Exercise 1

a) Encrypt the first eight characters of PLAINTEXT

```
PLAINTEXT = generate_plaintext('045')
input = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
output = "ACEGIKMOQSUWYBDFHJLNPRTVXZ"

print(PLAINTEXT[:8])
transTable = str.maketrans(input,output)
cipherText = PLAINTEXT.translate(transTable)
print(cipherText)
```

b) Decipher CIPHERTEXT_1B

```
print_frequency_order(CIPHERTEXT_1B)
#print(decrypt_sub('Z___V______________G______',CIPHERTEXT_1B))
#print(decrypt_sub('Z___V__S_________I_G______',CIPHERTEXT_1B))
#print(decrypt_sub('Z___V_TS______L__I_G______',CIPHERTEXT_1B))
#print(decrypt_sub('Z__WV_TS______L__IHGF_____',CIPHERTEXT_1B))
#print(decrypt_sub('ZYXWVUTSR____ML__IHGF_____',CIPHERTEXT_1B))
#print(decrypt_sub('ZYXWVUTSR__ONML__IHGF_____',CIPHERTEXT_1B))
print(decrypt_sub('ZYXWVUTSRQPONMLKJIHGFEDCBA',CIPHERTEXT_1B).capitalize())

OUTPUT:
Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to…
```

c) Decipher CIPHERTEXT_1C

```
#print(print_frequency_order(CIPHERTEXT_1C))
#print(decrypt_sub('M___C______________U______',CIPHERTEXT_1C))
#print(decrypt_sub('M___C__LK_____S____U______',CIPHERTEXT_1C))
#print(decrypt_sub('M___C__LK_____S__OIU__R___',CIPHERTEXT_1C))
#print(decrypt_sub('M__VCX_LK____DS__OIUY_R___',CIPHERTEXT_1C))
#print(decrypt_sub('MN_VCXZLK___FDS__OIUYTR___',CIPHERTEXT_1C))
print(decrypt_sub('MNBVCXZLKJHGFDSAPOIUYTREWQ',CIPHERTEXT_1C))

OUTPUT:
THERE WAS A TABLE SET OUT UNDER A TREE IN FRONT OF THE HOUSE, AND THE MARCH HARE AND THE HATTER WERE
HAVING TEA AT IT:…
```

d) Decipher CIPHERTEXT_1D

```
#print_coincidences_for_key_lengths(CIPHERTEXT_1D,12)
#print(find_cosets(CIPHERTEXT_1D,5))
#print_letter_frequency(find_cosets(CIPHERTEXT_1D,5)[0])
#print_letter_frequency(find_cosets(CIPHERTEXT_1D,5)[1])
#print_letter_frequency(find_cosets(CIPHERTEXT_1D,5)[2])
#print_letter_frequency(find_cosets(CIPHERTEXT_1D,5)[3])
#print_letter_frequency(find_cosets(CIPHERTEXT_1D,5)[4])
print(decrypt_vigenere('ALICE',CIPHERTEXT_1D).capitalize())
```

```
OUTPUT:
Thecaterpillarandalicelookedateachotherforsometimeinsilenceatlastthecaterpillar…
```

Exercise 2

a) Encrypt the first sixteen characters of PLAINTEXT using the Columnar Transposition

```
PLAINTEXT = generate_plaintext('045')
columns = print_tabulation(4,PLAINTEXT[:16])

#AEHR
#----
#EOZR
#ORFU
#IEFV
#EOZR

#EOIEOREOZFFZRUVR

print(decrypt_columnar_transposition('HARE','EOIEOREOZFFZRUVR'))

OUTPUT:
ZEROFOURFIVEZERO
```

b) Encrypt the ASCII binary encoding of the first two characters of PLAINTEXT

```
#Z = 0101 1010
#E=  0100 0101
#0 1 0 1  1 0 1  0  0 1  0  0  0  1  0  1 | Original
#1 2 3 4  5 6 7  8  9 10 11 12 13 14 15 16
#1 5 9 13 2 6 10 14 3 7  11 15 4  8  12 16
#0 1 0 0  1 0 1  1  0 1  0  0  1  0  0  1 | Encrypted

#original = 0101  1010  0100  0101
#encrypted= 0100  1011  0100  1001
```

