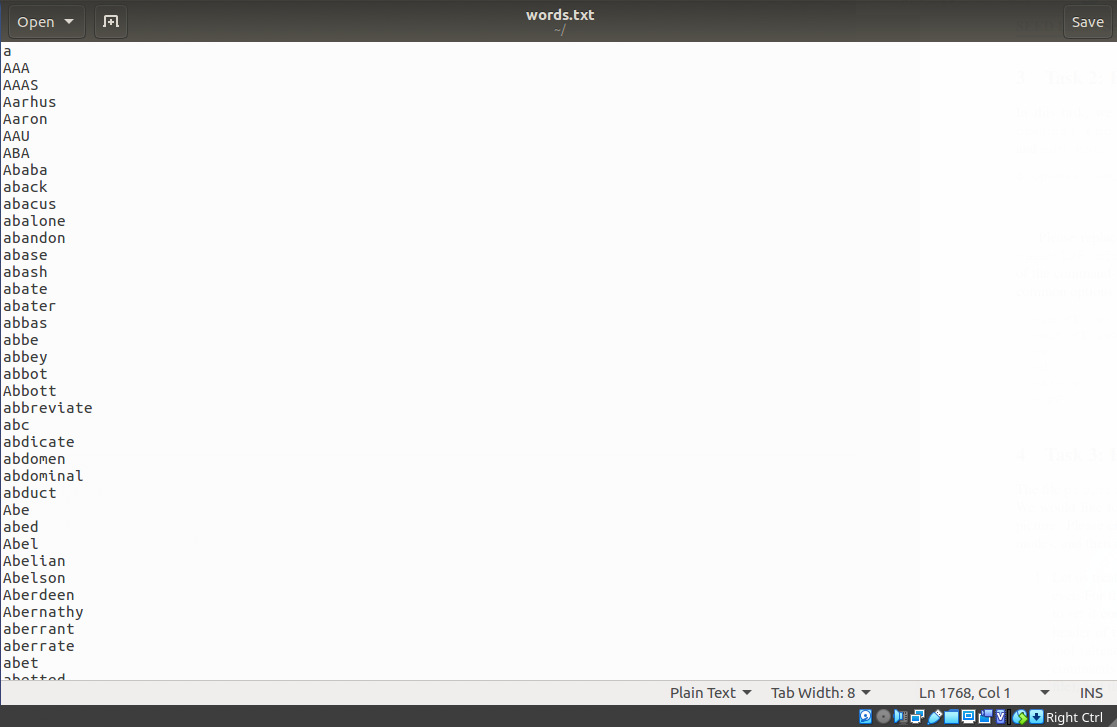


**Observation:**  Using a tool to detect how the original cipher text was coded, we were able to quickly decode the message. The tool utilized a word count method, comparing the original text to the most often used letters or combinations of letters in English.

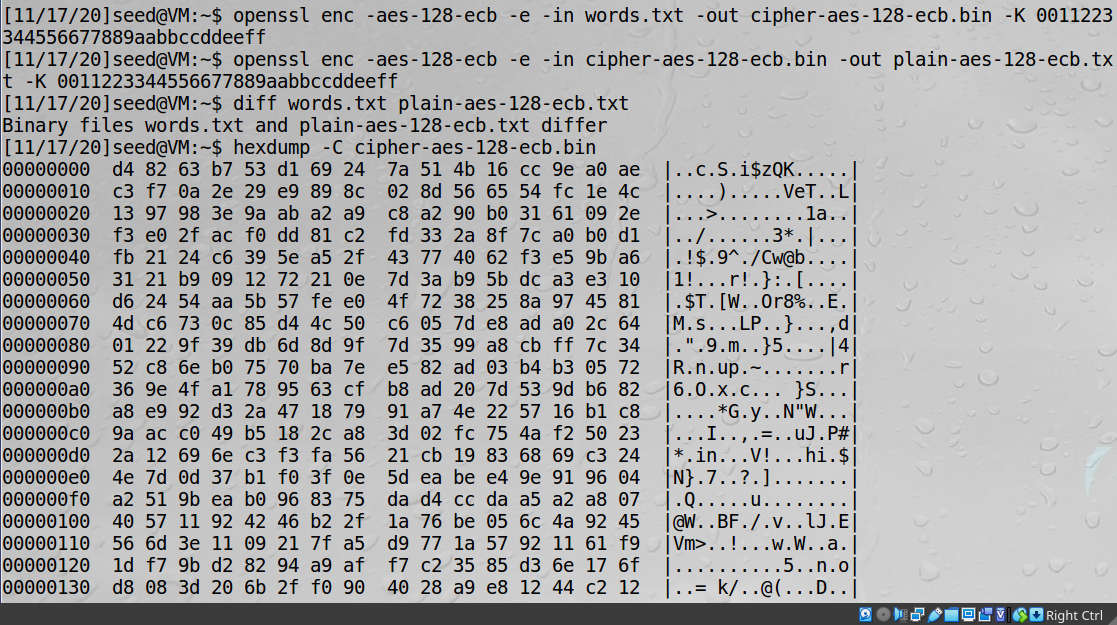
**Explanation:** Because this is a monoalphabetic substitution cipher, the decryption process was not difficult. Each letter is mapped to another letter, and by using the frequency method, it is a quick decryption process.

