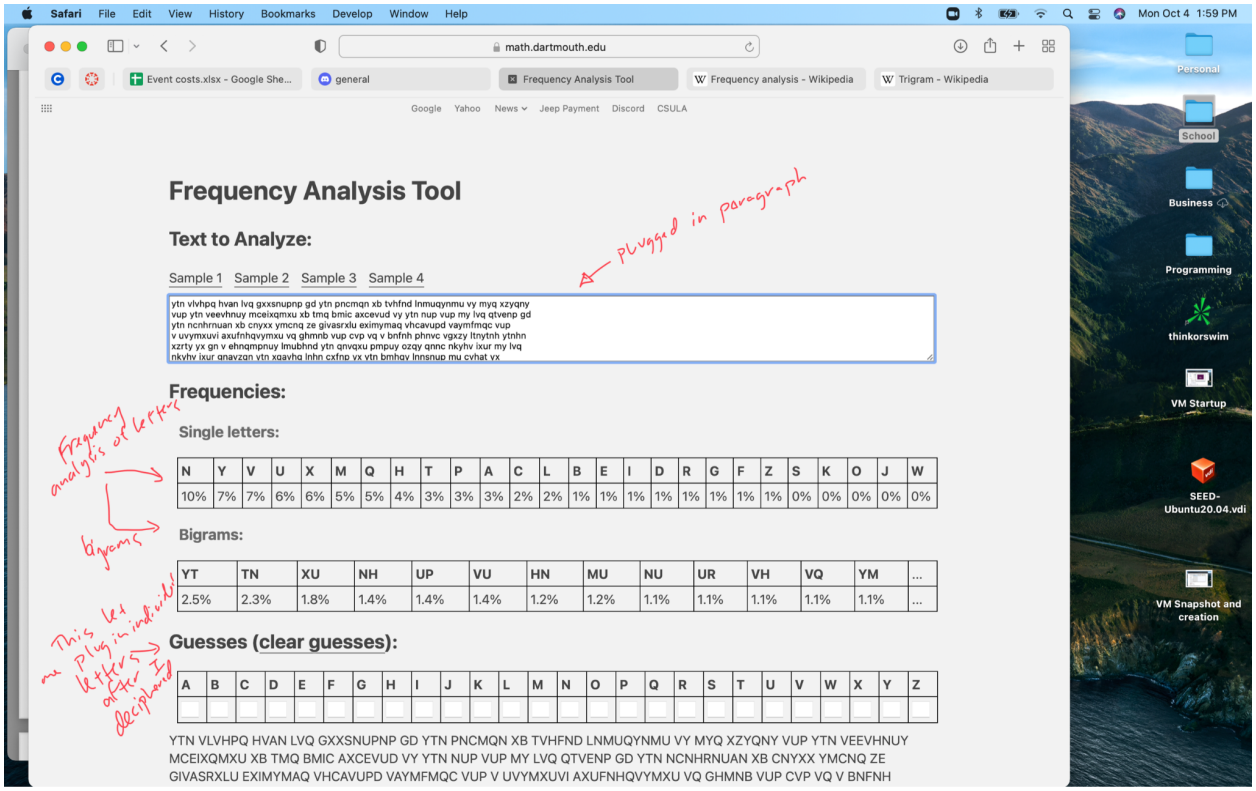
Hugo Izquierdo

Task 1: Frequency Analysis

For task one, the first thing I did was download the virtual box program and the Ubuntu seed. I set up the virtual machine with the instructions provided and made sure it was working. Next, I downloaded the “files” from canvas, which included the ciphertext.txt. I then used a frequency analysis tool website in order to conduct the analysis. I plugged a long paragraph of the text into the analysis field, and it gave me the breakdown of word and letter frequency, as shown below:



N	Y	V	U	X	M	Q	H	T	P	A	C	L	B	E	I	D	R	G	F	Z	S	K	O	J	W
10%	7%	7%	6%	6%	5%	5%	4%	3%	3%	3%	2%	2%	1%	1%	1%	1%	1%	1%	1%	1%	0%	0%	0%	0%	0%

YT	TN	XU	NH	UP	VU	HN	MU	NU	UR	VH	VQ	YM	...
2.5%	2.3%	1.8%	1.4%	1.4%	1.4%	1.2%	1.2%	1.1%	1.1%	1.1%	1.1%	1.1%	...

