

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 - david.byrne@myci.csuci.edu

Marco Cabrera

email - marco.cabrera561@myci.csuci.edu, studentID - 002456715

Kaveh Arashvand

email - kaveh.arashvand075@myci.csuci.edu, studentID - 002377350

September 20, 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 approach was used to solve this problem. From one of the three basic classical attacks, we applied *Ciphertext only attack*, also called as *recognizable plaintext attack*. Considering we are dealing with Caesar Cipher(CC) base64 encrypted text, we were sure that its using one of the 63 possible keys from 64 character alphabet list to encrypt the plaintext. From Dr. soltys notes, we used decryption function

$$D_n(x) = (x + n)(mod64)$$

Also, we ignored keys $n = 0$ and $n = 64$, since the obtained ciphertext is same as plaintext. Each key was used to break the cipher followed by base64 decoding on the decrypted text to get the plaintext. On offset 19 i.e key $n = 19$, we got the plaintext. It is a poem from *William Blake*.

Python code snippet we used to obtain plaintext:

```
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 2

MAC cipher Base64 encoded text

File - [assignment-1-b.txt](#)

For this problem, we tried to divide the base 64 encrypted text into 4 character chunks. Because, we know that each 3 character plaintext transfer to 4 letter cipher text. The biggest issue was that because of this transformation the location of each letter in base64 was important. We had to find a frequency function from the base64 to the plaintext.

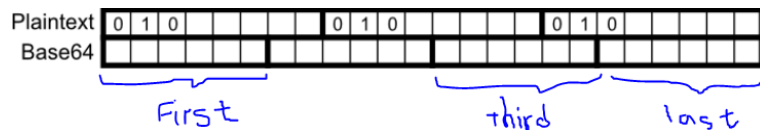


Figure 2: base 64 representation of 8 bit character

From our understanding of above figure, we created the below table which has the frequency of letters according to their locations in the ciphertext.

Now please consider table below: (TABLE OF FREQUENCY)

All Alphabets in cipher	First Location in cipher	Third location in cipher	Last location in cipher
M: 197 (important)	Y:141	V: 140	I: 115 (lower el)
Q: 152	U:128	Z:79	Q: 92
U: 149	I: 126 (capital i)	W:56	e:65
V: 144	S: 107	X:51	F: 58
Y:141	/: 104	L:51	O: 57 (capital oh)
I: 132 (capital i)	Z:88	Q:50	z:47
I:131 (lower el)	b:80	F:47	R:45
S:129	i: 36	d:41	x: 44
/:110	5: 14	T:38	6:43
p: 108	2:13	O:37 (ZERO)	A:40
F:105		s: 37	8:37
m:102		r: 33	P:34
Z: 93			D:30

Figure 3: Cipher frequency analysis

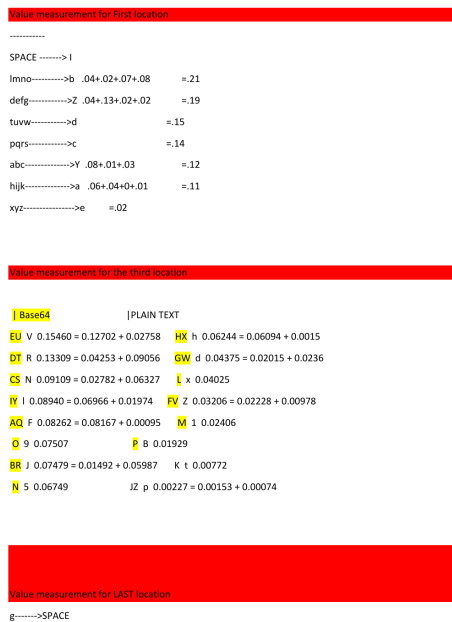


Figure 4: Frequency measurement value - Diagram 1

Problem 3

Caesar cipher Base64 encoded image

File - [assignment-1-c.txt](#)

To crack this secret, we used *recognizable plaintext attack*, also called *Ciphertext only* attack. We performed brute force on the ciphertext - scanning all the keys and for each key we used the Decryption function

$$D_n(x) = (x + n)(mod64)$$

where, x is the character from ciphertext and n is the key for that iteration. Also, note that the key $n = 0$ or $n = 64$ were not considered because it doesn't change the original plaintext. For each key, after running the decryption function, base64 decoding was applied to the text, and that base64 data was saved to disk as an image. Doing this, we saved altogether 63 images to the disk. Finally, we found key $n = 9$ was used to encrypt the plaintext which gave us the resulting image during decryption.

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

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