**Task 2 – Encryption Using Different Ciphers and Modes**

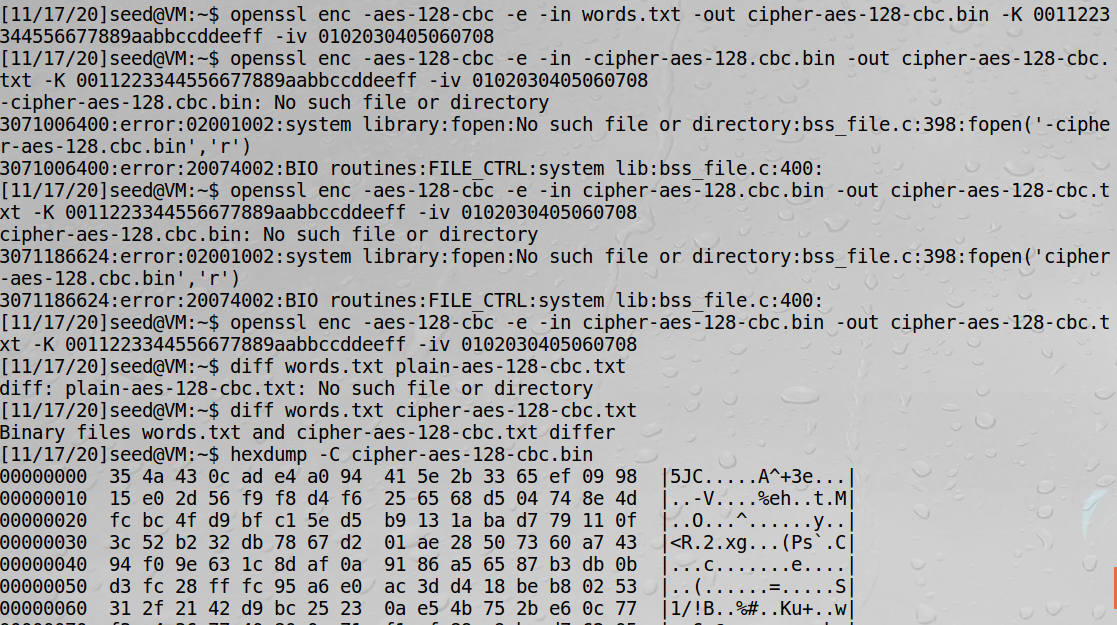
I was unable to locate the suggested **plain.txt** file, so instead I used the provided **words.txt**. I just used all the words that began with the letter “A” as my input file:



