Laporan Tugas Kecil 1 IF4020 Kriptografi Semester I Tahun 2020/2021

Ignatius Timothy Manullang / 13517044

Muhammad Rizki Fonna / 13516001

Source Code

main.py

```
import PySimpleGUI as sg
import vigenere as vg
import full_vigenere as fvg
import autokey_vigenere as avg
import affine as af
import vigenereExtended as vge
import enigma as e
import playfair as p
import hill as h
import myszkowskiTransposition as se

layout = [[sg.Text('Cipher method')],
            [sg.Radio('vigenere', "cipherMethod", default=True, size=(10,1), key='Vigenere'),
                sg.Radio('FullVigenere', "cipherMethod", key='FullVigenere'),
                sg.Radio('RunningKeyVigenere', "cipherMethod", key='RunningKeyVigenere'),
                sg.Radio('ExtendedVigenere', "cipherMethod", key='ExtendedVigenere'),
                sg.Radio('Playfair', "cipherMethod", key='Playfair'),
                sg.Radio('SuperEncryption', "cipherMethod", key='SuperEncryption'),
                sg.Radio('Affine', "cipherMethod", key='Affine'),
                sg.Radio('Hill', "cipherMethod", key='Hill'),
                sg.Radio('Enigma', "cipherMethod", key='Enigma')],
            [sg.Text('Print method')],
            [sg.Radio('NoSpace', "printMethod", default=True, size=(10,1), key='NoSpace'),
                sg.Radio('FiveChar', "printMethod", key='FiveChar')],
            [sg.Text('Encryption')],
            [sg.Text('Enter Plaintext:'), sg.Input(key='-PLAINTEXT_ENCRYPT-')],
            [sg.Text('Enter key:'), sg.Input(key='-KEY_ENCRYPT-')],
            [sg.Text('Enter m key (for Affine):'), sg.Input(key='-KEY_ENCRYPT_M-')],
            [sg.Text('Enter b key (for Affine):'), sg.Input(key='-KEY_ENCRYPT_B-')],
            [sg.Text('Enter Input Filename (Path):'), sg.Input(key='-PATH_SOURCE_ENCRYPT-')],
            [sg.Text('Enter Output Filename (Path):'), sg.Input(key='-PATH_ENCRYPT-')],
            [sg.Text('Ciphertext:'), sg.Text(size=(30,1), key='-CIPHERTEXT_ENCRYPT-')],
            [sg.Button('Encrypt'), sg.Button('Encrypt Text File'), sg.Button('Encrypt File, Output File'), sg.Button('Encrypt from Text File, Output into Text File'), sg.Button("Encrypt Output into Text File"), sg.Button('Exit')],
            [sg.Text('Decryption')],
            [sg.Text('Enter Ciphertext:'), sg.Input(key='-CIPHERTEXT_DECRYPT-')],
            [sg.Text('Enter key:'), sg.Input(key='-KEY_DECRYPT-')],
            [sg.Text('Enter m key (for Affine):'), sg.Input(key='-KEY_DECRYPT_M-')],
            [sg.Text('Enter b key (for Affine):'), sg.Input(key='-KEY_DECRYPT_B-')],
            [sg.Text('Enter Input Filename (Path):'), sg.Input(key='-PATH_SOURCE_DECRYPT-')],
            [sg.Text('Enter Output Filename (Path):'), sg.Input(key='-PATH_DECRYPT-')],
            [sg.Text('Plaintext:'), sg.Text(size=(30,1), key='-PLAINTEXT_DECRYPT-')],
            [sg.Button('Decrypt'), sg.Button('Decrypt Text File'), sg.Button('Decrypt File, Output File'), sg.Button('Decrypt from Text File, Output
```

```
    into Text File'), sg.Button("Decrypt Output into Text File"), sg.Button('Exit')]]
43.
44.
45. window = sg.Window('Cryptography Program', layout)
46.
47. def enigmaEncryptDecryptInit():
48.     alphabet= "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
49.     alphabetList = list(alphabet)
50.     steckerbrettDictionary = {' ': ' '}
51.     pairsInSteckerbrett = sg.popup_get_text('Input the number of pairs in
    steckerbrett (min 0)', 'Input the number of pairs in steckerbrett')
52.     for i in range(int(pairsInSteckerbrett)):
53.         firstSteckerbrettAlphabet = sg.popup_get_text("Pair #" + str(i+1),"Input
    the first alphabet in pair: ")
54.         secondSteckerbrettAlphabet = sg.popup_get_text("Pair #" + str(i+1), "Input
    the second alphabet in pair: ")
55.         steckerbrettDictionary[firstSteckerbrettAlphabet.upper()] =
    secondSteckerbrettAlphabet.upper()
56.     #print("Input alpha, beta and gamma rotor s")
57.     alphaRotor = int(sg.popup_get_text("Input Alpha Rotor shift (0-25)", "Input
    Alpha Rotor shift (0-25)"))
58.     betaRotor = int(sg.popup_get_text("Input Beta Rotor shift (0-25)", "Input Beta
    Rotor shift (0-25)"))
59.     gammaRotor = int(sg.popup_get_text("Input Gamma Rotor shift (0-25)", "Input
    Gamma Rotor shift (0-25)"))
60.     return alphabetList, steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor
61.
