Assignment 2 Solution – ENGI 9807 (Part 03)

Sourav Barua

20199158

1. Symmetric key cryptography (CBC Encryption)

I wrote a python program and made a function named *blockCipher(plaintext,IV,key).* This function takes the plaintext, the initialization vector and the key as input parameters. It then transforms the characters from the plain text to their respective ASCII value and stores them in an array. Then for each of the characters, it applies the given cryptographic function using CBC mode. The applied function is –

$$C = k \cdot P \bmod 256$$

The function uses the IV (Initialization vector) to apply XOR with the first character of the given plaintext and then for the next characters it XORs the character's binary representation with the binary representation of the cipher of the previous character. Below is a snap of the python program –

```
cbc_blockCipher.py > ...
def blockCipher(plaintext,IV,key):
    initVector = IV
    P_ASCII = []
    CIPTEXT = []
    for i in range(len(plaintext)):
        P_ASCII.append(ord(plaintext[i]))
    # Applying the CBC mode block cipher
    for i in range(len(P_ASCII)):
        if(i==0):
            CIPTEXT.append(((P_ASCII[i]^IV)*key)%256);
        else:
            CIPTEXT.append(((P_ASCII[i]^CIPTEXT[i-1])*key)%256);


    print("Plain Text: "+ plaintext)
    print("ASCII Form of Plain Text: ")
    for item in P_ASCII:
        print(item, end=" ")
    print("\n")

    print("Cipher text generated from Plain Text (Integer Representation): ")

    for item in CIPTEXT:
        print(item, end=" ")
    print("\n")
    print("Cipher text generated from Plain Text (Binary Representation): ")

    for item in CIPTEXT:
        print(bin(item)[2:].zfill(8), end=" ")
    print("\n")
    return CIPTEXT

########### main() ###############################
def main():
    cipherText = blockCipher("hello",201,197)


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

Figure 1: Python program to get cipher text using block cipher in CBC mode

Then I tried to use the function to encrypt the word “hello” using initialization vector 201 and key 197.

```
(base) D:\MUN\SPRING 2020 Computer Security\Assignments\Assignment 2 files\Tiny_blockcipher>C:/Users/User/Anaconda3/python.exe "d:/MUN/SPRING 2020 Computer Security/Assignments/Assignment 2 files/Tiny_blockcipher/cbc_blockCipher.py"
Plain Text: hello
ASCII Form of Plain Text:
104 101 108 108 111

Cipher text generated from Plain Text (Integer Representation):
229 128 156 176 155

Cipher text generated from Plain Text (Binary Representation):
11100101 10000000 10011100 10110000 10011011
```

Figure 2: Output of the program when called blockCipher(“hello”,201,197)

So, the resulting cipher text is –
Integer Representation:
229 128 156 176 155
Binary Representation:
11100101 10000000 10011100 10110000 10011011

2. Symbol Frequency

1. I have written a python program named calcSymbolFreq.py. This file takes Jones and Mewhord’s data collection as an input file. It then calculates the total number of times each symbol is observed (a) and the relative frequency(b). Below is a snapshot of the program’s code –

```
cbc_blockCipher.py   calcSymbolFreq.py ×   charFrequencyAnalysis.tsv   test.txt   symbolFreq.tsv
calcSymbolFreq.py > main
import pandas as pd
import sys
import csv

def main():
    filename = sys.argv[1]
    symbolFreqDF = pd.read_csv(filename,sep='\t')
    symbolStat = pd.DataFrame(columns=["Char", "Total Observations", "Relative Frequency"])
    symbolStat["Char"] = symbolFreqDF["Char"]
    symbolStat["Total Observations"] = symbolFreqDF.sum(axis = 1)
    sumOfTotalObservations = symbolStat["Total Observations"].sum()
    symbolStat["Relative Frequency"] = (symbolStat["Total Observations"]/sumOfTotalObservations) * 100
    symbolStat.at[1,"Char"] = '"'

    print("Result Table \n")
    print("########################################")
    print(symbolStat.to_string())
    symbolStat.to_csv("charFrequencyAnalysis.tsv", sep="\t", index=False,float_format='%.4f')

