PART A

Task 1

The first of step in the procedure of encryption via DES and AES, both of which require the creation of random Keys with AES having the additional requirement of an initialization vector.

```
tabasco-slim@sneaky-breeki:~$ openssl rand -hex 8
73745ef64e2c9800
tabasco-slim@sneaky-breeki:~$ openssl rand -hex 8
e3d383b531ad9edd
tabasco-slim@sneaky-breeki:~$ openssl rand -hex 16
be942347a422a302a5850e74f059ea2d
tabasco-slim@sneaky-breeki:~$ openssl rand -hex 16
d6df31e4b9002da841d62e71a6f9b17c
tabasco-slim@sneaky-breeki:~$
```

Figure 1: The generation of the keys and IVs via openssl

Figure 1 shows the user name and the domain name of the machine which will be used for the duration of this lab. As this lab was completed over several days the specific keys and initialization vectors may change depending on which task is being executed.

Task 2

Part (a)

Figure 2: CS4331.bmp

Viewing the initial .bmp file with a basic image viewer, the file appears as follows:

Part (b)

Here are the commands executed to encrypt the CS4331.bmp file. he commands excited in the terminal where:

```
tabasco-slim@sneaky-breeki:~$ openssl enc -des-ecb -in CS4331.bmp -out CS4331_des_ecd.bmp -K 73745ef64e2
c9800
```

Figure 3: des-ecb

oepnssl enc -des-ecb -in CS4331.bmp -out CS433_des_ecd.bmp -K 73745ef64e2c9800

Note: there was a typo in the naming of the file, this did not impact performance but is worth mentioning here.

```
tabasco-slim@sneaky-breeki:~$ openssl enc -des-cbc -in CS4331.bmp -out CS4331_des_cbc.bmp -K 73745ef64
e2c9800 -iv e3d383b531ad9edd
```

Figure 4: des-cbc

oepnssl enc -des-cbc -in CS4331.bmp -out CS433_des_cdc.bmp -K 73745ef64e2c9800 -iv

```
tabasco-slim@sneaky-breeki:~$ openssl enc -aes-128-ecb -in CS4331.bmp -out CS4331_aes_ecd.bmp -K be942
347a422a302a5850e74f059ea2d
```

Figure 5: aes-128-ecb

e3d383b531ad9edd

oepnssl enc -aes-128-ecb -in CS4331.bmp -out CS433_des_ecd.bmp -K

```
tabasco-slim@sneaky-breeki:~$ openssl enc -aes-128-cbc -in CS4331.bmp -out CS4331_aes_cbc.bmp -K be942
347a422a302a5850e74f059ea2d -iv d6df31e4b9002da841d62e71a6f9b17c
```

Figure 6: aes-128-cbc

be942347a422a302a5850e74f059ea2d

oepnssl enc -aes-128-ecb -in CS4331.bmp -out CS433_des_ecd.bmp -K be942347a422a302a5850e74f059ea2d -iv d6df31e4b90002da841d62e62e71a6f9b17c

Note, the encpted files followed a somewhat different naming scheme as suggested in the Lab 2 documentation

Part (c)

After the execution of the commands in Task 2 Part (b) the .bmp files are not viewable via the iamge viewer used in part (a). Figure 7 shows the error message when a basic image viewer attempts to open one of the newly created .bmp files.

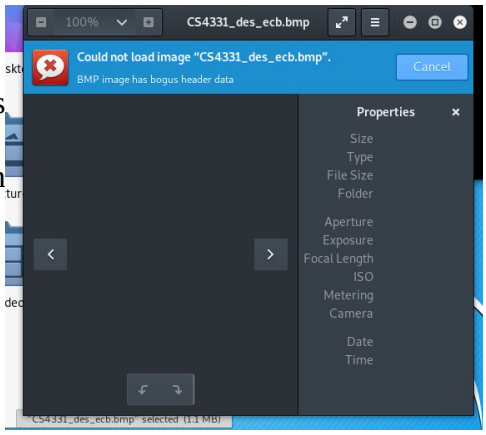

Figure 7: Image viewer error

Figure 8 shows the four new .bmp files in the directory, later views of the files will actaully show thumbnail images as CS4331.bmp shows in figure 8.

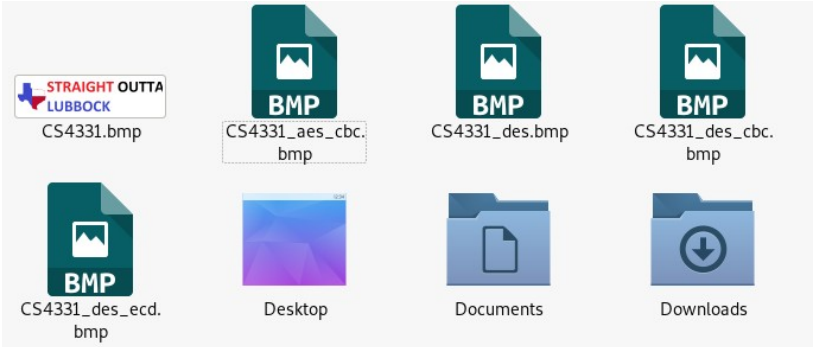

Figure 8: Unedited .bmp files

Though having been recommended in the documentation for Lab 2, the Bless hex editor was deemed insufficient for use as on the rolling version of Kali Linux results in the program consistently crashing the Bless. The solution to this problem was using Vim as Vim is an incredibly stable text editor which is more than capable of acting as a hex editor. To covert transform a file to hex representation in vim the command is: :%!xxd

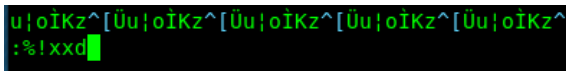

Figure 9: The xxd command in Vim

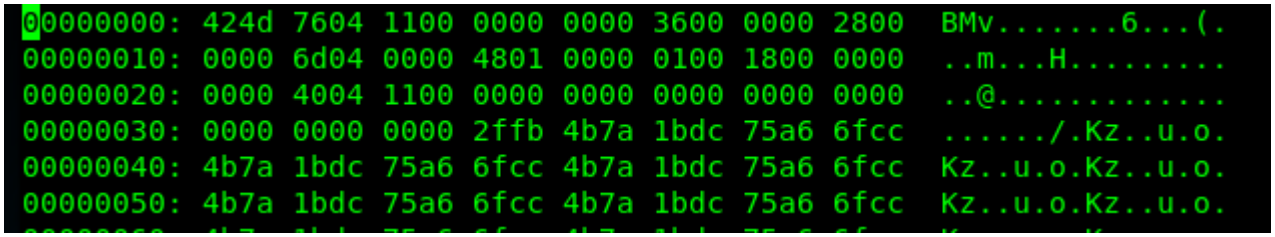

Figure 10: Vim being used as a hex editor

Figure 10 shows Vim being used as a hex editor. Specifically, the above image is from the post-edited CS_4331_des_ecd.bmp file. The first 54 bytes of the file which where copied over from the here are shown now as the the bytes contained in the first 3 lines, as well as the bytes in the first three columns of the fourth line. Figure 11 shows the bytes as coppied over from the CS4311.bmp file.

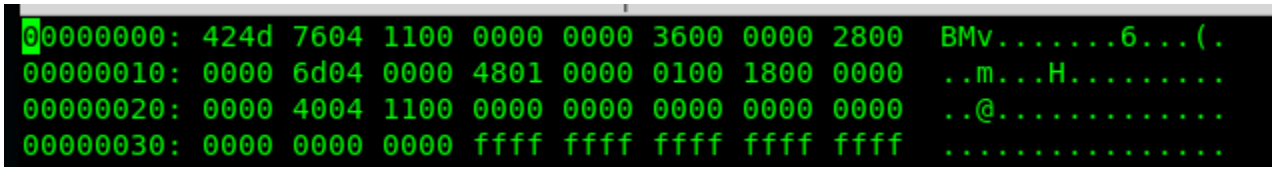

Figure 11: CS4311.bmp Viewed in Vim

Figure 12 Shows the post-edit of the first 54 of the .bmp file for the des-ecb encrypted file.

As per instruction, this process is repeated on the aes-128-ecb encrypted .bmp file, the post edited first 54 bytes of said file as viewed in Vim is shown in Figure

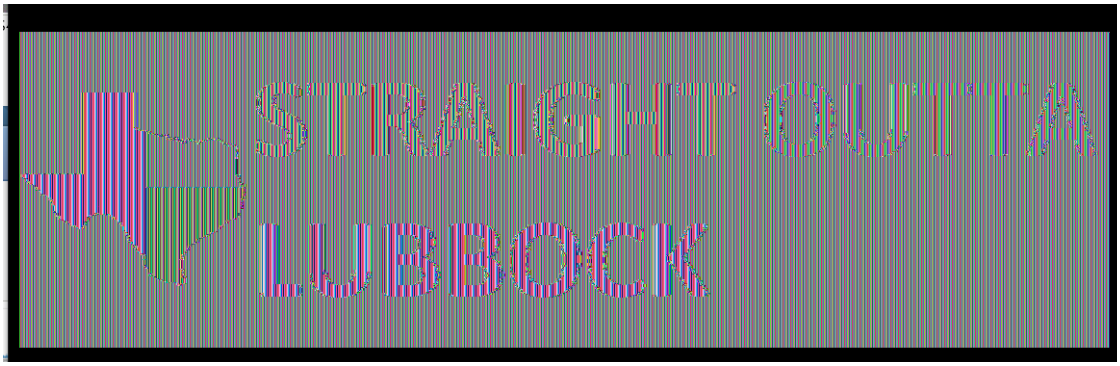

Figure 12: Post-edit of .bmp file

```
00000000: 424d 7604 1100 0000 0000 3600 0000 2800  BMv.......6...(.
00000010: 0000 6d04 0000 4801 0000 0100 1800 0000  ..m...H.........
00000020: 0000 4004 1100 0000 0000 0000 0000 0000  ..@.............
00000030: 0000 0000 0000 be20 18b4 bf4e 0bcc 549d  ....... ...N..T.
00000040: 3647 326d d27d 8f8c b6c7 1bb9 e5b2 08b2  6G2m.}..........
00000050: 3647 326d d27d 8f8c b6c7 1bb9 e5b2 08b2  6G2m.}..........
```

Figure 13: aes-ecb post-edit

13. Figure 14 shows the the .bmp file as it now appears when viewed through basic image viewing software.

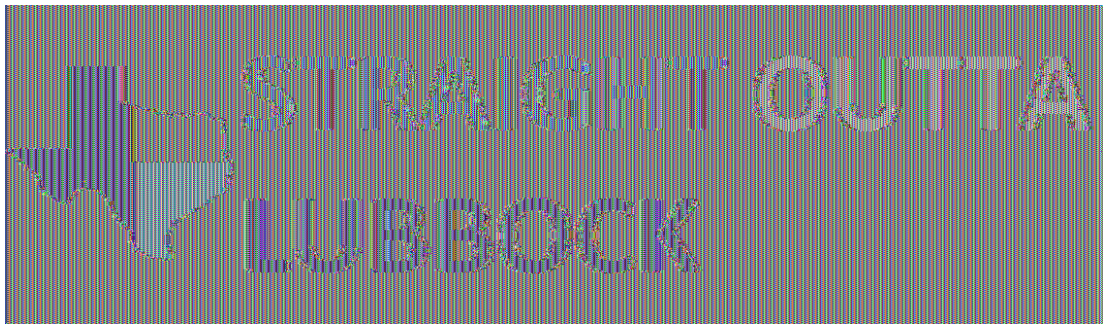

Figure 14: aes-ecb viewed post-edit

Subsection I

Why is it that you are able to view the images now when you where unable to do so previously in (b)?

The reason for why the images are able to be viewed now is likely due to how the .bmp files are formatted and how this particular image viewer operates. As part of the objective of encryption is not only ensuring that the content of any given information be transmitted is obfuscated from an attack but also that said information's exact type is not made easily known to an attacker either. The initial encryption of the bytes resulted in data which is not identifiable as an image to a basic image viewer software.

As seen in figure 11, the only contents of the .bmp file when viewed through a hex editor that contain any variation of bytes that are not "ff" are the first 54 bytes of the file. What can be deduced from this is that the first 54 bytes are the real content of the the .bmp file, having these bytes results in effectively having the message.

Subsection II

Are you impressed by the encryption? Explain why (why not).

No, the encryption is not impressive as the ciphered .bmp files could be brute forced to confirm the message being transmitted in the hypothetical scenario of described in this lab. If an attacker was attempting to discover who you were sending the .bmp file to and said attacker already had

the CS4331.bmp file then the attacker would be able to confirm the content of this message after it had been transmitted, thus compromising the confidentiality of the system.