c) Decipher CIPHERTEXT_2C

d) Decipher CIPHERTEXT_2D

Exercise 3

a) Encrypt the first two letters of PLAINTEXT, using just one round of the Heys cipher

```
Heys Round 1
STEP1: XOR Conversion
k     : A71C
PLAIN: Z         E
HEX  : 5    A    4    5
BINp : 0101 1010 0100 0101
BINk : 1010 0111 0001 1100
--------------------------
BINo : 1111 1101 0101 1001
HEXo : F    D    5    9



HEXo : F    D    5    9
SBOX : 7    9    F    A
BINo : 0111 1001 1111 1010



BINi: 0 1 1 1  1 0 0  1  1 1  1  1  1  0  1  0
INP : 1 2 3 4  5 6 7  8  9 10 11 12 13 14 15 16
OUT : 1 5 9 13 2 6 10 14 3 7  11 15 4  8  12 16
PBOX: 0 1 1 1  1 0 1  0  1 0  1  1  1  1  1  0
HEXo: 7ABE """

decrypt_heys_round('A71C','7ABE',True)

OUTPUT:
INPUT      0111 [7] | 1010 [A] | 1011 [B] | 1110 [E] |
AFTER-PBOX 0111 [7] | 1001 [9] | 1111 [F] | 1010 [A] |
AFTER-SBOX 1111 [F] | 1101 [D] | 0101 [5] | 1001 [9] |
AFTER-XOR  0101 [5] | 1010 [A] | 0100 [4] | 0101 [5] |
ZE       5A   45
```

b) Encrypt the first two letters of PLAINTEXT, using all four rounds of the Heys cipher

```
4 rounds of heys
STEP1: XOR Conversion

A71C 1
EA71 2
CEA7 3
1CEA 4
```

```
71CE 5

Round 1:
HEXp: 5A 45
HEXk: A71C
--------------------
HEXo: FD59
SBOX: 79FA
PBOX: 0 1 1 1  1 0 0  1  1 1  1  1  1  0  1  0
INP : 1 2 3 4  5 6 7  8  9 10 11 12 13 14 15 16
OUT : 1 5 9 13 2 6 10 14 3 7  11 15 4  8  12 16
BINo: 0 1 1 1  1 0 1  0  1 0  1  1  1  1  1  0


#xor_hex_hex('7ABE','EA71',prnt=True)

Round 2:
HEXp: 7ABE
HEXk: EA71
--------------------
HEXo: 90CF
SBOX: AE57
PBOX: 1 0 1 0  1 1 1  0  0 1  0  1  0  1  1  1
INP : 1 2 3 4  5 6 7  8  9 10 11 12 13 14 15 16
OUT : 1 5 9 13 2 6 10 14 3 7  11 15 4  8  12 16
BINo: 1 1 0 0  0 1 1  1  1 1  0  1  0  0  1  1

#xor_hex_hex('C7D3','CEA7',prnt=True)

xor_hex_hex('7ABE','EA71',prnt=True)
Round 3:
HEXp: C7D3
HEXk: CEA7
--------------------
HEXo: 0974
SBOX: EA82
PBOX: 1 1 1 0  1 0 1  0  1 0  0  0  0  0  1  0
INP : 1 2 3 4  5 6 7  8  9 10 11 12 13 14 15 16
OUT : 1 5 9 13 2 6 10 14 3 7  11 15 4  8  12 16
BINo: 1 1 1 0  1 0 0  0  1 1  0  1  0  0  0  0

#xor_hex_hex('E8D0','1CEA',prnt=True)

Round 4:
HEXp: E8D0
```

```
HEXk: 1CEA
--------------------
HEXo: F43A
SBOX: 7216

#xor_hex_hex('7216','71CE',prnt=True)

Round 5:
HEXp: 7216
HEXk: 71CE
--------------------
HEXo: 03D8
 """
CTEXT = '03D8'
bloc1 = decrypt_heys_final_round('1CEA', '71CE', CTEXT[0:4],True)
bloc1 = decrypt_heys_round('CEA7', bloc1, True)
bloc1 = decrypt_heys_round('EA71', bloc1, True)
bloc1 = decrypt_heys_round('A71C', bloc1, True)

OUTPUT:
INPUT      0000 [0] | 0011 [3] | 1101 [D] | 1000 [8] |
ROUND 4
AFTER-XOR  0111 [7] | 0010 [2] | 0001 [1] | 0110 [6] |
AFTER-SBOX 1111 [F] | 0100 [4] | 0011 [3] | 1010 [A] |
AFTER-XOR  1110 [E] | 1000 [8] | 1101 [D] | 0000 [0] |
èÐ        E8    D0
INPUT      1110 [E] | 1000 [8] | 1101 [D] | 0000 [0] |
AFTER-PBOX 1110 [E] | 1010 [A] | 1000 [8] | 0010 [2] |
AFTER-SBOX 0000 [0] | 1001 [9] | 0111 [7] | 0100 [4] |
AFTER-XOR  1100 [C] | 0111 [7] | 1101 [D] | 0011 [3] |
ÇÓ        C7    D3
INPUT      1100 [C] | 0111 [7] | 1101 [D] | 0011 [3] |
AFTER-PBOX 1010 [A] | 1110 [E] | 0101 [5] | 0111 [7] |
AFTER-SBOX 1001 [9] | 0000 [0] | 1100 [C] | 1111 [F] |
AFTER-XOR  0111 [7] | 1010 [A] | 1011 [B] | 1110 [E] |
z¾        7A    BE
INPUT      0111 [7] | 1010 [A] | 1011 [B] | 1110 [E] |
AFTER-PBOX 0111 [7] | 1001 [9] | 1111 [F] | 1010 [A] |
AFTER-SBOX 1111 [F] | 1101 [D] | 0101 [5] | 1001 [9] |
AFTER-XOR  0101 [5] | 1010 [A] | 0100 [4] | 0101 [5] |
ZE        5A    45
```