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

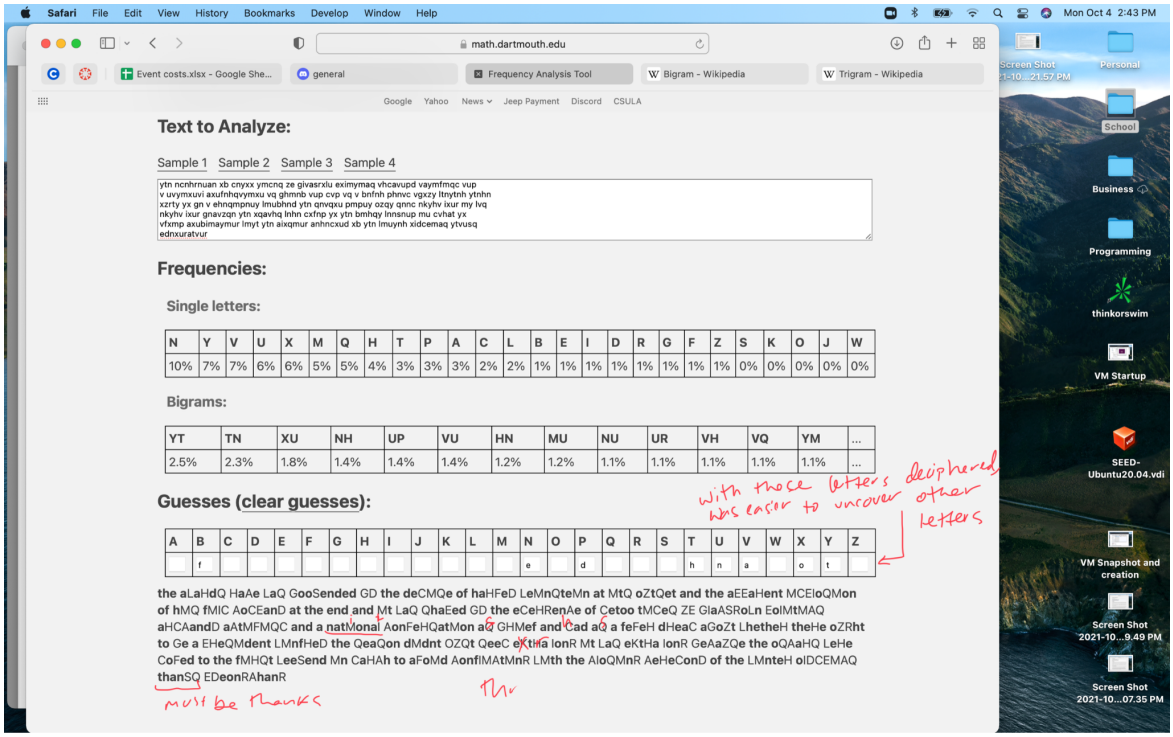
Seeing how the most common letter in the English language is ‘e’ , the most common bigram is ‘th’ and the most common trigram is ‘the’, it stood to reason that the cipher letter ‘N’ stood for e (N was most common letter in ciphertext), and YTN would thus be “The” based on the first word of this passage being ‘YTN’. It also stood to reason that the only letter in English which can be used as a word on it’s own is the letter ‘a’, therefore cipher-V must stand for ‘a’.

