Documentation Crypthography Tool

Skill: Programming – Introduction Level

Fabien Ruf

Contents

1 Instructions

Prerequisites: Python 3

To see what the tool is capable of and in order to test it, follow these steps:

1. Download the whole folder and open the file **main.py** and run the programm.
   $\Longrightarrow$ The tool should be opened automatically in a new window.
2. Enter any text you want to encrypt. You can use letters (capital or small), umlauts, numbers, spaces, wrap and the usual special characters (, ; . : _ - ! ? % ” ’).
3. Enter a password of any length. The same characters are allowed to be used as befor. Show or hide the password by clicking on the Show button.
4. Press the ENCRYPT button. $\Longrightarrow$ Your text should now be displayed successfully encrypted below.
5. Copy the encrypted text and paste it into the top. Press DECRYPT and the decrypted text should appear at the bottom again. Choose a completely different password and press DECRYPT. The text should not be decrypted correctly.
6. Enter a character that you are not allowed to enter (e.g. @) and press ENCRYPT/DECRYPT.
7. Click on LOAD and go to the directory where you saved the tool. Select the file example.txt and open it. The content of the file was now directly imported into the program and can now be encrypted.

8. Choose a new password or have one generated randomly. You can choose the length yourself. Enter any letter (e.g. a) and see what happens.

9. Encrypt the text. It should now be displayed encrypted below. Click on SAVE to save the encrypted content again. It makes sense to overwrite the file immediately to remove the unencrypted text. Otherwise choose a new file name and click on Save. See the result by opening the text file with a text editor (Possibly activate line break in menu under format).

10. Go back to the tool and load the previously saved file and decrypt it with the correct key to see if it works accurately.

With this tool it is possible to encrypt and decrypt any text. The encryption is based on a self-definable key. It is also possible to directly import and encrypt entire .txt files.

2 Functionality

Basically all code is self-written, nothing existing was taken over. We start with the **main.py** file, where generally everything except the specific functions for the crypting system is written in.

2.1 GUI

We Import all necessary packages:

```
from tkinter import *
from tkinter import scrolledtext as st
from tkinter import messagebox
from tkinter import filedialog

import random
import string

import crypting
```

The package tkinter is used to create the GUI. For input and output I used special text fields with scroll mechanism and therefor needed to install scrolledtext. The package filedialog allows us to handle files. Messagebox is needed for warning messages. The packages random and string are used to generate a random key. Import crypting imports another own written Python file. There the whole mechanism of encryption is programmed and nothing else. This file is described in chapter 3.

I skip the functions for the first time and take a look at the GUI section. Here all single elements are created and positioned. The positioning is done with the .place() method. This makes you quite inflexible and static, but its sufficient for this project. The individual buttons indicate which functions are to be executed by clicking.

```
style=("Arial Bold", 10)
h=40

window = Tk()

window.title("Cryptography tool")
window.geometry("470x505")


lbl = Label(window, text="Enter the text you want to encrypt or decrypt here: (or load it)",
     font=style)
lbl.place(x=20, y=10, height=h, width=360)

```

```
txt = st.ScrolledText(window)
txt.focus()
txt.place(x=20,y=60, height=100, width=430)

lbl2 = Label(window, text="Enter any password here:", font=style)
lbl2.place(x= 20, y=170, height=h)

check_var = IntVar()
check_show_psw = Checkbutton(window, text = "Show", variable = check_var, \
                 onvalue = 1, offvalue = 0,command = show_hide_psd)
check_show_psw.place(y=170,x=200, height=h)

btn0 = Button(window, text="Generate", command=generatePassword)
btn0.place(y=170,x=380, height=h, width=70)

digits = Entry(window,justify='center')
digits.insert(INSERT, 8 )
digits.place(x=340,y=180, height=20, width=20)

ky = Entry(window, show='*')
ky.place(x=20,y=220, height=25, width=415)

btn1 = Button(window, text="ENCRYPT", command=on_encrypt_clicked)
btn1.place(y=265,x=20, height=h, width=70)

btn2 = Button(window, text="DECRYPT", command=on_decrypt_clicked)
btn2.place(y=265,x=150, height=h, width=70)

outp = st.ScrolledText(window)
outp.place(x=20, y=325, height=100, width=430)


btn4 = Button(window, text="Save", command=on_save_clicked)
btn4.place(x=380,y=445,height=h, width=70)

btn5 = Button(window, text="Load", command=on_load_clicked)
btn5.place(x=20, y=445, height=h, width=70)

window.mainloop()
```

2.2 GUI Functions

The function randomStringDigits() generates a random string of numbers and letters. The input specifies the length. The function returns a string.