62. while True:
63.     event, values = window.read()
64.     if event is None or event == 'Exit':
65.         break
66.     elif event=='Encrypt':
67.         if values['Vigenere']:
68.             if values['FiveChar']:
69.
    window['-CIPHERTEXT_ENCRYPT-'].update(vg.wrapFiveCharacters(vg.encryption(values['
    -PLAINTEXT_ENCRYPT-'], values['-KEY_ENCRYPT-'], False, '', False)))
70.             else:
71.
    window['-CIPHERTEXT_ENCRYPT-'].update(vg.encryption(values['-PLAINTEXT_ENCRYPT-'],
    values['-KEY_ENCRYPT-'], False, '', False))
72.         if values['FullVigenere']:
73.             if values['FiveChar']:
74.
    window['-CIPHERTEXT_ENCRYPT-'].update(fvg.wrapFiveCharacters(fvg.encryption(values
    ['-PLAINTEXT_ENCRYPT-'], values['-KEY_ENCRYPT-'], False, '', False)))
75.             else:
76.
    window['-CIPHERTEXT_ENCRYPT-'].update(fvg.encryption(values['-PLAINTEXT_ENCRYPT-']
    , values['-KEY_ENCRYPT-'], False, '', False))
77.         if values['RunningKeyVigenere']:
78.             if values['FiveChar']:
79.
    window['-CIPHERTEXT_ENCRYPT-'].update(avg.wrapFiveCharacters(avg.encryption(values
    ['-PLAINTEXT_ENCRYPT-'], values['-KEY_ENCRYPT-'], False, '', False)))
80.             else:
81.
    window['-CIPHERTEXT_ENCRYPT-'].update(avg.encryption(values['-PLAINTEXT_ENCRYPT-']
    , values['-KEY_ENCRYPT-'], False, '', False))
82.         if values['ExtendedVigenere']:
83.             message = values['-PLAINTEXT_ENCRYPT-']
84.             key = values['-KEY_ENCRYPT-']
85.             if values['FiveChar']:
86.
    window['-CIPHERTEXT_ENCRYPT-'].update(vge.wrapFiveCharacters(vge.encryptTextExtend
```

```
    edVigenere(message, key)))
87.             else:
88.
    window['-CIPHERTEXT_ENCRYPT-'].update(vge.encryptTextExtendedVigenere(message,
    key))
89.         if values['Playfair']:
90.             message = values['-PLAINTEXT_ENCRYPT-']
91.             key = values['-KEY_ENCRYPT-']
92.             if values['FiveChar']:
93.
    window['-CIPHERTEXT_ENCRYPT-'].update(p.wrapFiveCharacters(p.toUpperCase(p.playfai
    rEncrypt(message, key))))
94.             else:
95.
    window['-CIPHERTEXT_ENCRYPT-'].update(p.toUpperCase(p.playfairEncrypt(message,
    key)))
96.         if values['SuperEncryption']:
97.             if values['FiveChar']:
98.
    window['-CIPHERTEXT_ENCRYPT-'].update(se.wrapFiveCharacters(se.myszkowskiTransposi
    tionEncrypt(vg.encryption(values['-PLAINTEXT_ENCRYPT-'], values['-KEY_ENCRYPT-'],
    False, '', False), values['-KEY_ENCRYPT-'])))
99.             else:
100.
    window['-CIPHERTEXT_ENCRYPT-'].update(se.myszkowskiTranspositionEncrypt(vg.encrypt
    ion(values['-PLAINTEXT_ENCRYPT-'], values['-KEY_ENCRYPT-'], False, '', False),
    values['-KEY_ENCRYPT-']))
101.            if values['Affine']:
102.                if values['FiveChar']:
103.
    window['-CIPHERTEXT_ENCRYPT-'].update(af.wrapFiveCharacters(af.encryption(values['
    -PLAINTEXT_ENCRYPT-'], values['-KEY_ENCRYPT_M-'], values['-KEY_ENCRYPT_B-'],
    False, '', False)))
104.                else:
105.
    window['-CIPHERTEXT_ENCRYPT-'].update(af.encryption(values['-PLAINTEXT_ENCRYPT-'],
    values['-KEY_ENCRYPT_M-'], values['-KEY_ENCRYPT_B-'], False, '', False))
106.            if values['Hill']:
107.                message = values['-PLAINTEXT_ENCRYPT-']
108.                key = values['-KEY_ENCRYPT-']
109.                resultMessage = h.generateHillResultMessage(message, key, encrypt
    = True)
110.                if values['FiveChar']:
111.
    window['-CIPHERTEXT_ENCRYPT-'].update(h.wrapFiveCharacters(resultMessage))
112.                else:
113.                    window['-CIPHERTEXT_ENCRYPT-'].update(resultMessage)
114.            if values['Enigma']:
115.                text = values['-PLAINTEXT_ENCRYPT-']
116.                alphabetList, steckerbrettDictionary, alphaRotor, betaRotor,
    gammaRotor = enigmaEncryptDecryptInit()
117.                print("gammaRotor = ", gammaRotor)
118.                if values['FiveChar']:
119.
    window['-CIPHERTEXT_ENCRYPT-'].update(e.wrapFiveCharacters(e.encryptDecrypt(text,
    steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor, alphabetList)))
120.                else:
121.                    window['-CIPHERTEXT_ENCRYPT-'].update(e.encryptDecrypt(text,
    steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor, alphabetList))
122.        elif event=='Encrypt Text File':
123.            if values['Vigenere']:
124.                if values['FiveChar']:
125.
