Information Security Assignment 1

Stijn Kammer (S4986296) Ramon Kits (S5440769)

September 2022

Exercise 1

The University of Groningen's Acceptable Use Policy, or also known as the AUP of the RUG, is a document describing general rules for using the computer systems of the University of Groningen.
Below we will discuss some questions about the AUP.

1. **What (if any) are the differences between the responsibilities of 'ordinary' users and systems managers? Do systems managers have special privileges and responsibilities (if so, what are they)?**

   The rights and duties of both users and systems managers are the same. But the position of the systems manager comes with additional security requirements. The main difference boils down to one of the later points in the list in the AUP; "The systems manager will consider any information about the system, as well as any information stored in the system as confidential." Which means that the systems manager should not share private information which he or she might encounter while working.

2. **What is the ground-rule upon which the RUG's AUP is based?** "The users of the university computer systems may not endanger these systems, nor may they hinder other users." This is stated near the start of the document.

3. **Mention four advices for users of 'RuGnet';**

   (a) **Data is being monitored.** in- and outbound traffic of RuGnet is constantly monitored, this means it not adviced to use RuGnet for illegal activities or to sent private information. Any use of the university computer systems is restricted to research or education.

   (b) **Copying software.** Any software stored on the university computer systems may not be copied for use elsewhere, unless explicit and written permission was granted by proper authorities.

   (c) **Do not share your password.** The password of your account should not be shared with anyone. If you suspect that your password has been compromised, you should change it immediately. To minimize the risk of your password being compromised, you should use a strong password.

   (d) **Check for any hardware abnormalities.** If you notice any abnormalities in the hardware of the computer systems, you should report it immediately to the ICT department and not use that device. abnormalities can be anything from unexpected extra periffirals connected to the device, to the keyboard being plugged in a seperate device before it connects to the computer or device.

4. **Describe four actions that are prohibited by the RuG's AUP;** Following actions are taken directly from the secion "The abuse of facilities and privileges" in the AUP:

(a) **Removing soft- or hardware.** The removal of any software or hardware from the university computer systems is prohibited.

(b) **Using computer systems without permission.** It is prohibited to use any computer system without permission from proper authorities.

(c) **Executing DoS attacks.** It is prohibited to execute any kind of Denial of Service attacks on any computer system or the network around it. Examples of this are, submitting extremely large printjobs, storing extremely large amounts of data, or executing programs using grossly inefficient algorithms or requiring excessively large resources.

(d) **Distributing information that belongs to the university.** It is prohibited to distribute information that belongs to the university, unless explicit and written permission was granted by proper authorities or owner of the information.

5. **What sanctions can be applied to those who violate the AUP?** As stated in Consequences of abusing the University computer systems, Abusing university computer systems may result in disciplinary action. This includes the access rights of the suspect being restricted or suspended, awaiting the results of the investigation which will take place by. Either way, the faculty or department responsible for the suspect will be informed.

6. **If a sanction is applied to you, where can you go to challenge that sanction?** You may file an objection to this restriction or suspension with the chair of your department.