generatePassword() is triggered by clicking the GENERATE button. It takes the input from the entry field, checks if it is a number, otherwise it triggers a warning message and is done. If not it triggers the function randomStringDigits() and gives the input with it. Then it deletes the input field for the key and inserts the received random key. It also generates a reminder message.

```
def randomStringDigits(stringLength):
    lettersAndDigits = string.ascii_letters + string.digits
    return ''.join(random.choice(lettersAndDigits) for i in range(stringLength))

def generatePassword():
    try:
        var=int(digits.get())
    except ValueError:
        messagebox.showinfo('Warning',"No Integer entered")
        return

    ky.delete(0, END)
    ky.insert(INSERT,randomStringDigits(var))
    check_show_psw.select()
```

```
    show_hide_psd()
    messagebox.showinfo('Warning',"Don't forget this password")
```

The function on_encrypt_clicked() is triggered by clicking the ENCRYPT button, and calls the function crypt() with input 0. The function on_decrypt_clicked() is triggered by clicking the ENCRYPT button, and calls the function crypt() with Input 1.

crypt() is the most important function. It takes the input from the input field and the key field and triggers the functions to encrypt or decrypt (depending on the input), and and gives with the text and key. Each of these functions returns a string that is stored in res. If there was an error during the functions (more on this later), this variable corresponds to 'error' and the function then triggers an error message. If the variable does not correspond to 'error', the string is inserted into the output field.

```
def on_encrypt_clicked():
    crypt(0)

def on_decrypt_clicked():
    crypt(1)

def crypt(p):
    text = txt.get(1.0, END).strip()
    key = ky.get().strip()

    outp.delete(1.0,END)

    if p == 0:
        res = crypting.encrypt(text,key) #calls function to encrypt
    else:
        res = crypting.decrypt(text,key) #calls function to encrypt

    if res == 'error':
        messagebox.showinfo('Warning', 'A character is invalid')
    else:
        outp.insert(INSERT,res)
```

on_load_clicked() is triggered by pressing the LOAD button. It clears the input field and allows the user to load a file. This file is read and inserted into the input field.

on_save_clicked() is triggered by pressing the SAVE button. It allows a file to be stored in a specific location. It reads out the contents of the output field and writes it into the file, which is then closed.

```
def on_load_clicked():
    txt.delete(1.0,END) #maybe need deletation
    filename = filedialog.askopenfilename( initialdir="C:/", title="select file",
                                          filetypes=(("text files", "*.txt"), ("all files",
     "*.*")))
    f = open(filename, "r")
    input= f.read()
    f.close()
    txt.insert(INSERT,input)

def on_save_clicked():
    f = filedialog.asksaveasfile(mode='w', title="Save the file",
            defaultextension=".txt", filetypes=(('all files', '.*'), ('text files', '.txt'))
    ,
                                 initialfile='cryptography_example')
    if f is None:
        return
    text2save = outp.get(1.0, END).strip()
    f.write(text2save)
    f.close()

    messagebox.showinfo('Save','The text has successfully been saved')
```

The function below changes the settings of the key input field by the status of the check button, to what

extent the input should be displayed (normal or with *).

```
def show_hide_psd():
    if(check_var.get()):
        ky.config(show="")
    else:
        ky.config(show="*")
```

2.3 Cryptioning

The encryption is based on the multiplicative cipher method and uses Caesar cipher technique. The text and the key are superimposed (the key simply repeats itself if it is shorter than the text). Each character is assigned a number. The individual characters of text and keys are converted into their numbers. These are added together and this number then specifies the character in the encryption. If the new number is larger than the table, it simply starts counting again, i.e. 15 is O but 30 is O as well.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Text:	**H**	**E**	**L**	**L**	**O**
	8	5	12	12	15
Key:	**H**	**O**	**I**	**H**	**O**
	8	15	9	8	15
	16	20	21	20	30
Encrypt:	**A**	**E**	**F**	**E**	**O**

The decryption takes place in exactly the opposite direction. An encrypted text is assigned to the corresponding numbers, and the numbers of the key are subtracted from it, which then converted again results in the decrypted text.

The functions regarding the encryption are written int the **crypting.py** file. First the code ist defined in a list. The list is derived from other lists for structural reasons. This code lists all characters that can be used with. However, it has a double function, because the index of a character in the list is the number assigned to it to crypt. Theoretically any character can be added here easily, and otherwise no adjustments have to be made. The numbers can also simply be perceived as characters to which a number (the index) is assigned.[1]

```
alphabet=['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']
umlauts=['ae ','oe','ue']
numbers=['1','2','3','4','5','6','7','8','9','0']
symbols=[',','.','!','?',':','-','_','"',"'",'%',' ']
reserved=['^','$']

code = alphabet + umlauts + symbols + reserved + numbers
```

The reserved symbols can not be used by the user, because they are needed in the encryption, to send additional informations (see below) with the crypted text. But if you want to use this symbols, just modify the list and change them with other symbols.