```
import mac
import os
import base64
```

```
def decrypt_cc_images():
```

l (el)	e	0.127
0 (Zero)	t	0.090
K	a	0.081
v	o	0.075
p	i	0.069
u	n	0.067
z	s	0.063
o (oh)	h	
y	r	

Figure 5: Frequency measurement value - Diagram 2

```

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()

```

```

PEate(n0z#(^X#(,3EAWj $! OucfaH
8A0iN is nsd<90>tf0Npe dIdg-$'A F0ow000EjLK# *Np0et*Un+eyHara nk# cop0id*Ay Npe lJáv0z ht0el eF0izg0e0a j0w cswHlK%p^? dejnde0
wovey tf0an+ep0aný f0ap
rsj^vjáte nvep0Npe m-)dHN *Any dczpEsfo lcsgr lFáthe eayHh *om0epwinyánuz#Dzsvate lIdg-^M
Thd<90>Bn0Wm0i1Kindá ncnuz no deoth
>0f0sgp the <ey00nmK00srghtlH
3Inuzote#<jpnXo> pia gan2BVk0o^Np Npe tHtle lFofk*g0Fbb["
ÁtheP0BmHil sn^?2That ho- gY)v0s h nuXw0zd<90>(no<98>
6iny that t1GBY-3d at xrk0 *wpcN hBdE r^jdI nccg0A#0lZghnpathio Z10^
8ank#00hd<90>avm^Xrn pk4v^ZwpcN cXl lHt1J0piXgY%noXfczn8dTht0E ti tHtl<90>ev0X il dtn0$D1 gvil in0zgtB0p^ *$Gxilec 0hvgldicg- pK%v^ZAlh^that a#xafejp jna 0Cl
3cnuM^Dennj0p0x<90>d0k#<ho^? dijtCnN), dt *Tpe fjäg)^Xx<90>zvm#swH#0hgxâtcXr^Zto an0aücNpeNrn fj!v0^M
3snj10Ec cXl in0hs- npts^dThe eslizs0en0Wm gY)v0s cgox0
Thd<90>hnm)t _n)^rk^Z#0ara<98>
0000jn0Zf^Ma nk# gZreo0
wd<90>(wwN#<Bf0ep0Npe( cXl adCin1H
9sti thiY sguNp0X cibp
wpezd<90>clisd lryin)!0ezY B0E$Thesr eXl hfizsB ti ýY4lJárcHThe#*Nb0sfth no^Gtljántiv0 ¶cs *Mngh ján+8nge#0tw), iNy Dnis
7topetiNsve eFnulF0S0WN#<Xho jna lIdD<90>onr lnsf^Ztn0an0ewphn))o cp0an0$¶uN no#0hd<90>(s):nr thex<90>ltczd<98>d&'ig))nj0UíA<83>il falE$0k##00hd<90>im#ep0!titn0X#<xzLK^Zc
$F0jg- nj0nyáthe mlp
rlisg No Npeip0av0znge dazú$The jédHNý {wwN k%v0z gt nBt(dThe ^<82>0iJ ^<82>p^? nj0coyl#0Xox0
3Xl Npe lNsv0stilm0 lNsp0ip0
Tc ^pE thiY om)^N cibp(g0Tpe eXzn0Ww)% no htne^M
:áB^? tE 0HtvjS#<xed<90>gpezd<90>gd<90>ap0^X
:0p^? ra l<90>houstec#<t10<98>
7hrjáb0s cop0id tf0Npe n0ZsBHTHT hav0 n0Dgr n0en0hopax<90>tp0ZlNc0
$The#<in0igW ^0linNk^N#0zowh
9(0nztst -g),lK,*<XhnuN
9l k4t lo jYwcd<90>ay twr tils*$THat {no ¶0Ark,rxáv}4te$0m4w^? DiadHileD
9l#0zve lFáthe k4p0il hfczw&$Fpz#0hd<90>ey-4p0cp0c is#0hd<90>clK%
>o ealH tN(na nk# ealH ncs
7zovej gpcN iW gand0t hove *)jn^? vk)v0z0CX j0v0$0Wm#0t m% j0v0c cXlK%0
$E-4l<98>Npe lN0Ezv0Nsv0e dab0$0k^No Npe gthrgal Xrfe
Thd<90>cRk,d<90>jl0wtey0 lN(d<98>dR0afcnrk^Z#0hfiz ^p0)nYávpg0^Wá<83>J0#<ilh'n0 wY^Bd<90>No Npe tIod<98>d%0
%j0 gtnem#zote ^p0 lA oxàtNzs^c
3sd hfjál0Znvep04y0 tC_g0To zgwBne thesr lN(y0X0UNzx<90>Zn^W0
wpi lCs#Y%lea0E thel<90>sn^?0
Who jna p0alH the denf0
Who jna ppenf fpz#0hd<90>cv`EX^Z$0j0^HâpcDd<90>iy a#0trge
Tt#00di the rIded Xre^X
Thd<90>z10jm0io Xre ra the m:nis
>o#0hd<90>lfk,vcX ^in0on0<8>e:pe:ptzfeN
3sd Npe lie no^CBthtp0N(0wpnld<90>cvlXdisf- f)40e the#<X_zú$Thep0 s- n0 bujp Npigs ay Npe Rtate$0k# k4 nsd<90>exilw0 nj0n0^M
8Bn+ep0alj0v~ n0 lHtrge
Tt#0hd<90>jstiz0s nz#0hd<90>alJ0lE^M
We ^<82>p^? lnd<90>tn0 nk4th0p0tp0cre3