Section (d)

When repeating the process outlined in sections (b) and (c) on the .bmp files encrypted with cbc mode the results are are considerably different.

Figures 15 and 16 show the first 8 lines of the .bmp CBC mode encrypted files with firgure 15 specifically showing the file created via a DES encryption in CBC mode where as figure 16 shows the first eight lines of the file created via the AES encryption in CBC mode.

Initially there is no obvious difference when using CBC mode as apposed to using ECB modes. The difference between the two modes become apparent when the same edit of the first 54 bytes of the .bmp files is made.

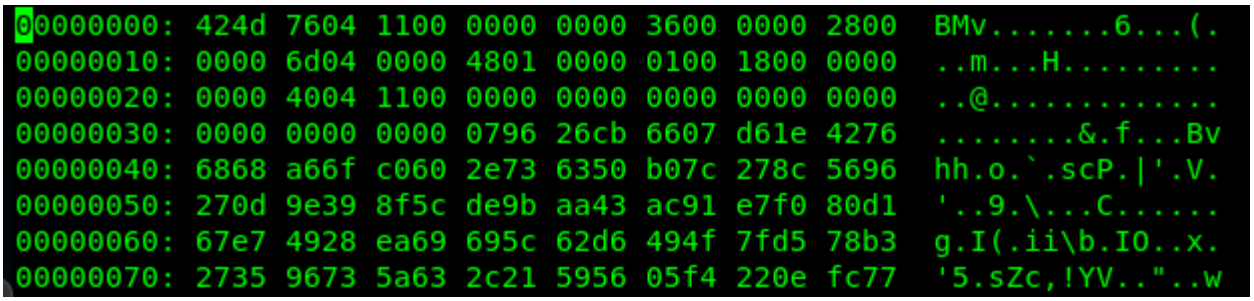

Figure 15: vim CS4331_des_cbc.bmp, post edit

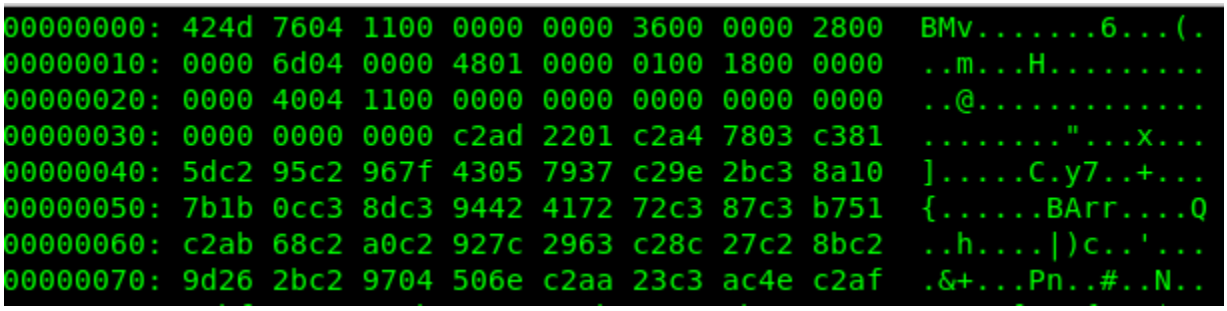

Figure 16: vim CS4331_aes_cbc.bmp, post edit

Explain the difference in observations between the case when you used ecb optiond and when you used the cbc options.

Figures 17 and 18 show an image that is almost indistinguishable to general observation with the only clearly recongnisable difference between the imageges being made apparent when the .bmp files are compated side by side. Consider that the 55th adnd 56th bytes of the DES-CBC file as shown in Figure 15 are **07** and **96** where as the 55th and 56th bytes of the DES-CBC file as shown in Figure 16 are **c2** and **ad**.

Figure 17: CS4331_aes_cbc.bmp in image viewer

Figure 18: CS4331_des_cbc.bmp in image viewer

Another interesting observation can be found when comparing Figure 10 to Figure 15 as well as when comparing Figure 13 to figure 16. To clarify, Figure 10 shows the hex view of a .bmp file created via DES in ECB mode and Figure 13 shows the hex view of a .bmp file created via the encryption of AES in ECB mode. What is noteworthy is that the fifth and six line of these files in the hex view are repeating; the fifth line is repeated on the sixth line. By allowing for any repetition in the document, the files are giving away some information as to the nature of the contents which where encrypted, thus presenting the documents in a less secure cipher text. This repletion is not observed in either the CBC modes, which may have thus contributed to the image viewing software's inability to present the data in any meaningful way, despite being able to understand that the data of the cipher text was indeed an image.

Task 3

The initial document which underwent encryption was a simple .txt file which contained the first five line of the 2007 DreamWorks Animation film: *Bee Movie.* Figure 19 shows a the contents of this file when viewed in the terminal.

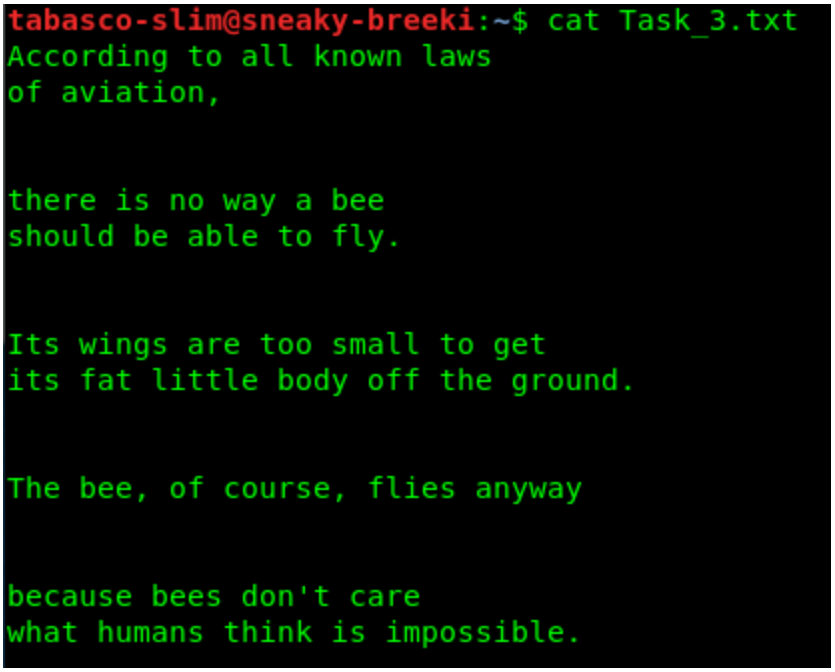

Figure 19: Task_3.txt in terminal

Note: As the documentation for this lab did not specify what the contents of the text file should be, I chose to use the opening line of a film which I enjoy.

The Lab documentation instruction state:

You will create a small text file that is several blocks long, encrypt the file using DES or an AES configuration of your choice, flip a single bit in the cipher text and then decrypt the corrupted cipher text to observe the impact of the corruption on each of the above 4 modes of operation.

This appears to contain a contradiction, as choosing a single DES of AES configuration does not yeil four different sets of corruption along the 4 different combinations of DES and AES in it's various modes. Subseqently I have performed the bit modification of the on the aforementioned .txt file with all combinations of DES and EAS with CBC mode and ECB mode.

The flipping of a bit is done on the same place of the bit in each version of the encrypted .txt files. Here the two bytes: **7c 1f8.** This can be represented as the binary number: **0111 1100 111 1000.**

Flipping the most significant bit of this number will give the number: **1111 1100 111 1000.** This can be represented as hexadecimal number: **fc f8.**

```
tabasco-slim@sneaky-breeki:~$ openssl enc -aes-128-cbc -in Task_3.txt -e -out Task_3_aes_cbc.txt -K b
e942347a422a302a5850e74f059ea2d -iv d6df31e4b9002da841d62e71a6f9b17c
tabasco-slim@sneaky-breeki:~$ openssl enc -aes-128-ecb -in Task_3.txt -e -out Task_3_aes_ecb.txt -K b
e942347a422a302a5850e74f059ea2d
tabasco-slim@sneaky-breeki:~$ openssl enc -des-cbc -in Task_3.txt -e -out Task_3_des_cbc.txt -K 73745
ef64e2c9800 -iv e3d383b531ad9edd
tabasco-slim@sneaky-breeki:~$ openssl enc -des-ecb -in Task_3.txt -e -out Task_3_des_ecb.txt -K 73745
ef64e2c9800
tabasco-slim@sneaky-breeki:~$ vim Task_3_aes_ecb.txt
tabasco-slim@sneaky-breeki:~$ vim Task_3_aes_cbc.txt
tabasco-slim@sneaky-breeki:~$ vim Task_3_des_cbc.txt
tabasco-slim@sneaky-breeki:~$ vim Task_3_des_ecb.txt
```

Figure 20: Encryption of the Task_3.txt and Edit of output files

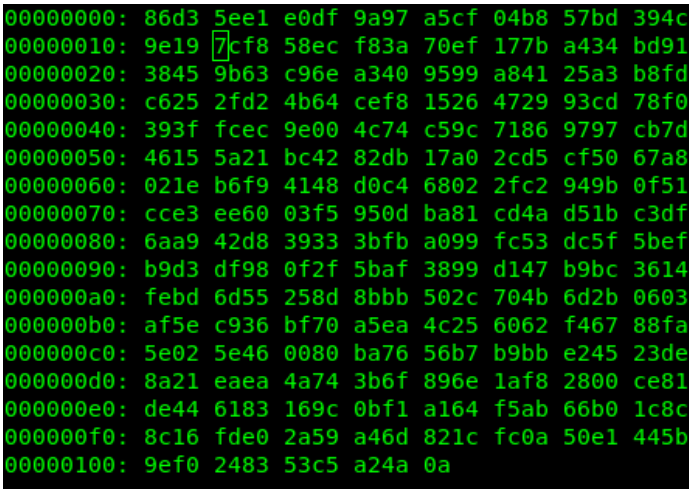

Figure 21: Task_3_des_ecb.txt pre-edit

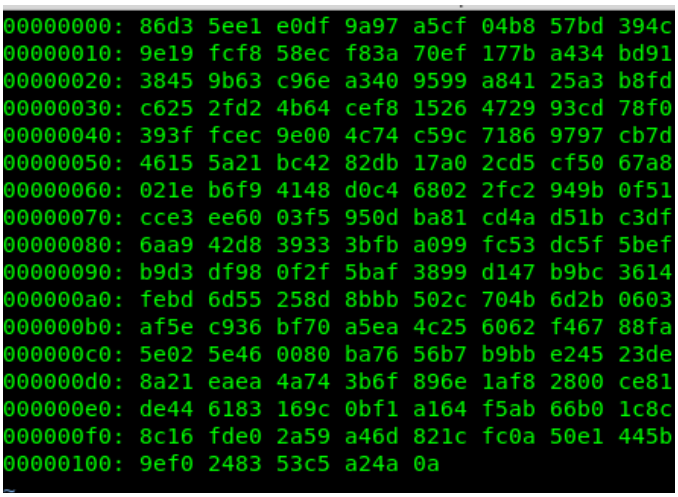

Figure 22: Task_3_des_ecb.txt post-edit

This exact process is shown here on the Task_3_des_ecb.txt file and is conducted on all other Task _3.txt file variants. Figure 22 and Figure 21 shows the bit flip on as described in above.

