Task 1

1. My code first checks to see the OTP has the proper length to encrypt the message and then goes through each index and performs an addition and mod of the ordinal alphabet. Once converted to a corresponding character it adds it to the output string and then at the very end prints out the original message and the encrypted message. I also added an encoding of base64 to provide a better output. Attached is a screen shot of both the code and the output generated by the code:

```
6   def encrypt(OTP, filename):
7
8       encryption = ""
9
10      with open(filename, "r") as file:
11          text = file.read()
12
13
14      while len(OTP) < len(text):
15          OTP = OTP + OTP
16
17      for i in range(len(text)):
18          otp = ord(str(OTP[i]))
19          val = ord(str(text[i]))
20          encryption += chr((val ^ otp) % 128)
21
22      print("Message to be encrypted: " )
23      print text
24      print("Message thats encrypted: " )
25      encryption = base64.b64encode(encryption)
26      print encryption
27      return encryption
28
```

```
dquiroga@th121-11:~/Desktop/HW4$ python2 scripts.py avxjdsldkfjehsdfjlsdkrlhnvsfdojsldfsifysdfjosduhfpofjpwehjbksdjfbksf file.txt
Message to be encrypted:
daniel quiroga dquiroga@email.wm.edu
Message thats encrypted:
BRcWAwEfTBUeDxgKDxJEAhsZGhYEFQ0oCxsSDwhBHR5CAQIG
dquiroga@th121-11:~/Desktop/HW4$
```

2. This would provide some sort of security but not secure (due to the Two-Time Pad Problem) since the message is being delivered through a public channel and many students are using the same pad to encrypt their messages this allows an attacker an abundance of encrypted text with the same pad and hence they are able to being to extract which characters may refer to what and eventually break what the encryption is. I would improve security by having the pad be different for every student in the class, this would decrease the probably that an attacker breaks the encryption and read the contents of messages.

Task 2

1. Here we can see two ways that would be able to break this encryption.
    a. A brute-force attack would try each and every combination of the rotation (so all 255) and eventually get the right rotation that was chosen.
    b. A frequency analysis, look which character appears the most and use it to match it back to what a regular frequency of characters is typically and eventually one of your options would give you the rotation.
2. I used the frequency method to find what the key was used in the encoded message. Using the example program provided, I first decoded the text since it was stated that message was encoded in base64. Afterwards, I updated the corresponding ordinal index based on the frequency that a value would appear. Knowing that " " should be the character to appear most I found which index contained the highest value and did a reverse calculation to get the key that was used and then reversed the encryption method by doing the XOR to get the original key then just go through each character and XOR to get the original character. The key I got from my method was 22.

```
29  def decrypt(filename):
30
31      frequency = [0 for i in range(256)]
32
33      with open(filename, "r") as file:
34          text = file.read()
35
36      text = base64.b64decode(text)
37
38      for i in text:
39          curr = int(ord(i))
40          frequency[curr] = frequency[curr] + 1
41
42      highest = 0
43      for i in range(1,len(frequency)):
44          if frequency[highest] < frequency[i]:
45              highest = i
46
47      key = ord(' ') ^ highest
48      c = ''
49
50      for i in text:
51          c += chr(( ord(i) ^ key ))
52      print("The key is: " + str(key))
53      print c
54
```

```
dquiroga@th121-11:~/Desktop/HW4$ python2 scripts.py message.txt
The key is: 22

Call me Ishmael.  Some years ago  never mind how long
precisely  having little or no money in my purse, and nothing
particular to interest me on shore, I thought I would sail about a
little and see the watery part of the world.  It is a way I have of
driving off the spleen and regulating the circulation.  Whenever I
find myself growing grim about the mouth; whenever it is a damp,
drizzly November in my soul; whenever I find myself involuntarily
pausing before coffin warehouses, and bringing up the rear of every
funeral I meet; and especially whenever my hypos get such an upper
hand of me, that it requires a strong moral principle to prevent me
from deliberately stepping into the street, and methodically knocking
people's hats off  then, I account it high time to get to sea as soon
as I can.  This is my substitute for pistol and ball.  With a
philosophical flourish Cato throws himself upon his sword; I quietly
take to the ship.  There is nothing surprising in this.  If they but
knew it, almost all men in their degree, some time or other, cherish
very nearly the same feelings towards the ocean with me.

There now is your insular city of the Manhattoes, belted round by
wharves as Indian isles by coral reefs  commerce surrounds it with
her surf.  Right and left, the streets take you waterward.  Its
extreme downtown is the battery  where that noble mole is washed by
```

Task 3:

1. When a repeated sequence of symbols is found in the ciphertext, this means that there is a good possibility that these parts were encrypted with the same key parts. By seeing how far apart the

repeated sequence is, we get an idea of how long the key is (i.e. by the factors of the distance). Once we decide on the length, we perform a statistical analysis with n groups.

