Q1 Break Simple Substitution

25 Points

Write a program (in any language you like) to decrypt a **simple substitution cipher**.
Your program should **take the ciphertext as input**, then
1. Compute and display letter frequency counts.
2. Allow the user to guess a key and display the results of the corresponding "decryption" with the putative key.
3. Allow the user to try step 2 as many times as needed until getting the correct plaintext (the text that makes sense).
Alternatively, you can allow the user to try each mapping one by one. For example, the user can choose to substitute 'C' as 'a', and see the result after substitution. Your goal is to be able to find the key.
Finally, **output the correct key in all UPPERCASES**, so that the first letter is the corresponding ciphertext for plaintext 'a', the second letter is mapping to 'b', ..., the 26th letter is mapping to 'z'.

For example, key=LHFGKTMPZECBQRIYAUSODJWNVX means...

$$\begin{bmatrix} plaintext & a & b & c & d & e & f & g & h & i & j & k & l & m & n & o & p & q & r & s & t & u & v & w & x & y & z \\ CIPHERTEXT & L & H & F & G & K & T & M & P & Z & E & C & B & Q & R & I & Y & A & U & S & O & D & J & W & N & V & X \end{bmatrix}$$

Q1.1 Simple subs. Code
23 Points

Upload your code for break simple substitution.

simple_sub.py Download

```python
"""
Problem 1: Break Simple Substitution

Sources:

import string: https://www.geeksforgeeks.org/python/alphabet-range-in-python/

"""
import string

ciphertext = input("Enter the ciphertext: ")

ciphertext = ciphertext.replace(" ", "")

# print(ciphertext)

letter_dict = {}

for letter in ciphertext:
  letter_dict[letter] = letter_dict.get(letter, 0) + 1

sorted_letter_dict = {k: v for k, v in sorted(letter_dict.items(), key = lambda item: item[1], reverse=True)}

print("Letter Frequency Counts:")

for letter, freq in sorted_letter_dict.items():
  print(letter + ": " + str(freq))

while True:
  key = input("Guess a key (type exit to quit): ")
  if key == 'exit':
    break
  elif len(key) == 26:
    alphabet = string.ascii_lowercase # library from https://www.geeksforgeeks.org/python/alphabet-range-in-python/
    mapping = {c: p for (c, p) in zip(key, alphabet)}
    # print(mapping)
    decrypted_plain = ""
    for c in ciphertext:
      if c in mapping:
        decrypted_plain += mapping[c]
```

```
      else:
        decrypted_plain += c
    print(decrypted_plain)
  else:
    print("Enter a valid key that is exactly 26 characters long")

```

Q1.2 Simple subs. key
2 Points

Given the following ciphertext, use your program to find the key. Remember to enter the **key in all CAPS**! See the example in the instruction.
Also, note that the spaces are for easier reading (otherwise it will be a very long line), so please **remove all spaces in the ciphertext first** (before you do the letter frequency counts).

> GBSXUCGSZQGKGSQPKQKGLSKASPCGBGBKGUKGCEUKUZKGGBSQ
> EICACGKGCEUERWKLKUPKQQGCIICUAEUVSHQKGCEUPCGBCGQOE
> VSHUNSUGKUZCGQSNLSHEHIEEDCUOGEPKHZGBSNKCUGSUKUASE
> RLSKASCUGBSLKACRCACUZSSZEUSBEXHKRGSHWKLKUSQSKCHQT
> XKZHEUQBKZAENNSUASZFENFCUOCUEKBXGBSWKLKUSQSKNFKQQ
> KZEHGEGBSXUCGSZQGKGSQKUZBCQAEIISKOXSZSICVSHSZGEGB
> SQSAHSGKHMERQGKGSKREHNKIHSLIMGEKHSASUGKNSHCAKUNSQ
> QKOSPBCISGBCQHSLIMQGKGSZGBKGCGQSSNSZXQSISQQGEAEUG
> CUXSGBSSJCQGCUOZCLIENKGCAUSOEGCKGCEUQCGAEUGKCUSZU
> EGBHSKGEHBCUGERPKHEHKHNSZKGGKAD

KFAZSROBCWDINUELTHQGXVPJMY

Q2 A5/1
15 Points

Write a program (in any language you like) to implement the **A5/1** algorithm. Your program should **take the values in the registers (X, Y, and Z) as input**, and then:
1. Generate and output a keystream of any given length.
2. Output the updated content of registers (X, Y, Z) after the keystream with the desired length is generated.

