Lebanese American University

School of Arts and Sciences

Department of Computer Science and Mathematics

# CSC435 (Computer Security)

# Lab #04 (Cryptography)

A Report Done By

**Ahmad Hussein** ahmad.hussein03@lau.edu

**Mahmoud Joumaa** mahmoud.joumaa@lau.edu

And presented to Dr. Ayman Tajeddine

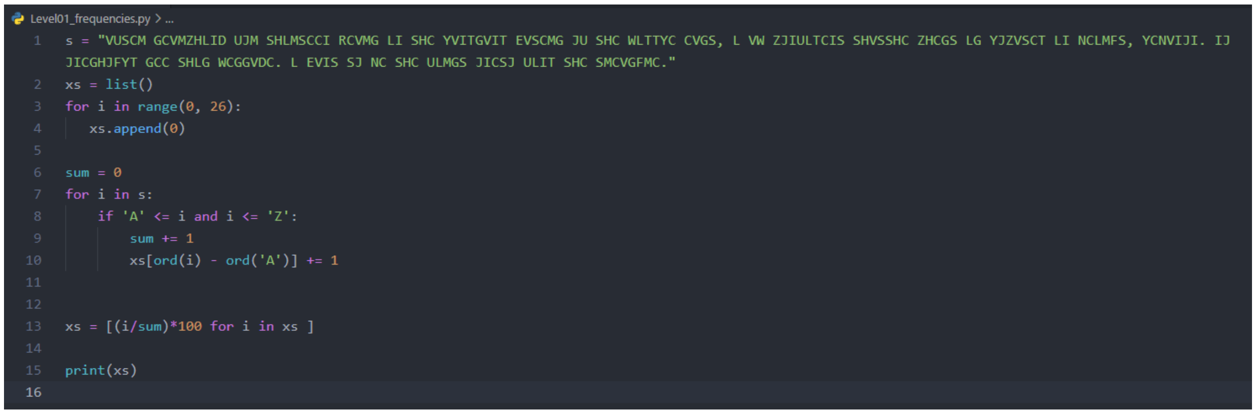
December 2023

## Table of Contents

## I. Level 1: Finding the Location of a Treasure Chest

1. The deciphered message is:
   "AFTER SEARCHING FOR THIRTEEN YEARS IN THE LAND SAND WATERS OF THE MIDDLE EAST, I AM CONFIDENT THAT THE CHEST IS LOCATED IN BEIRUT, LEBANON. NO ONE SHOULD SEE THIS MESSAGE. I WANT TO BE THE FIRST ONE TO FIND THE TREASURE."

   We first computed the frequency (by pasting the received letter from Captain Jack as input to the code snippet shown below) of each character in the encrypted message.



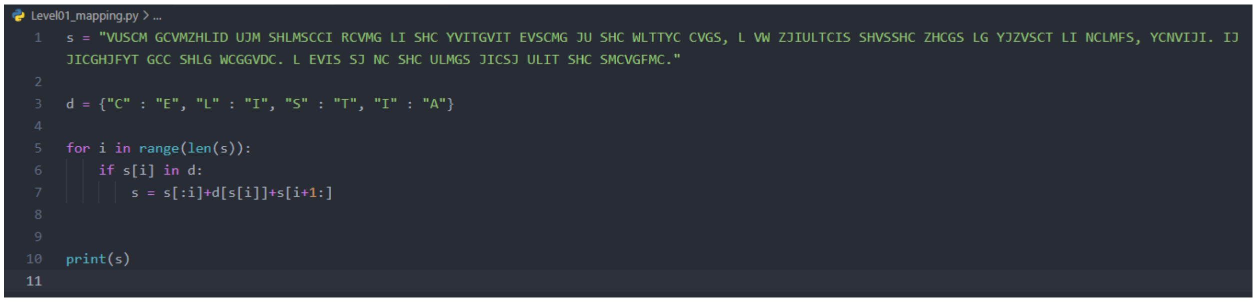
```
Level01_frequencies.py > ...
s = "VUSCM GCVMZHLID UJM SHLMSCCI RCVMG LI SHC YVITGVIT EVSCMG JU SHC WLTTYC CVGS, L VW ZJIULTCIS SHVSSHC ZHCGS LG YJZVSCT LI NCLMFS, YCNVIJI. IJ JICGHJFYT GCC SHLG WCGGVDC. L EVIS SJ NC SHC ULMGS JICSJ ULIT SHC SMCVGFMC."
xs = list()
for i in range(0, 26):
    xs.append(0)

sum = 0
for i in s:
    if 'A' <= i and i <= 'Z':
        sum += 1
        xs[ord(i) - ord('A')] += 1


xs = [(i/sum)*100 for i in xs ]

print(xs)
```

The results are shown in the table below. The second row shows the frequency of each character and the third row shows that frequency as a percentage rounded to the nearest ones (the sample is not large enough to predict accurate estimations reflecting the provided statistics, which is why we opted to round the percentages):

| 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 27 | 2 | 2 | 3 | 14 | 11 | 15 | 11 | 0 | 13 | 10 | 3 | 0 | 0 | 0 | 1 | 22 | 8 | 6 | 14 | 3 | 0 | 5 | 4 |
| 0 | 0 | 16 | 1 | 1 | 2 | 8 | 6 | 9 | 6 | 0 | 7 | 6 | 2 | 0 | 0 | 0 | 1 | 13 | 5 | 3 | 8 | 2 | 0 | 3 | 2 |

We compared the above frequencies with the ones provided in the bar chart provided in the lab instructions.