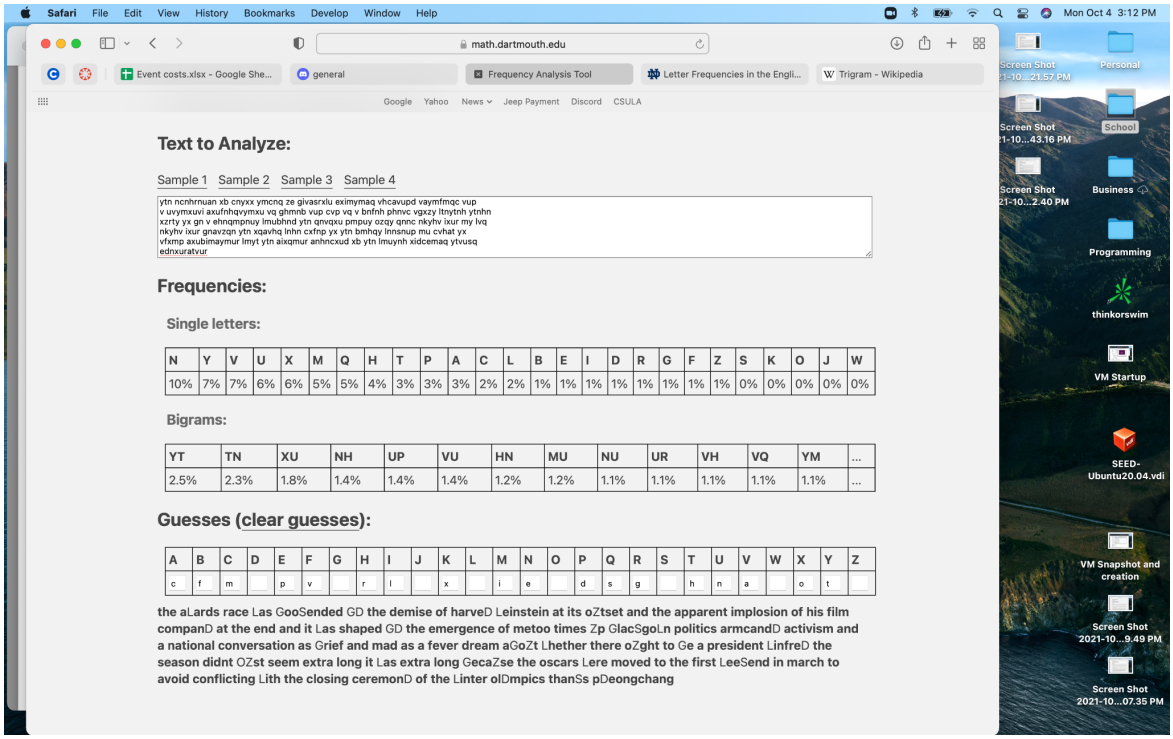
After these letters were deciphered, I tried to find a relationship between the distance of each cipher letter to its plaintext letter, but couldn’t find one, so I kept chipping away at deciphering through frequency analysis.

Next, I started looking for bigrams which contained the same cipher letters. I found EE, and XX were most common. Since ‘X’ is the 5th most common cipher letter, and it appears together in form xx, and it appears as a bigram word with other letters, we can deduce that it must be a vowel, since all words must include a vowel. None of the other vowels except for ‘ee’ are frequently paired together, and since we have already found ‘N’ to represent e, we can assume that cipher-X must represent ‘o’.

Given the frequency of the trigram ‘VUP’ as a standalone word in our cipher text, and the fact that V stands for the letter ‘a’, it is safe to assume ‘VUP’ stands for ‘and’, another very

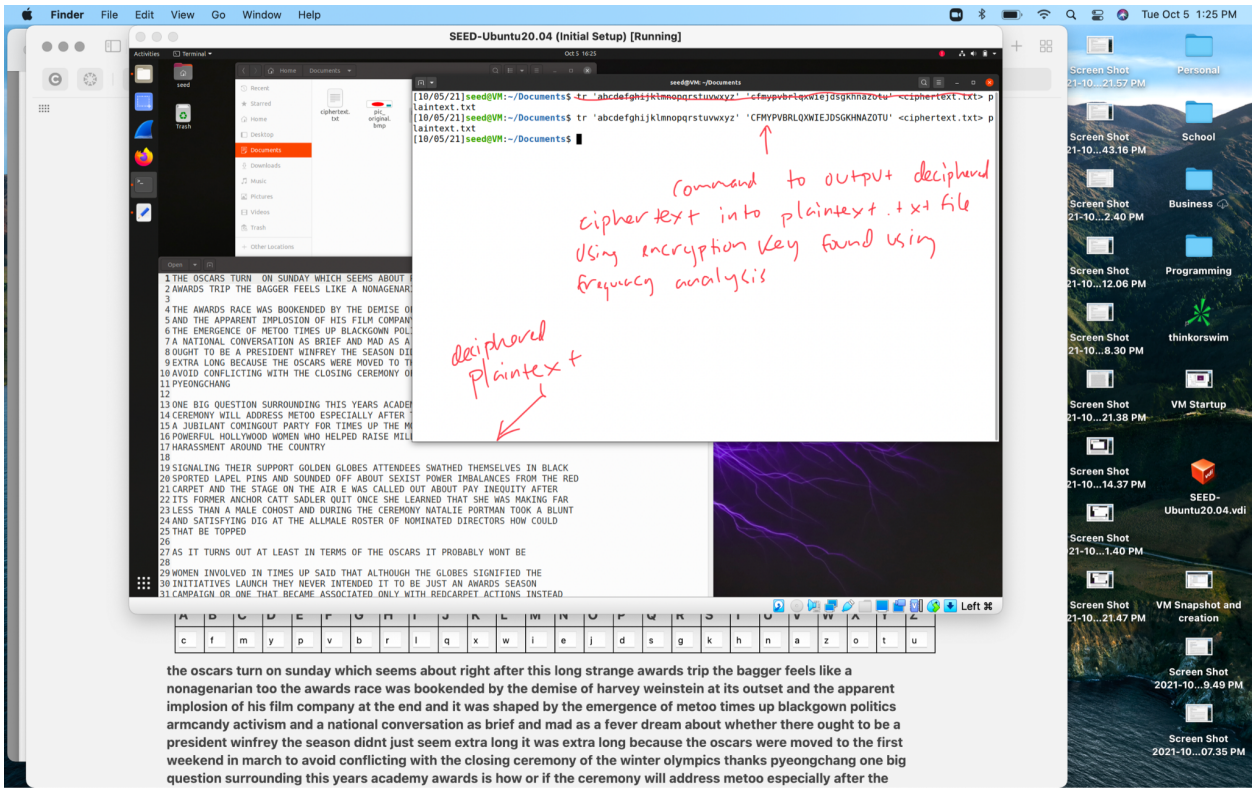
common trigram in English. That gives us U standing for ‘n’ and P standing for ‘d’. Deciphering the plaintext letter n was very helpful, and after that I was able to sort out other words that made sense, until the solutions to last letters was trivial.