'ffenselely Bn0ep0Npe n0ZsBHTfuzd^rtp00 is#<Nwptp0Xrel0Zy0^?<90> dtw0ed ev0Z(wH)% ^t0bsrn 0nimHl no jédHN
6Xowh nBt gpezgvep0Npe %<it^Mxghnk^Zd<90>Npeip0(gwlingeP0$¶cl^H0 gt0pt0Ec j0pE the0e0f0M0ppl csd tf0cwwN ^BfjängB0)%d cx<90>Npe 0C(g0)fgatitn0aný delücip0dPHtw an0af0ip00nyáfu0áne3
~
~
~

```

Figure 6: MAC Cipher partially decrypted base64 data

```
decrypt_cc_images()
```

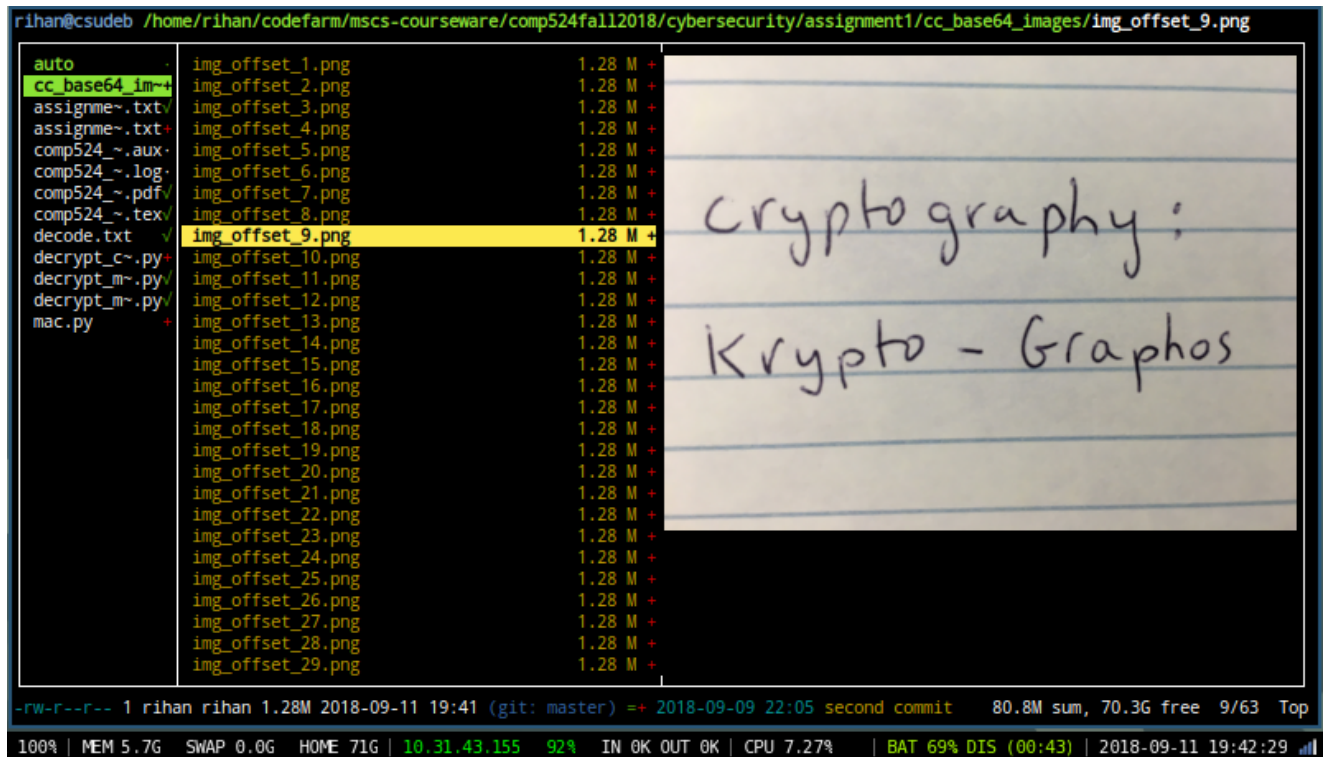


