

Cryptography Assignment 2

1. Plaintext = ITSROS cipher text = JKPDNO
From state in B.

674 I → 5 → 16 → 11 → J
675 T → 15 → 4 → 4 → K
676 S → 13 → 2 → 12 → P
677 R → 11 → 17 → 26 → D
678 O → 7 → 15 → 5 → N
679 S → 10 → 9 → 24 → O

A	19	2	18	13
B	20	25	19	2
C	21	6	20	25
D	22	24	21	6
E	23	13	22	24
F	24	21	23	13
G	25	3	24	21
H	26	15	25	3
I	1	19	26	15
J	2	10	1	19
K	3	14	2	10
L	4	14	3	14
M	5	26	4	14
N	6	20	5	26
O	7	8	6	20
P	8	16	7	8
Q	9	7	8	16
R	10	22	9	7
S	11	4	10	22
T	12	5	11	4
U	13	17	12	5
V	14	17	13	17
W	15	9	14	17
X	16	23	15	9
Y	17	18	16	23
Z	18	2	17	18
After 677	18	17	17	23
After 678	17	17	17	23

ROTOR CHANGES									
A	22	24	21	6	20	25	25	17	26
B	23	13	22	21	21	6	26	20	1
C	24	21	23	13	22	24	1	1	2
D	25	3	24	21	23	13	2	2	3
E	26	15	25	3	24	21	3	4	4
F	1	1	26	15	25	3	4	15	5
G	2	19	1	1	26	15	5	3	6
H	3	10	2	19	1	1	6	14	7
I	4	14	3	10	2	19	7	12	8
J	5	26	4	14	3	10	8	23	9
K	6	30	5	26	4	14	9	5	10
L	7	8	6	30	5	26	10	16	11
M	8	16	7	8	6	30	11	2	12
N	9	7	8	16	7	8	12	23	13
O	10	20	9	7	8	16	13	19	14
P	11	4	10	20	9	7	14	11	15
Q	12	11	11	4	10	20	15	18	16
R	13	5	12	11	11	4	16	25	17
S	14	17	13	5	12	11	17	24	18
T	15	9	14	17	13	5	18	13	19
U	16	12	15	9	14	17	19	7	20
V	17	23	16	12	15	9	20	10	21
W	18	18	17	23	16	12	21	8	22
X	19	2	18	18	17	23	22	21	23
Y	20	25	19	2	18	18	23	9	24
Z	21	6	20	25	19	2	24	26	25
FAST First medium SLOW									
#2 #1 #1									
After 674	After 675	After 676							

Hilroy

2 Most Frequent letters : E, A, R, I, O, T, N, S ...

" " bigrams : th, he, in, en, nt, re, er, an, ti, es, on, at ...

trigrams : the, and, tha, ent, double letter : ee, ll, ss, oo

A Botanical Code .

B → E, common trigram = the; BJB → THE

R → A, common trigram = and; RIX → AND

oo → LL try Z → W for ZBOO → WELL

freg BU → ER, -tandard → standard : N → S

- was → I was : K → I, thin- → think : P → K

ed--ated → educated : LC → UC, --R the → FOR THE FA → FO

sentimental → sentimental : E → M, lan_a_e → language : S → G

stud- → study : m → y, TULL → tulip : W → P

go_erness → governess : T → V, -ut → but : V → B

-calousy → Jealousy : Q → J,

Alphabet decoded : 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 E C J M F T N H I U Y S L K J A G V R B P D W

There was no way to test D, H, Y.

The message is .

I was, I think, well educated. for the standard of the day.
my sister and I had a German governess, a very sentimental
creature. She taught us the language of flowers, a forgotten
study nowadays, but most charming. A yellow Tulip, for
instance, means hopeless love; while a China Aster means I
die of jealousy at your feet.

No Ho's!

2.b. $V \rightarrow E, XC \rightarrow TH$.

$TH-T \rightarrow That : S \rightarrow A$, matches probability

no the thus the-e \rightarrow these : $O \rightarrow S$

se_a-ate \rightarrow separate : $P \rightarrow P, N \rightarrow R$

appare_t_ \rightarrow apparently : $A \rightarrow N, U \rightarrow L, B \rightarrow Y$

se-ret \rightarrow secret : $U \rightarrow C$, sketches \rightarrow sketches : $R \rightarrow K$

c-phers \rightarrow ciphers : $G \rightarrow I$, -airly \rightarrow fairly : $E \rightarrow F$

fa-iliar \rightarrow familiar : $Z \rightarrow M$, -ith \rightarrow with : $K \rightarrow W$

P-rms \rightarrow forms : $Q \rightarrow O$, writin- \rightarrow writing : $H \rightarrow G$

an- \rightarrow and : $W \rightarrow D$, -pon \rightarrow upon : $I \rightarrow U$

analy-e \rightarrow analyze : $F \rightarrow Z$, -ut \rightarrow but : $D \rightarrow B$

sub-ect \rightarrow subject : $L \rightarrow J$, con-ey \rightarrow convey : $M \rightarrow V$

alphabet decoded: ABCDEFGHIJKLMNOPQRSTUVWXYZ

NYHBFZIGUCWJVRSPOKA LEDT

T and Y map to : Q and X, unknown order

The message is.