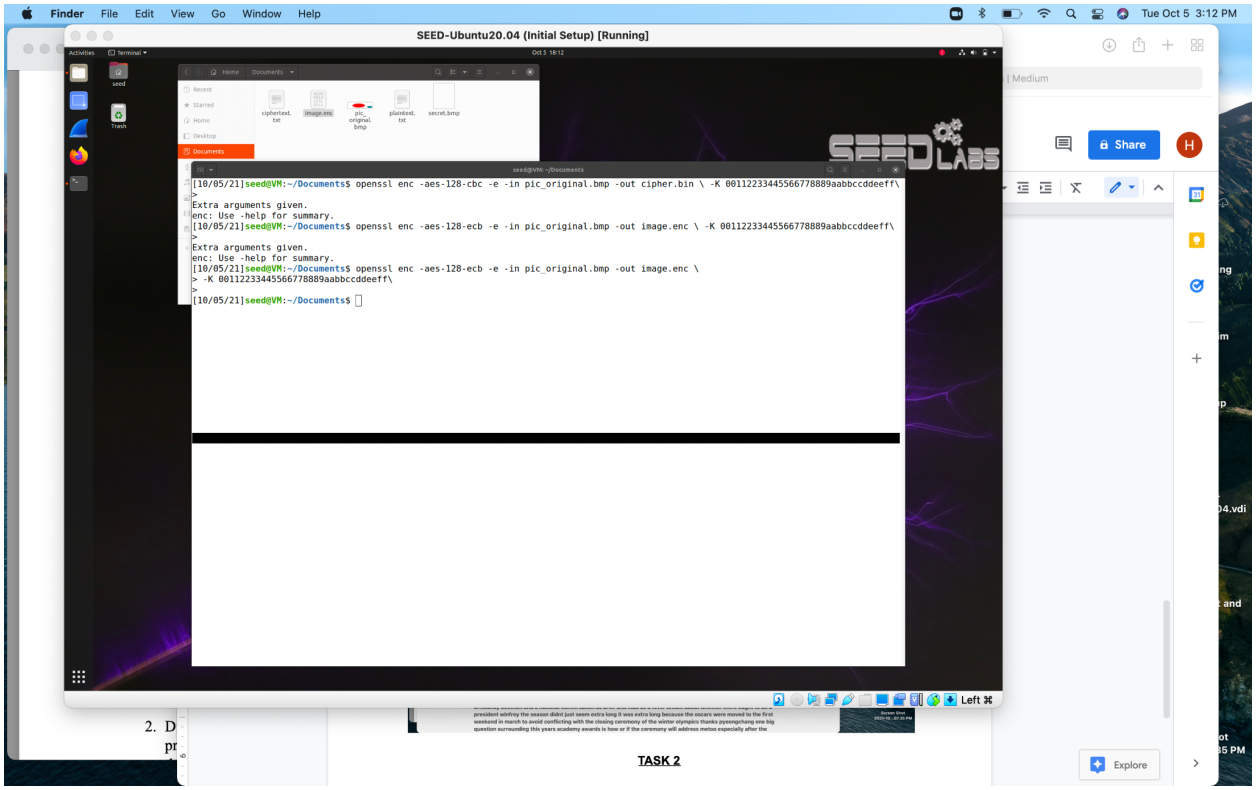
After deciphering all the letters, and plugging in the entire cipher-text back into this program to verify that the code worked for the rest of the text, I had the encryption key: ‘CFMYPVBRLQXWIEJDSGKHNAZOTU’, where each letter stands for it’s alphabetic