Figure 7: assignment-1-c.txt Caesar cipher decrypted and decoded image

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+ObWCKotacldw7FE5BXHV4puUmPS

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',
```



```
'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:

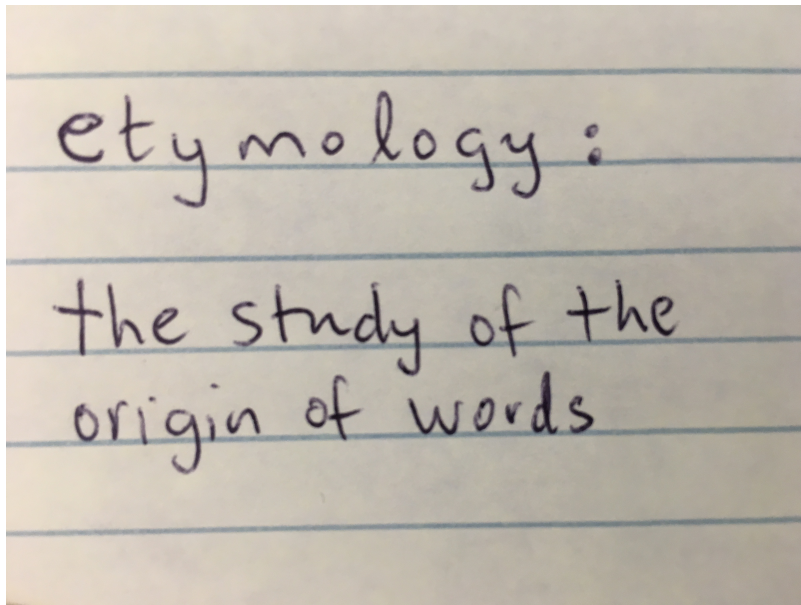


Figure 8: assignment-1-d.txt MAC cipher decrypted and decoded jpeg