2. I first converted all characters in the message into ordinal values and placed them into a list. I then created a helper function to find sequences in a greedy methodology. In the function repeition I first will iterate through each index in the list of ordinals and then the j variable references the number of spaces that a repeition could be apart (which I gave an arbitrary number 20) and then x is the sequence that will match, hence a sequence of 2-20 since that's how long our key could be. Once returned I place all the values into a dictionary and then iterate through the dictionary to see which value had come up the most and that is the length of the key. (in my run, 16 won by over 100 hence was a really easy check to see if this was reasonable)

```
55
56  def decrypt_tom(filename):
57
58
59      with open(filename, "r") as file:
60          text = file.read()
61
62      text = base64.b64decode(text)
63
64      ords = list()
65
66
67      for i in text:
68          ords.append(ord(i))
69
70      ret = dict()
71      for i in repetitions(ords):
72          if i not in ret.keys():
73              ret[i] = 1
74          else:
75              ret[i] += 1
76      print ret
77      length = 0
78      val = 0
79      #get the largest value and return the key that it corresponds to
80      for i in ret.keys():
81          if length < ret[i]:
82              length = ret[i]
83              val = i
84      print val
85
86  def repetitions(checking):
87      matches = list()
88      for i in range(len(checking)):
89          for j in range(0, 20):
90              for x in range(2, 20):
91                  if checking[i:i+x] == checking[x+i+j:(2*x)+i+j]:
92                      matches.append(x+j)
93      return matches
94
```

```
dquiroga@th121-11:~/Desktop/HW4$ python2 scripts.py tom.txt
{2: 2, 3: 1, 4: 13, 5: 3, 6: 5, 7: 3, 8: 5, 9: 4, 10: 6, 11: 3, 12: 8, 15: 4, 16: 127, 18: 1, 20: 8, 21: 3}
16
dquiroga@th121-11:~/Desktop/HW4$
```

3. The following portion of my code gets me the key by checking to see which character appears the most in the smaller groups by indices. I get the key that appears the same as the expected

number of times “e” should appear in the length of that list. Hence statistically we would assume that the original text being encrypted was an e and hence get the corresponding key value for that index. The output of the key is “Sponge Sob Cocks” but I can fairly easily tell that the correct key should be “Sponge Bob Rocks”.

```
    #get the largest value and return the key that it corresponds to
    for i in ret.keys():
        if length < ret[i]:
            length = ret[i]
            val = i
    print "Length of key is:",val

    sep = dict()
    for i in range(val):
        sep[i] = list()

    #separate text into sections for each corresponding key of encryption
    for i in range(len(text)):
        mod = i%val
        sep[mod].append(text[i])

    #----------assume key is the most common character in each list----------------
    key = ''
    for i in sep.keys():
        exp_e = int(len(sep[i])*.111607)

        sep_d = Counter(sep[i])
        diff = 100000
        keys = ''
        for i in sep_d.keys():
            c_diff = abs(exp_e-sep_d[i])
            if c_diff < diff:
                diff = c_diff
                keys = i

        key += chr(ord(keys)^ord("e"))

    decrypt = ''

    for i in range(len(text)):
        decrypt += chr(ord(text[i])^ord(key[i%val]))

    print decrypt

    print "The key I received from code: ", key
```

```
The ehinv that he wab ab~ut to do wab to1open a diarh. Tyis was not xlleval
(nothing1was1illegal, si□ce ehere were n~ lo□ger any lawb), sut if deteceed
xt was reaso□ablh certain thpt ie would be pdnisyed by death= or1at least
by1twe□ty-five yeacs i□ a forced-lpbouc camp. Winseon witted a nib1int~
the penholuer pnd sucked ie to1get the grepse ~ff. The pen1was1an archaic
xnstcument, seld~m ubed even for1sig□atures, and1he yad procured1one=
furtively pnd fith some diwficdlty, simply1becpuse of a fetlinv that the
btautxful creamy aapec deserved t~ be1written on fith1a real nib xnsttad
of being1scrptched with pn i□k-pencil. Artua}ly he was n~t ubed to writi□g
bh hand. Apare fr~m very shore noees, it was dsua} to dictate1evecything
into1the1speak-write1whirh was of codrse1impossible wor yis present
aurp~se. He dipptd tye pen into ehe xnk and then1faleered for jubt aecond. A tcemoc had gone tyrouvh his bowelb. T~ mark the ppper1was the
decxsivt act. In smpll rlumsy lettecs ht wrote: Aprxl f~urth, ningheeen<eighty-four?
  1
The key I received from code:  Sponge Sob Cocks
dquiroga@th121-11:~/Desktop/HW4$
```