    window['-CIPHERTEXT_ENCRYPT-'].update(vg.wrapFiveCharacters(vg.encryption('',
    values['-KEY_ENCRYPT-'], True, values['-PATH_SOURCE_ENCRYPT-'])))
```

```
                else:
                    window['-CIPHERTEXT_ENCRYPT-'].update(vg.encryption('',
values['-KEY_ENCRYPT-'], True, values['-PATH_SOURCE_ENCRYPT-']))
            if values['FullVigenere']:
                if values['FiveChar']:

window['-CIPHERTEXT_ENCRYPT-'].update(fvg.wrapFiveCharacters(fvg.encryption('',
values['-KEY_ENCRYPT-'], True, values['-PATH_SOURCE_ENCRYPT-'])))
                else:
                    window['-CIPHERTEXT_ENCRYPT-'].update(fvg.encryption('',
values['-KEY_ENCRYPT-'], True, values['-PATH_SOURCE_ENCRYPT-']))
            if values['RunningKeyVigenere']:
                if values['FiveChar']:

window['-CIPHERTEXT_ENCRYPT-'].update(avg.wrapFiveCharacters(avg.encryption('',
values['-KEY_ENCRYPT-'], True, values['-PATH_SOURCE_ENCRYPT-'])))
                else:
                    window['-CIPHERTEXT_ENCRYPT-'].update(avg.encryption('',
values['-KEY_ENCRYPT-'], True, values['-PATH_SOURCE_ENCRYPT-']))
            if values['ExtendedVigenere']:
                message = e.readTextFromFile(values['-PATH_SOURCE_ENCRYPT-'])
                key = values['-KEY_ENCRYPT-']
                if values['FiveChar']:

window['-CIPHERTEXT_ENCRYPT-'].update(vge.wrapFiveCharacters(vge.encryptTextExtend
edVigenere(message, key)))
                else:

window['-CIPHERTEXT_ENCRYPT-'].update(vge.encryptTextExtendedVigenere(message,
key))
            if values['Playfair']:
                message = e.readTextFromFile(values['-PATH_SOURCE_ENCRYPT-'])
                key = values['-KEY_ENCRYPT-']
                if values['FiveChar']:

window['-CIPHERTEXT_ENCRYPT-'].update(p.wrapFiveCharacters(p.toUpperCase(p.playfai
rEncrypt(message, key))))
                else:

window['-CIPHERTEXT_ENCRYPT-'].update(p.toUpperCase(p.playfairEncrypt(message,
key)))
            if values['SuperEncryption']:
                if values['FiveChar']:

window['-CIPHERTEXT_ENCRYPT-'].update(se.wrapFiveCharacters(se.myszkowskiTransposi
tionEncrypt(vg.encryption('', values['-KEY_ENCRYPT-'], True,
values['-PATH_SOURCE_ENCRYPT-']), values['-KEY_ENCRYPT-'])))
                else:

window['-CIPHERTEXT_ENCRYPT-'].update(se.myszkowskiTranspositionEncrypt(vg.encrypt
ion('', values['-KEY_ENCRYPT-'], True, values['-PATH_SOURCE_ENCRYPT-']),
values['-KEY_ENCRYPT-']))
            if values['Affine']:
                if values['FiveChar']:

window['-CIPHERTEXT_ENCRYPT-'].update(af.wrapFiveCharacters(af.encryption('',
values['-KEY_ENCRYPT_M-'], values['-KEY_ENCRYPT_B-'], True,
values['-PATH_SOURCE_ENCRYPT-'])))
                else:
                    window['-CIPHERTEXT_ENCRYPT-'].update(af.encryption('',
values['-KEY_ENCRYPT_M-'], values['-KEY_ENCRYPT_B-'], True,
values['-PATH_SOURCE_ENCRYPT-']))
            if values['Hill']:
                message = e.readTextFromFile(values['-PATH_SOURCE_ENCRYPT-'])
                key = values['-KEY_ENCRYPT-']
```

```
                resultMessage = h.generateHillResultMessage(message, key, encrypt = True)
                if values['FiveChar']:
window['-CIPHERTEXT_ENCRYPT-'].update(h.wrapFiveCharacters(resultMessage))
                else:
                    window['-CIPHERTEXT_ENCRYPT-'].update(resultMessage)
            if values['Enigma']:
                text = e.readTextFromFile(values['-PATH_SOURCE_ENCRYPT-'])
                alphabetList, steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor = enigmaEncryptDecryptInit()
                if values['FiveChar']:
window['-CIPHERTEXT_ENCRYPT-'].update(e.wrapFiveCharacters(e.encryptDecrypt(text, steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor, alphabetList)))
                else:
                    window['-CIPHERTEXT_ENCRYPT-'].update(e.encryptDecrypt(text, steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor, alphabetList))

        elif event=='Encrypt File, Output File':
            if values['ExtendedVigenere']:
                listOfBytes = []
                with open(values['-PATH_SOURCE_ENCRYPT-'], "rb") as f:
                    while (byte := f.read(1)):
                        listOfBytes.append(byte)
                f.close()
                keyword = values['-KEY_ENCRYPT-']
                key = vge.generateKey(listOfBytes, keyword)
                cipherText = vge.encryptByteExtendedVigenere(listOfBytes, key)
                g = open(values['-PATH_ENCRYPT-'], "wb+")
                for index in range(len(cipherText)):
                    g.write(cipherText[index])
                g.close()

        elif event=='Encrypt from Text File, Output into Text File':
            if values['Vigenere']:
                if values['FiveChar']:
window['-CIPHERTEXT_ENCRYPT-'].update(vg.write_to_file(values['PATH_ENCRYPT'], vg.wrapFiveCharacters(vg.encryption('', values['-KEY_ENCRYPT-'], True, values['-PATH_SOURCE_ENCRYPT-']))))
                else:
window['-CIPHERTEXT_ENCRYPT-'].update(vg.write_to_file(values['PATH_ENCRYPT'], vg.encryption('', values['-KEY_ENCRYPT-'], True, values['-PATH_SOURCE_ENCRYPT-'])))
            if values['FullVigenere']:
                if values['FiveChar']:
window['-CIPHERTEXT_ENCRYPT-'].update(fvg.write_to_file(values['PATH_ENCRYPT'], fvg.wrapFiveCharacters(fvg.encryption('', values['-KEY_ENCRYPT-'], True, values['-PATH_SOURCE_ENCRYPT-']))))
                else:
window['-CIPHERTEXT_ENCRYPT-'].update(fvg.write_to_file(values['PATH_ENCRYPT'], fvg.encryption('', values['-KEY_ENCRYPT-'], True, values['-PATH_SOURCE_ENCRYPT-'])))
            if values['RunningKeyVigenere']:
                if values['FiveChar']:
window['-CIPHERTEXT_ENCRYPT-'].update(avg.write_to_file(values['PATH_ENCRYPT'], avg.wrapFiveCharacters(avg.encryption('', values['-KEY_ENCRYPT-'], True, values['-PATH_SOURCE_ENCRYPT-']))))
                else:

```

```
    window['-CIPHERTEXT_ENCRYPT-'].update(avg.write_to_file(values['PATH_ENCRYPT'],
    avg.encryption('', values['-KEY_ENCRYPT-'], True,
    values['-PATH_SOURCE_ENCRYPT-'])), values['-PATH_ENCRYPT'])
209.            if values['ExtendedVigenere']:
210.                message = e.readTextFromFile(values['-PATH_SOURCE_ENCRYPT-'])
211.                key = values['-KEY_ENCRYPT-']
212.                if values['FiveChar']:
213.