counterpart. Next, I created a plaintext.txt file in the virtual machine, opened the terminal, and typed in the command ‘cd Documents’ to go to that folder (which is where the ciphertext.txt and plaintext.txt were located). I then typed in the tr command given in the handout, using my encryption key, and that outputted the deciphered ciphertext into the plaintext file. Message deciphered.

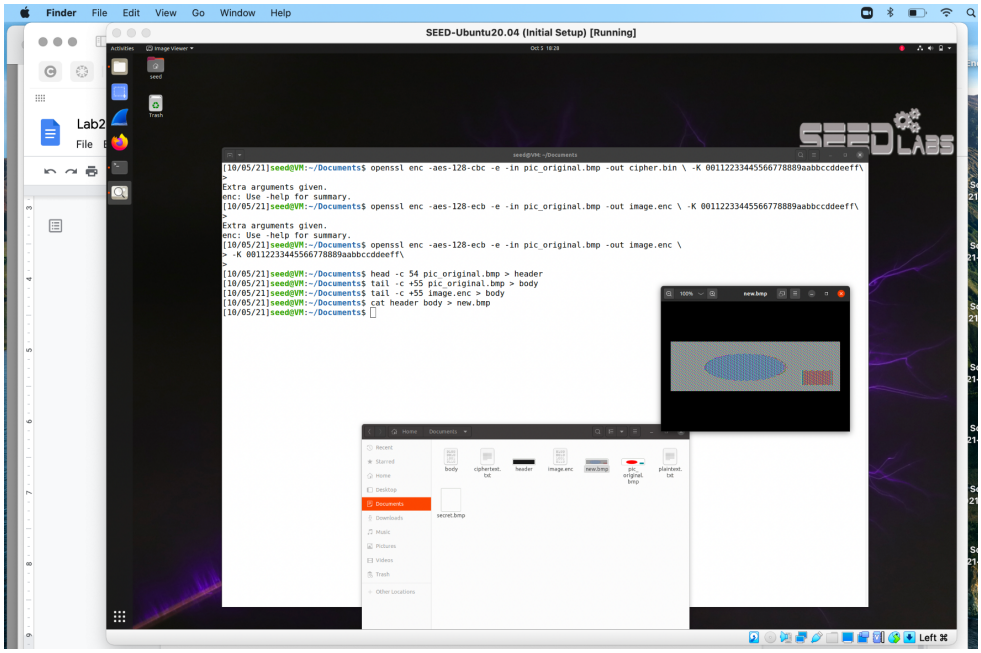


TASK 2: Encryption

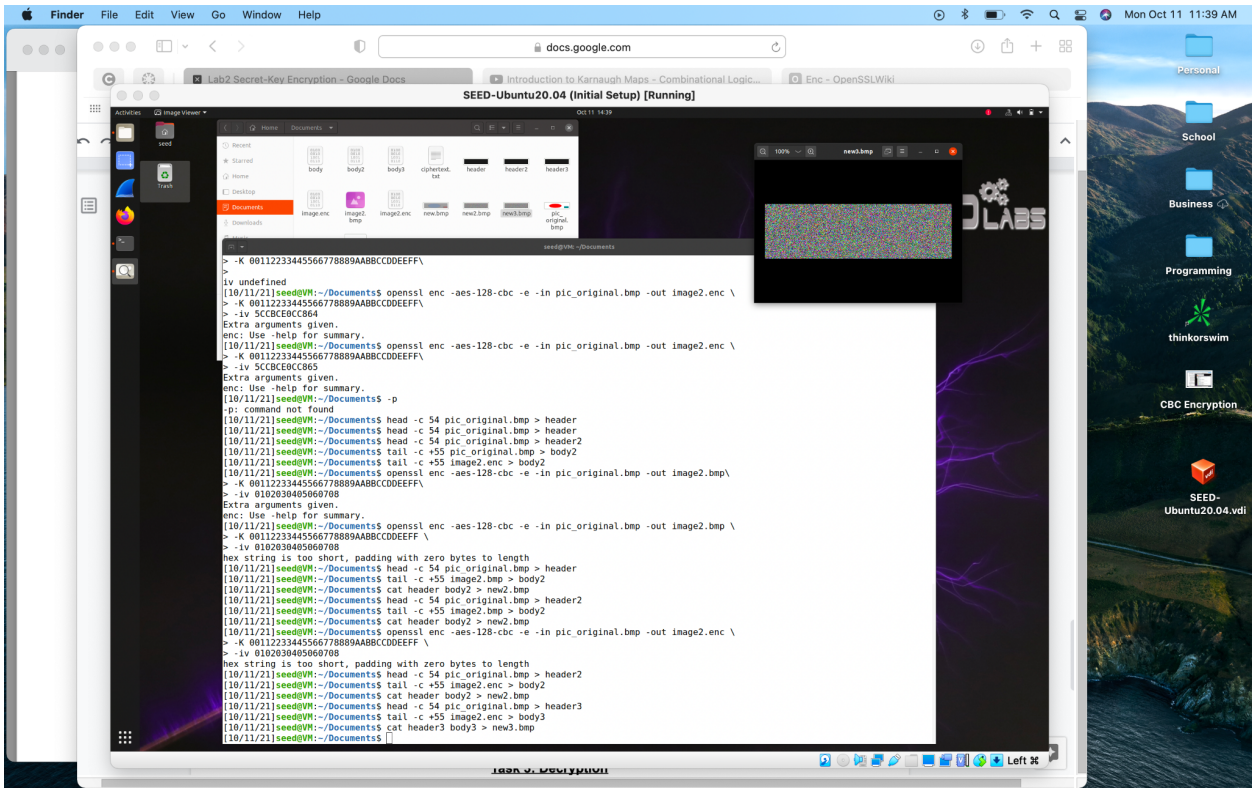
For task 2, the first thing I did was open the terminal and type in the openssl command to understand what it was and dig through the different available commands and available help. I read the chapters on ECB and CBC modes, then went into terminal to try out commands. I used the -aes-128-ecb command first, with the key being the one provided on the handout. I typed in everything the same as on handout, except changed the input file to be the image file, and made the output be simply image.enc. This command created a file which cannot be seen (image.enc).



Next, I used the commands to get the header of the original image file, and the body of the image.enc file, and put them together so the encoded image can still be viewed as a legitimate image. When I opened the image, it still had the same elements, but looked greyed out (staticky), and it was interesting to find that the colors of the shapes seemed to almost be inverted. The circle was originally red, and on new picture was blue, and the teal square became red.



Next, I encrypted using CBC. I used the same steps as the ecb, except that I used the label -cbc in first command, and I had to enter a -K value and and -iv value as well. I tried for about an hour to get this working, only to realize that I was missing a space between the end of the -K value and the ‘ \ ‘ symbol, a source of much frustration, but was a good lesson in paying attention to the