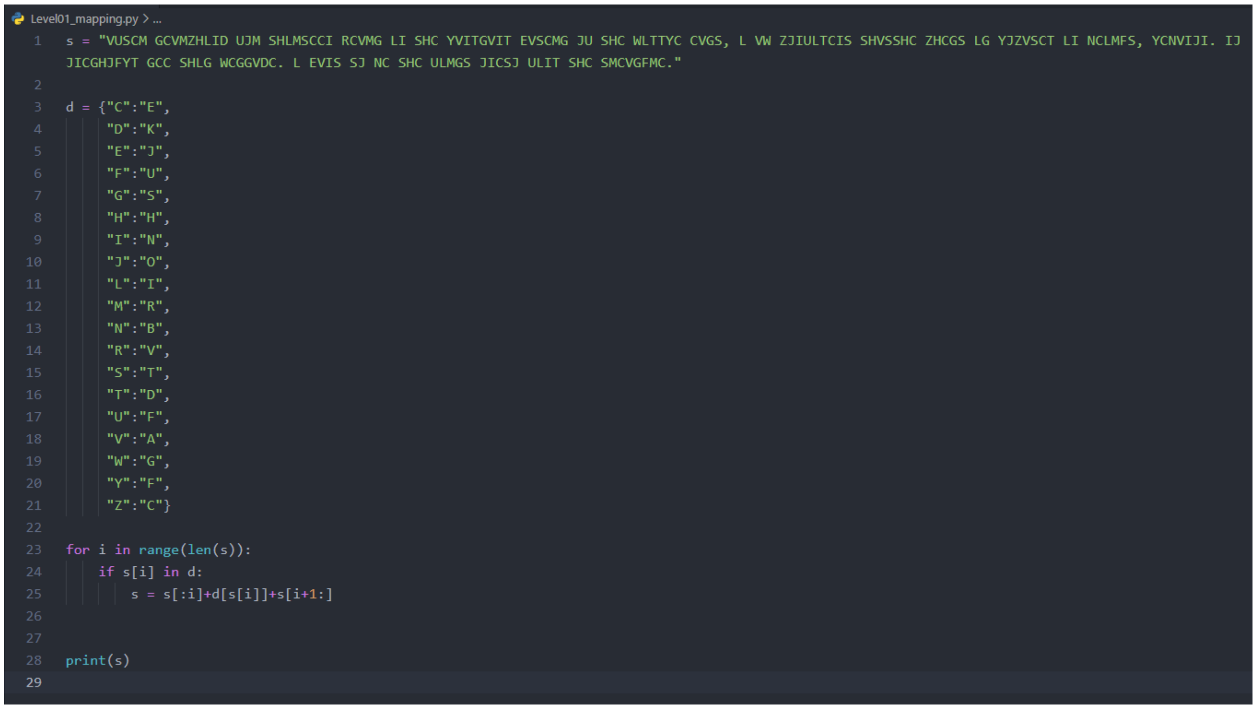
We first opted to match the ones with the highest frequencies and the most occurring words that are less than 3 letters long in an attempt to clarify parts of the message.
The resulting partial deciphered message was: "VUTEM GEVMZHIAD UJM THIMTEEA REVMG IA THE YVATGVAT EVTEMG JU THE WITTYE EVGT, I VW ZJAUITEAT THVTTHE ZHEGT IG YJZVTET IA NEIMFT, YENVAJA. AJ JAEGHJFYT GEE THIG WEGGVDE. I EVAT TJ NE THE UIMGT JAETJ UIAT THE TMEVGFME."



Level01_mapping.py > ...

```python
s = "VUSCM GCVMZHLID UJM SHLMSCCI RCVMG LI SHC YVITGVIT EVSCMG JU SHC WLTTYC CVGS, L VW ZJIULTCIS SHVSSHC ZHCGS LG YJZVSCT LI NCLMFS, YCNVIJI. IJ JICGHJFYT GCC SHLG WCGGVDC. L EVIS SJ NC SHC ULMGS JICSJ ULIT SHC SMCVGFMC."

d = {"C" : "E", "L" : "I", "S" : "T", "I" : "A"}

for i in range(len(s)):
    if s[i] in d:
        s = s[:i]+d[s[i]]+s[i+1:]


print(s)
```

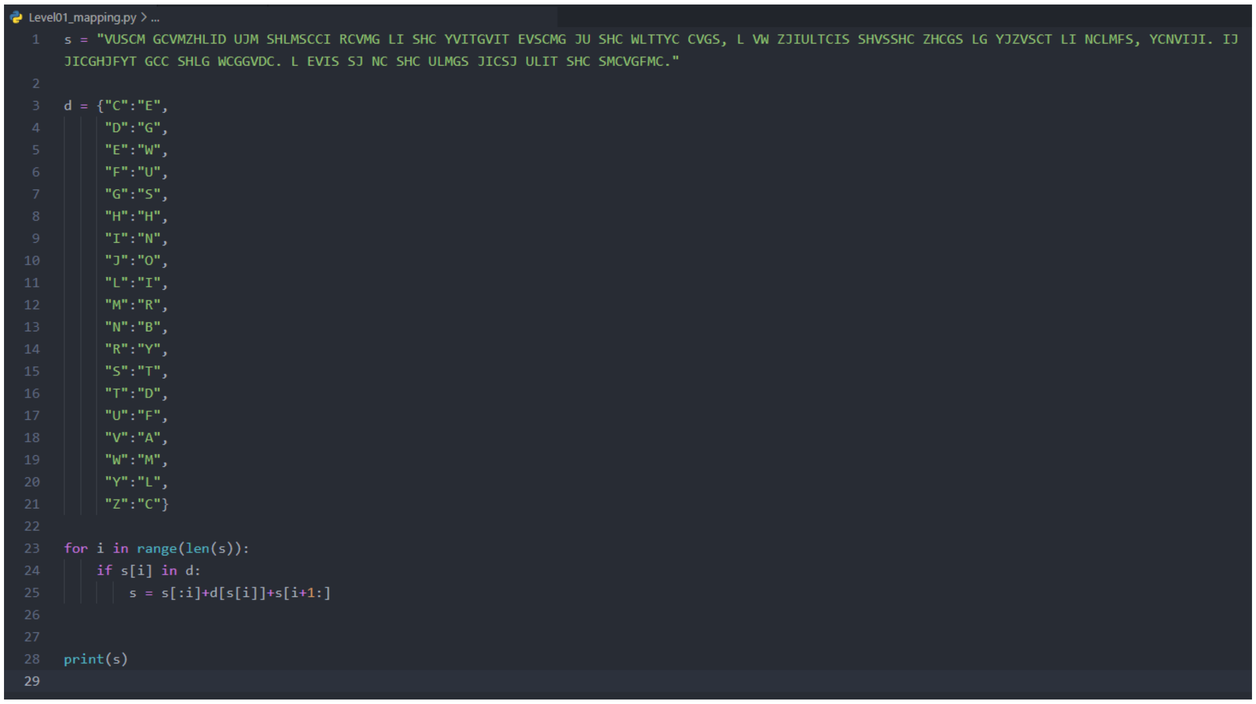
It was then a matter of trial and error as we attempted matching different letters to different combinations until the following statement was output: "AFTER SEARCHINK FOR THIRTEEN VEARS IN THE FANDSAND JATERS OF THE GIDDFE EAST, I AG CONFIDENT THATTHE CHEST IS FOCATED IN BEIRUT, FEBANON. NO ONESHOUFD SEE THIS GESSAKE. I JANT TO BE THE FIRST ONETO FIND THE TREASURE."



