COMP 524 - Cybersecurity: Assignment #1

Due on September 19th, 2018

Dr. Michael Soltys

Rihan Stephen Pereira
email - rihanstephen.pereira576@myci.csuci.edu, studentID - 002497665
David Bryne
`email - something@myci.csuci.edu, studentID - 012345`
Marco Cabrera
email - marco.cabrera561@myci.csuci.edu, studentID - 002456715
Kaveh Arashvand
`email - something@myci.csuci.edu, studentID - 012345`

September 19, 2018

Problem 1

Decrypt the following four documents. To submit the assignment, write a two page PDF report, which explains, with code snippets, how you managed to decrypt the documents, and what the documents were.

- assignment-1-a.txt: Caesar cipher Base64 encoded text
- assignment-1-b.txt: MAC cipher Base64 encoded text
- assignment-1-c.txt: Caesar cipher Base64 encoded jpeg
- assignment-1-d.txt: MAC cipher Base64 encoded jpeg

Solution:
Caesar cipher Base64 encoded text
File - assignment-1-a.txt

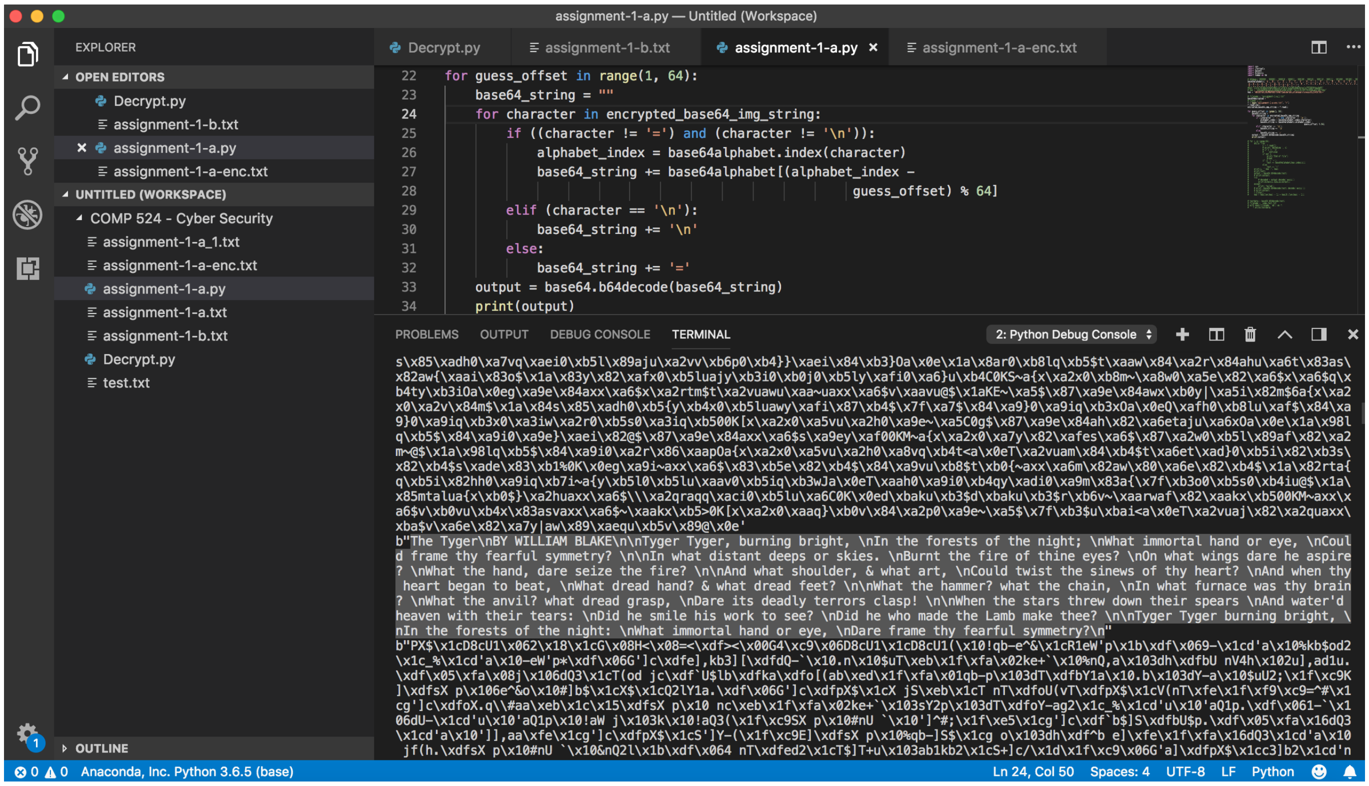
A brute-force approached was used to solve this problem. Considering we are dealing with Caesar cipher Base64 encoded and encrypted text, and since we know that there are only 64 possible combinations in Base64, we have enough information to decrypt the cipher text by brute-forcing it. As part of the brute-force approach, we had to decrypt and decode for every single key, therefore we decrypted and decoded 64 times. This same process was repeated over and over again until we were able to read the cipher text by shifting it 19 times. It was at the 19 shift where we finally decrypted and decoded the Base64 version of the file back to plain text.