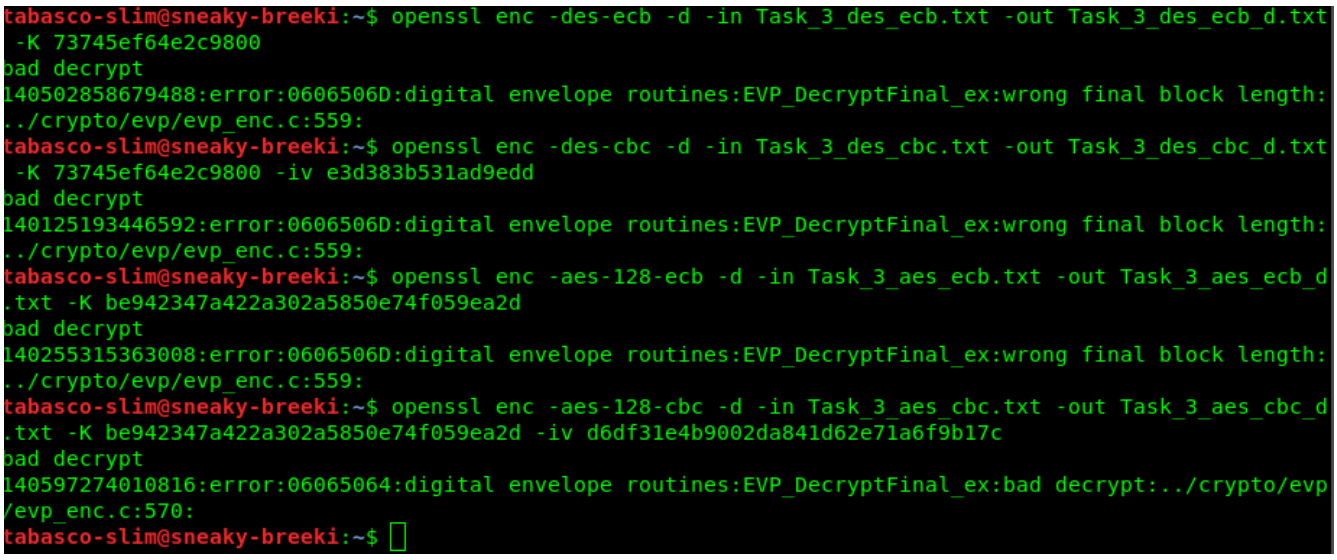

Figure 23: The decryption of the variants of Task_3.txt

```
tabasco-slim@sneaky-breeki:~$ cat Task_3_des_ecb_d.txt
According to all@V@,d@Vuaws
of aviation,

there is no way a bee
should be able to fly.

Its wings are too small to get
its fat little body off the ground.

The bee, of course, flies anyway

because bees don't care
what humans think is impossible.

tabasco-slim@sneaky-breeki:~$
```

Figure 25: cat Task_3_des_ecb_d.txt

```
tabasco-slim@sneaky-breeki:~$ cat Task_3_des_cbc_d.txt
According to all@@@@aw@
of aviation,

there is no way a bee
should be able to fly.

Its wings are too small to get
its fat little body off the ground.

The bee, of course, flies anyway

because bees don't care
what humans think is impossible.

tabasco-slim@sneaky-breeki:~$
```

Figure 24: cat Task_3_des_cbc.txt

```
tabasco-slim@sneaky-breeki:~$ cat Task_3_aes_ecb_d.txt
According to all@@@TF@9@@@@(viation,

there is no way a bee
should be able to fly.

Its wings are too small to get
its fat little body off the ground.

The bee, of course, flies anyway

because bees don't care
what humans think is impossible.
```

Figure 26: cat Task_3_aes_ecb_d.txt

Though the single bit flips have little effect on most of the combinations of the encryption and modes, it is worth noting that the cat Task_3_aes_cbc_d.txt file is completely incoherent. The conclusion here is thus that the combination of AES and CBC results in a cipher text that can be completely destroyed with the flipping of a single bit.

```
tabasco-slim@sneaky-breeki:~$ cat Task_3_aes_cbc_d.txt
@?$@$a@2L@9v@@@@@wlP@@@@l#@.@6L@F0=@@@5`@>B↴@@ow2Vg@[\Fv@0@S@X$@!@S:
                @@@@@@I:@?@@TG@@V+@@A@^a^E@I'@@@@@@JY@h@_@@@Q@@@@k@X@>
%@,mH#@@@mQA@d@O@@@ayo@@@_@(⊦@a1@{J@@@5^5@@@m@@@@@@Y;M@@-@@'@5@
                                                        @@n@@@@@d@R@0@@@7q@x@i@p@@VнR@@л@3,@'G@M
%X@l.gn5%@@@@Wv<@@@@9"@@
@@@>'@v@@@@vPX)
                @7@@8@@:<3@ZB@hb@@@]'@@,@@J@@@@ @V@U@tabasco-slim@sneaky-breeki:~$
```

Figure 27: cat Task_3_aes_cbc_d.txt

PART B (Undergraduate extra credit)

The objective of this section is to ascertain if the phenomena observed in the encryption and editing of an image .bmp file can be found in performing the same actions on a document file. The document which I crafted for the first phase of Part B was a simple .doc file was a simple .doc file titled: secret_song.doc. This document contain the song lyrics to “Hey Jude” by the Beatles, a song which is regarded as being one of the most well received yet highly repetitive songs of the 20th century. One may hypothesize that the highly repetitive nature of this song could yield is easily identifiable paterns in the syntax analysis of the encrypted documents.

```
tabasco-slim@sneaky-breeki:~$ openssl enc -des-ecb -e -in secret_song.doc -out secret_song_des_ecd.doc
 -k a9b1c06713c099d7
```

Figure 28: des-ecb on secre_song.doc

```
tabasco-slim@sneaky-breeki:~$ openssl enc -aes-128-ecb -e -in secret_song.doc -out secret_song_aes_ecb
.doc -K fcf8c5a6ca90d5b7d782a73da19496ec
```

Figure 29: aes-ecb on secre_song.doc

```
tabasco-slim@sneaky-breeki:~$ openssl enc -des-cbc -e -in secret_song.doc -out secret_song_des_cbc.doc
 -k a9b1c06713c099d7 -iv e940524ef4584672
```

Figure 30: des-cbc on secre_song.doc

```
tabasco-slim@sneaky-breeki:~$ openssl enc -aes-128-cbc -e -in secret_song.doc -out secret_song_aes_ecb
.doc -K fcf8c5a6ca90d5b7d782a73da19496ec -iv
```

Figure 31: aes-cbc on secre_song.doc

For the section of documents which I did not compose, I chose to download several Literary classics from the Gutenberg Project in the for of .txt files. Specifically I chose to used Time Machine by H. G. Wells, The Idiot by Fyodor Dostoyevsky, and War and Peace by Leo Tolstoy. These three documents can be found here:

https://www.gutenberg.org/files/35/35-0.txt
https://www.gutenberg.org/files/2638/2638-0.txt
https://www.gutenberg.org/files/2600/2600-0.txt

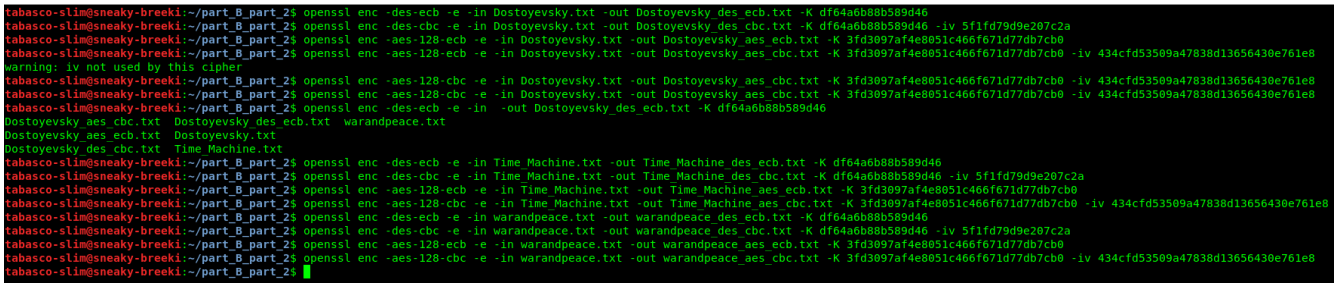

Figure 32: The Encryption of several files available online

The Method for performing an analysis on the encrypted documents consisted of two phases, one using C++ and another using Python 3.6. Once the documents are encrypted, they appear considerably

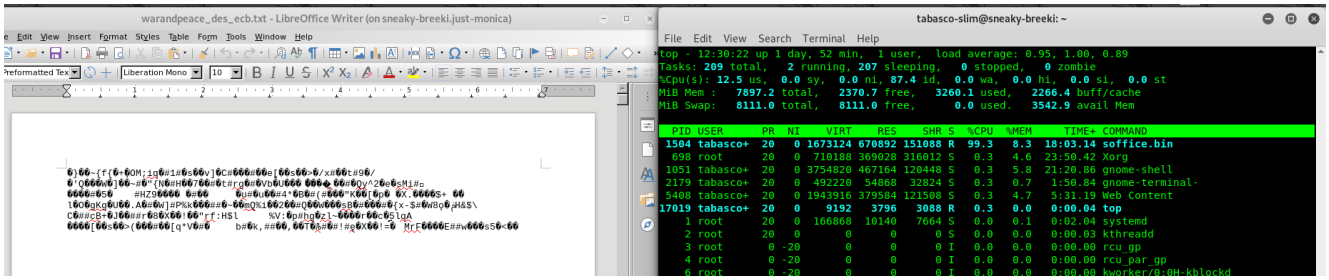

Figure 33: War and Peace, DES ECB, LibreOffice

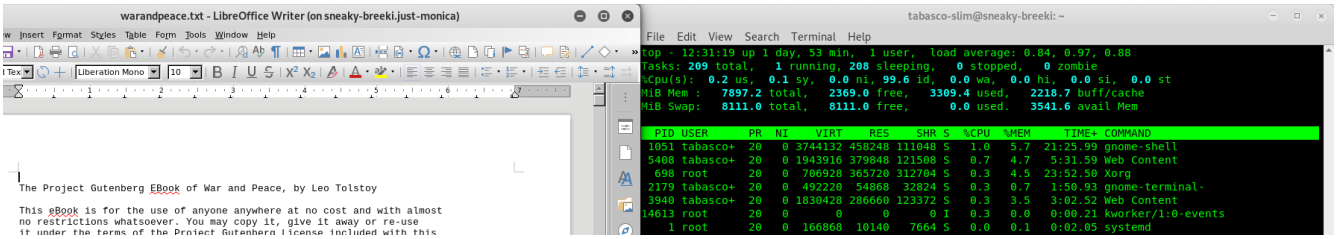

Figure 34: War and Peae, Plain Text, LibreOffice

different when viewed by basic office software. See Figure 33 and Figure 34 for comparision.

> *Note: An interesting observation can be made when attempting to open one of the encrypted files in office sofrware as apposed to attempting to open the plain text file. Given that in LibreOffice, Tolstoy's War and Peace is about over 1,100 pages long, it is understandable that opening the document would be very taxing on the machine. Though opening the plain text edition of war and peace is taxing, the CPU usage does not remain at above 95% for more than a few seconds where as all encrpted versions of the document when viewed in the safe software with the same background world load do cause a considerable impact on the machine's workload. This phenomena was initially noted through the activation of the test machine's internal fans durring the viewing of documents for screen capturing purposes and was then document using the top command in the the terminal. See Figure 33 and Figure 34 for more details.*

The first phase of analysis involves the counting of character in the encrypted files. Here the analysis breaks down into an in depth analysis and a general analysis, with an in-depth annalists having been successfully performed on the secrete_song.doc while only a general analysis was successfully conducted on the various online available files. The reason for there being a variation in the analysis of the two section of Part B rests in two issue. The first issue being that this experiment as well as this report are limited in resources, time, and technical guidance regarding applied encryption in a Linux environment; thus further development of this project is not possible at this time. The second issue is due to the fact that the representation of individual characters in a .doc file, a .txt file, and the cpp

methods for handling this characters made the multi-character analysis of documents using the methods used in the .doc portion of this experimenter did not work when evaluating .txt files. Such an issue is outside the domain of the Texas Tech University, Fall 2019, CS4311 course.

The design decision to execute the first phase of the analysis in C++ was made with the understanding that C++ is a considerably faster language to execute tasks when compared to Python. The complete analysis of War and Peace on a machine with a 7th generation i7 processor took approximately 4 seconds.

When anylizing the secret_song.doc file, three categories of charters where examined, those being individual characters, charters groupings of two, and charter groupings of three.