Level01_mapping.py > ...

```python
s = "VUSCM GCVMZHLID UJM SHLMSCCI RCVMG LI SHC YVITGVIT EVSCMG JU SHC WLTTYC CVGS, L VW ZJIULTCIS SHVSSHC ZHCGS LG YJZVSCT LI NCLMFS, YCNVIJI. IJ JICGHJFYT GCC SHLG WCGGVDC. L EVIS SJ NC SHC ULMGS JICSJ ULIT SHC SMCVGFMC."

d = {"C":"E",
     "D":"K",
     "E":"J",
     "F":"U",
     "G":"S",
     "H":"H",
     "I":"N",
     "J":"O",
     "L":"I",
     "M":"R",
     "N":"B",
     "R":"V",
     "S":"T",
     "T":"D",
     "U":"F",
     "V":"A",
     "W":"G",
     "Y":"F",
     "Z":"C"}

for i in range(len(s)):
    if s[i] in d:
        s = s[:i]+d[s[i]]+s[i+1:]


print(s)
```

After that point, we were able to predict the remaining letters as follows:



Level01_mapping.py > ...

```python
s = "VUSCM GCVMZHLID UJM SHLMSCCI RCVMG LI SHC YVITGVIT EVSCMG JU SHC WLTTYC CVGS, L VW ZJIULTCIS SHVSSHC ZHCGS LG YJZVSCT LI NCLMFS, YCNVIJI. IJ JICGHJFYT GCC SHLG WCGGVDC. L EVIS SJ NC SHC ULMGS JICSJ ULIT SHC SMCVGFMC."

d = {"C":"E",
     "D":"G",
     "E":"W",
     "F":"U",
     "G":"S",
     "H":"H",
     "I":"N",
     "J":"O",
     "L":"I",
     "M":"R",
     "N":"B",
     "R":"Y",
     "S":"T",
     "T":"D",
     "U":"F",
     "V":"A",
     "W":"M",
     "Y":"L",
     "Z":"C"}

for i in range(len(s)):
    if s[i] in d:
        s = s[:i]+d[s[i]]+s[i+1:]


print(s)
```

The final result displays the complete deciphered message, revealing the location of the chest.

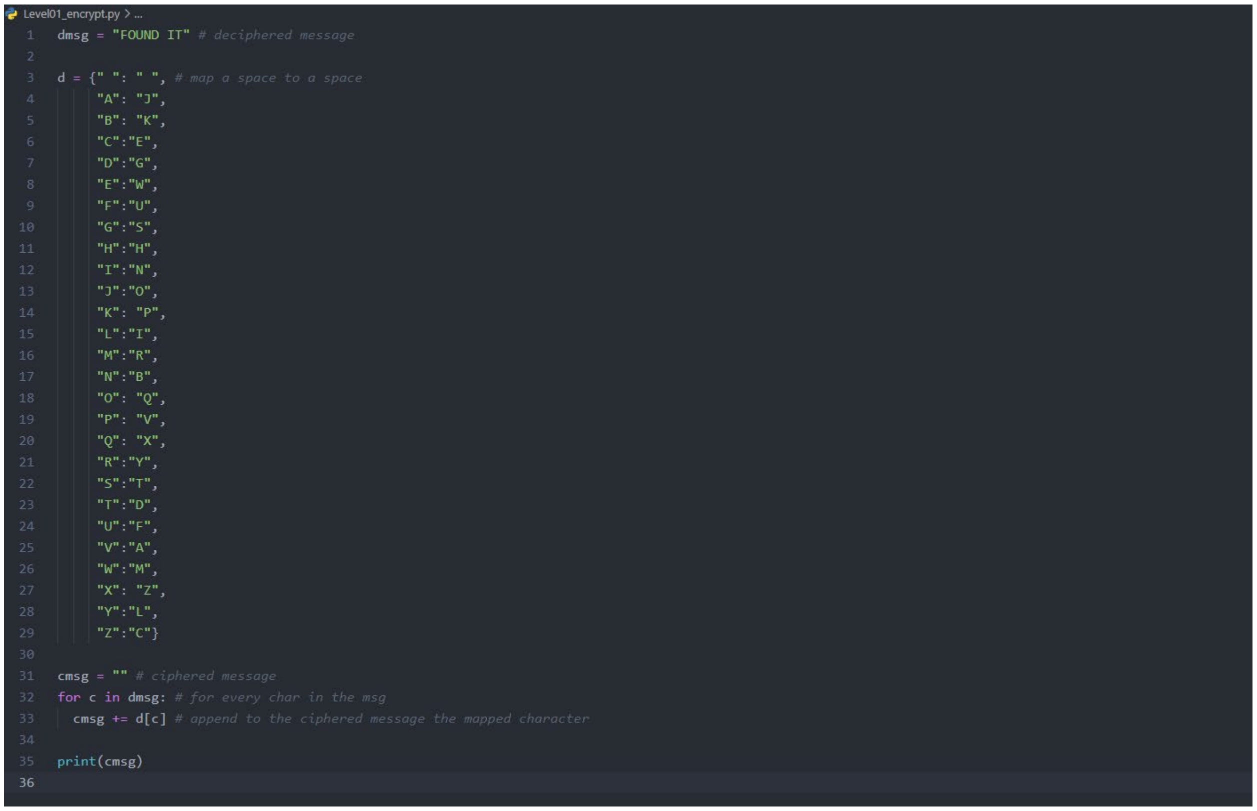
2. The cryptographic technique used is monoalphabetic cipher.

   The following table maps each character in the ciphered message (first row) to its character in the deciphered message (second row). Characters 'A', 'B', 'K', 'O', 'P', 'Q', and 'X' were randomly assigned different mappings as they did not appear in the ciphered message.