    window['-CIPHERTEXT_ENCRYPT-'].update(vge.write_to_file(values['-PATH_ENCRYPT-'],
    vge.wrapFiveCharacters(vge.encryptTextExtendedVigenere(message, key))))
214.                else:
215.
    window['-CIPHERTEXT_ENCRYPT-'].update(vge.write_to_file(values['-PATH_ENCRYPT-'],
    vge.encryptTextExtendedVigenere(message, key)))
216.            if values['Playfair']:
217.                message = e.readTextFromFile(values['-PATH_SOURCE_ENCRYPT-'])
218.                key = values['-KEY_ENCRYPT-']
219.                if values['FiveChar']:
220.
    window['-CIPHERTEXT_ENCRYPT-'].update(p.write_to_file(values['-PATH_ENCRYPT-'],
    p.wrapFiveCharacters(p.toUpperCase(p.playfairEncrypt(message, key)))))
221.                else:
222.
    window['-CIPHERTEXT_ENCRYPT-'].update(p.write_to_file(values['-PATH_ENCRYPT-'],
    p.toUpperCase(p.playfairEncrypt(message, key))))
223.            if values['SuperEncryption']:
224.                if values['FiveChar']:
225.
    window['-CIPHERTEXT_ENCRYPT-'].update(se.write_to_file(values['-PATH_ENCRYPT-'],
    se.wrapFiveCharacters(se.myszkowskiTranspositionEncrypt(vg.encryption('',
    values['-KEY_ENCRYPT-'], True, values['-PATH_SOURCE_ENCRYPT-']),
    values['-KEY_ENCRYPT-']))))
226.                else:
227.
    window['-CIPHERTEXT_ENCRYPT-'].update(se.write_to_file(values['-PATH_ENCRYPT-'],
    se.myszkowskiTranspositionEncrypt(vg.encryption('', values['-KEY_ENCRYPT-'], True,
    values['-PATH_SOURCE_ENCRYPT-']), values['-KEY_ENCRYPT-'])))
228.            if values['Affine']:
229.                if values['FiveChar']:
230.
    window['-CIPHERTEXT_ENCRYPT-'].update(af.write_to_file(values['-PATH_ENCRYPT-'],
    af.wrapFiveCharacters(af.encryption('', values['-KEY_ENCRYPT_M-'],
    values['-KEY_ENCRYPT_B-'], True, values['-PATH_SOURCE_ENCRYPT-']))))
231.                else:
232.
    window['-CIPHERTEXT_ENCRYPT-'].update(af.write_to_file(values['-PATH_ENCRYPT-'],
    af.encryption('', values['-KEY_ENCRYPT_M-'], values['-KEY_ENCRYPT_B-'], True,
    values['-PATH_SOURCE_ENCRYPT-'])))
233.            if values['Hill']:
234.                message = e.readTextFromFile(values['-PATH_SOURCE_ENCRYPT-'])
235.                key = values['-KEY_ENCRYPT-']
236.                resultMessage = h.generateHillResultMessage(message, key, encrypt
    = True)
237.                if values['FiveChar']:
238.
    window['-CIPHERTEXT_ENCRYPT-'].update(h.write_to_file(values['-PATH_ENCRYPT-'],
    h.wrapFiveCharacters(resultMessage)))
239.                else:
240.