Python code snippet we used to obtain plaintext:

```python
import base64

base64alphabet = ['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','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','1','2','3',
                  '4','5','6','7','8','9','+','/']

f = open("assignment-1-a-enc.txt", "r")
f.readline()
encrypted_base64_string = f.read()

for guess_offset in range(1, 64):
    base64_string = ""
    for character in encrypted_base64_string:
        if ((character != '=') and (character != '\n')):
            alphabet_index = base64alphabet.index(character)
            base64_string += base64alphabet[(alphabet_index -
                                             guess_offset) % 64]
        elif (character == '\n'):
            base64_string += '\n'
        else:
            base64_string += '='
    output = base64.b64decode(base64_string)
    print(output)
```

Problem 1 continued on next page. . .

The below images shows the plaintext that we obtained after running the above code fragment.



```python
for guess_offset in range(1, 64):
    base64_string = ""
    for character in encrypted_base64_img_string:
        if ((character != '=') and (character != '\n')):
            alphabet_index = base64alphabet.index(character)
            base64_string += base64alphabet[(alphabet_index -
                                             guess_offset) % 64]
        elif (character == '\n'):
            base64_string += '\n'
        else:
            base64_string += '='
    output = base64.b64decode(base64_string)
    print(output)
```

```
s\x85\xadh0\xa7vq\xaei0\xb5l\x89aju\xa2vv\xb6p0\xb4}}\xaei\x84\xb3}Oa\x0e\x1a\x8ar0\xb8lq\xb5$t\xaaw\x84\xa2r\x84ahu\xa6t\x83as\
x82aw{\xaai\x83o$\x1a\x83y\x82\xafx0\xb5luajy\xb3i0\xb0j0\xb5ly\xafi0\xa6}u\xb4C0KS~a{x\xa2x0\xb8m~\xa8w0\xa5e\x82\xa6$x\xa6$q\x
b4ty\xb3iOa\x0eg\xa9e\x84axx\xa6$x\xa2rtm$t\xa2vuawu\xaa~uaxx\xa6$v\xaavu@$\x1aKE~\xa5$\x87\xa9e\x84awx\xb0y|\xa5i\x82m$6a{x\xa2
x0\xa2v\x84m$\x1a\x84s\x85\xadh0\xb5{y\xb4x0\xb5luawy\xafi\x87\xb4$\x7f\xa7$\x84\xa9}0\xa9iq\xb3xOa\x0eQ\xafh0\xb8lu\xaf$\x84\xa
9}0\xa9iq\xb3x0\xa3iw\xa2r0\xb5s0\xa3iq\xb500K[x\xa2x0\xa5vu\xa2h0\xa9e~\xa5C0g$\x87\xa9e\x84ah\x82\xa6etaju\xa6xOa\x0e\x1a\x98l
q\xb5$\x84\xa9i0\xa9e}\xaei\x82@$\x87\xa9e\x84axx\xa6$s\xa9ey\xaf00KM~a{x\xa2x0\xa7y\x82\xafes\xa6$\x87\xa2w0\xb5l\x89af\x82\xa2
m~@$\x1a\x98lq\xb5$\x84\xa9i0\xa2r\x86\xaapOa{x\xa2x0\xa5vu\xa2h0\xa8vq\xb4t<a\x0eT\xa2vuam\x84\xb4$t\xa6et\xad}0\xb5i\x82\xb3s\