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |
| J | K | E | G | W | U | S | H | N | O | P | I | R | B | Q | V | X | Y | T | D | F | A | M | Z | L | C |

3. The message, after applying the cipher technique, sent to Captain Jack will be: "UQFBG ND".

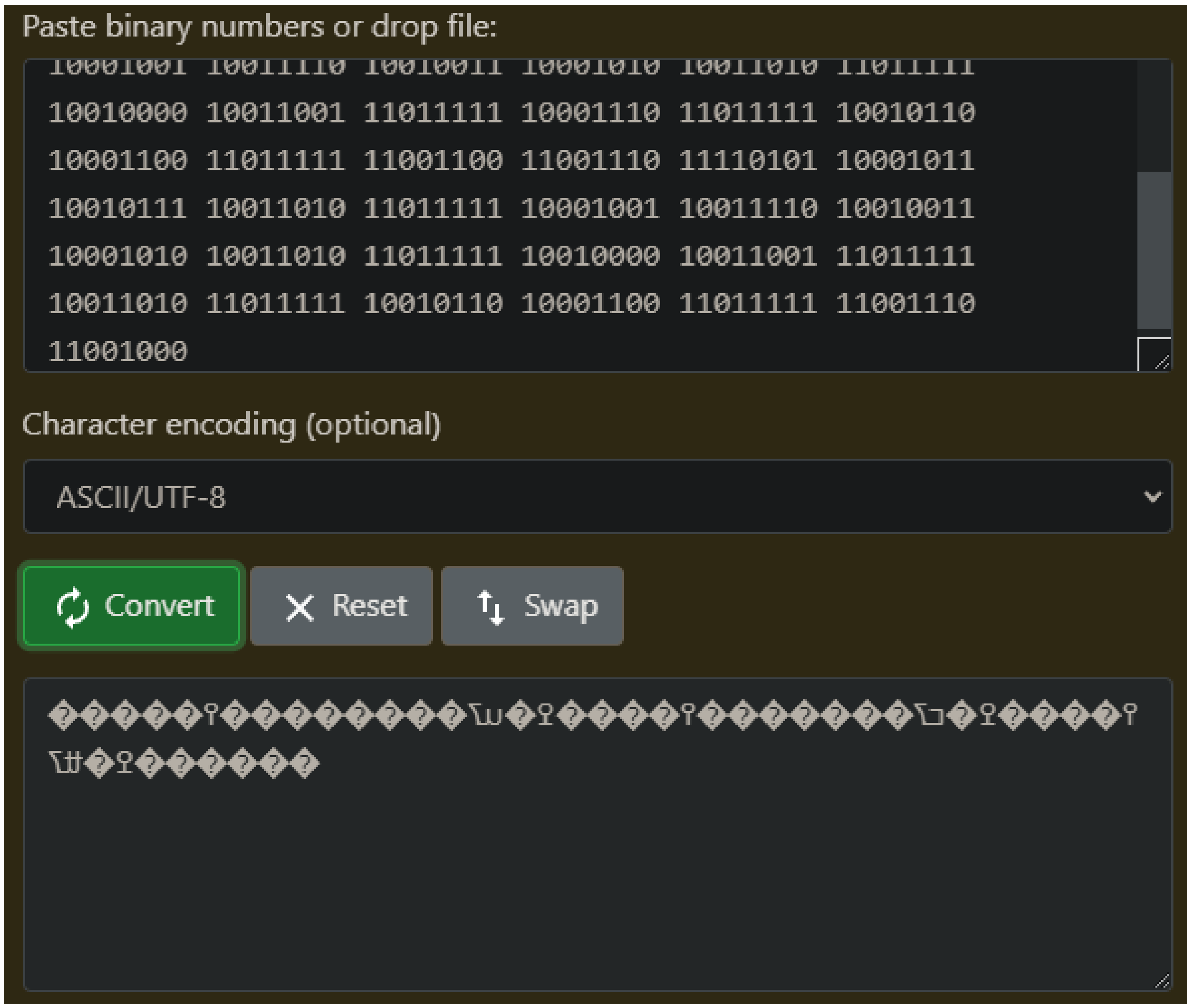
   It was computed via the below script:


Level01_encrypt.py > ...

```python
dmsg = "FOUND IT" # deciphered message

d = {" ": " ", # map a space to a space
     "A": "J",
     "B": "K",
     "C":"E",
     "D":"G",
     "E":"W",
     "F":"U",
     "G":"S",
     "H":"H",
     "I":"N",
     "J":"O",
     "K": "P",
     "L":"I",
     "M":"R",
     "N":"B",
     "O": "Q",
     "P": "V",
     "Q": "X",
     "R":"Y",
     "S":"T",
     "T":"D",
     "U":"F",
     "V":"A",
     "W":"M",
     "X": "Z",
     "Y":"L",
     "Z":"C"}

cmsg = "" # ciphered message
for c in dmsg: # for every char in the msg
  cmsg += d[c] # append to the ciphered message the mapped character

print(cmsg)
```

## II. Level 2: Removing a Ransomware

1. Despite our rather humble, and limited, knowledge in regards to computer security, we would not pay the $1000 ransom. Besides the fact that our to-do-list is not worth $1000 to begin with, the attacker doesn't have any access to the information within the text file. There is also no guarantee that the attacker will indeed provide the key to decrypt the file once the payment is settled.
   RSA is a rather challenging technique to decrypt without knowing the key, but it is possible that, with enough research, we may be able to crack it.
2. The file contains groups of 8-bit bundles. Upon attempting to convert the given binary to various different encodings using an online converter, the following outputs were obtained:

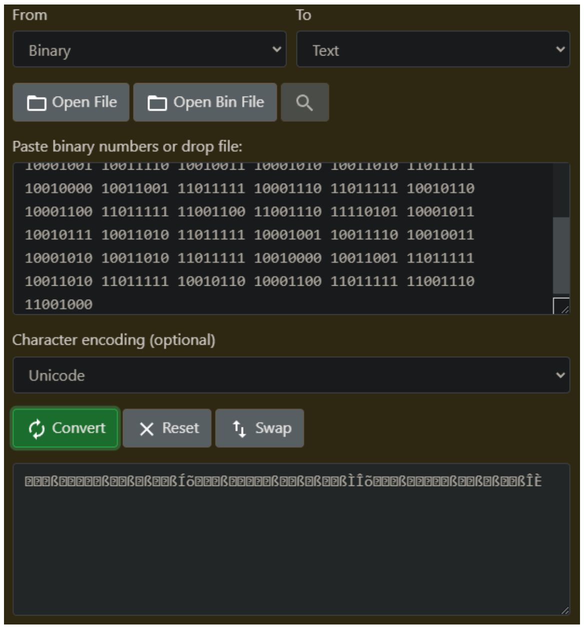

Paste binary numbers or drop file:

```
10001001 10011110 10010011 10001010 10011010 11011111
10010000 10011001 11011111 10001110 11011111 10010110
10001100 11011111 11001100 11001110 11110101 10001011
10010111 10011010 11011111 10001001 10011110 10010011
10001010 10011010 11011111 10010000 10011001 11011111
10011010 11011111 10010110 10001100 11011111 11001110
11001000
```