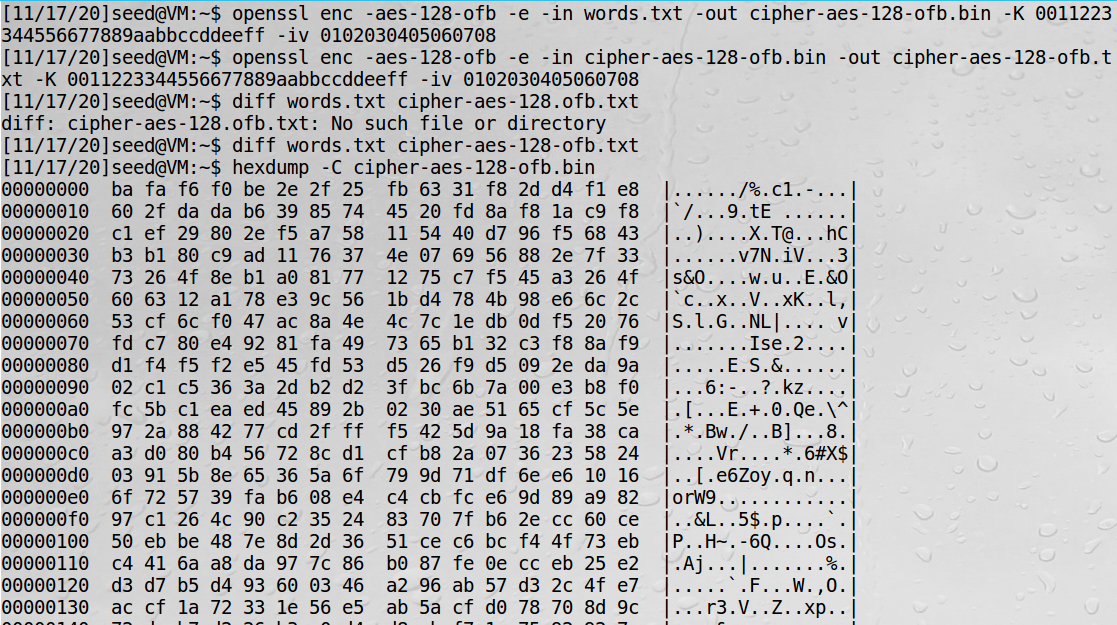
Next, I use the **openssl enc** command to encrypt the text in different ways. First, **128-bit AES**, with the Linux command **diff** used to show that the encryption works:



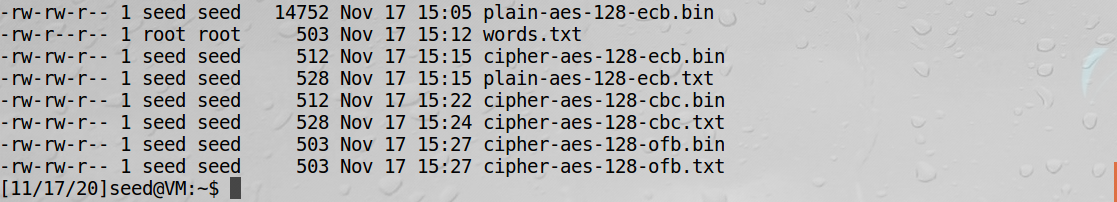
Then I apply the same steps for the **128-bit AES cbc** encryption (I had some issues with the correct file names):



For the third method, **128-bit AES OFB**:



Finally, looking at the size of each file, we see the OFB encryption is the smallest, matching the original plain text file:

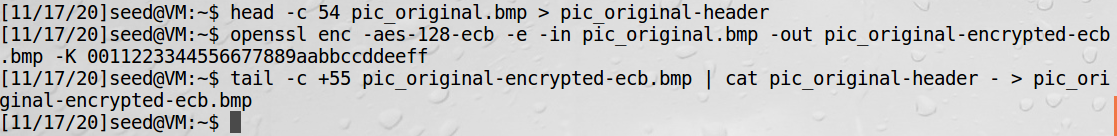


**Observation:** Using the **openssl enc** command, we are able to appply a variety of encryption methods. We are able to varify using the **diff** command that the files were indeed different, and we can use a **hexdump** to see how the encryption was applied.