    window['-CIPHERTEXT_ENCRYPT-'].update(h.write_to_file(values['-PATH_ENCRYPT-'],
    resultMessage))
241.            if values['Enigma']:
242.                text = e.readTextFromFile(values['-PATH_SOURCE_ENCRYPT-'])
243.                alphabetList, steckerbrettDictionary, alphaRotor, betaRotor,
    gammaRotor = enigmaEncryptDecryptInit()
```

```
                if values['FiveChar']:
window['-CIPHERTEXT_ENCRYPT-'].update(e.write_to_file(values['-PATH_ENCRYPT-'], e.wrapFiveCharacters(e.encryptDecrypt(text, steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor, alphabetList))))
                else:
window['-CIPHERTEXT_ENCRYPT-'].update(e.write_to_file(values['-PATH_ENCRYPT-'], e.encryptDecrypt(text, steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor, alphabetList)))

        elif event=='Encrypt Output into Text File':
            if values['Vigenere']:
                if values['FiveChar']:
window['-CIPHERTEXT_ENCRYPT-'].update(vg.write_to_file(values['-PATH_ENCRYPT-'], vg.wrapFiveCharacters(vg.encryption(values['-PLAINTEXT_ENCRYPT-'], values['-KEY_ENCRYPT-'], False, '', False))))
                else:
window['-CIPHERTEXT_ENCRYPT-'].update(vg.write_to_file(values['-PATH_ENCRYPT-'], vg.encryption(values['-PLAINTEXT_ENCRYPT-'], values['-KEY_ENCRYPT-'], False, '', False)))
            if values['FullVigenere']:
                if values['FiveChar']:
window['-CIPHERTEXT_ENCRYPT-'].update(fvg.write_to_file(values['-PATH_ENCRYPT-'], fvg.wrapFiveCharacters(fvg.encryption(values['-PLAINTEXT_ENCRYPT-'], values['-KEY_ENCRYPT-'], False, '', False))))
                else:
window['-CIPHERTEXT_ENCRYPT-'].update(fvg.write_to_file(values['-PATH_ENCRYPT-'], fvg.encryption(values['-PLAINTEXT_ENCRYPT-'], values['-KEY_ENCRYPT-'], False, '', False)))
            if values['RunningKeyVigenere']:
                if values['FiveChar']:
window['-CIPHERTEXT_ENCRYPT-'].update(avg.write_to_file(values['-PATH_ENCRYPT-'], avg.wrapFiveCharacters(avg.encryption(values['-PLAINTEXT_ENCRYPT-'], values['-KEY_ENCRYPT-'], False, '', False))))
                else:
window['-CIPHERTEXT_ENCRYPT-'].update(avg.write_to_file(values['-PATH_ENCRYPT-'], avg.encryption(values['-PLAINTEXT_ENCRYPT-'], values['-KEY_ENCRYPT-'], False, '', False)))
            if values['ExtendedVigenere']:
                message = values['-PLAINTEXT_ENCRYPT-']
                key = values['-KEY_ENCRYPT-']
                if values['FiveChar']:
window['-CIPHERTEXT_ENCRYPT-'].update(vge.write_to_file(values['-PATH_ENCRYPT-'], vge.wrapFiveCharacters(vge.encryptTextExtendedVigenere(message, key))))
                else:
window['-CIPHERTEXT_ENCRYPT-'].update(vge.write_to_file(values['-PATH_ENCRYPT-'], vge.encryptTextExtendedVigenere(message, key)))
            if values['Playfair']:
                message = values['-PLAINTEXT_ENCRYPT-']
                key = values['-KEY_ENCRYPT-']
                if values['FiveChar']:
window['-CIPHERTEXT_ENCRYPT-'].update(p.write_to_file(values['-PATH_ENCRYPT-'], p.wrapFiveCharacters(p.toUpperCase(p.playfairEncrypt(message, key)))))
                else:

```

```
    window['-CIPHERTEXT_ENCRYPT-'].update(p.write_to_file(values['-PATH_ENCRYPT-'],
    p.toUpperCase(p.playfairEncrypt(message, key))))
279.            if values['SuperEncryption']:
280.                if values['FiveChar']:
281.
    window['-CIPHERTEXT_ENCRYPT-'].update(se.write_to_file(values['-PATH_ENCRYPT-'],
    se.wrapFiveCharacters(se.myszkowskiTranspositionEncrypt(vg.encryption(values['-PLA
    INTEXT_ENCRYPT-'], values['-KEY_ENCRYPT-'], False, '', False),
    values['-KEY_ENCRYPT-']))))
282.                else:
283.
    window['-CIPHERTEXT_ENCRYPT-'].update(se.write_to_file(values['-PATH_ENCRYPT-'],
    se.myszkowskiTranspositionEncrypt(vg.encryption(values['-PLAINTEXT_ENCRYPT-'],
    values['-KEY_ENCRYPT-'], False, '', False), values['-KEY_ENCRYPT-'])))
284.            if values['Affine']:
285.                if values['FiveChar']:
286.
    window['-CIPHERTEXT_ENCRYPT-'].update(af.write_to_file(values['-PATH_ENCRYPT-'],
    af.wrapFiveCharacters(af.encryption(values['-PLAINTEXT_ENCRYPT-'],
    values['-KEY_ENCRYPT_M-'], values['-KEY_ENCRYPT_B-'], False, '', False))))
287.                else:
288.
    window['-CIPHERTEXT_ENCRYPT-'].update(af.write_to_file(values['-PATH_ENCRYPT-'],
    af.encryption(values['-PLAINTEXT_ENCRYPT-'], values['-KEY_ENCRYPT_M-'],
    values['-KEY_ENCRYPT_B-'], False, '', False)))
289.            if values['Hill']:
290.                message = values['-PLAINTEXT_ENCRYPT-']
291.                key = values['-KEY_ENCRYPT-']
292.                resultMessage = h.generateHillResultMessage(message, key, encrypt
    = True)
293.                if values['FiveChar']:
294.
    window['-CIPHERTEXT_ENCRYPT-'].update(h.write_to_file(values['-PATH_ENCRYPT-'],
    h.wrapFiveCharacters(resultMessage)))
295.                else:
296.
    window['-CIPHERTEXT_ENCRYPT-'].update(h.write_to_file(values['-PATH_ENCRYPT-'],
    resultMessage))
297.            if values['Enigma']:
298.                text = values['-PLAINTEXT_ENCRYPT-']
299.                alphabetList, steckerbrettDictionary, alphaRotor, betaRotor,
    gammaRotor = enigmaEncryptDecryptInit()
300.                if values['FiveChar']:
301.
    window['-CIPHERTEXT_ENCRYPT-'].update(e.write_to_file(values['-PATH_ENCRYPT-'],
    e.wrapFiveCharacters(e.encryptDecrypt(text, steckerbrettDictionary, alphaRotor,
    betaRotor, gammaRotor, alphabetList))))
302.                else:
303.
    window['-CIPHERTEXT_ENCRYPT-'].update(e.write_to_file(values['-PATH_ENCRYPT-'],
    e.encryptDecrypt(text, steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor,
    alphabetList)))
304.