Character encoding (optional)

ASCII/UTF-8

```
�����[illegible]��������[illegible]�[illegible]�����[illegible]��������[illegible]�[illegible]�����[illegible]
[illegible]�[illegible]�������
```



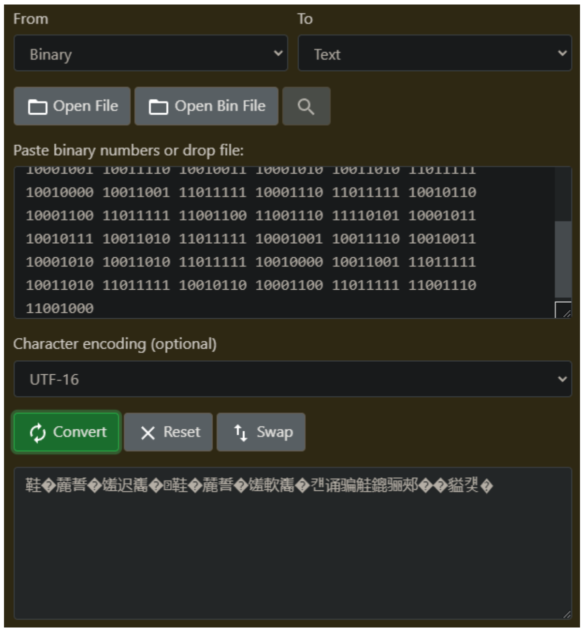
From: Binary
To: Text

Paste binary numbers or drop file:

```
10001001 10011110 10010011 10001010 10011010 11011111
10010000 10011001 11011111 10001110 11011111 10010110
10001100 11011111 11001100 11001110 11110101 10001011
10010111 10011010 11011111 10001001 10011110 10010011
10001010 10011010 11011111 10010000 10011001 11011111
10011010 11011111 10010110 10001100 11011111 11001110
11001000
```

Character encoding (optional): Unicode

```
□□□ß□□□□□□ß□□□ß□ß□□□ßÍõ□□□ß□□□□□□ß□□□ß□ß□□□ßÌÎõ□□□ß□□□□□□ß□□□ß□ß□□□ßÎÈ
```



From: Binary
To: Text

Paste binary numbers or drop file:

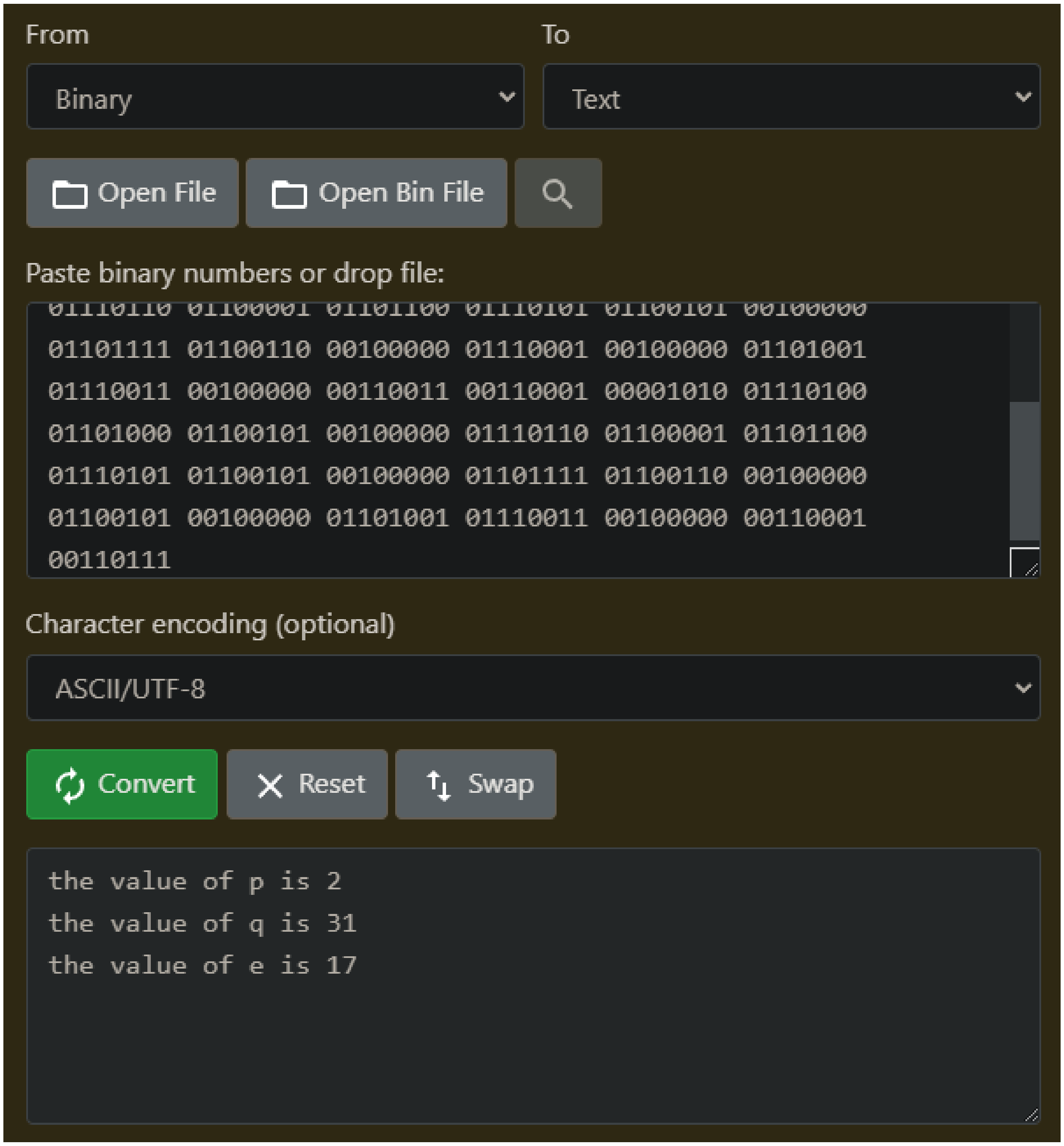
```
10001001 10011110 10010011 10001010 10011010 11011111
10010000 10011001 11011111 10001110 11011111 10010110
10001100 11011111 11001100 11001110 11110101 10001011
10010111 10011010 11011111 10001001 10011110 10010011
10001010 10011010 11011111 10010000 10011001 11011111
10011010 11011111 10010110 10001100 11011111 11001110
11001000
```

Character encoding (optional): UTF-16

```
鞋�蘼誓�馐迟蔫�□鞋�蘼誓�馐軟蔫�갠诵骗鲑鑣骊郏��貖갶�
```

As the output was invalid, we considered different approaches. We opted to attempt to flip the bits before converting, following the name of the file “flip-the-bits”.