x82\xb4$s\xade\x83\xb1%0K\x0eg\xa9i~axx\xa6$\x83\xb5e\x82\xb4$\x84\xa9vu\xb8$t\xb0{~axx\xa6m\x82aw\x80\xa6e\x82\xb4$\x1a\x82rta{
q\xb5i\x82hh0\xa9iq\xb7i~a{y\xb5l0\xb5lu\xaav0\xb5iq\xb3wJa\x0eT\xaah0\xa9i0\xb4qy\xadi0\xa9m\x83a{\x7f\xb3o0\xb5s0\xb4iu@$\x1a\
x85mtalua{x\xb0$}\xa2huaxx\xa6$\\\xa2qraqq\xaci0\xb5lu\xa6C0K\x0ed\xbaku\xb3$d\xbaku\xb3$r\xb6v~\xaarwaf\x82\xaakx\xb500KM~axx\x
a6$v\xb0vu\xb4x\x83asvaxx\xa6$~\xaakx\xb5>0K[x\xa2x0\xaaq}\xb0v\x84\xa2p0\xa9e~\xa5$\x7f\xb3$u\xbai<a\x0eT\xa2vuaj\x82\xa2quaxx\
xba$v\xa6e\x82\xa7y|aw\x89\xaequ\xb5v\x89@\x0e'
b"The Tyger\nBY WILLIAM BLAKE\n\nTyger Tyger, burning bright, \nIn the forests of the night; \nWhat immortal hand or eye, \nCoul
d frame thy fearful symmetry? \n\nIn what distant deeps or skies. \nBurnt the fire of thine eyes? \nOn what wings dare he aspire
? \nWhat the hand, dare seize the fire? \n\nAnd what shoulder, & what art, \nCould twist the sinews of thy heart? \nAnd when thy
 heart began to beat, \nWhat dread hand? & what dread feet? \n\nWhat the hammer? what the chain, \nIn what furnace was thy brain
? \nWhat the anvil? what dread grasp, \nDare its deadly terrors clasp! \n\nWhen the stars threw down their spears \nAnd water'd
heaven with their tears: \nDid he smile his work to see? \nDid he who made the Lamb make thee? \n\nTyger Tyger burning bright, \
nIn the forests of the night: \nWhat immortal hand or eye, \nDare frame thy fearful symmetry?\n"
b"PX$\x1cD8cU1\x062\x18\x1cG\x08H<\x08=<\xdf><\x00G4\xc9\x06D8cU1\x1cD8cU1(\x10!qb-e^&\x1cR1eW'p\x1b\xdf\x069-\x1cd'a\x10%kb$od2
\x1c_%\x1cd'a\x10-eW'p*\xdf\x06G']c\xdfe],kb3][\xdfdQ-`\x10.n\x10$uT\xeb\x1f\xfa\x02ke+`\x10%nQ,a\x103dh\xdfbU nV4h\x102u],ad1u.
\xdf\x05\xfa\x08j\x106dQ3\x1cT(od jc\xdf`U$lb\xdfka\xdfo[(ab\xed\x1f\xfa\x01qb-p\x103dT\xdfbY1a\x10.b\x103dY-a\x10$uU2;\x1f\xc9K
]\xdfsX p\x106e^&o\x10#]b$\x1cX$\x1cQ2lY1a.\xdf\x06G']c\xdfpX$\x1cX jS\xeb\x1cT nT\xdfoU(vT\xdfpX$\x1cV(nT\xfe\x1f\xf9\xc9=^#\x1
cg']c\xdfoX.q\\#aa\xeb\x1c\x15\xdfsX p\x10 nc\xeb\x1f\xfa\x02ke+`\x103sY2p\x103dT\xdfoY-ag2\x1c_%\x1cd'u\x10'aQ1p.\xdf\x061-`\x1
06dU-\x1cd'u\x10'aQ1p\x10!aW j\x103k\x10!aQ3(\x1f\xc9SX p\x10#nU `\x10']^#;\x1f\xe5\x1cg']c\xdf`b$]S\xdfbU$p.\xdf\x05\xfa\x16dQ3
\x1cd'a\x10']],aa\xfe\x1cg']c\xdfpX$\x1cS']Y-(\x1f\xc9E]\xdfsX p\x10%qb-]S$\x1cg o\x103dh\xdf^b e]\xfe\x1f\xfa\x16dQ3\x1cd'a\x10
 jf(h.\xdfsX p\x10#nU `\x10&nQ2l\x1b\xdf\x064 nT\xdfed2\x1cT$]T+u\x103ab1kb2\x1cS+]c/\x1d\x1f\xc9\x06G'a]\xdfpX$\x1cc3]b2\x1cd'n
```