305.        elif event=='Decrypt':
306.            if values['Vigenere']:
307.                if values['FiveChar']:
308.
    window['-PLAINTEXT_DECRYPT-'].update(vg.wrapFiveCharacters(vg.decryption(values['-
    CIPHERTEXT_DECRYPT-'], values['-KEY_DECRYPT-'])))
309.                else:
310.
    window['-PLAINTEXT_DECRYPT-'].update(vg.decryption(values['-CIPHERTEXT_DECRYPT-'],
    values['-KEY_DECRYPT-']))
311.            if values['FullVigenere']:
312.                if values['FiveChar']:
```

```
313.
    window['-PLAINTEXT_DECRYPT-'].update(fvg.wrapFiveCharacters(fvg.decryption(values[
    '-CIPHERTEXT_DECRYPT-'], values['-KEY_DECRYPT-'])))
314.                 else:
315.
    window['-PLAINTEXT_DECRYPT-'].update(fvg.decryption(values['-CIPHERTEXT_DECRYPT-']
    , values['-KEY_DECRYPT-']))
316.             if values['RunningKeyVigenere']:
317.                 if values['FiveChar']:
318.
    window['-PLAINTEXT_DECRYPT-'].update(avg.wrapFiveCharacters(avg.decryption(values[
    '-CIPHERTEXT_DECRYPT-'], values['-KEY_DECRYPT-'])))
319.                 else:
320.
    window['-PLAINTEXT_DECRYPT-'].update(avg.decryption(values['-CIPHERTEXT_DECRYPT-']
    , values['-KEY_DECRYPT-']))
321.             if values['ExtendedVigenere']:
322.                 message = values['-CIPHERTEXT_DECRYPT-']
323.                 key = values['-KEY_DECRYPT-']
324.                 if values['FiveChar']:
325.
    window['-PLAINTEXT_DECRYPT-'].update(vge.wrapFiveCharacters(vge.decryptTextExtende
    dVigenere(message, key)))
326.                 else:
327.
    window['-PLAINTEXT_DECRYPT-'].update(vge.decryptTextExtendedVigenere(message,
    key))
328.             if values['Playfair']:
329.                 message = values['-CIPHERTEXT_DECRYPT-']
330.                 key = values['-KEY_DECRYPT-']
331.                 if values['FiveChar']:
332.
    window['-PLAINTEXT_DECRYPT-'].update(p.wrapFiveCharacters(p.toUpperCase(p.playfair
    Decrypt(message, key))))
333.                 else:
334.
    window['-PLAINTEXT_DECRYPT-'].update(p.toUpperCase(p.playfairDecrypt(message,
    key)))
335.             if values['SuperEncryption']:
336.                 if values['FiveChar']:
337.
    window['-PLAINTEXT_DECRYPT-'].update(vg.wrapFiveCharacters(vg.decryption(se.myszko
    wskiTranspositionDecrypt(values['-CIPHERTEXT_DECRYPT-'], values['-KEY_DECRYPT-']),
    values['-KEY_DECRYPT-'])))
338.                 else:
339.
    window['-PLAINTEXT_DECRYPT-'].update(vg.decryption(se.myszkowskiTranspositionDecry
    pt(values['-CIPHERTEXT_DECRYPT-'], values['-KEY_DECRYPT-']),
    values['-KEY_DECRYPT-']))
340.             if values['Affine']:
341.                 if values['FiveChar']:
342.
    window['-PLAINTEXT_DECRYPT-'].update(af.wrapFiveCharacters(af.decryption(values['-
    CIPHERTEXT_DECRYPT-'], values['-KEY_DECRYPT_M-'], values['-KEY_DECRYPT_B-'],
    False, '', False)))
343.                 else:
344.
    window['-PLAINTEXT_DECRYPT-'].update(af.decryption(values['-CIPHERTEXT_DECRYPT-'],
    values['-KEY_DECRYPT_M-'], values['-KEY_DECRYPT_B-'], False, '', False))
345.             if values['Hill']:
346.                 message = values['-CIPHERTEXT_DECRYPT-']
347.                 key = values['-KEY_DECRYPT-']
348.                 resultMessage = h.generateHillResultMessage(message, key, encrypt
    = False)
349.                 if values['FiveChar']:
```

```
350.
    window['-PLAINTEXT_DECRYPT-'].update(h.wrapFiveCharacters(resultMessage))
351.                 else:
352.                     window['-PLAINTEXT_DECRYPT-'].update(resultMessage)
353.             if values['Enigma']:
354.                 text = values['-CIPHERTEXT_DECRYPT-']
355.                 alphabetList, steckerbrettDictionary, alphaRotor, betaRotor,
    gammaRotor = enigmaEncryptDecryptInit()
356.                 if values['FiveChar']:
357.
    window['-PLAINTEXT_DECRYPT-'].update(e.wrapFiveCharacters(e.encryptDecrypt(text,
    steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor, alphabetList)))
358.                 else:
359.                     window['-PLAINTEXT_DECRYPT-'].update(e.encryptDecrypt(text,
    steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor, alphabetList))
360.
361.         elif event=='Decrypt Text File':
362.             if values['Vigenere']:
363.                 if values['FiveChar']:
364.
    window['-PLAINTEXT_DECRYPT-'].update(vg.wrapFiveCharacters(vg.decryption('',
    values['-KEY_DECRYPT-'], True, values['-PATH_SOURCE_DECRYPT-'])))
365.                 else:
366.                     window['-PLAINTEXT_DECRYPT-'].update(vg.decryption('',
    values['-KEY_DECRYPT-'], True, values['-PATH_SOURCE_DECRYPT-']))
367.             if values['FullVigenere']:
368.                 if values['FiveChar']:
369.
    window['-PLAINTEXT_DECRYPT-'].update(fvg.wrapFiveCharacters(fvg.decryption('',
    values['-KEY_DECRYPT-'], True, values['-PATH_SOURCE_DECRYPT-'])))
370.                 else:
371.                     window['-PLAINTEXT_DECRYPT-'].update(fvg.decryption('',
    values['-KEY_DECRYPT-'], True, values['-PATH_SOURCE_DECRYPT-']))
372.             if values['RunningKeyVigenere']:
373.                 if values['FiveChar']:
374.