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

Figure 3: The calcSymbolFreq.py file

Running this python file and supplying the filename as first command line argument generates a file named charFrequencyAnalysis.tsv file. These are the content of that file –

```
Char    Total Observations  Relative Frequency
!   636553  0.0649
""""    3323331 0.3387
#   201113  0.0205
$   495185  0.0505
%   266890  0.0272
&   239640  0.0244
'   3377384 0.3442
(   2085979 0.2126
```

```
)   2142308 0.2183
*   752611  0.0767
+   234124  0.0239
,   10074601    1.0267
-   10947554    1.1157
.   16444697    1.6759
/   3320184 0.3384
0   6511154 0.6636
1   5166554 0.5265
2   4035391 0.4113
3   3288586 0.3352
4   2455656 0.2503
5   2296191 0.2340
6   1877213 0.1913
7   1674985 0.1707
8   1950750 0.1988
9   2971693 0.3029
:   2967137 0.3024
;   2470716 0.2518
<   930860  0.0949
=   2641886 0.2692
>   85852   0.0087
?   866561  0.0883
@   769603  0.0784
A   5031723 0.5128
B   2308278 0.2352
C   3809278 0.3882
D   2984941 0.3042
E   3068242 0.3127
F   2022803 0.2062
G   1619925 0.1651
H   2034148 0.2073
I   5004871 0.5101
J   1014208 0.1034
K   663229  0.0676
L   2121771 0.2162
M   2891665 0.2947
N   2659109 0.2710
O   2323268 0.2368
P   2557089 0.2606
Q   200371  0.0204
R   2582550 0.2632
S   4741083 0.4832
T   5085655 0.5183
U   1460694 0.1489
```

```
V   703443  0.0717
W   1948044 0.1985
X   253121  0.0258
Y   852848  0.0869
Z   159215  0.0162
[   292949  0.0299
\   62120   0.0063
]   285414  0.0291
^   56593   0.0058
_   1570378 0.1600
`   71570   0.0073
a   66151947    6.7418
b   11698024    1.1922
c   26475250    2.6982
d   30035729    3.0610
e   101601923   10.3546
f   16763191    1.7084
g   16555383    1.6872
h   36511435    3.7210
i   60485020    6.1642
j   1259963 0.1284
k   5891236 0.6004
l   34297439    3.4954
m   21779233    2.2196
n   59681186    6.0823
o   65122881    6.6369
p   17443337    1.7777
q   1050576 0.1071
r   53072396    5.4088
s   53804421    5.4834
t   74784318    7.6215
u   24341343    2.4807
v   8908357 0.9079
w   14400662    1.4676
x   1962589 0.2000
y   15047067    1.5335
z   1015126 0.1035
{   138062  0.0141
|   546703  0.0557
}   156144  0.0159
~   300046  0.0306
```

First column represents the total number of observations of that symbol and the second column indicates the relative frequency of that symbol in percent