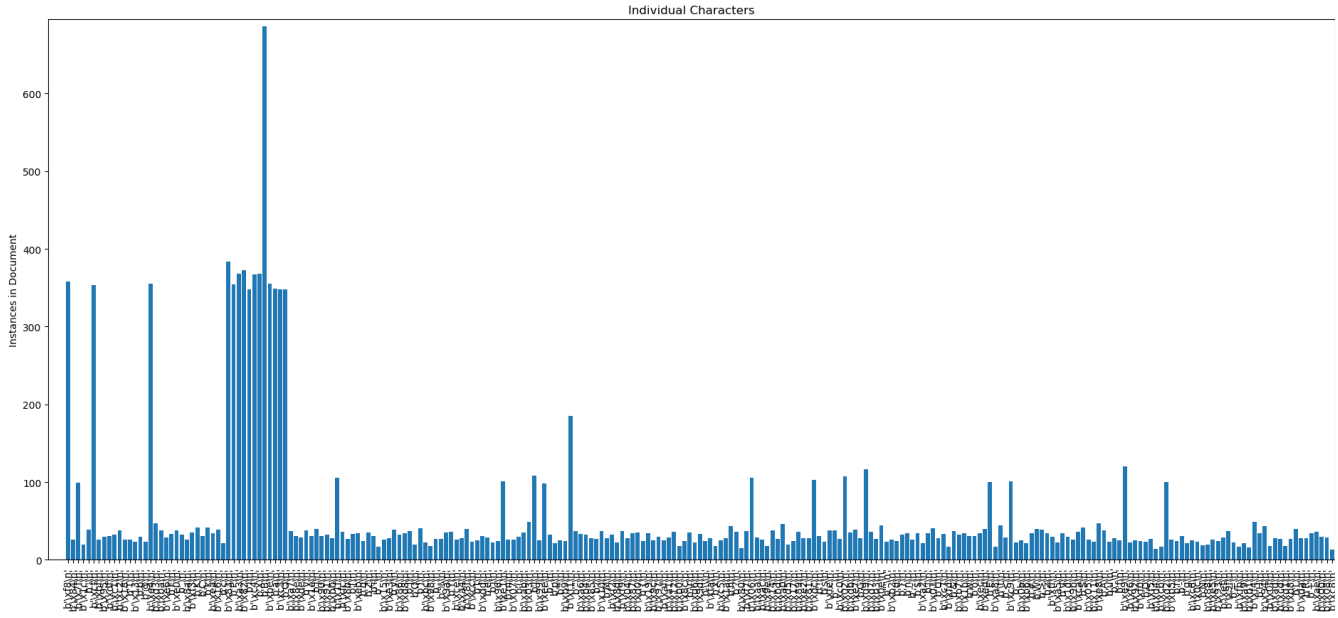

Figure 35: Secret_song.doc, AES ECB, 1 char

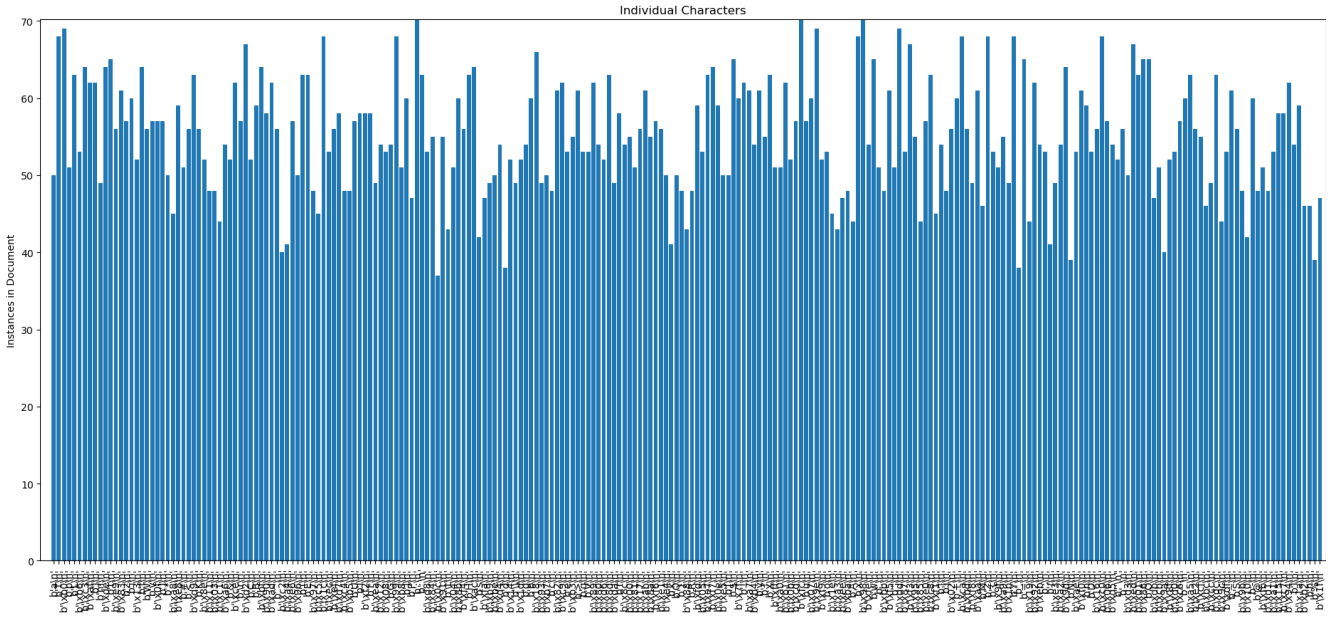

Figure 36: Secret_song.doc, AES CBC, 1 char

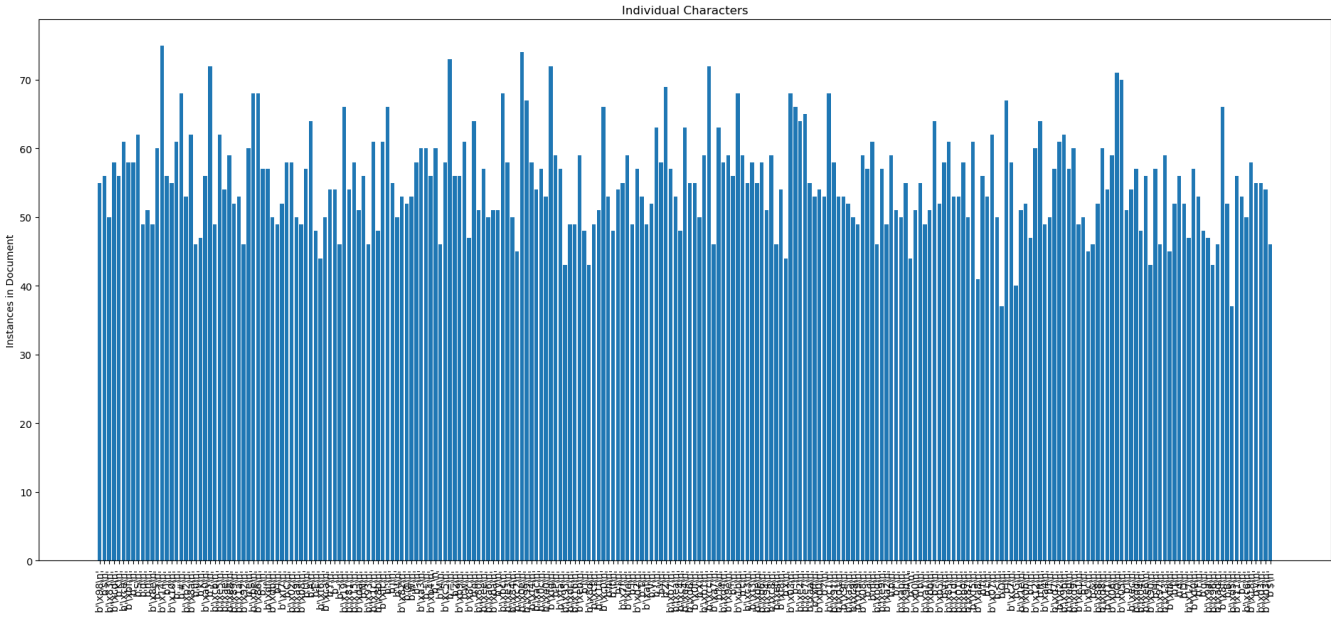

Figure 37: Secret_song.doc, DES CBC, 1 char

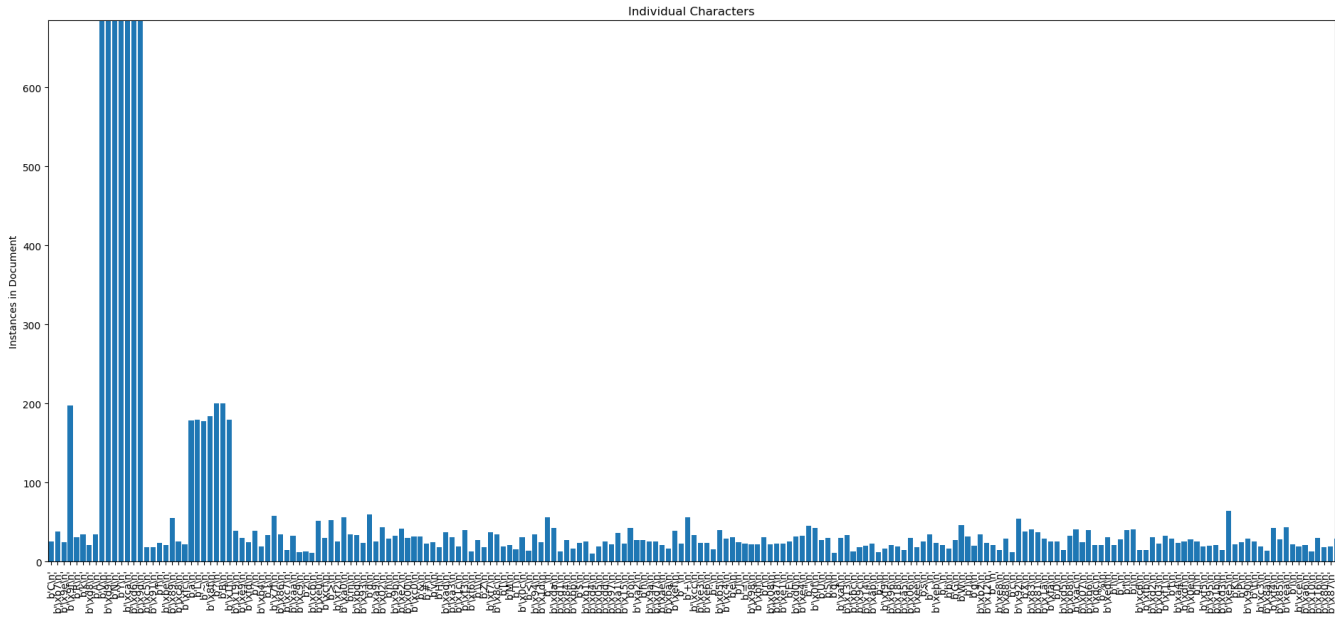

Figure 38: Secret_song.doc, DES ECB, 1 char

Next, groupings of two characters were examined.