c) Decrypt CIPHERTEXT_3C
d) Decipher CIPHERTEXT_3D

Exercise 4

a) Encrypt pt using just the first round of the Heys cipher in ECB mode with key k

```
pt = PLAINTEXT[:4]*4 #plaintext (ZEROZERO)
k = b'A71CA71CA71CA71C' #key
#iv = b'A71C'
ciph = AES.new(k, AES.MODE_ECB) #creates cipher using key
ct = ciph.encrypt(pt.encode()) #encrypts plaintext using cipher assigning to ct variable
print(f"Plaintext: {pt}\nKey: {k}\nCiphertext: {ct.hex()}\n") #Print the outputs
OUTPUT
[:4]*4
8d4640567641b5c686aac7437bd47e70

[:16]*4
59732743b558bb3d1c845e9514635acc
59732743b558bb3d1c845e9514635acc
59732743b558bb3d1c845e9514635acc
59732743b558bb3d1c845e9514635acc
```

b) Encrypt pt using just the first round of the Heys cipher in CBC mode with key k and initialisation vector iv

```
pt = PLAINTEXT[:4]*4 #plaintext (ZEROZERO)
k = b'A71CA71CA71CA71C' #key
iv = b'D0D0D0D0D0D0D0D0'
ciph = AES.new(k, AES.MODE_CBC, iv) #creates cipher using key
ct = ciph.encrypt(pt.encode()) #encrypts plaintext using cipher assigning to ct variable
print(f"Plaintext: {pt}\nKey: {k}\nCiphertext: {ct.hex()}") #Print the outputs

OUTPUT
[:4]*4
cf09dede272f01f7523ef3c794b564c8

[:16]*4
2cfaaaf6af663ce4db7a47ea5fc052cd
af11313c1c0f733908b5780399f1230c
6bc92f0a23707dba4481b6b9ca5c63ab
1fa75aa97fafcf30c179e4aaf0d4fa0c
```

c) What do you observe when comparing the two ciphertexts generated in answer to exercise 5A and Exercise 5B?

EBC and CBC produce different outputs. However, when investigating further, increasing the bit size to [:16]*4 for each, EBC mode produces a repetitive hash whereas the hash produced using CBC does not repeat.