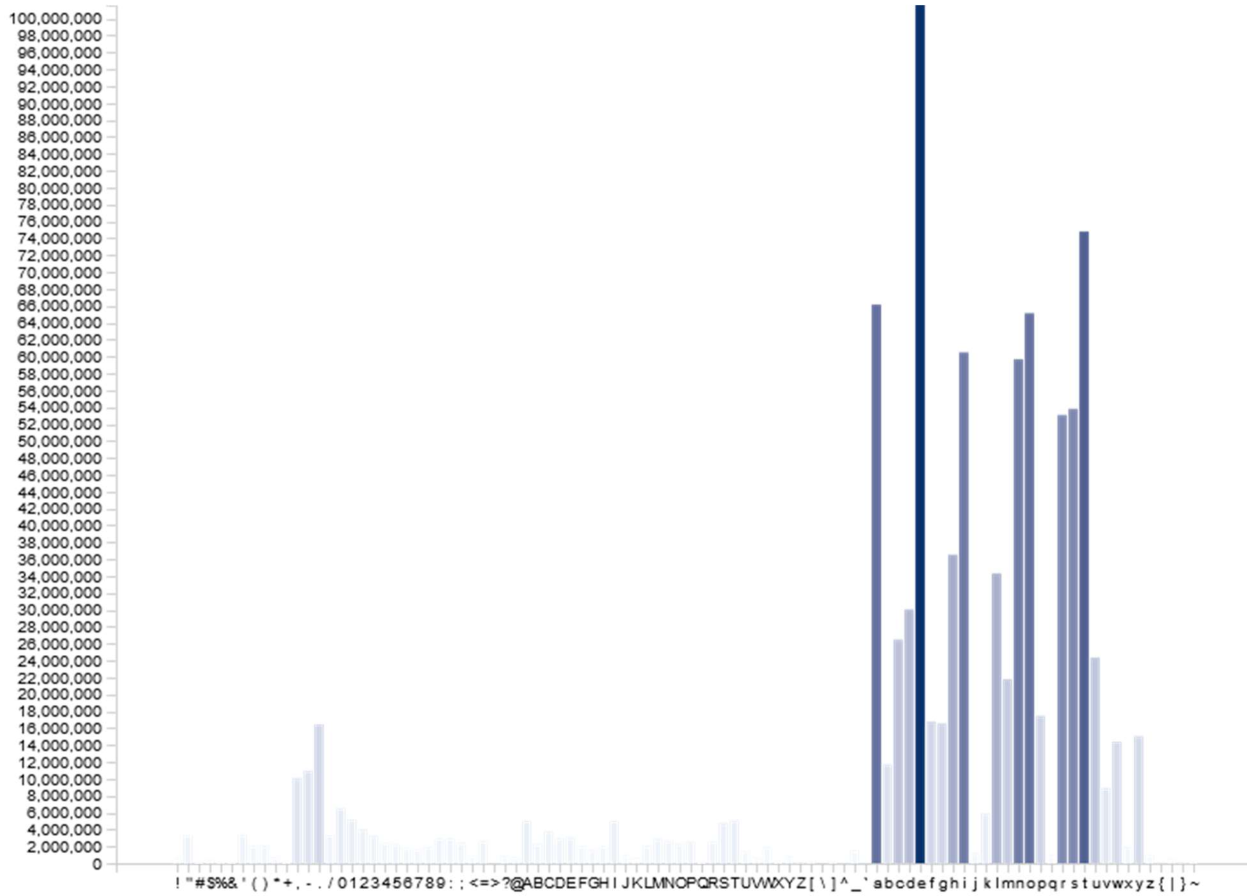

Figure 4: Symbol vs relative frequency bar plot

From the visual, it can be clearly seen that, the characters e, a, i, t has the most relative frequency on the provided dataset.

3. Shannon Entropy

3.1 password - This password only contains the lowercase alphabetic letters. So, it can possibly have 26 different choices and it is 8 characters long. The Shannon Entropy for this password would be –

$$log_2|26^8|= 37.60\ Sh\ (Bits)$$

3.2 password1 – This password contains lowercase letters (26 choices) and numeric values (10 choices) and it is 9 characters long. So,

$$log_2|36^9|= 46.53\ Sh\ (Bits)$$

3.3 password!6@ - This password has 11 characters, containing lowercase letters (26), numbers (10) and special characters (32).

$$log_2|68^{11}|= 66.96\ Sh\ (Bits)$$

3.4 y9]z'626:g – This password has 10 characters and it has lowercase, numbers and special characters.

$$log_2|68^{10}|= 60.87\ Sh\ (Bits)$$