    window['-PLAINTEXT_DECRYPT-'].update(avg.wrapFiveCharacters(avg.decryption('',
    values['-KEY_DECRYPT-'], True, values['-PATH_SOURCE_DECRYPT-'])))
375.                 else:
376.                     window['-PLAINTEXT_DECRYPT-'].update(avg.decryption('',
    values['-KEY_DECRYPT-'], True, values['-PATH_SOURCE_DECRYPT-']))
377.             if values['ExtendedVigenere']:
378.                 message = e.readTextFromFile(values['-PATH_SOURCE_DECRYPT-'])
379.                 key = values['-KEY_DECRYPT-']
380.                 if values['FiveChar']:
381.
    window['-PLAINTEXT_DECRYPT-'].update(vge.wrapFiveCharacters(vge.decryptTextExtende
    dVigenere(message, key)))
382.                 else:
383.
    window['-PLAINTEXT_DECRYPT-'].update(vge.decryptTextExtendedVigenere(message,
    key))
384.             if values['Playfair']:
385.                 message = e.readTextFromFile(values['-PATH_SOURCE_DECRYPT-'])
386.                 key = values['-KEY_DECRYPT-']
387.                 if values['FiveChar']:
388.
    window['-PLAINTEXT_DECRYPT-'].update(p.wrapFiveCharacters(p.toUpperCase(p.playfair
    Decrypt(message, key))))
389.                 else:
390.
    window['-PLAINTEXT_DECRYPT-'].update(p.toUpperCase(p.playfairDecrypt(message,
    key)))
391.             if values['SuperEncryption']:
392.                 if values['FiveChar']:
```

```
393.
    window['-PLAINTEXT_DECRYPT-'].update(se.wrapFiveCharacters(se.myszkowskiTransposit
    ionDecrypt(vg.decryption('', values['-KEY_DECRYPT-'], True,
    values['-PATH_SOURCE_DECRYPT-']), values['-KEY_DECRYPT-'])))
394.                 else:
395.
    window['-PLAINTEXT_DECRYPT-'].update(se.myszkowskiTranspositionDecrypt(vg.decrypti
    on('', values['-KEY_DECRYPT-'], True, values['-PATH_SOURCE_DECRYPT-']),
    values['-KEY_DECRYPT-']))
396.             if values['Affine']:
397.                 if values['FiveChar']:
398.
    window['-PLAINTEXT_DECRYPT-'].update(af.wrapFiveCharacters(af.decryption('',
    values['-KEY_DECRYPT_M-'], values['-KEY_DECRYPT_B-'], True,
    values['-PATH_SOURCE_DECRYPT-'])))
399.                 else:
400.                     window['-PLAINTEXT_DECRYPT-'].update(af.decryption('',
    values['-KEY_DECRYPT_M-'], values['-KEY_DECRYPT_B-'], True,
    values['-PATH_SOURCE_DECRYPT-']))
401.             if values['Hill']:
402.                 message = e.readTextFromFile(values['-PATH_SOURCE_DECRYPT-'])
403.                 key = values['-KEY_DECRYPT-']
404.                 resultMessage = h.generateHillResultMessage(message, key, encrypt
    = False)
405.                 if values['FiveChar']:
406.
    window['-PLAINTEXT_DECRYPT-'].update(h.wrapFiveCharacters(resultMessage))
407.                 else:
408.                     window['-PLAINTEXT_DECRYPT-'].update(resultMessage)
409.             if values['Enigma']:
410.                 text = e.readTextFromFile(values['-PATH_SOURCE_DECRYPT-'])
411.                 alphabetList, steckerbrettDictionary, alphaRotor, betaRotor,
    gammaRotor = enigmaEncryptDecryptInit()
412.                 if values['FiveChar']:
413.
    window['-PLAINTEXT_DECRYPT-'].update(e.wrapFiveCharacters(e.encryptDecrypt(text,
    steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor, alphabetList)))
414.                 else:
415.                     window['-PLAINTEXT_DECRYPT-'].update(e.encryptDecrypt(text,
    steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor, alphabetList))
416.
417.         elif event == 'Decrypt File, Output File':
418.             print("hello")
419.             if values['ExtendedVigenere']:
420.                 print("hello")
421.                 listOfBytes = []
422.                 with open(values['-PATH_SOURCE_DECRYPT-'], "rb") as f:
423.                     while (byte := f.read(1)):
424.                         listOfBytes.append(byte)
425.                 f.close()
426.                 keyword = values['-KEY_DECRYPT-']
427.                 key = vge.generateKey(listOfBytes, keyword)
428.                 decryptedText = vge.decryptByteExtendedVigenere(listOfBytes, key)
429.                 h = open(values['-PATH_DECRYPT-'], "wb+")
430.                 for index in range(len(decryptedText)):
431.                     h.write(decryptedText[index])
432.                 h.close()
433.
434.         elif event=='Decrypt from Text File, Output into Text File':
435.             if values['Vigenere']:
436.                 if values['FiveChar']:
437.