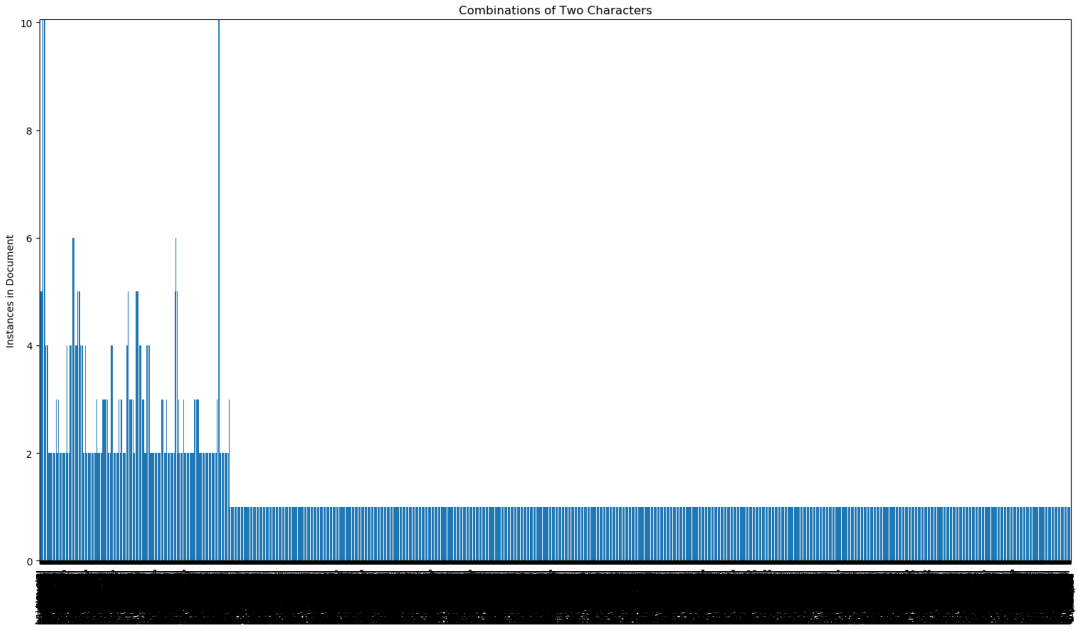

Figure 39: Secret_song.doc, AES ECB, 2 chars

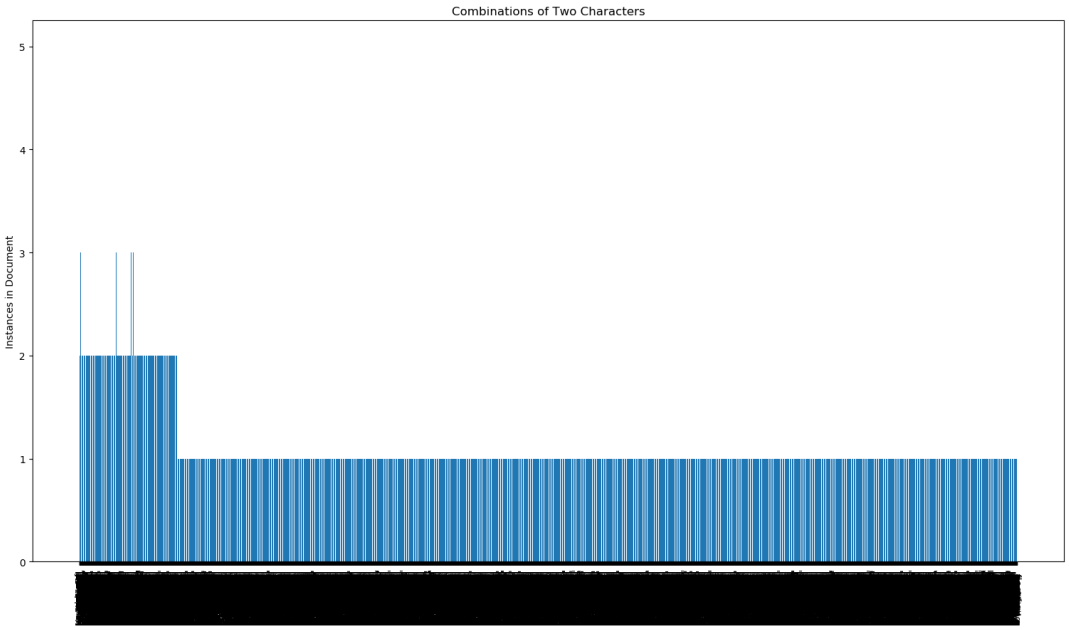

Figure 40: Secret_song.doc, AES CBC, 2 chars

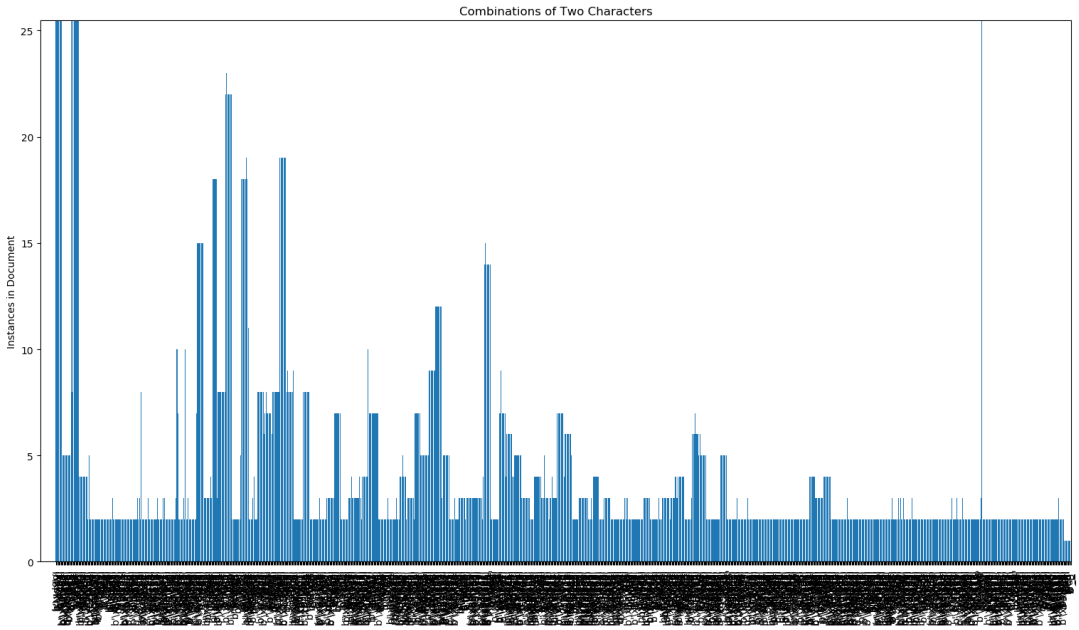

Figure 41: Secret_song.doc, DES ECB, 2 chars

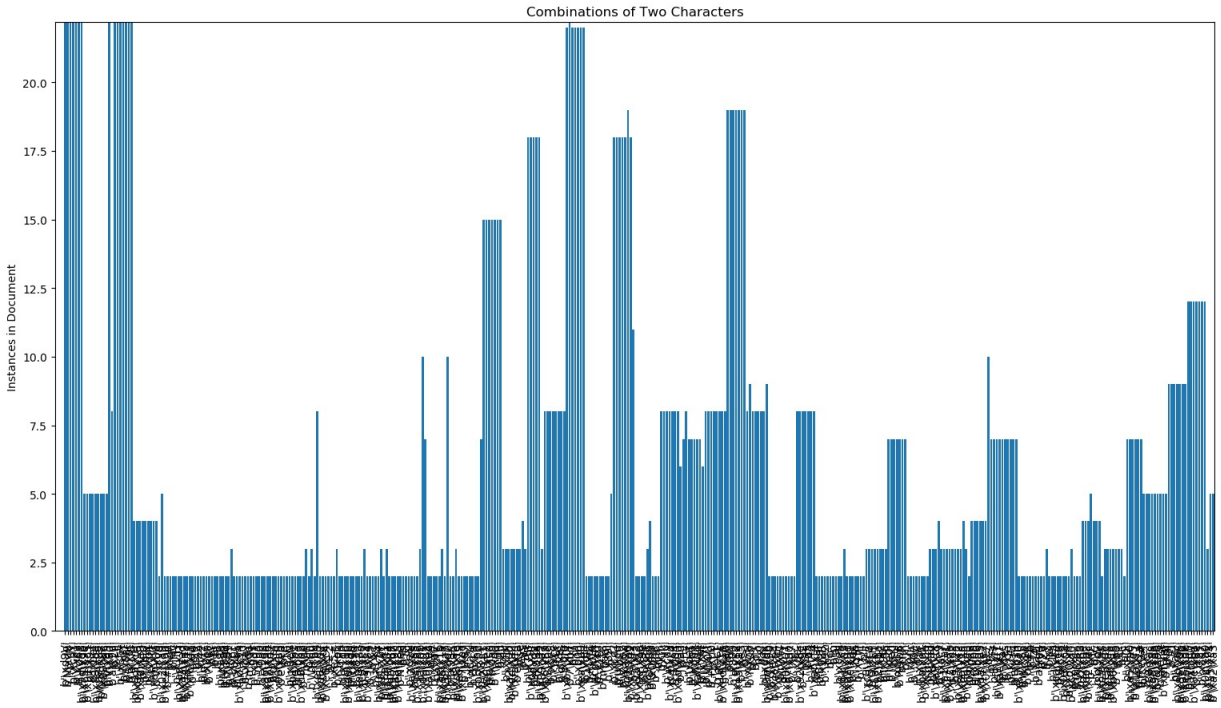

Figure 42: Secret_song.doc, DES CBC, 2 chars

Finally, groupings of three characters where examined.