Figure 1: assignment-1-a.txt deciphered and decoded output

Problem 2

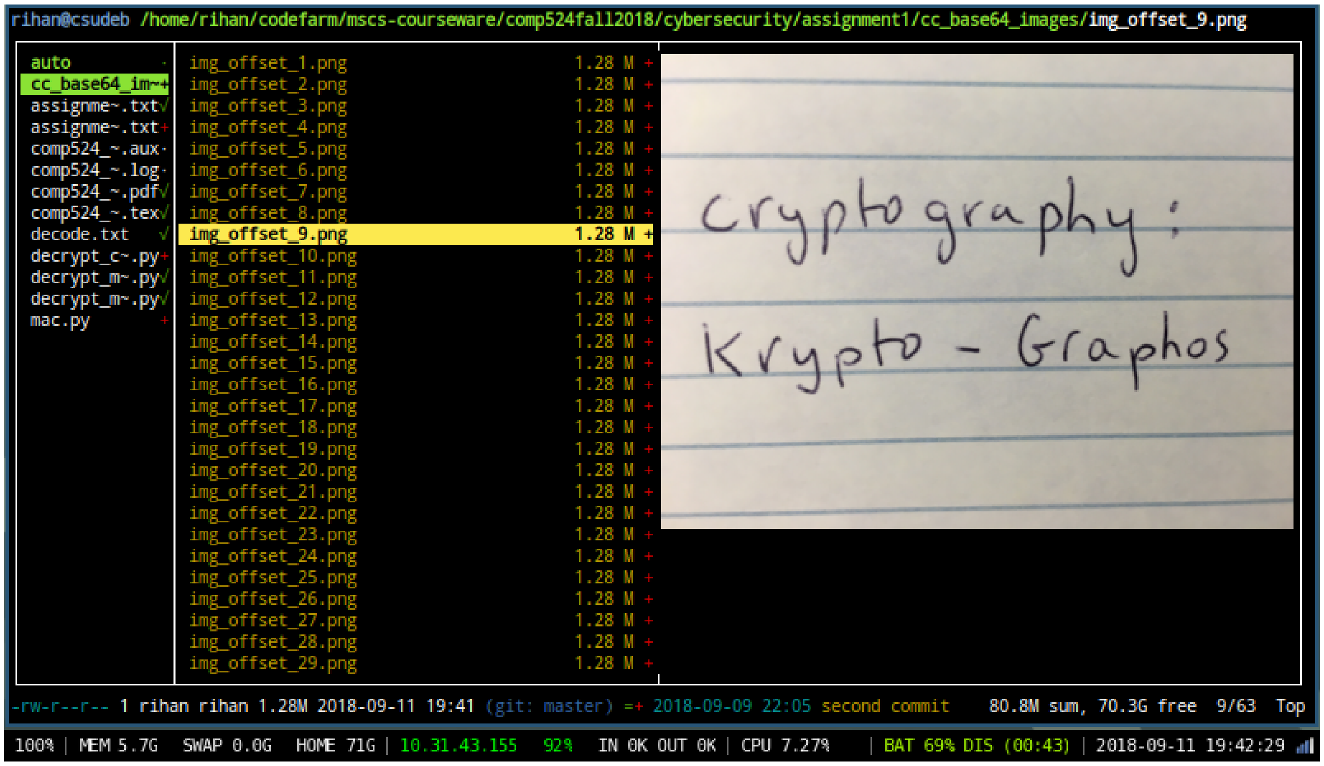
MAC cipher Base64 encoded text
File - assignment-1-b.txt