The following image shows the resulting output:

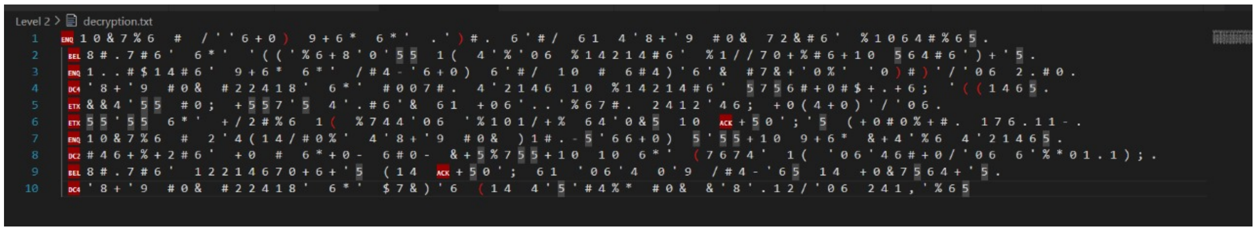


```
01110110 01100001 01101100 01110101 01100101 00100000
01101111 01100110 00100000 01110001 00100000 01101001
01110011 00100000 00110011 00110001 00001010 01110100
01101000 01100101 00100000 01110110 01100001 01101100
01110101 01100101 00100000 01101111 01100110 00100000
01100101 00100000 01101001 01110011 00100000 00110001
00110111
```

```
the value of p is 2
the value of q is 31
the value of e is 17
```

The contents of the file appear to facilitate the decryption process as it provides the values for *p*, *q*, and *e*. These can be used to also compute *n*, *z*, and *d* which will be used in the decryption process of RSA.

NOTE that due to the unexpected format of the output that follows in part 3 of this level (shown in the next screenshot), we have contacted Mr. Akram Tabbaa to confirm the output.



```
Level 2 > decryption.txt
ENQ 1 0 & 7 % 6   #   / ' ' 6 + 0 )   9 + 6 *   6 * '   . ' ) # .   6 ' # /   6 1   4 ' 8 + ' 9   # 0 &   7 2 & # 6 '   % 1 0 6 4 # % 6 5 .
BEL 8 # . 7 # 6 '   6 * '   ' ( ( ' % 6 + 8 ' 0 ' 5 5   1 (   4 ' % ' 0 6   % 1 4 2 1 4 # 6 '   % 1 / / 7 0 + % # 6 + 1 0   5 6 4 # 6 ' ) + ' 5 .
ENQ 1 . . # $ 1 4 # 6 '   9 + 6 *   6 * '   / # 4 - ' 6 + 0 )   6 ' # /   1 0   #   6 # 4 ) ' 6 ' &   # 7 & + ' 0 % '   ' 0 ) # ) ' / ' 0 6   2 . # 0 .
DC4 ' 8 + ' 9   # 0 &   # 2 2 4 1 8 '   6 * '   # 0 0 7 # .   4 ' 2 1 4 6   1 0   % 1 4 2 1 4 # 6 '   5 7 5 6 # + 0 # $ + . + 6 ;   ' ( ( 1 4 6 5 .
ETX & & 4 ' 5 5   # 0 ;   + 5 5 7 ' 5   4 ' . # 6 ' &   6 1   + 0 6 ' . . ' % 6 7 # .   2 4 1 2 ' 4 6 ;   + 0 ( 4 + 0 ) ' / ' 0 6 .
ETX 5 5 ' 5 5   6 * '   + / 2 # % 6   1 (   % 7 4 4 ' 0 6   ' % 1 0 1 / + %   6 4 ' 0 & 5   1 0   ACK + 5 0 ' ; ' 5   ( + 0 # 0 % + # .   1 7 6 . 1 1 - .
ENQ 1 0 & 7 % 6   #   2 ' 4 ( 1 4 / # 0 % '   4 ' 8 + ' 9   # 0 &   ) 1 # . - 5 ' 6 6 + 0 )   5 ' 5 5 + 1 0   9 + 6 *   & + 4 ' % 6   4 ' 2 1 4 6 5 .
DC2 # 4 6 + % + 2 # 6 '   + 0   #   6 * + 0 -   6 # 0 -   & + 5 % 7 5 5 + 1 0   1 0   6 * '   ( 7 6 7 4 '   1 (   ' 0 6 ' 4 6 # + 0 / ' 0 6   6 ' % * 0 1 . 1 ) ; .
BEL 8 # . 7 # 6 '   1 2 2 1 4 6 7 0 + 6 + ' 5   ( 1 4   ACK + 5 0 ' ;   6 1   ' 0 6 ' 4   0 ' 9   / # 4 - ' 6 5   1 4   + 0 & 7 5 6 4 + ' 5 .
DC4 ' 8 + ' 9   # 0 &   # 2 2 4 1 8 '   6 * '   $ 7 & ) ' 6   ( 1 4   4 ' 5 ' # 4 % *   # 0 &   & ' 8 ' . 1 2 / ' 0 6   2 4 1 , ' % 6 5
```

We, then, repeated the previous steps to decrypt the new set of files that were provided for this level.

The newly found values of *p*, *q*, and *e* are *13*, *17*, and *5* respectively.

The decryption key *d* is then computed to be *77*.