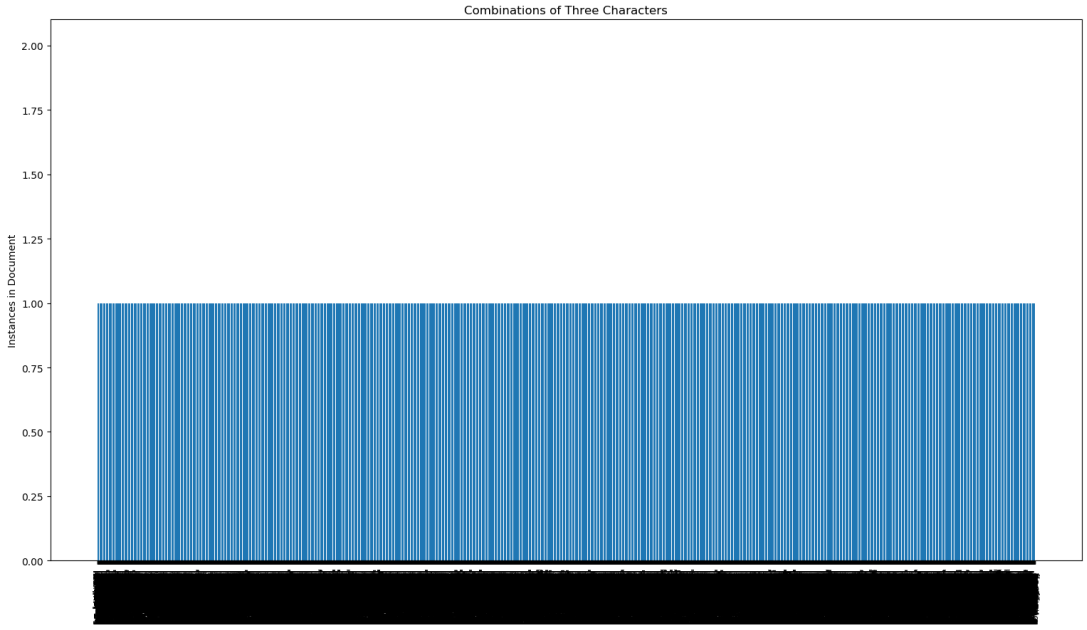

Figure 43: Secret_song.doc, AES CBC, 3 chars

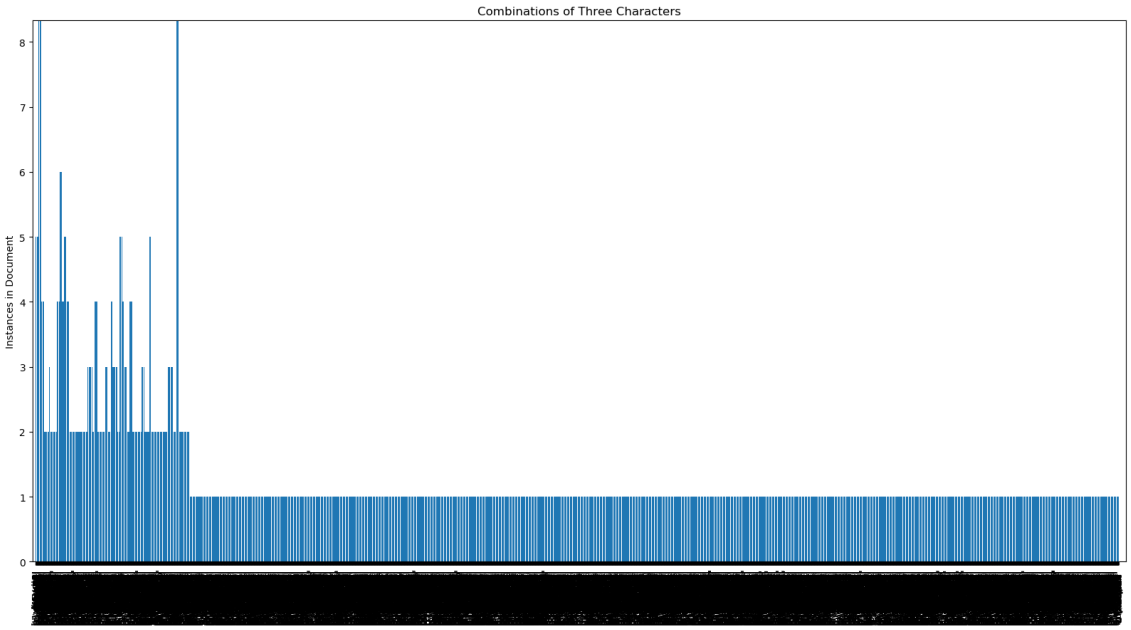

Figure 44: Secret_song.doc, AES ECB, 3 chars

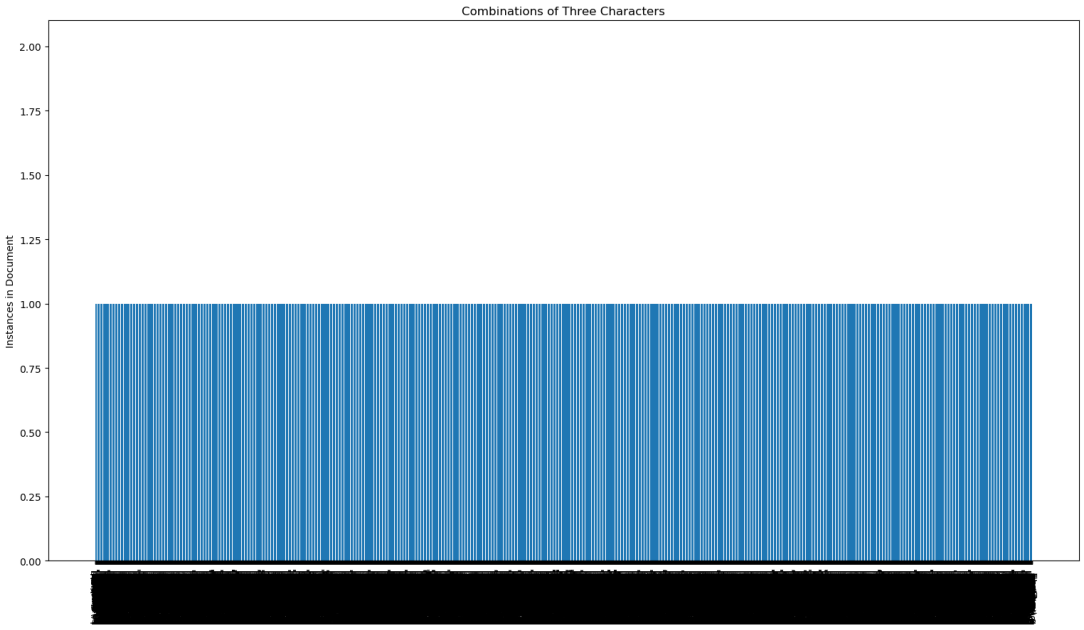

Figure 45: Secret_song.doc, DES CBC, 3 chars

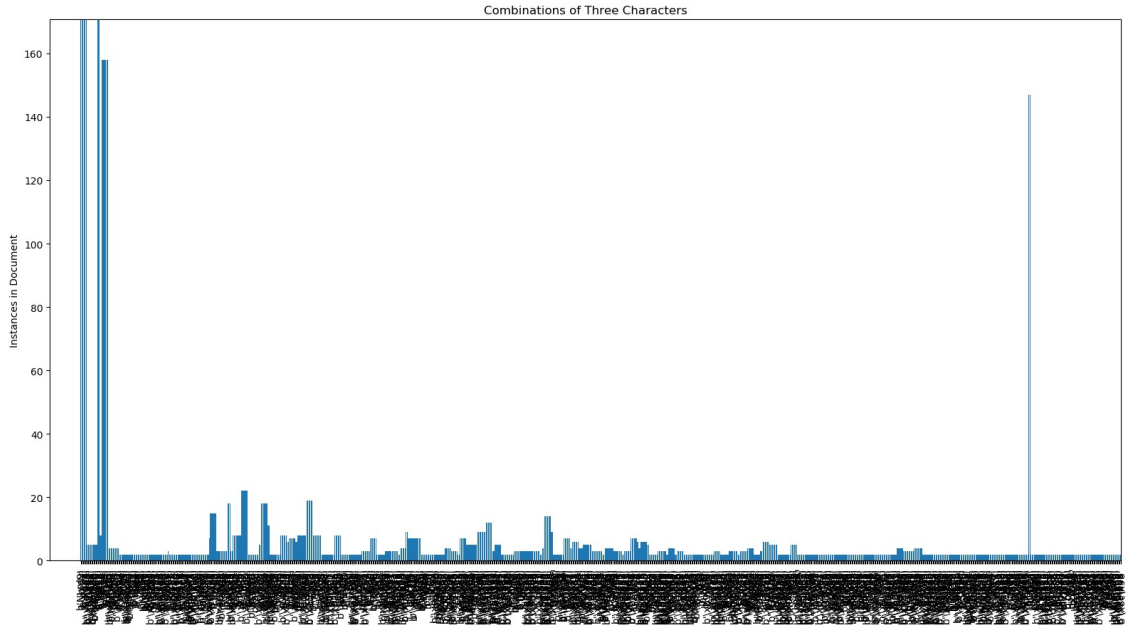

Figure 46: Secret_song.doc, DES ECB, 3 chars

As previously stated, an analysis was only conducted on the online available documents with groupings of a single character, the results are as follows.

Time Machine by H. G. Wells:

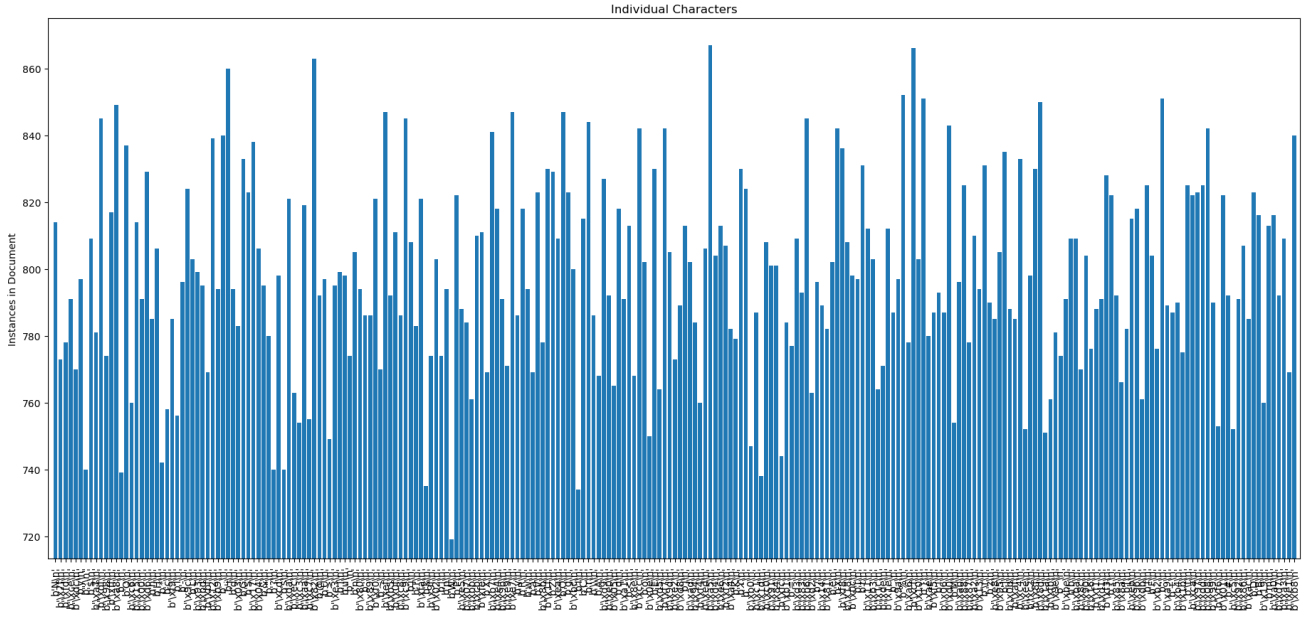

Figure 47: Time Machine, AES CBC, 1 char

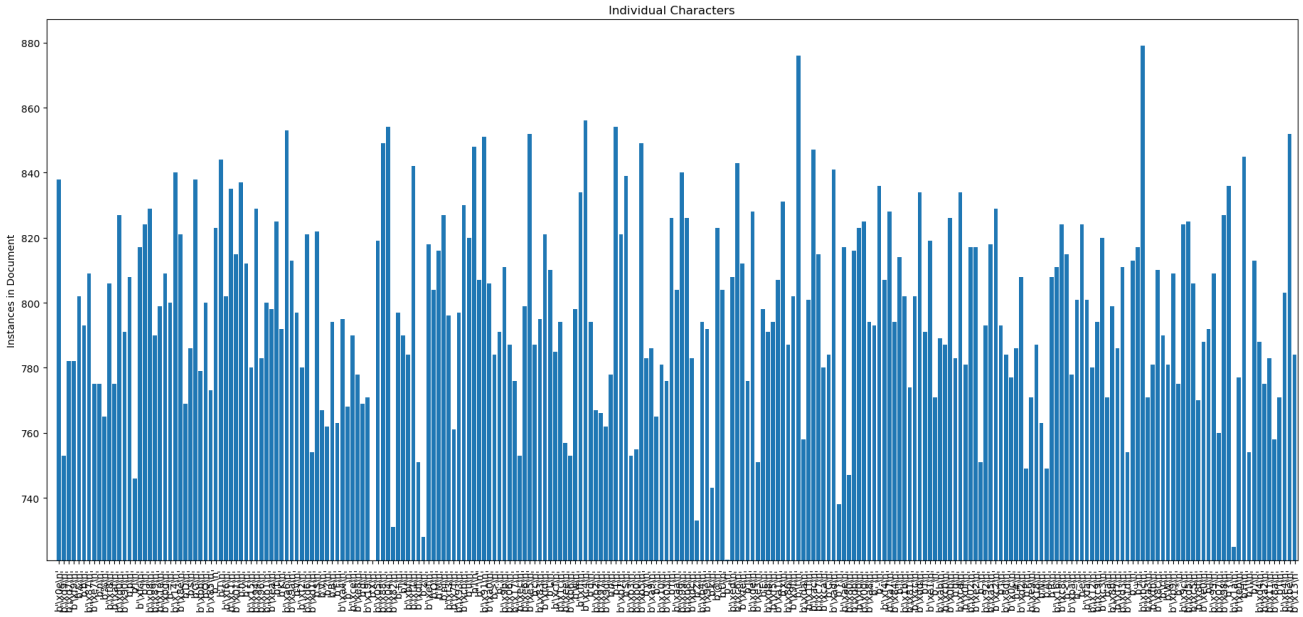

Figure 48: Time Machine, AES ECB, 1 char

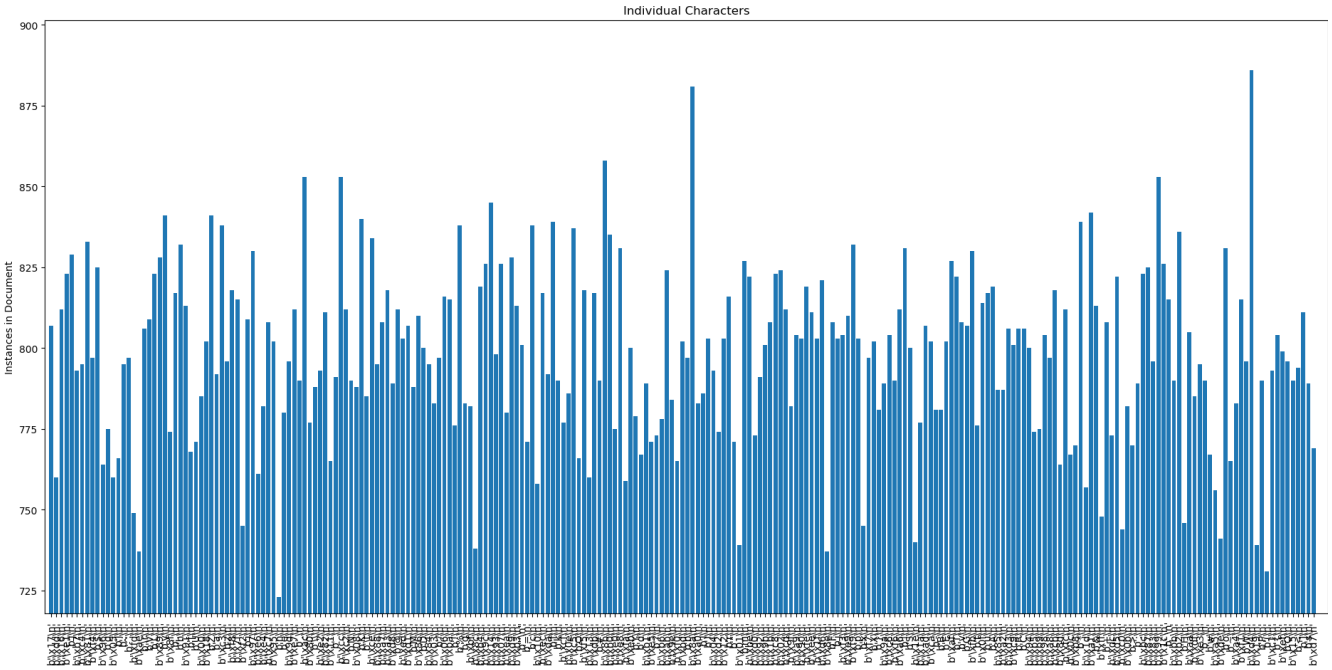

Figure 49: Time Machine, DES CBC, 1 char

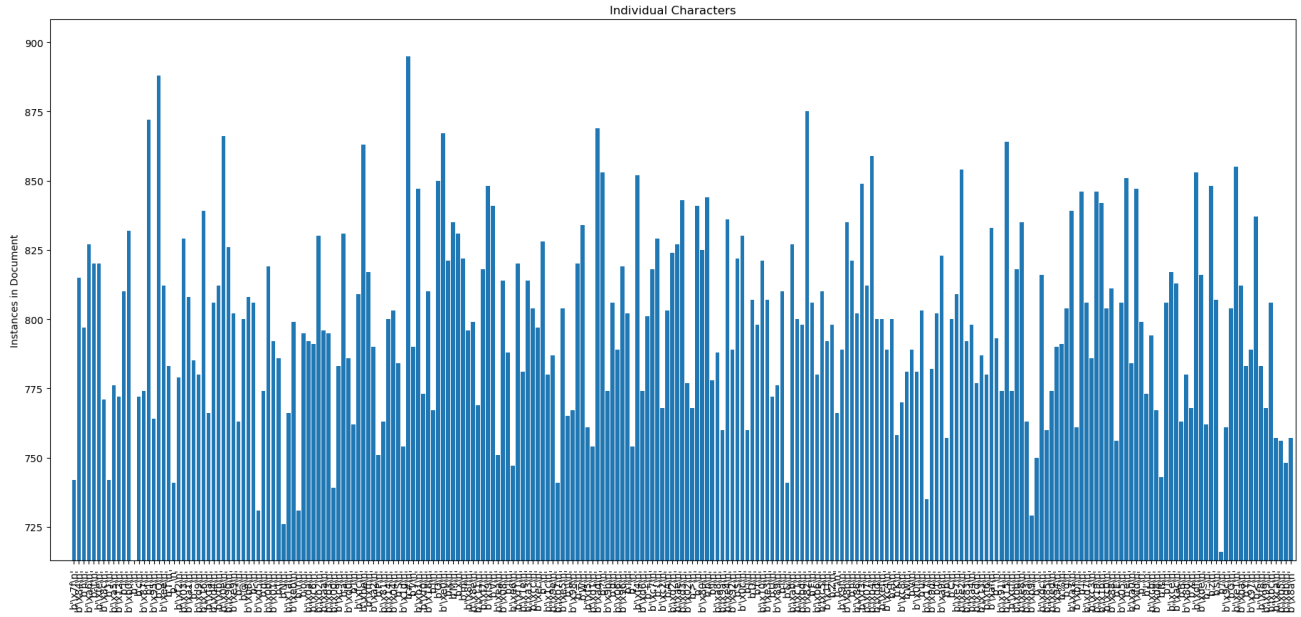

Figure 50: Time Machine, DES ECB, 1 char

The Idiot by Fyodor Dostoyevsky:

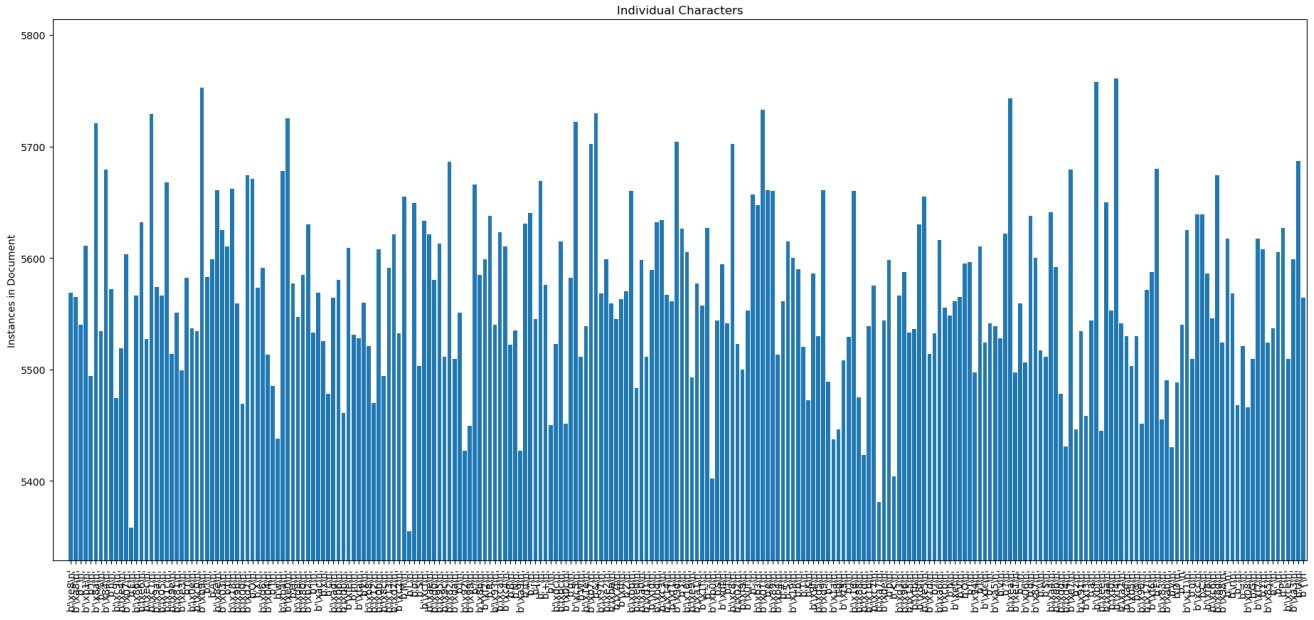

Figure 51: The Idiot, AES CBC, 1 char

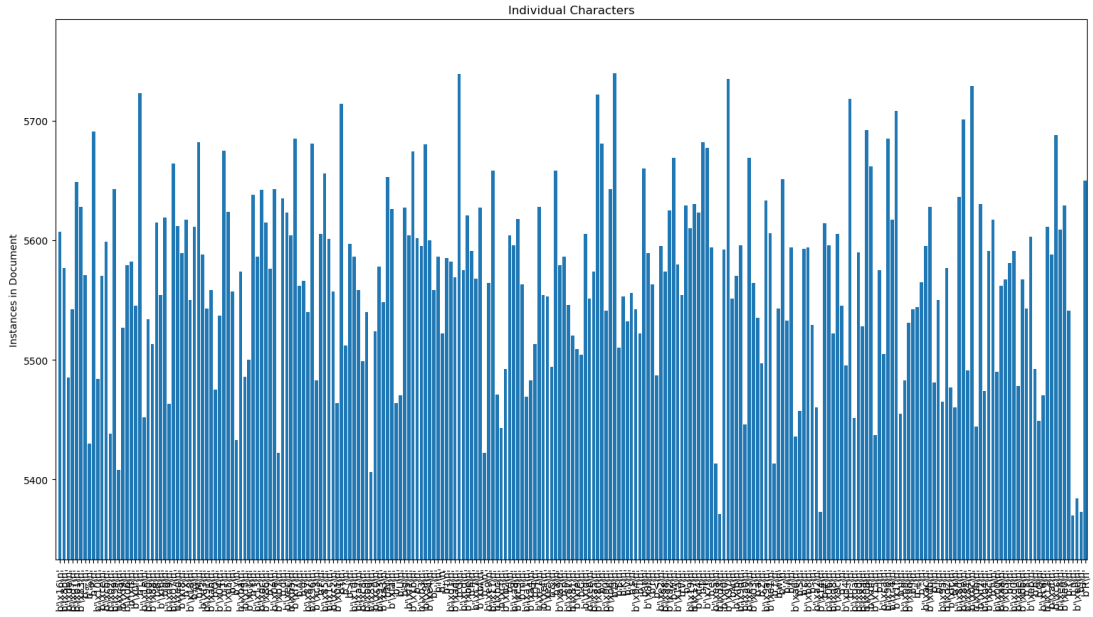

Figure 52: The Idiot, AES ECB, 1 char

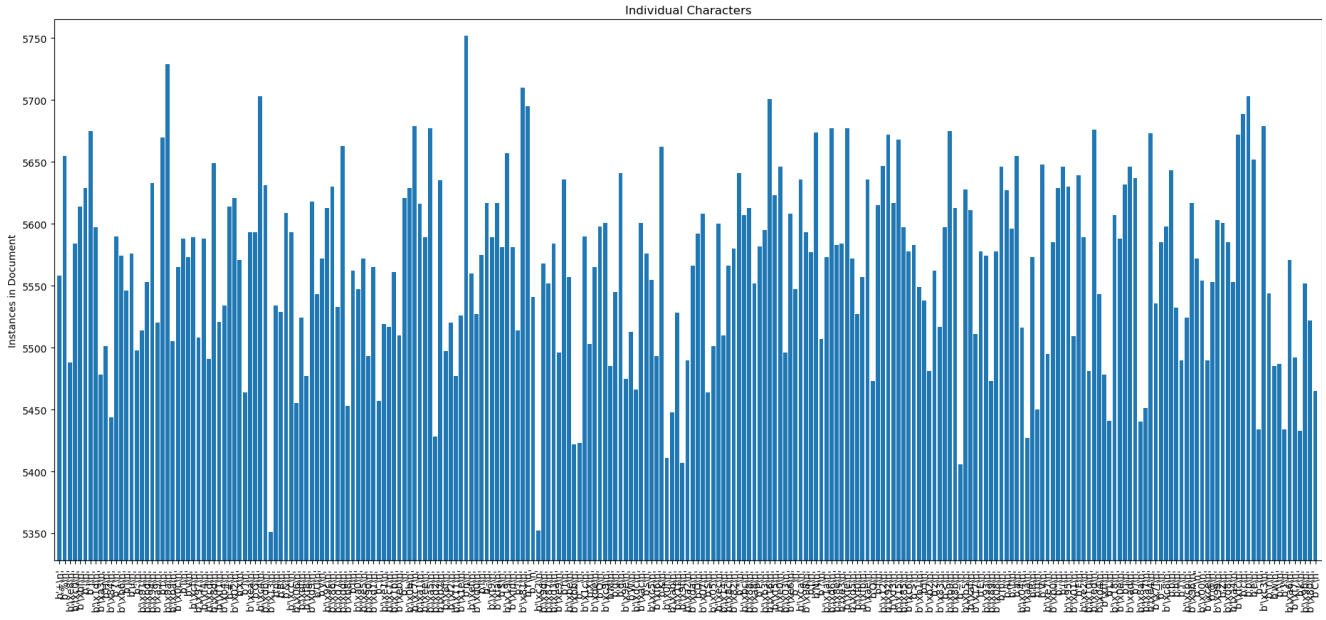

Figure 53: The Idiot, DES CBC, 1 char

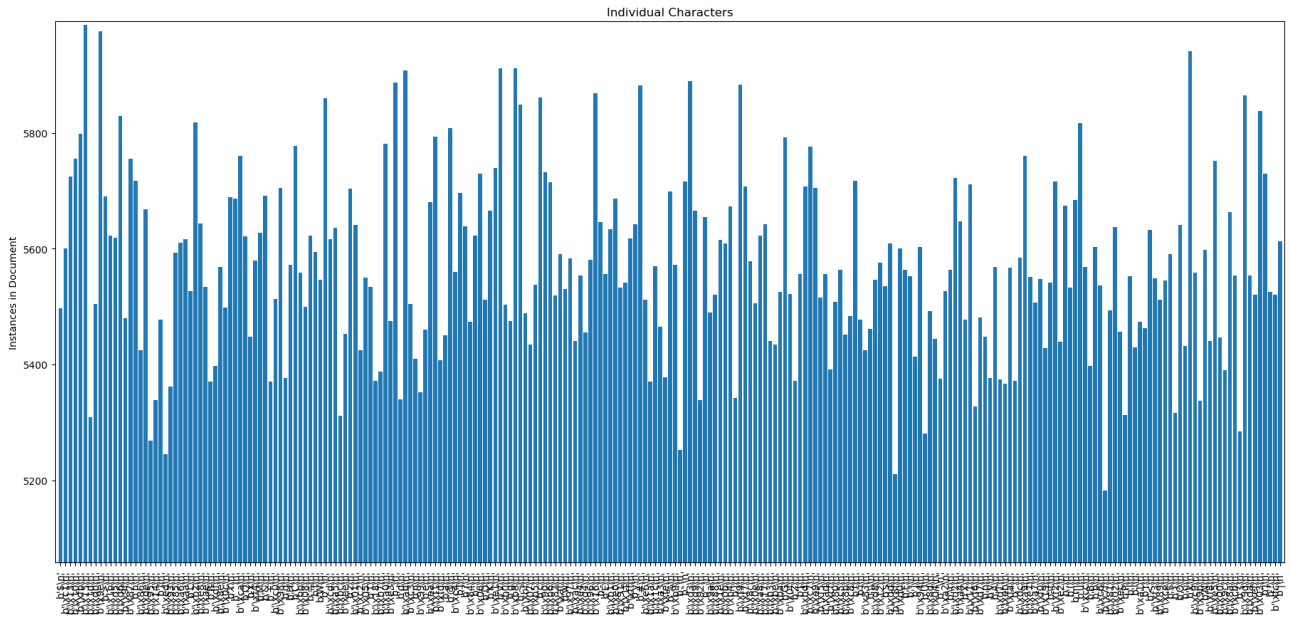

Figure 54: The Idiot, DES ECB, 1 char

War and Peace by Leo Tolstoy:

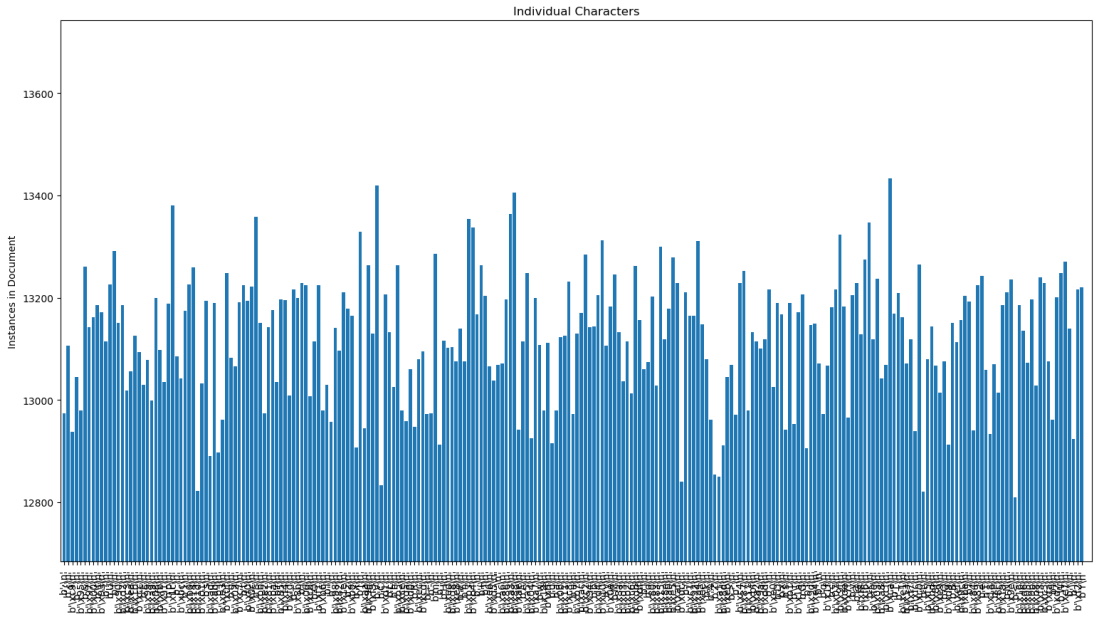

Figure 55: War and Peace, AES CBC, 1 char

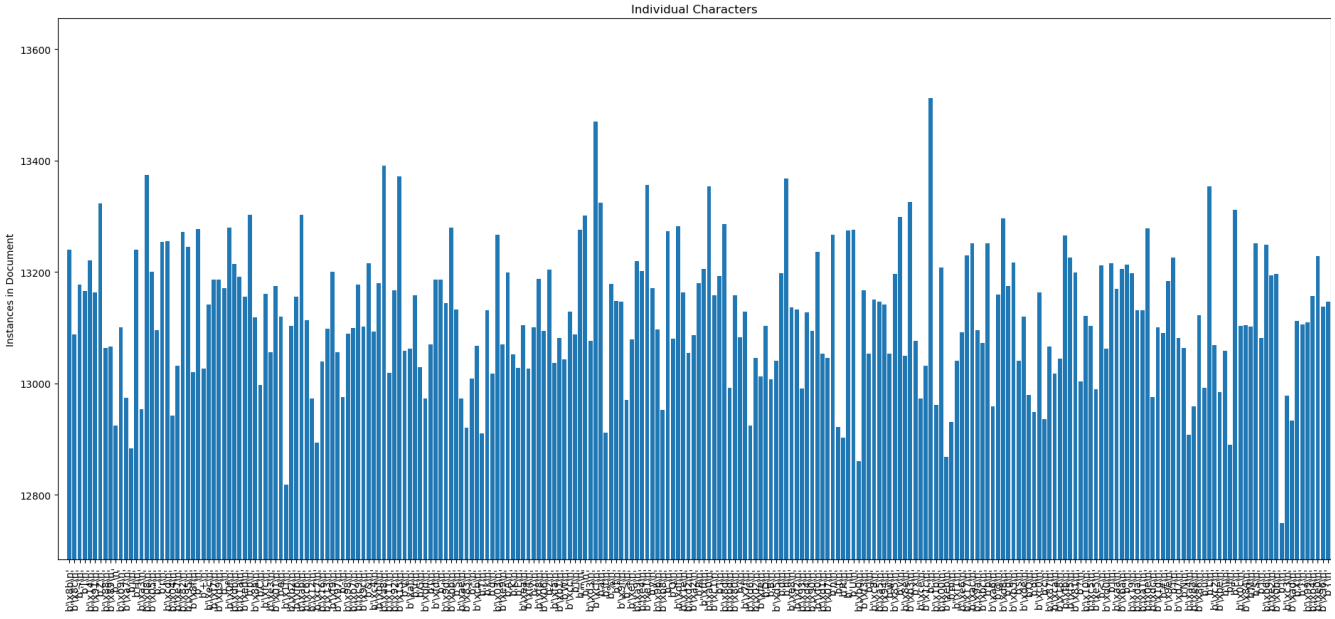

Figure 56: War and Peace, AES ECB, 1 char

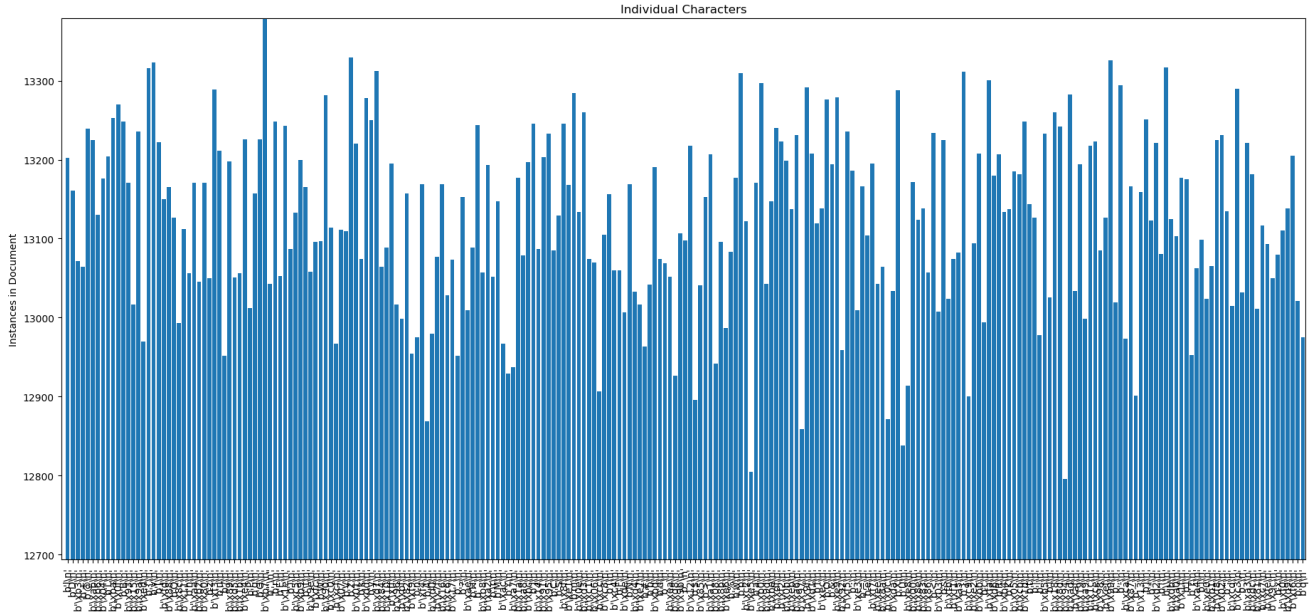

Figure 57: War and Peace, DES CBC, 1 char

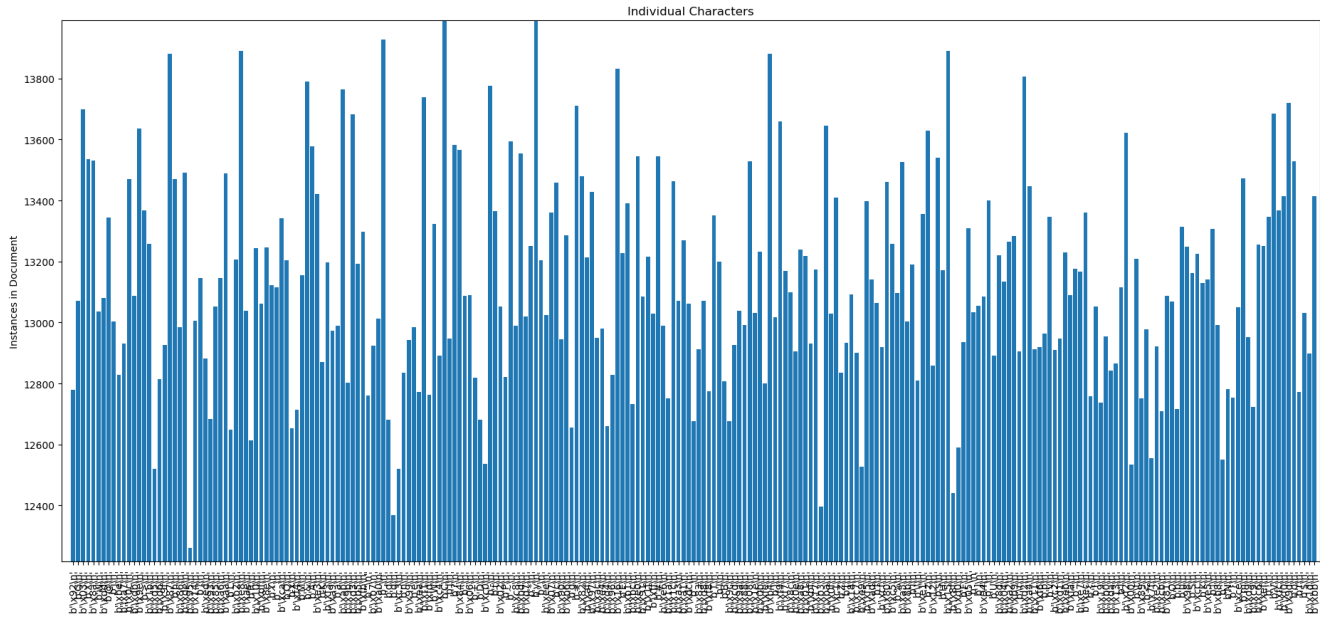

Figure 58: War and Peace, DES ECB, 1 char

Python 3.6 was chosen as the method for visualization due to the relative easy to use nature of the nump and matplotlib libraries. Given the the non-specific nature of the instructions for this lab via "using plots, tables, summary metrics, etc." the data visualization portion of this report was deemed a low enough priority that Python was the only appropriate option.

Summary of findings.

As illustrated in Figure 43 and Figure 45, CBC is a far stronger mode when compared to ECB in the way of privacy protection. Consider that when patterns arise in the frequent usage of three combinations of characters in these documents, an attack may be able to better target the contents of a document. Though both Figure 43 and Figure 44 show the use of the AES encryption algorithm, the AES ECB scheme does cause a very high spike in several three letter combinations. Though the combination of AES and ECB does still provide a stronger encryption scheme than DES ECB, the AEA ECB scheme still fails to be as level of a distribution as the AES CBC scheme.

When examine Figures 47 through 58 this conclusion is not obvious though there does appear to be somewhat less of a gap between the characters of highest and least frequency when using CBC mode than when using ECB mode. For all Encryption schemes depicted in Figures 47 through 58, the same number of characters, 245, is recorded. In all schemes, the strenght of CBC mode over ECB mode is shown as CBC mode consistently shows a shorter gap between the least frequently occurring character and the most frequently occurring character.

In conclusion, though both the in depth analysis and general analysis do not all explore the same character combination variations, both analyzes help support the claim that CBC provides a stronger mode for privacy protection.

Note: As the the scripts written for the analysis of this lab have undergone many changes to generate each visual, some sections of the scripts will need to be edited with the appropriate input and output files to replicate this experiment. For an easy in understanding how this works, all scripts for the general analysis will be submitted with this lap while only the last edited variations of the scripts used for the in depth analysis will be submitted. All scripts will also be saved on my personal github page in the event that the submission size is too large for Texas Tech’s document submission client.