First there are some general functions that I have written and that are used both for encrypting and decrypting:

The following function replaces first of all text breaks corresponding to \n in Python with $, because I didn't want to have text breaks in the encrypted text and since this can also easly lead to errors when copying and

[1] It is not possible to display ä, ö, ü in this file here, because of LaTex. Of course in the code it is displayed as normal.

pasting. By replacing it I can give this information with the code and read it out again when decrypting and insert text breaks at the specific places. The fact that I can add additional information to the encrypted text, without visually seeing this process, I found interesting. Therefore, instead of including all capital letters in the code, I decided to make them in lower case and to insert the character în front of them.

```
def transformString (string):
    string = string.replace('\n','$')
    k=len(string)
    i=0
    while i < (k):
        if string[i].isupper() == True:
            string=string[0:i]+'^'+string[i].lower()+string[(i+1):k]
            k+=1
        i+=1
    return string
```

The next function takes the previously transformed string and creates a new list by assigning each character its number (the index of the character in the code list).

```
def CodeIt(string, lst_code):
    var= []
    for x in string:
        var.append(lst_code.index(x))
    return var
```

The calculateLists() function takes two lists of numbers and adds (encrypts) or subtracts (decrypts) the respective elements of the list and generates a new list.

```
def calculateLists(lst_input,task,lst_key):
    lst_var=[]
    i=0
    for x in range(len(lst_input)):

        if task == 'crypt':
            lst_var.append(lst_input[x]+lst_key[i])
        else:
            lst_var.append(lst_input[x]-lst_key[i])

        if (i+1) == len(lst_key): #when list from key is shorter, it starts again
            i=0
        else:
            i+=1

    return lst_var
```

Finally, the StringOutOfList() function converts the elements of a list into a string:

```
def StringOutOfList (lst):
    return ''.join(lst)
```

Finally, but probably most importantly, the actual functions for encrypting and decrypting come into play:

The encrypt() triggered function is the first function to transform both the text and the key (capital letters and breaks). Thereupon a check mechanism comes, whether the characters are contained in the code and otherwise sends back an error, which triggers an error message as explained above. Then both strings are converted into lists with the function CodeIt(). This triggers the calculateLists() function, which adds both lists together. For each number in this list, the corresponding character is taken after the index and added to a new list, which now contains the encrypted text. Finally, the function StringOutOfList() converts this list into a string and returns this string, which is then displayed in the output as described above.

```
def encrypt(txt,key):
    txt = transformString(txt)
    key = transformString(key)

```

```
    var = txt
    for _ in range(2):
        for x in var:
            if x not in code:
                error = 'error'
                return error
        var = key

    lst_txt = CodeIt(txt,code)
    lst_key = CodeIt(key,code)


    lst_added = calculateLists(lst_txt,'crypt',lst_key)


    lst_crypted=[]
    for x in lst_added:
        if x > (len(code)-1):
            x-=len(code)
        lst_crypted.append(code[x])

    isCrypted= StringOutOfList(lst_crypted)

    return isCrypted
```

The decrypt function for decrypting is similarly structured. Since the encrypted text already has the correct format, only the key is transformed. Then both are checked again with the code, if all characters are correct. Then both strings are converted into numbers with the CodeIt() function and processed with the calculateLists() function. This list with numbers must now be decrypted again with the code, capital letters and text breaks are made again. This list is then converted back into a string and returned.

```
def decrypt(crypted,key):
    key = transformString(key)

    var = key
    for _ in range(2):
        for x in var:
            if x not in code:
                error = 'error'
                return error
        var = crypted

    lst_crypted = CodeIt(crypted,code)
    lst_key = CodeIt(key,code)

    lst_dif = calculateLists(lst_crypted,'decrypt',lst_key)


    lst_decrypted=[]
    for x in lst_dif:
        if x < 0:
            x+= len(code)
        lst_decrypted.append(code[x])

    for x in lst_decrypted:
        if x == '^':
            k = lst_decrypted.index(x)+1
            lst_decrypted[k] = lst_decrypted[k].upper()
            lst_decrypted.remove(x)
        elif x == '$':
            k = lst_decrypted.index(x)
            lst_decrypted[k] ='\n'

    isDecrypted= StringOutOfList(lst_decrypted)

    return isDecrypted
```

3 Bugs and Improvements

If a .txt file is imported into the tool, errors can occur, because the function .read() for example cannot read umlauts (and certain symbols).

The GUI is very unflexible and ugly. The window is to small, so its not easy to read the texts

The error messages could be precise with the failure, but it was getting complex and the additional effect was not worth the effort.