Following is a brief overview of the formulas referenced:

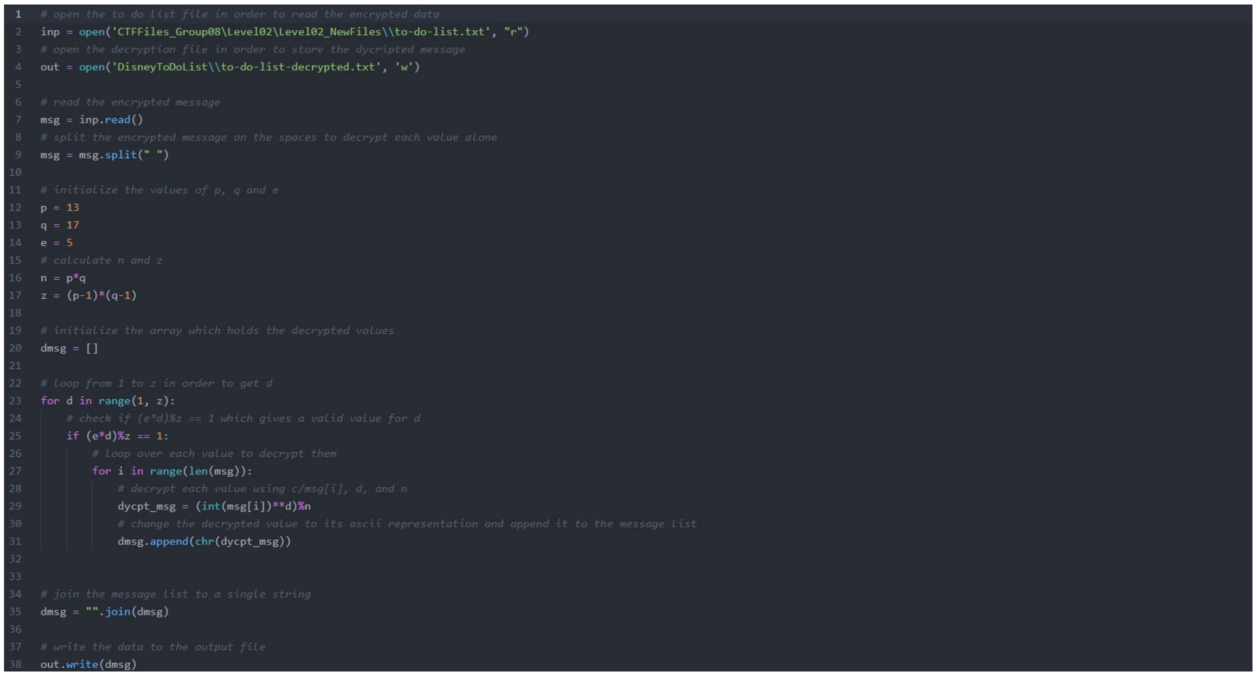
*n = p.q*

*z = (p-1).(q-1)*

*d is calculated such that ed%z == 1*

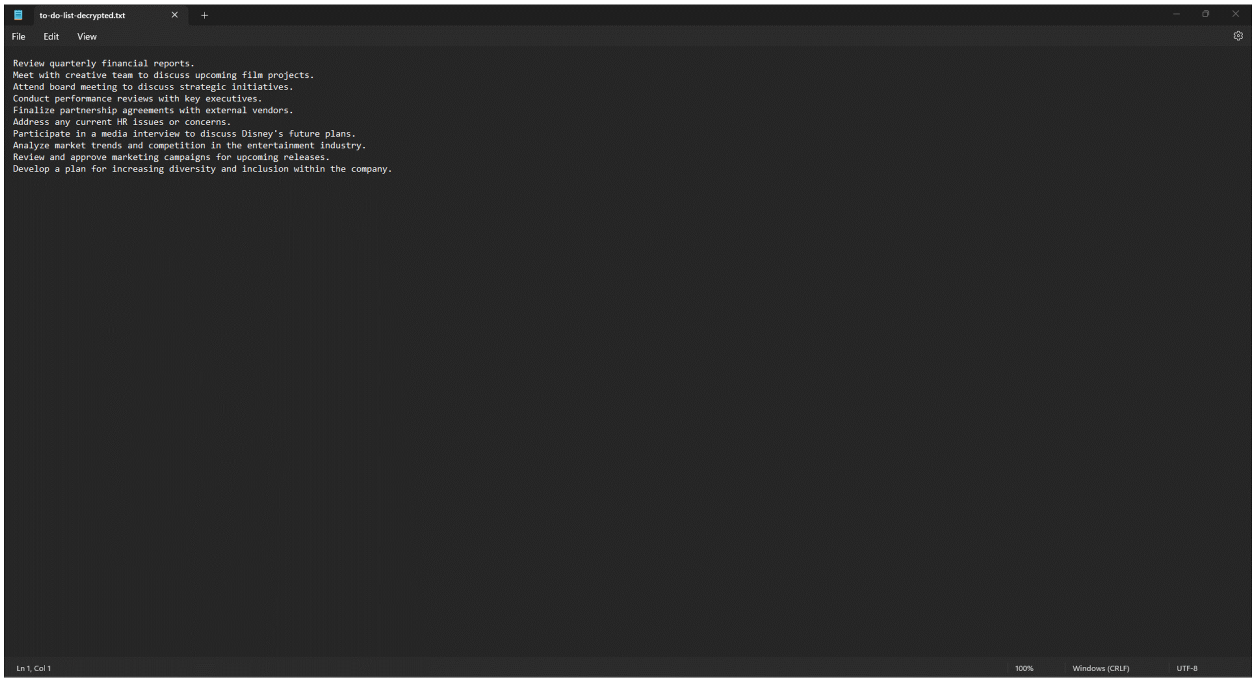
NOTE that the codes provided in the submitted *zip* file reflect those of the new set of files rather than the ones of invalid output.