I am fairly familiar with all forms of secret writing and am myself author of a brilliant monograph upon subject which I analyze one hundred separate ciphers but I confess that this is entirely new to me. Object of those who invented this system has apparently been to conceal that these characters convey a message and to give idea that they are mere random sketches of children

The code I used to test and solve question 2 is as follows:

```
def map_chars(input_string, source_alphabet, target_alphabet):  
    # Create a dictionary for mapping source characters to target characters  
    mapping_dict = {src: tgt for src, tgt in zip(source_alphabet, target_alphabet)}  
    # Replace characters in the input string according to the mapping  
    mapped_string = ''.join([mapping_dict.get(char, char) for char in input_string])  
    return mapped_string
```

For parts a and b, the source alphabet was the regular A to Z characters and the target alphabet was 26 of the character “-” where each corresponds to an A to Z character. With each attempt at solving the code a letter was added to the mapped alphabet in place of the dash of the corresponding letter. After some parts of words were found, I was able to infer more letters until the message was fully deciphered.

For part c, I switched the target and source alphabet to encode my plaintext after removing the common word ‘the’

Sample Outputs:

b.

```
PS C:\Users\mika> & C:/Python312/python.exe c:/Users/mika/Desktop/Fifth/4108/probciphers.py  
Original string: GSZES GNUBE SZGUG SNKGX CSUUE QNZOQ EOVDN VVKNG XGAHS  
AWSZZ BOVUE SIXCQ NQESX NGEUG AHZQA QHNSP CIPQA OIDL  
JXGAK CGJCG SASUB FVQAV CIAWN VWOVP SNSXV JGPCV NODIX  
GJQAE VOOXC SXXCG OGOVA XGNVU BAVKX QZVQD LVJXQ EXCQO  
VKCQG AMVAX VWXCG OOBX VZCSO SPPSN VAXUB DVVAX QJQAJ  
VSUXC SXXCV OVJCS NSJXV NOJQA MVBSZ VOOSH VSAWX QHGMV  
GWVSX CSXOC VBSNV ZVNVN SAWQZ ORVXJ CVOQE JCGUN NVA  
Mapped string: IAMFA IRLYF AMILI ARWIT HALLF ORMSO FSECR ETWRI TINGA  
NDAMM YSELF AUTHO ROFAT RIFLI NGMON OGRAP HUPON SUBJE  
CTINW HICHI ANALY ZEONE HUNDR EDSEP ARATE CIPHE RS  
BUT ICONF ESSTH ATTHI SISEN TIREL YNEWT OMEOB JECTO FTHOS  
EWHOI NVENT EDTHI SSYST EMHAS APPAR ENTLY BEENT OCONC  
EALTH ATTHE SECHA RACTE RSCON VEYAM ESSAG EANDT OGIVE  
IDEAT HATTH EYARE MERER ANDOM SKETC HESOF CHILD REN  
PS C:\Users\mika> █
```

c.

```
PS C:\Users\mika> & C:/Python312/python.exe c:/Users/mika/Desktop/Fifth/4108/probciphers.py  
Original string: QUICK BROWN FOXJ MPSOV ERLAZ YDOGI SASEN TENCE  
WITHA LLETT TERSO FALPH ABETI NIT  
Mapped string: TIGJR DNQKA EQYLI ZPOQM VNUSF BWQHG OSOVA XVAJV  
KGXCS UUVX XVNOQ ESUPC SDVXG AGX  
PS C:\Users\mika> █
```


Qc plaintext: "The quick brown fox jumps over the lazy dog is a sentence with all the letters of the alphabet in it"

plaintext "quick brown fox jumps over lazy dog is a sentence with all letters of alphabet in it"
- "the"

using alphabet "NYHBFZIGUCWJVRSPQKAOLEDTXM"
where TY maps to QX.

ciphertext = TIGJR DNQKA EQYLI ZPOQM VNUSE
BWQHG OSOVA XVAJV KGXCS UUVVX
ESUPC SDVXG AGX

3. Using attached Python Script.

Let $P = 8219$.

$$8219 \equiv 16363 \equiv 3 \pmod{4}$$

Let $Q = 16363$

$n = 134487497$ seed = 29373821

The first 15 generated outputs are

$[0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0]$

- b. From my understanding the security of BBS is roughly equivalent to $n/2$ as the security relies on the factorability of n . thus to be \geq the security of Des - with a 56 bit key, each p_i, q_i will need to be at least 56 bits and because Aes supports up to 256 bit keys, p_i, q_i needs to be at least 256 bit sizes.