d) Decrypt CIPHERTEXT_4D

```
iv = 'D0D0'
key = 'A71CEA71CEA71CEA71CE'

bloc1 = CIPHERTEXT_4D[0:4]
```

```
bloc1 = decrypt_heys_final_round(key[12:16], key[16:20], bloc1)
bloc1 = decrypt_heys_round(key[8:12], bloc1)
bloc1 = decrypt_heys_round(key[4:8], bloc1)
bloc1 = decrypt_heys_round(key[0:4], bloc1)
bloc1 = xor_hex_hex(bloc1, iv, True)

bloc2 = CIPHERTEXT_4D[4:8]
bloc2 = decrypt_heys_final_round(key[12:16], key[16:20], bloc2)
bloc2 = decrypt_heys_round(key[8:12], bloc2)
bloc2 = decrypt_heys_round(key[4:8], bloc2)
bloc2 = decrypt_heys_round(key[0:4], bloc2)
bloc2 = xor_hex_hex(bloc2, CIPHERTEXT_4D[0:4], True)

bloc3 = CIPHERTEXT_4D[8:12]
bloc3 = decrypt_heys_final_round(key[12:16], key[16:20], bloc3)
bloc3 = decrypt_heys_round(key[8:12], bloc3)
bloc3 = decrypt_heys_round(key[4:8], bloc3)
bloc3 = decrypt_heys_round(key[0:4], bloc3)
bloc3 = xor_hex_hex(bloc3, CIPHERTEXT_4D[4:8], True)

bloc4 = CIPHERTEXT_4D[12:16]
bloc4 = decrypt_heys_final_round(key[12:16], key[16:20], bloc4)
bloc4 = decrypt_heys_round(key[8:12], bloc4)
bloc4 = decrypt_heys_round(key[4:8], bloc4)
bloc4 = decrypt_heys_round(key[0:4], bloc4)
bloc4 = xor_hex_hex(bloc4, CIPHERTEXT_4D[8:12], True)

bloc5 = CIPHERTEXT_4D[16:20]
bloc5 = decrypt_heys_final_round(key[12:16], key[16:20], bloc5)
bloc5 = decrypt_heys_round(key[8:12], bloc5)
bloc5 = decrypt_heys_round(key[4:8], bloc5)
bloc5 = decrypt_heys_round(key[0:4], bloc5)
bloc5 = xor_hex_hex(bloc5, CIPHERTEXT_4D[12:16], True)

RAW OUTPUT:
CH          4348
ES          4553
HI          4849
RE          5245
_C          5F43

DECRYPTED OUTPUT:
CHESHIRE_C
```

Exercise 5

a) Encrypt your student number (as an integer)

```
stu_No = 2009045
(e,n) = (65537, 38083040388480806513)
ciph_No = pow(stu_No,e,n)
print(ciph_No)

OUTPUT:
31843809755463598485
```

b) Generate a SHA1 hash of hatter.pdf and of flamingo.pdf

Both Flamingo and Hatter have the same SHA1 hashes. When compared to a different file (Heys) it can be seen the hash value change.

```
print('Flamingo: ',sha1_file('flamingo.pdf'))
print('Hatter   : ',sha1_file('hatter.pdf'))
print('Heys     : ',sha1_file('heys.pdf'))
Flamingo:  baba2841e2db8c8b94005848441a095e67456fc6
Hatter  :  baba2841e2db8c8b94005848441a095e67456fc6
Heys    :  b6271755f4a81af038a910e141e15ede41871708
```

c) Generate a SHA256 hash of hatter.pdf and of flamingo.pdf && Compare the two hashes, what do you observe?

The sha256 hashes are different for flamingo and hatter. (Heys included for further comparison)

```
print('Flamingo: ',sha256_file('flamingo.pdf'))
print('Hatter   : ',sha256_file('hatter.pdf'))
print('Heys     : ',sha256_file('heys.pdf'))
Flamingo:  59e99bff26d0322a70e3fb3e193267a397d825afd97bca692ff4954d13267850
Hatter  :  611134263e283e028ffce61a3fc1ac38d2f66eb8b71a074608a6fc41b877e43c
Heys    :  7879e32bac43ba8d92ef300f3b60b002d2b509bf7302420c8f8156aa4da98ef7
```

d) Verify which of the following three messages is the genuine message

```
pub =
rsa.PublicKey(7831380548017175338915258319833911893931253330893508365879836914869334842132321469618467
903977349122973376473368256628270012195036837460719988392463450933, 65537)

MESSAGE_1 = "Would you tell me, please, which way I ought to go from here? That depends a good deal on
where you want to get to".encode('utf8')

MESSAGE_2 = "Its no use going back to yesterday, because I was a different person
then.".encode('utf8')

MESSAGE_3 = "Begin at the beginning, the King said, very gravely, and go on till you come to the end:
then stop.".encode('utf8')

try : rsa.verify(MESSAGE_1, SIGNATURE_5D, pub)
```

```
except: print("Message 1 not verified")
else : print("Message 1 verified")

try: rsa.verify(MESSAGE_2, SIGNATURE_5D, pub)
except : print("Message 2 not verified")
else : print("Message 2 verified")

try: rsa.verify(MESSAGE_3, SIGNATURE_5D, pub)
except : print("Message 3 not verified")
else: print("Message 3 verified")

OUTPUT:
Message 1 not verified
Message 2 not verified
Message 3 verified
```