Exercise 2

```
1
2  import string
3
4
5  # substitution cypher
6
7  # Encrypts a string using a shift cypher
8  def encryptShift(plain_text, key):
9      cipher_text = ""
10     for char in plain_text:
11         if char.isalpha():
12             if char.isupper():
13                 cipher_text += chr((ord(char) + int(key) - 65) % 26 + 65)
14             else:
15                 cipher_text += chr((ord(char) + int(key) - 97) % 26 + 97)
16         else:
17             cipher_text += char
```

```
    return cipher_text

# Decrypts a string using a shift cypher
def decryptShift(cipher_text, key):
    plain_text = ""
    for char in cipher_text:
        if char.isalpha():
            if char.isupper():
                plain_text += chr((ord(char) - int(key) - 65) % 26 + 65)
            else:
                plain_text += chr((ord(char) - int(key) - 97) % 26 + 97)
        else:
            plain_text += char
    return plain_text

# Encrypts a string using a mapping cypher
def encryptMapping(plain_text, mapping):
    cipher_text = ""
    for char in plain_text:
        if char.isalpha():
            if char.isupper():
                cipher_text += mapping[ord(char) - 65].upper()
            else:
                cipher_text += mapping[ord(char) - 97]
        else:
            cipher_text += char
    return cipher_text

# Decrypts a string using a mapping cypher
def decryptMapping(cipher_text, mapping):
    plain_text = ""
    for char in cipher_text:
        if char.isalpha():
            if char.isupper():
                plain_text += chr(mapping.index(char.lower()) +
                    97).upper()
            else:
                plain_text += chr(mapping.index(char) + 97)
        else:
            plain_text += char
    return plain_text

# Handles the input from the user
def handle_input(query):
    mappingoffset = string.ascii_lowercase
    steps = query.split(" ")
    for i in range(0, len(steps), 2):
        map_or_shift = stringToInt(steps[i+1])
        if steps[i] == "d":
            if isinstance(map_or_shift, int):
```

```
                mappingoffset = decryptShift(mappingoffset, map_or_shift)
            else:
                mappingoffset = decryptMapping(mappingoffset,
                    map_or_shift)
        elif steps[i] == "e":
            if isinstance(map_or_shift, int):
                mappingoffset = encryptShift(mappingoffset, map_or_shift)
            else:
                mappingoffset = encryptMapping(mappingoffset,
                    map_or_shift)
    return mappingoffset


# Converts a string to an int if possible
def stringToInt(string):
    try:
        value = int(string)
        return value
    except ValueError:
        return string

# Main function
def main():
    query = input()
    handled_input = handle_input(query)
    while(True):
        try:
            plain_text = input()
            print(encryptMapping(plain_text, handled_input))
        except EOFError:
            break

# Run main function
if __name__ == "__main__":
    main()
```

Exercise 3

1. **How many possible alphabets could be used in a substitution cipher that uses shifting? What about the number of possible alphabets in a substitution cipher that uses a mixed alphabet?** For a substitution cipher that uses shifting, there are 26 possible alphabets. Starting from the normal alphabet, it can be shifted by one inder 26 times before it is at its begin state again. For a substitution cipher that uses a mixed alphabet, there are 26! possible alphabets, that translates to about 4.0329×10^{26} possible alphabets.

2. **Does applying a significant number of consecutive simple substitution cipher encryptions/decryptions with a mixed or shifted alphabet make it harder to break the original plaintext? Justify your answer.** No, applying a significant number of consecutive simple substitution cipher encryptions/decryptions with both mixed or shifted alphabet does not make it harder to break the original plaintext. It will not make it easier either. This is because every character is still directly mapped to another character. An R mapped to a Q which is then mapped to a W, is the same as an R mapped to a W. The same goes for a shifted alphabet, A shift of 8 followed by a shift of -2 is the same as a shift of 6. The one and only possible benefit is that with multiple iterations you can use multiple alphabets. So you can give two or more people different alphabets and they can work together to encrypt the plaintext and decrypt when done in reverse order. The power it would take to break the text is the same with any amount of encryptions/decryptions. The only difference is that it would take longer to decrypt the text with the set of keys/alphabets.

3. **Can the encryption function of the substitution cipher also be used for decryption? If so, how?** Yes, the encryption function of the substitution cipher can also be used for decryption. By running the encryption function with the same key multiple times, the plaintext will eventualy be returned. With a shift of 1, the plaintext will be returned after 26 iterations for example. This will also apply to mixed alphabets, after a maximum of 26 iterations, 26 individual mapping loops have been made. So after 26 iterations, the plaintext will definitely be returned and in some cases earlier.