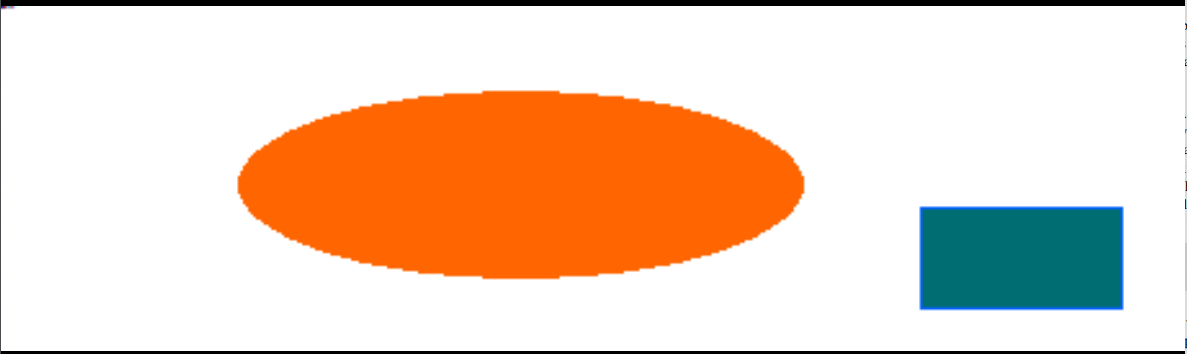
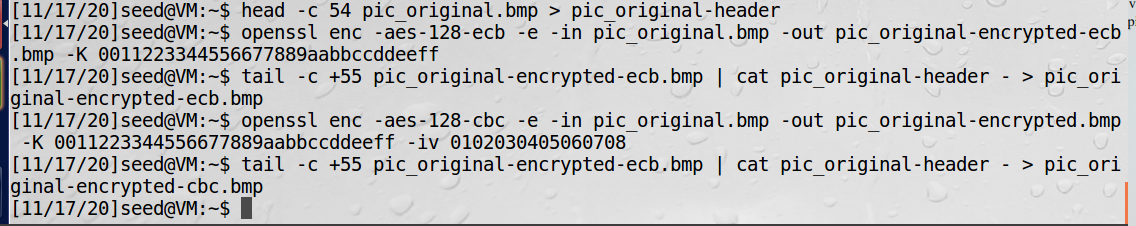
**Explanation:**  The different algorithms needed are all built in to **openssl** which allows for quick and easy encryption methdology. It is a flexible program that supports a variety of formats.

**Task 3 – Encryption Mode: ECB vs. CBC**

To encrypt the image, **pic\_original.bmp**, I save the header of the image, and tghen replace it with the remainder of the file, and finally recombine to create a new picture. First I try the AES-ECB method:

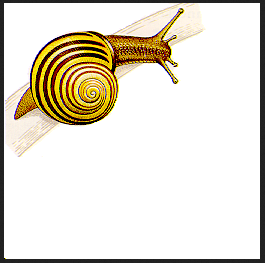
****

Next, we try the CBC encryption:

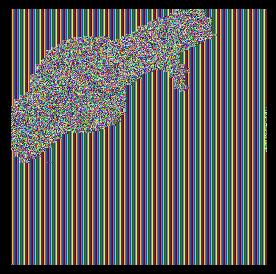
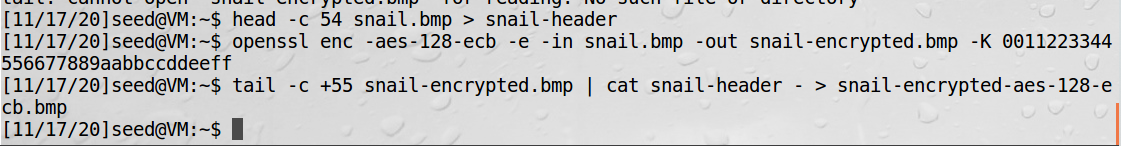


Note: the image viewer in my VM was giving different results each time I opened the image. It appeared to be an issue with how it was reading the file. This was one of the more interesting attempts as it appears to have slightly distorted the original colors.