3. After decoding the *flip-the-bits.txt* file and computing the new values of *p, q, n, z, e,* and *d*, we were able to easily decrypt the *to-do-list.txt* file using the decryption key (*n, d*) via the python script provided in the submitted *zip*. Find a snapshot of the script below.



```python
# open the to do list file in order to read the encrypted data
inp = open('CTFFiles_Group08\Level02\Level02_NewFiles\\to-do-list.txt', "r")
# open the decryption file in order to store the dycripted message
out = open('DisneyToDoList\\to-do-list-decrypted.txt', 'w')

# read the encrypted message
msg = inp.read()
# split the encrypted message on the spaces to decrypt each value alone
msg = msg.split(" ")

# initialize the values of p, q and e
p = 13
q = 17
e = 5
# calculate n and z
n = p*q
z = (p-1)*(q-1)

# initialize the array which holds the decrypted values
dmsg = []

# loop from 1 to z in order to get d
for d in range(1, z):
    # check if (e*d)%z == 1 which gives a valid value for d
    if (e*d)%z == 1:
        # loop over each value to decrypt them
        for i in range(len(msg)):
            # decrypt each value using c/msg[i], d, and n
            dycpt_msg = (int(msg[i])**d)%n
            # change the decrypted value to its ascii representation and append it to the message list
            dmsg.append(chr(dycpt_msg))


# join the message list to a single string
dmsg = "".join(dmsg)

# write the data to the output file
out.write(dmsg)
```

The output is then saved in the *DisneyToDoList* directory as a *txt* file (shown in the next screenshot)



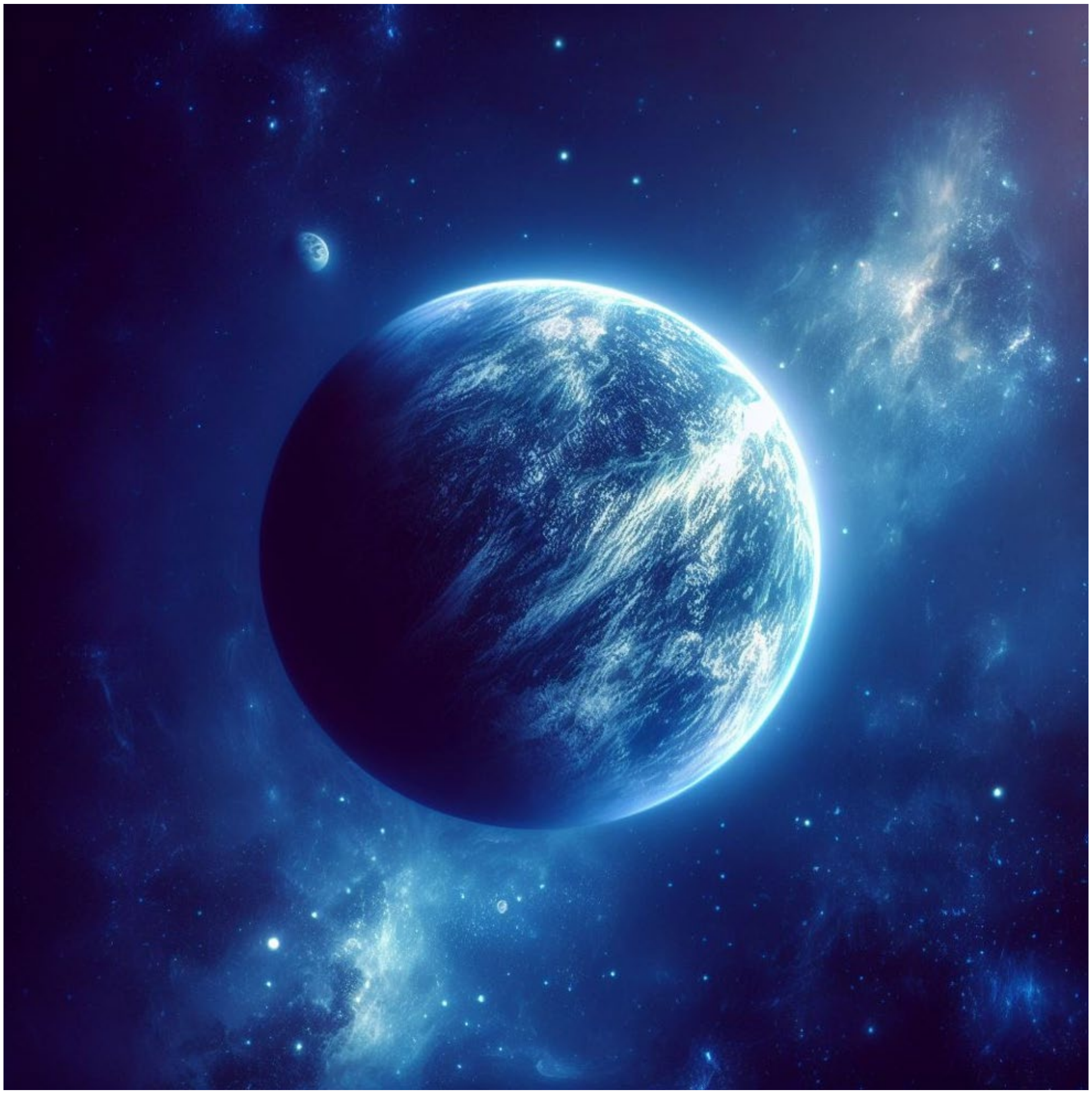
```
Review quarterly financial reports.
Meet with creative team to discuss upcoming film projects.
Attend board meeting to discuss strategic initiatives.
Conduct performance reviews with key executives.
Finalize partnership agreements with external vendors.
Address any current HR issues or concerns.
Participate in a media interview to discuss Disney's future plans.
Analyze market trends and competition in the entertainment industry.
Review and approve marketing campaigns for upcoming releases.
Develop a plan for increasing diversity and inclusion within the company.
```

## III. Level 3: Finding Planet X

Given that the encryption technique used is Ceasar Cipher, and that the image can handle a range of 0-255, there are 256 possible keys (i.e. the range of values from 0 to 255).

As the key is unknown, we attempted to brute force all 256 possible keys. For each key *k*, we substituted the current character (of size 1 byte) with whatever character that falls *k* steps after it, so we opted to skip *k=0* since the characters would basically stay the same.

The final decrypted image is shown below at the corresponding key of *k = 157*.



## IV. Conclusion

This lab was generally less-demanding than its predecessor, lab 3 (web attacks), so we reverted back to working on this lab together rather than distribute the work as individual tasks to be completed separately. We only struggled with the newly added challenge of submitting the lab report “fast enough” to gain points as our availability was very limited during the weekend. (It’s still a nice idea in theory though. Maybe it would be possible to organize an on-site CTF event next time.)

This lab, as all others, was really fun to work on. As this is the last lab of the semester, we’d like to acknowledge how fun and vital these labs were to complete the Computer Security course. Although the theoretical part is important, we can’t deny the added benefit of having a practical aspect to the course as well. Thank you both, Dr. Ayman and Mr. Akram for a fun semester.

Kindly refer to the other files in the submitted *zip* for the codes used to complete this lab.

## V. References

Binary to Text Converter: https://www.rapidtables.com/convert/number/binary-to-ascii.html