Problem 3

Caesar cipher Base64 encoded text
File - assignment-1-c.txt
explaination for 1c goes here

Python code snippet for decrypting and decoding Caesar Cipher base64 jpeg:

```
#! /usr/bin/python3

import mac
import os
import base64


def decrypt_cc_images():
    f = open("assignment-1-c-enc.txt", "r")
    f.readline()
    encrypted_base64_img_string = f.read()

    for guess_offset in range(1, 64):
        base64_string = ""
        for character in encrypted_base64_img_string:
            if ((character != '=') and (character != '\n')):
                alphabet_index = mac.base64alphabet.index(character)
                base64_string += mac.base64alphabet[(alphabet_index -
                                                      guess_offset) % 64]
            elif (character == '\n'):
                base64_string += '\n'
            else:
                base64_string += '='

        temp_filename = "img_offset_" + str(guess_offset) + ".png"

        print('file %s written to disk'.format(temp_filename))
        img2disk = open(os.path.join("cc_base64_images", temp_filename),
                        "wb")
        img2disk.write(base64.b64decode(base64_string))
        img2disk.close()


decrypt_cc_images()
```



```
rihan@csudeb /home/rihan/codefarm/mscs-courseware/comp524fall2018/cybersecurity/assignment1/cc_base64_images/img_offset_9.png

 auto          -  img_offset_1.png        1.28 M +
 cc_base64_im~+   img_offset_2.png        1.28 M +
 assignme~.txt√   img_offset_3.png        1.28 M +
 assignme~.txt+   img_offset_4.png        1.28 M +
 comp524_~.aux-   img_offset_5.png        1.28 M +
 comp524_~.log-   img_offset_6.png        1.28 M +
 comp524_~.pdf√   img_offset_7.png        1.28 M +
 comp524_~.tex√   img_offset_8.png        1.28 M +
 decode.txt    √  img_offset_9.png        1.28 M +
 decrypt_c~.py+   img_offset_10.png       1.28 M +
 decrypt_m~.py√   img_offset_11.png       1.28 M +
 decrypt_m~.py√   img_offset_12.png       1.28 M +
 mac.py        +  img_offset_13.png       1.28 M +
                  img_offset_14.png       1.28 M +
                  img_offset_15.png       1.28 M +
                  img_offset_16.png       1.28 M +
                  img_offset_17.png       1.28 M +
                  img_offset_18.png       1.28 M +
                  img_offset_19.png       1.28 M +
                  img_offset_20.png       1.28 M +
                  img_offset_21.png       1.28 M +
                  img_offset_22.png       1.28 M +
                  img_offset_23.png       1.28 M +
                  img_offset_24.png       1.28 M +
                  img_offset_25.png       1.28 M +
                  img_offset_26.png       1.28 M +
                  img_offset_27.png       1.28 M +
                  img_offset_28.png       1.28 M +
                  img_offset_29.png       1.28 M +
-rw-r--r-- 1 rihan rihan 1.28M 2018-09-11 19:41 (git: master) =+ 2018-09-09 22:05 second commit    80.8M sum, 70.3G free  9/63  Top
100% | MEM 5.7G  SWAP 0.0G  HOME 71G | 10.31.43.155  92%  IN 0K OUT 0K | CPU 7.27%   | BAT 69% DIS (00:43) | 2018-09-11 19:42:29
```