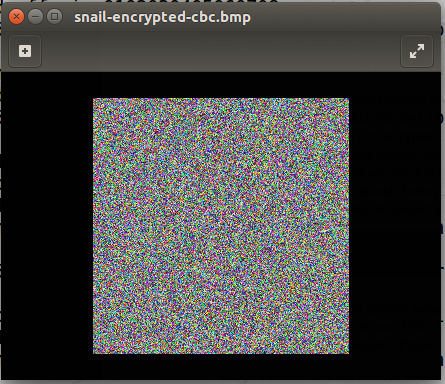
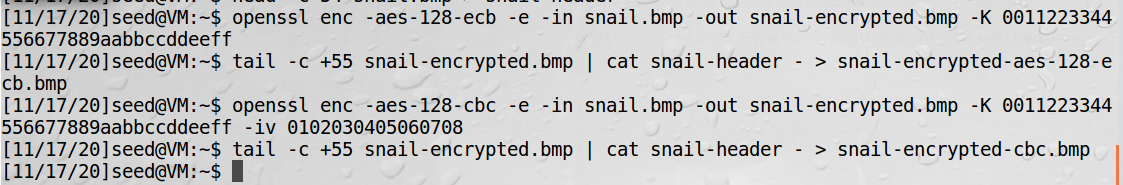
I run the two encryption tests again, with a different starting image:



ECB:



CBC:



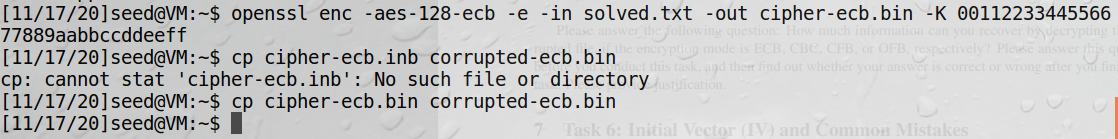
**Observation**: both encryption methods clearly worked better on the second image. My belief is there was an error in how I initially encrypted or recorded the head metadata. Regardless, we can visually see the relative effectiveness of each encryption method.

**Explanation:** The encryption methods provided by **openssl** can also be applied to bmp images to effectively manipulate them as well. CBC appears to provide better encryption that ECB, as the image is more distorted under this approach.

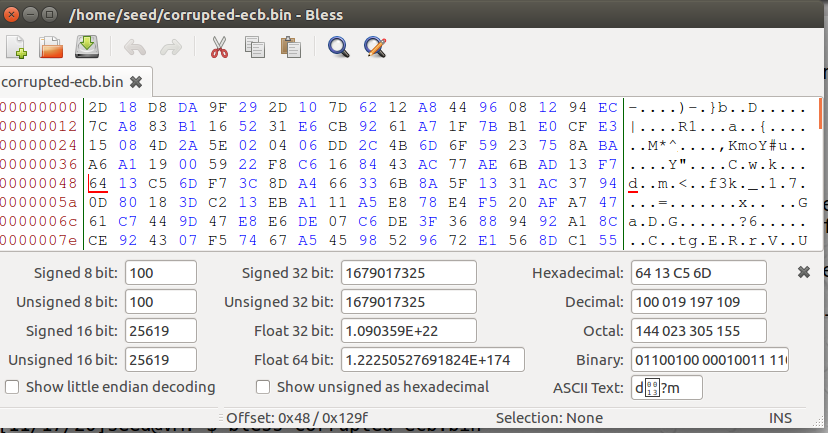
**Task 5 – Error Propagation – Corrupted Cipher Text**

To corrupt a single bit of a file, I will use the **solved.txt** file from earlier, which is 4.8kb (over the 1,000 byte recommendation). First, I encrypt the file using **128-bit AES ECB** encryption, following the same procedure as earlier. Then, I “corrupt” the 55th bit using **bless**. Finally, I decrypt the file using the correct key and IV.

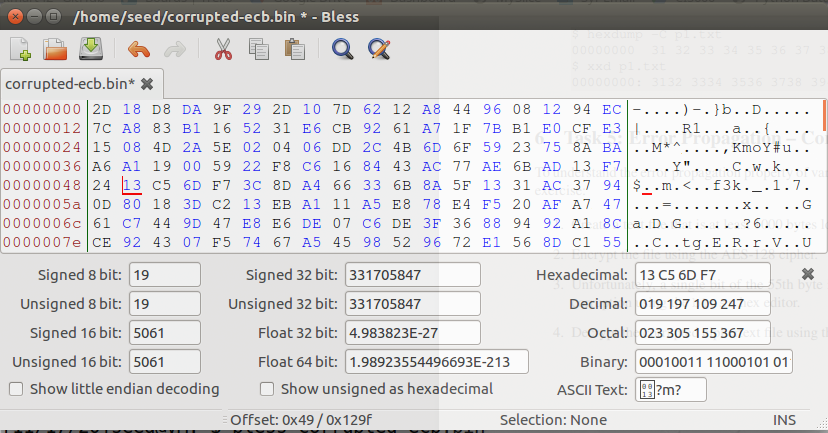
First, the encrypted file and copy:



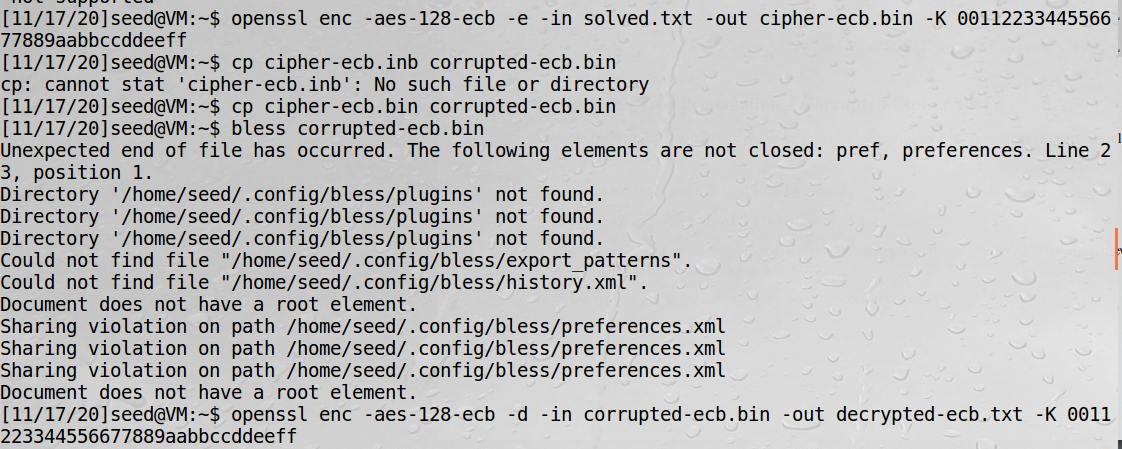
We can see that in the position of hex 35 (54 in decimal, or the 55th bit), the value is originally 0x64 (100):



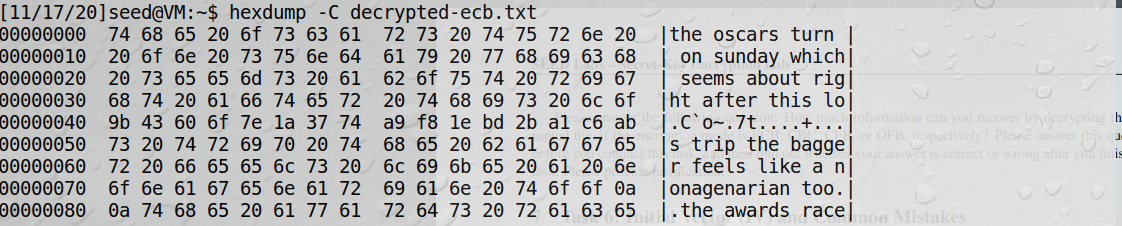
We replace it with the value of 24 in hex (34 in decimal):



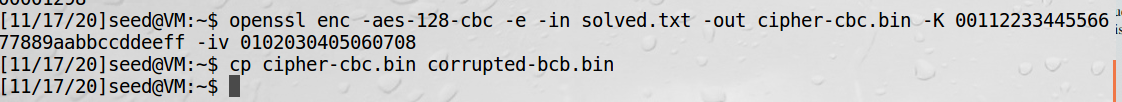
Now we decrypt:



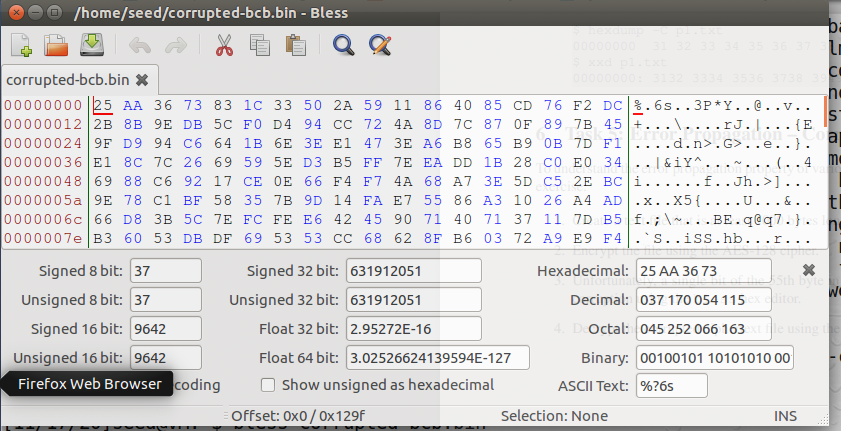
And the final hexdump. Note the corruption is still preserved, and indeed across the entire line of text:



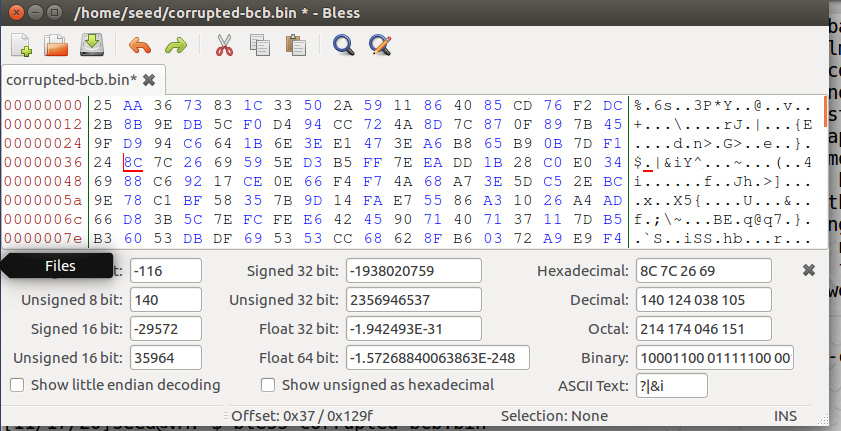
I recreate the steps, only this time using the **128-AES in CBC** to try and recover as much as possible:



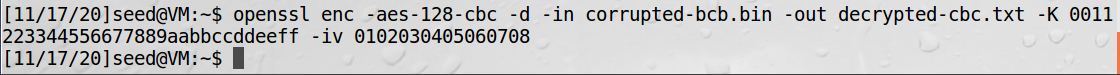
We see the 55th bit is E1:



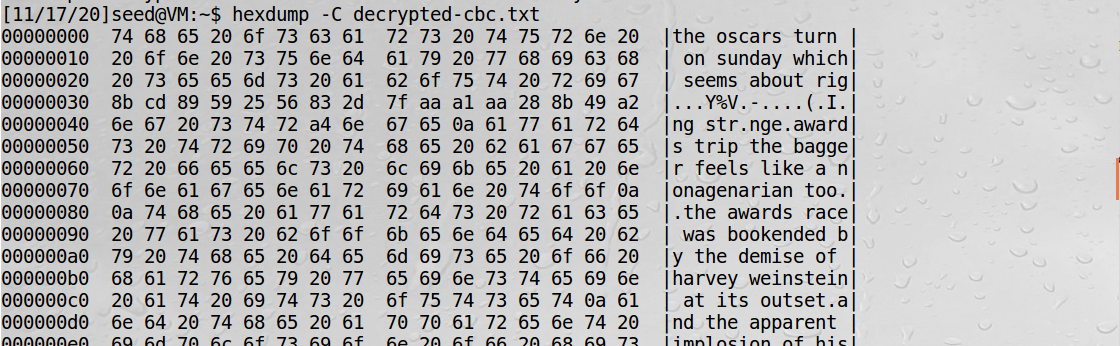
And we change to 24 again:



Decrypt the file:



Finally, the hexdump:



**Observation:** By simply changing one byte in a 4,000+ bite file, the entire line of text is corrupted. In both cases, some of the data was lost. In ECB mode, the entire block was lost, while in CBC mode, it actually extended even further to the line below it.

**Explanation:** Due to the different methods of encryption and decryption, more than just a single byte of data can be lost. Because the methods rely on the surrounding data, this can make it difficult to restore even the surrounding bits.