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

ROTOR CHANGES

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

A	19	2	18	18
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	1	26	15
J	2	19	1	1
K	3	10	2	19
L	4	14	3	10
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	11	11	4
U	13	5	12	11
V	14	17	13	5
W	15	9	14	17
X	16	12	15	9
Y	17	23	16	12
Z	18	18	17	23
	After 677		After 678	

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 ; GJB → THE
R → A, common trigram = and, RIX → AND
oo → LL try Z → W for ZBOO → WELL
freq BG → 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
senti_ental → sentimental : E → M, lan_a_e → language : S → G
stud_ → study : m → y, TULL_ → tulip : W → P.
go_erness → governess : T → V, _ut → but : Y → B
_ealousy → 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
O E C _ 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 the's!

2.b. V → E, XC → TH.

TH_T → that : S → A, matches probability

no the' thus the_e → these : O → S

se_a_ate → separate : P → P, N → R

appare_t_ _ → apparently : A → N, U → L, B → Y

se_ret → secret : U → C, sketches → sketches : R → K

c_phers → ciphers : G → I, _airly → fairly : E → F

fa_iliar → familar : Z → M, _ith → with K → W

f_rms → forms : Q → O, writin_ → writing : H → G

an_ → and : W → D, _pon → upon : I → U

analy_e → analyze : F → Z, _ut → but : D → B

sub_ect → subject : L → J, con_ey → convey : M → V

alphabet decoded: ABCDEFGHIJKLMNOPQRSTUVWXYZ

NYHBFZIGUCWJVRSPOKA LEDT M

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

The message is.

I am fairly familiar with all forms of secret writting and am myself author of a trifling monograph upon subject which I analyze one hundred separate ciphers but I confe that this is entirely new to me object of those who invented this system has appartently been to concea that these characters convey a message and to give idea that they are mere random sketches of childre

Hilroy

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\mikae> & C:/Python312/python.exe c:/Users/mikae/Desktop/Fifth/4108/probciphers.py
Original string: GSZES GNUBE SZGUG SNKGX CSUUE QNZOQ EOVJN VXKNG XGAHS
 AWSZZ BOVUE SIXCQ NQESX NGEUG AHZQA QHNSP CIPQA OIDLV
JXGAK CGJCG SASUB FVQAV CIAWN VWOVP SNSXV JGPCV NODIX
GJQAE VOOXC SXXCG OGOVA XGNVU BAVKX QZVQD LVJXQ EXCQO
VKCQG AMVAX VWXCG OOBOX VZCSO SPPSN VAXUB DVVAX QJQAJ
VSUXC SXXCV OVJCS NSJXV NOJQA MVBSZ VOOSH VSAWX QHGMV
GWVSX CSXXC VBSNV ZVNVN SAWQZ ORVXJ CVOQE JCGUW 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 RSBUT
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\mikae>
```

c.

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

2c plaintext: "The quick brown fox jumps over the lazy dog is a sentence with all the letters of the alphabet in it".

plaintext - "the" "quick brownfoxjumps over lazy dog is a sentence with all letters of alphabet in it.

using alphabet "N4HBFZIGUCWJVRSPOKAQLEDTXM" where TY maps to QX.

ciphertext = TIGJR DNQKA EQYLI ZPOQM VNUSF BWQHG OSOVA XVAJV KGXCS UUUVX 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, q, will need to be at least 56 bits and because AES supports up to 256 bit keys, $\therefore p, q$ 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). amd 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