details of these commands. I was finally able to get it to work and outputted to a image2.enc file. I used the same head and tail concatenation as in the ecb, but made new files so I wouldn’t be confused (header2, body2). Once I ran the cat command and saw that the image could not be seen, I tried it once again with new files and got the same result, and so I can say that no useful information can be gathered from the encrypted CBC picture.



Task 3: Decryption

For Task 3, the first thing I did was map out with pen and paper the exact sequence I used for the encryption steps above, so I can work the reverse for this task. Since I am only given the encryption key for task 3, I figured it would be best to use ebc standard. I used the command with -d for decryption, and outputted to a new file secret_dec.bmp . I used the -K 123456 as the encryption code. During this, I learned that when you add a ‘ \ ‘ symbol to the end of these lines, it starts a prompt for another code or line (This part more for my notes than for assignment). I then clicked on the new outputted image file, but it gave an error message that there was bogus header data. When I googled why this might be, it told me that since the entire file had been encrypted, this also included the header information. So I used the command to get the header information from the original secret.bmp file and assigned it to ‘ headerdec ‘, and used the command to get tail information from the decryption output file secret_dec.bmp and assigned it

to ‘ bodydec’ . Next I used the cat command to concatenate the header information to the body information, with a new output file named newsecret.bmp. This generated the new file with the viewable image! Go Golden Eagles! (P.S. I did not want to delete the previous code that gave me errors, that way I can see what I did wrong, so please only look at the bottom portion of commands without error messages).