Figure 2: assignment-1-c.txt Caesar cipher decrypted and decoded image

Problem 4

MAC cipher Base64 encoded jpeg
File - assignment-1-d.txt

The approach I used to solve this problem was a known text attack. By inspecting the other image that was decoded in problem 3 I was able to determine the header and byte offset that was included when the image was taken from an iphone. Then after we base64 encoded that known image I was able to count the exact offset of the header in base64 characters. Once I had this offset decrypting the key was just a matter of comparing the character at the same offset in both the encrypted file and the know file. Once we had a mapping for every character we were able to write a python script that decrypted and decoded the file.
The tool I used to calculate the byte offset of the image header was 0xd

The key is: 2JzvLTdIARk3nyDg9s0NhxjGf18YQrq/ei6ZM+ObWCKotaclw7FE5BXHV4puUmPS

The python program solution_1d.py will output the decrypted image to a file. You can run this code: python solution_1d.py

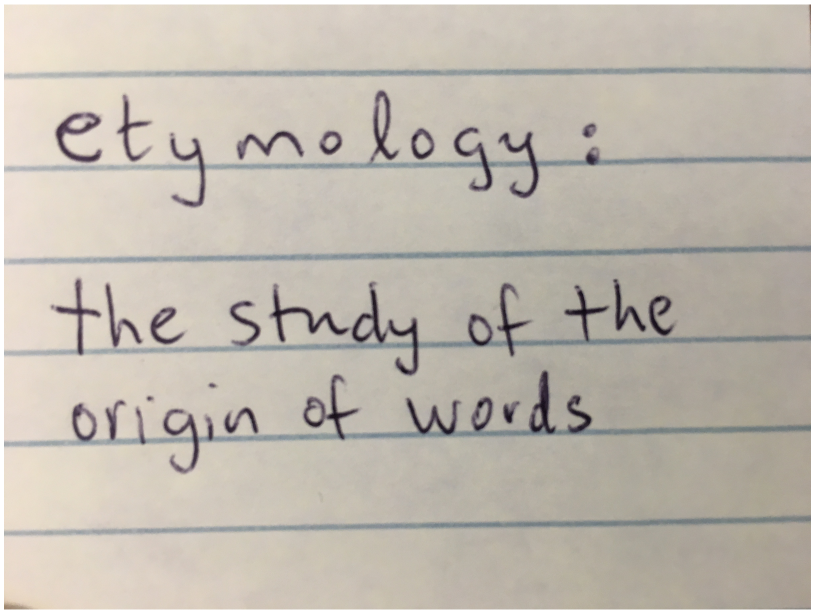
Here is the Python code snippet which uses above key to reveal the secret jpeg image.

```
import base64
base64alphabet = ['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','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','1','2','3',
```

Problem 4 continued on next page. . .

```
                    '4','5','6','7','8','9','+','/']

key="2JzvLTdIARK3nyDg9s0NhxjGf18YQrk/ei6ZM+ObWCqotaclw7FE5BXHV4puUmPS"
filename="assignment-1-d-enc-clean.txt"
image=""

with open(filename) as f:
  while True:
    c = f.read(1)
    print('decoding ',c)
    if not c:
      print("End of file")
      break
    if (c != '='):
      print(c,' ',key.index(c))
      image=image+base64alphabet[key.index(c)]
    else:
      image=image+"="

imgdata = base64.b64decode(image)
filename = 'solution_1d.jpg'  # I assume you have a way of picking
# unique filenames


with open(filename, 'wb') as f:
    f.write(imgdata)
```

The jpeg image secret that we found after breaking the ciphertext:



etymology:

the study of the
origin of words

Figure 3: assignment-1-d.txt MAC cipher decrypted and decoded jpeg