Exercise 4

```
1
2  import sys
3
4
5  def encrypt(key, text):
6      # encrypt text using given key
7      counter = 0
8      newText = []
9      for i in range(len(text)):
10         # continue if character is alphanumeric
11         if text[i].isalpha():
12             start = ord('A') if text[i].isupper() else ord('a')
13             newText.append(chr(
14                 ((ord(text[i]) - start + ord(key[counter % len(key)]) -
                       ord('a')) % 26) + start))
15             counter += 1
```

```
        else:
            newText.append(text[i])
    print(''.join(newText))


def decrypt(key, text):
    # decrypt text using given key
    counter = 0
    newText = []
    for i in range(len(text)):
        # continue if character is alphanumeric
        if text[i].isalpha():
            start = ord('A') if text[i].isupper() else ord('a')
            newText.append(chr(
                ((ord(text[i]) - start - ord(key[counter % len(key)]) +
                    ord('a')) % 26) + start))
            counter += 1
        else:
            newText.append(text[i])
    print(''.join(newText))


def handle_input(ende, key, input):
    if(ende == 'd'):
        decrypt(key, input)
    elif(ende == 'e'):
        encrypt(key, input)
    else:
        print("Invalid input")


def main():
    ende = ''
    key = ''
    input = ''
    # get input text
    for line in sys.stdin:
        if key == '':
            ende, key = line.strip().split(' ')
        elif line.strip() == '':
            handle_input(ende, key, input)
        else:
            input = input + line


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

Exercise 5

```
1
2  import re
3  import sys
4  import numpy as np
5
6  def textToKeyMatrix(text, keySize):
7      # transform EncText to ndarray with shape (n, keySize)
8      return np.array(list(text)[:len(text)-len(text) %
           keySize]).reshape(int(len(text)/keySize), keySize)
9
10
11 def replaceNonAlphaChars(text):
12     # replace non alpha chars with dash
13     return re.sub(r'[^a-zA-Z]', '-', text)
14
15
16 def occurencePerColumn(matrix):
17     # calculate occurrences of each alphabetic character per column
18     totalOccurences = []
19     for i in range(matrix.shape[1]):
20         occurences = np.unique(matrix[:, i], return_counts=True)
21         # remove dashes from occurences if present
22         if '-' in occurences[0]:
23             occurences = (np.delete(occurences[0], np.where(
24                 occurences[0] == '-')), np.delete(occurences[1],
                        np.where(occurences[0] == '-')))
25         totalOccurences.append(occurences)
26     return totalOccurences
27
28
29 def getMostFreqCharPerColumn(occurencePerColumn):
30     # get most frequent character per column
31     return
           [occurencePerColumn[i][0][np.argmax(occurencePerColumn[i][1])]
           for i in range(len(occurencePerColumn))]
32
33
34 def stdTotalSum(occurrences):
35     # calculate the standard deviation of each column
36     stds = np.array([np.std([int(count) for count in occurrences[i][1]])
37                      for i in range(len(occurrences))])
38
39     # return sum of max occurrences
40     return np.sum(stds)
41
42
43 def getKeyShifts(mostFreqChars, mostFreqChar='e'):
```

```
    # get the shift of each character to the most frequent character
    return [ord(mostFreqChars[i]) - ord(mostFreqChar) for i in
        range(len(mostFreqChars))]


def keyShiftsToKey(keyShifts):
    # get the key from the key shifts
    return ''.join([chr((keyShifts[i] + 26) % 26 + ord('a')) for i in
        range(len(keyShifts))])


def printResultOf(input):
    minKeySize = int(input[0])
    maxKeySize = int(input[1])
    highestSTDSum = 0
    bestCharFreq = None
    # iterate over all possible key sizes
    for ks in range(minKeySize, maxKeySize + 1):
        charFreq = occurencePerColumn(
            textToKeyMatrix(replaceNonAlphaChars(input[2]), ks))
        totalSTDSum = stdTotalSum(charFreq)
        # keep track of the highest std sum
        if totalSTDSum > highestSTDSum:
            highestSTDSum = totalSTDSum
            bestCharFreq = charFreq
        print("The sum of %d std. devs: %s" %
              (ks, "{:.2f}".format(totalSTDSum)))

    print("Key guess:")
    print(keyShiftsToKey(getKeyShifts(getMostFreqCharPerColumn(bestCharFreq),
        'e')))


def handle_input(input):
    if len(input) >= 3:
        printResultOf(input)
    else:
        print("Invalid input")


def main():
    totalInput = []
    # get input text
    for line in sys.stdin:
        if line == '\n':
            break
        totalInput.append(line.strip())
    handle_input([totalInput[0], totalInput[1], ''.join(totalInput[2:])])

```

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