Q2.1 A5/1 Code
13 Points

Upload your code for A5/1.

▾ a51.py Download

```
"""
Problem 2: A5/1 Code


i.e., 0b1010101010101010101
      0b1100110011001100110011
      0b11100001111000011110000  32
"""

import sys

L1, L2, L3 = 19, 22, 23
MASK1, MASK2, MASK3 = (1 << L1) - 1, (1 << L2) - 1, (1 << L3) - 1


MSB_TAPS_X = [13, 16, 17, 18]
MSB_TAPS_Y = [20, 21]
MSB_TAPS_Z = [7, 20, 21, 22]

MSB_CLK_X, MSB_CLK_Y, MSB_CLK_Z = 8, 10, 10


def my_to_lsb_positions(length, msb_indices):
   return [length - 1 - idx for idx in msb_indices]


def my_to_lsb_position(length, msb_idx):
    return length - 1 - msb_idx


def bit(value, pos_lsb):
    return (value >> pos_lsb) & 1


def majority(a, b, c):
    return (a & b) | (a & c) | (b & c)


def clock_right_feed_msb(state, length, taps_lsb):

    fb = 0
    for t in taps_lsb:
        fb ^= bit(state, t)
    state = (state >> 1) | (fb << (length - 1))
    return state
```

```


def parse_value(s):
    s = s.strip().lower()
    if s.startswith("0b"):
        return int(s[2:], 2)
    elif s.startswith("0x"):
        return int(s[2:], 16)
    return int(s)


def to_bin(v, width):
    b = bin(v)[2:]
    if len(b) < width:
        return "0" * (width - len(b)) + b
    return b[-width:]


def generate(x, y, z, n):

    state_x, state_y, state_z = x, y, z
    out = []

    #These will be Converted MSB indices to LSB positions
    X_taps = my_to_lsb_positions(L1, MSB_TAPS_X)
    Y_taps = my_to_lsb_positions(L2, MSB_TAPS_Y)
    Z_taps = my_to_lsb_positions(L3, MSB_TAPS_Z)
    X_clk = my_to_lsb_position(L1, MSB_CLK_X)
    Y_clk = my_to_lsb_position(L2, MSB_CLK_Y)
    Z_clk = my_to_lsb_position(L3, MSB_CLK_Z)

    for _ in range(n):
        maj = majority(bit(state_x, X_clk), bit(state_y, Y_clk), bit(state_z, Z_clk))

        if bit(state_x, X_clk) == maj:
            state_x = clock_right_feed_msb(state_x, L1, X_taps)
        if bit(state_y, Y_clk) == maj:
            state_y = clock_right_feed_msb(state_y, L2, Y_taps)
        if bit(state_z, Z_clk) == maj:
            state_z = clock_right_feed_msb(state_z, L3, Z_taps)

        out_bit = (state_x & 1) ^ (state_y & 1) ^ (state_z & 1)
        out.append(str(out_bit))

```

```
    keystream = "".join(out)
    return keystream, state_x, state_y, state_z


def main():
    if len(sys.argv) != 5:
        print("My usage: python a51.py <X> <Y> <Z> <N>")
        print("The example is:")
        print("python a51.py 0b1010101010101010101 0b1100110011001100110011 0b11100001111000011110000 32")
        sys.exit(0)

    X = parse_value(sys.argv[1]) & MASK1
    Y = parse_value(sys.argv[2]) & MASK2
    Z = parse_value(sys.argv[3]) & MASK3
    N = int(sys.argv[4])

    ks, Xf, Yf, Zf = generate(X, Y, Z, N)

    # Print four lines
    print(ks)
    print(to_bin(Xf, L1))
    print(to_bin(Yf, L2))
    print(to_bin(Zf, L3))


if __name__ == "__main__":
    main()
```

Q2.2 A5/1 32-bit key
0.5 Points

(Q2.2 - Q2.5) Given the following X, Y, Z, using your program, generate **32 bits** keystream, and the content of the registers after the 32 bits has been generated

> X = 1010101010101010101
> Y = 1100110011001100110011
> Z = 11100001111000011110000

Your answers should be binary numbers, no spaces!