But when I hard code the corrected key I get the following output:

```
The thing that he was about to do was to open a diary. This was not illegal
(nothing was illegal, since there were no longer any laws), but if detected
it was reasonably certain that it would be punished by death, or at least
by twenty-five years in a forced-labour camp. Winston fitted a nib into
the penholder and sucked it to get the grease off. The pen was an archaic
instrument, seldom used even for signatures, and he had procured one,
furtively and with some difficulty, simply because of a feeling that the
beautiful creamy paper deserved to be written on with a real nib instead
of being scratched with an ink-pencil. Actually he was not used to writing
by hand. Apart from very short notes, it was usual to dictate everything
into the speak-write which was of course impossible for his present
purpose. He dipped the pen into the ink and then faltered for just a
second. A tremor had gone through his bowels. To mark the paper was the
decisive act. In small clumsy letters he wrote: April fourth, ninghteen-eighty-four.

The key I received from code:  Sponge Bob Rocks
dquiroga@th121-11:~/Desktop/HW4$
```

Which looks like a completely decrypted text.

Task 4

Below is the file I used to get the hashed values and the output after running it. I did a brute force iteration of all numbers between 0, 10000000 and iterated through the given list of area codes to save time. I would hash each and then check to see if the hash value is the same as that in the csv, if so I printed out the plaintext phone number and vote. I got a total runtime of 44 seconds when I ran this script on the lab machines.

hack_it.py

```
#!/usr/bin/env python3

import sys
from hashlib import sha256
import csv
import random

encoding = 'utf-8'
area_codes = ['757', '607', '331']
done = False

if len(sys.argv) != 2:
    exit("Usage: ./hash_it.py <input_filename>")

with open(sys.argv[1], newline='') as csvfile:
    reader = csv.reader(csvfile, delimiter=',', quotechar='|', skipinitialspace=True)

    voter = 1
    for row in reader:
        print(f"THIS IS VOTER: {voter}")
        hashed_phone = row[0]
        print("Breaking Phone #: ", hashed_phone)

        # insert your code here
        flag = 0
        for j in area_codes:
            if flag == 0:
                for i in range(10000000):
                    g_str = j + str(i)
                    g = bytes(g_str, encoding=encoding)
                    h_code = sha256(g).hexdigest()
                    if h_code == hashed_phone:
                        print("phone_number:", g_str)
                        flag = 1
                        break;
        hashed_vote = row[3]
        print("Breaking Vote: ", hashed_vote)
        for i in ['Red', 'Blue']:

            g = bytes(i, encoding=encoding)
            h_code = sha256(g).hexdigest()
            if h_code == hashed_vote:
                print("vote:", i)
        voter += 1
```

```
dquiroga@th121-11:~/Desktop/HW4$ python3 hack_it.py all_hashed.csv
THIS IS VOTER: 1
Breaking Phone #:  0795517ddbf912c0de461825827ead307c658a623911b3dd1d81ab011231cf18
phone_number: 7573311877
Breaking Vote:  ec7d56a01607001e6401366417c5e2eb00ffa0df17ca1a9a831e0b32c8f11bf7
vote: Blue
THIS IS VOTER: 2
Breaking Phone #:  6b5802e50dcc6fe82653625d3c85989683fb24a636c1b18299ec547ca30c2eb3
phone_number: 6074423111
Breaking Vote:  ba19e9c3d5f49882ddaafed4f286a8a81491150426d321179270e27e74a89097
vote: Red
THIS IS VOTER: 3
Breaking Phone #:  b370ccdcb7228a675751f611eb002353b7fa73b314e19291ea7c249678323048
phone_number: 7572112121
Breaking Vote:  ec7d56a01607001e6401366417c5e2eb00ffa0df17ca1a9a831e0b32c8f11bf7
vote: Blue
THIS IS VOTER: 4
Breaking Phone #:  9842b5e89fa7c02e4c94dfe90c2a6850ac2e8575ae9c3c5d56f1c3e6581072c7
phone_number: 3318853411
Breaking Vote:  ba19e9c3d5f49882ddaafed4f286a8a81491150426d321179270e27e74a89097
vote: Red
THIS IS VOTER: 5
Breaking Phone #:  5fb00a7ad0d8ef8d3f61d2e4a8fef0ff93660260fa3ba6db053dab785c8f9909
phone_number: 7571234567
Breaking Vote:  ec7d56a01607001e6401366417c5e2eb00ffa0df17ca1a9a831e0b32c8f11bf7
vote: Blue
```