For q3:

```
from sympy import isprime

import random

from math import gcd

def find_primes_between(start, end):

    """Find all prime numbers between the given start and end (inclusive). and for the sake of
    run time, all that are congruent to 3 mod 4"""

    primes = [num for num in range(start, end + 1) if isprime(num) and num%4 ==3]

    return primes

def generate_seed(n):

    """Generate a seed x_0 such that gcd(x_0, n) = 1."""

    while True:

        seed = random.randint(1, n - 1)

        if gcd(seed, n) == 1:

            return seed

def bbs_generate_bits(n, seed, num_bits=15):

    """Generate the first 'num_bits' bits using the Blum Blum Shub generator."""

    # Initialize the sequence with the given seed (x_0)

    x = (seed ** 2) % n # First iteration

    #Generate the sequence and extract bits

    bits = []

    for _ in range(num_bits):

        x = (x ** 2) % n # Compute the next x

        bits.append(x % 2) # Extract the least significant bit (LSB)

    return bits

start = 2**13 # 8192

end = 2**14 - 1 # 16383

primes_14_bit = find_primes_between(start, end)
```

```

print(primes_14_bit)

### Choose two values p q from primes_14_bit

p = primes_14_bit[0]

q = primes_14_bit[-1]

print(f"Chosen p value: {p}")

print(f"Chosen q value: {q}")

n = p*q

seed = generate_seed(n)

print(f"Generated Seed: {seed}")

# Generate the first 15 bits

first_15_bits = bbs_generate_bits(n, seed, num_bits=15)

print(first_15_bits)

```

Sample Output

```

[8219, 8231, 8243, 8263, 8287, 8291, 8311, 8363, 8387, 8419, 8423, 8431, 8443, 8447, 8467, 8527, 8539, 8543, 8563, 8599, 8623, 8627, 8647, 8663, 8699, 8707, 8719, 8731, 8747, 8779, 87
83, 8803, 8807, 8819, 8831, 8839, 8863, 8867, 8887, 8923, 8951, 8963, 8971, 8999, 9007, 9011, 9043, 9059, 9067, 9091, 9103, 9127, 9151, 9187, 9199, 9203, 9227, 9239, 9283, 9311, 9319,
9323, 9343, 9371, 9391, 9403, 9419, 9431, 9439, 9463, 9467, 9479, 9491, 9511, 9539, 9547, 9551, 9587, 9619, 9623, 9631, 9643, 9679, 9719, 9739, 9743, 9767, 9787, 9791, 9803, 9811, 98
39, 9851, 9859, 9871, 9883, 9887, 9907, 9923, 9931, 9967, 10007, 10039, 10067, 10079, 10091, 10099, 10103, 10111, 10139, 10151, 10159, 10163, 10211, 10223, 10243, 10247, 10259, 10267,
10271, 10303, 10331, 10343, 10391, 10399, 10427, 10459, 10463, 10487, 10499, 10531, 10559, 10567, 10607, 10627, 10631, 10639, 10651, 10663, 10667, 10687, 10691, 10711, 10723, 10739,
10771, 10799, 10831, 10847, 10859, 10867, 10883, 10891, 10903, 10939, 10979, 10987, 11003, 11027, 11047, 11059, 11071, 11083, 11087, 11119, 11131, 11159, 11171, 11239, 11243, 11251, 1
1279, 11287, 11299, 11311, 11351, 11383, 11399, 11411, 11423, 11443, 11447, 11467, 11471, 11483, 11491, 11503, 11519, 11527, 11551, 11579, 11587, 11699, 11719, 11731, 11743, 11779, 11
783, 11807, 11827, 11831, 11839, 11863, 11867, 11887, 11903, 11923, 11927, 11939, 11959, 11971, 11987, 12007, 12011, 12043, 12071, 12107, 12119, 12143, 12163, 12203, 12211, 12227, 122
39, 12251, 12263, 12323, 12343, 12347, 12379, 12391, 12451, 12479, 12487, 12491, 12503, 12511, 12527, 12539, 12547, 12583, 12611, 12619, 12647, 12659, 12671, 12703, 12739, 12743, 1276
14683, 14699, 14723, 14731, 14747, 14759, 14767, 14771, 14779, 14783, 14827, 14831, 14843, 14851, 14867, 14879, 14887, 14891, 14923, 14939, 14947, 14951, 14983, 15031, 15083, 15091, 1
5107, 15131, 15139, 15187, 15199, 15227, 15259, 15263, 15271, 15287, 15299, 15307, 15319, 15331, 15359, 15383, 15391, 15427, 15439, 15443, 15451, 15467, 15511, 15527, 15551, 15559, 15
583, 15607, 15619, 15643, 15647, 15667, 15671, 15679, 15683, 15727, 15731, 15739, 15767, 15787, 15791, 15803, 15823, 15859, 15887, 15907, 15919, 15923, 15959, 15971, 15991, 16007, 160
63, 16067, 16087, 16091, 16103, 16111, 16127, 16139, 16183, 16187, 16223, 16231, 16267, 16319, 16339, 16363]
Chosen p value: 8219
Chosen q value: 16363
n (modulus) value: 134487497
Generated Seed: 29373821
[0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0]
PS C:\Users\mikae>

```