32-bits keystream =

10000011011100000111100000011001

Q2.3 A5/1 new X
0.5 Points

After generating 32-bits, X =

0001101000000000000

Q2.4 A5/1 new Y
0.5 Points

After generating 32-bits, Y =

1111101010101010101010

Q2.5 A5/1 new Z
0.5 Points

After generating 32-bits, Z =

01101010111100001010101

Q3 "Break" RSA

20 Points

> For RSA, if you **know p & q** that are used to get N, it's not hard to **find d** using the **extended Euclidean algorithm (EEA)**.
> Here's why:
> One rule for keys used in RSA is that:
> $d = e^{-1} \mod \phi(N)$.
> As we showed in class (lesson 5, page 15), $e^{-1} \mod \phi(N)$ can be computed by EEA (which is the a you got from the algorithm).
> So, once we know p & q, it's easy to get
> $\phi(N) = (p-1)*(q-1)$ (let's call this number as phi).
> Then the a we get from `EEA(e, phi)` is private key d, where the **algorithm for EEA is given in lesson 5 page 11**.

Based on the above information, write a program (any language you like) that **takes p, q, and e for RSA as inputs**, and **output the smallest possible** $d > 0$ that can be used as a private key **using the extended Euclidean algorithm**.
**The challenging part is how to make sure it's the smallest positive d. That is, if you get $d < 0$, what do you add to make $d > 0$? Or, if you get $d > 0$, what do you subtract to make it as small as possible (while keeping it > 0)?
(Hint: we want to know the difference between each possible d. Recall that d & e are multiplicative inverse over $\mod \phi(N)$. Think about the definition of multiplicative inverse, and then use transitivity and cancellation of common term properties of congruence)

Q3.1 RSA code
18 Points

Upload your code for getting d in RSA using the extended Euclidean algorithm.

▾ q3.py Download

```python
# 3. EEA
#https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm
#https://cp-algorithms.com/algebra/extended-euclid-algorithm.html
#Lesson 5: Public Key Crypto
def recursiveEEA(phi,e):
    if phi == 0:
        return [e, 1, 0]
    else:
        values = recursiveEEA(e % phi, phi)
        return [values[0], values[2], values[1] - (e // phi) * values[2]]

def dCalc(phi, e):
    #values[gcd, old_s, s]
    values = recursiveEEA(phi, e)
    print(values)
    d = values[1] % phi

    if d < 0:
        d += phi
    return d

#Question values:
#p = 233
#q = 331
#e = 221
p = int(input('Enter p: '))
q = int(input('Enter q: '))
e = int(input('Enter e: '))

print(f'd is {dCalc(((p - 1)*(q - 1)), e)}')
```

Q3.2 RSA d
2 Points

Use your program to find the **smallest possible d (> 0)** given

> p = 233, q = 331, e = 221.

46421

Programming Assignment 1 - Crypto

Graded

Select each question to review feedback and grading details.

Group
Brendan Ly
Tiernan Johnson
Rongjie Mai
View or edit group

Total Points
58 / 60 pts

Question 1

	Break Simple Substitution	**23** / 25 pts
1.1	Simple subs. Code	**21** / 23 pts
1.2	Simple subs. key	**2** / 2 pts

Question 2

A5/1 **15** / 15 pts

2.1	A5/1 Code	**Resolved**	**13** / 13 pts
2.2	A5/1 32-bit key		**0.5** / 0.5 pts
2.3	A5/1 new X		**0.5** / 0.5 pts
2.4	A5/1 new Y		**0.5** / 0.5 pts
2.5	A5/1 new Z		**0.5** / 0.5 pts

Question 3

"Break" RSA **20** / 20 pts

3.1	RSA code	**18** / 18 pts
3.2	RSA d	**2** / 2 pts