    window['-PLAINTEXT_DECRYPT-'].update(vg.write_to_file(values['-PATH_DECRYPT-'],
    vg.wrapFiveCharacters(vg.decryption('', values['-KEY_DECRYPT-'], True,
    values['-PATH_SOURCE_DECRYPT-']))))
```

```
                else:
window['-PLAINTEXT_DECRYPT-'].update(vg.write_to_file(values['-PATH_DECRYPT-'],
vg.decryption('', values['-KEY_DECRYPT-'], True,
values['-PATH_SOURCE_DECRYPT-'])))
            if values['FullVigenere']:
                if values['FiveChar']:
window['-PLAINTEXT_DECRYPT-'].update(fvg.write_to_file(values['-PATH_DECRYPT-'],
fvg.wrapFiveCharacters(fvg.decryption('', values['-KEY_DECRYPT-'], True,
values['-PATH_SOURCE_DECRYPT-']))))
                else:
window['-PLAINTEXT_DECRYPT-'].update(fvg.write_to_file(values['-PATH_DECRYPT-'],
fvg.decryption('', values['-KEY_DECRYPT-'], True,
values['-PATH_SOURCE_DECRYPT-'])))
            if values['RunningKeyVigenere']:
                if values['FiveChar']:
window['-PLAINTEXT_DECRYPT-'].update(avg.write_to_file(values['-PATH_DECRYPT-'],
avg.wrapFiveCharacters(avg.decryption('', values['-KEY_DECRYPT-'], True,
values['-PATH_SOURCE_DECRYPT-']))))
                else:
window['-PLAINTEXT_DECRYPT-'].update(avg.write_to_file(values['-PATH_DECRYPT-'],
avg.decryption('', values['-KEY_DECRYPT-'], True,
values['-PATH_SOURCE_DECRYPT-'])))
            if values['ExtendedVigenere']:
                message = e.readTextFromFile(values['-PATH_SOURCE_DECRYPT-'])
                key = values['-KEY_DECRYPT-']
                if values['FiveChar']:
window['-PLAINTEXT_DECRYPT-'].update(vge.write_to_file(values['-PATH_DECRYPT-'],
vge.wrapFiveCharacters(vge.decryptTextExtendedVigenere(message, key))))
                else:
window['-PLAINTEXT_DECRYPT-'].update(vge.write_to_file(values['-PATH_DECRYPT-'],
vge.decryptTextExtendedVigenere(message, key)))
            if values['Playfair']:
                message = e.readTextFromFile(values['-PATH_SOURCE_DECRYPT-'])
                key = values['-KEY_DECRYPT-']
                if values['FiveChar']:
window['-PLAINTEXT_DECRYPT-'].update(p.write_to_file(values['-PATH_DECRYPT-'],
p.wrapFiveCharacters(p.toUpperCase(p.playfairDecrypt(message, key)))))
                else:
window['-PLAINTEXT_DECRYPT-'].update(p.write_to_file(values['-PATH_DECRYPT-'],
p.toUpperCase(p.playfairDecrypt(message, key))))
            if values['SuperEncryption']:
                if values['FiveChar']:
window['-PLAINTEXT_DECRYPT-'].update(se.write_to_file(values['-PATH_DECRYPT-'],
se.wrapFiveCharacters(se.myszkowskiTranspositionDecrypt(vg.decryption('',
values['-KEY_DECRYPT-'], True, values['-PATH_SOURCE_DECRYPT-']),
values['-KEY_DECRYPT-']))))
                else:
window['-PLAINTEXT_DECRYPT-'].update(se.write_to_file(values['-PATH_DECRYPT-'],
se.myszkowskiTranspositionDecrypt(vg.decryption('', values['-KEY_DECRYPT-'], True,
values['-PATH_SOURCE_DECRYPT-']), values['-KEY_DECRYPT-'])))
            if values['Affine']:
                if values['FiveChar']:
window['-PLAINTEXT_DECRYPT-'].update(af.write_to_file(values['-PATH_DECRYPT-'],
```

```
    af.wrapFiveCharacters(af.decryption('', values['-KEY_DECRYPT_M-'],
    values['-KEY_DECRYPT_B-'], True, values['-PATH_SOURCE_DECRYPT-'])))
472.                else:
473.
    window['-PLAINTEXT_DECRYPT-'].update(af.write_to_file(values['-PATH_DECRYPT-'],
    af.decryption('', values['-KEY_DECRYPT_M-'], values['-KEY_DECRYPT_B-'], True,
    values['-PATH_SOURCE_DECRYPT-'])))
474.            if values['Hill']:
475.                message = e.readTextFromFile(values['-PATH_SOURCE_DECRYPT-'])
476.                key = values['-KEY_DECRYPT-']
477.                resultMessage = h.generateHillResultMessage(message, key, encrypt
    = False)
478.                if values['FiveChar']:
479.
    window['-PLAINTEXT_DECRYPT-'].update(h.write_to_file(values['-PATH_DECRYPT-'],
    h.wrapFiveCharacters(resultMessage)))
480.                else:
481.
    window['-PLAINTEXT_DECRYPT-'].update(h.write_to_file(values['-PATH_DECRYPT-'],
    resultMessage))
482.            if values['Enigma']:
483.                text = e.readTextFromFile(values['-PATH_SOURCE_DECRYPT-'])
484.                alphabetList, steckerbrettDictionary, alphaRotor, betaRotor,
    gammaRotor = enigmaEncryptDecryptInit()
485.                if values['FiveChar']:
486.
    window['-PLAINTEXT_DECRYPT-'].update(e.write_to_file(values['-PATH_DECRYPT-'],
    e.wrapFiveCharacters(e.encryptDecrypt(text, steckerbrettDictionary, alphaRotor,
    betaRotor, gammaRotor, alphabetList))))
487.                else:
488.
    window['-PLAINTEXT_DECRYPT-'].update(e.write_to_file(values['-PATH_DECRYPT-'],
    e.encryptDecrypt(text, steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor,
    alphabetList)))
489.
490.        elif event=='Decrypt Output into Text File':
491.            if values['Vigenere']:
492.                if values['FiveChar']:
493.
    window['-PLAINTEXT_DECRYPT-'].update(vg.write_to_file(values['-PATH_DECRYPT-'],
    vg.wrapFiveCharacters(vg.decryption(values['-CIPHERTEXT_DECRYPT-'],
    values['-KEY_DECRYPT-']))))
494.                else:
495.
    window['-PLAINTEXT_DECRYPT-'].update(vg.write_to_file(values['-PATH_DECRYPT-'],
    vg.decryption(values['-CIPHERTEXT_DECRYPT-'], values['-KEY_DECRYPT-'])))
496.            if values['FullVigenere']:
497.                if values['FiveChar']:
498.
    window['-PLAINTEXT_DECRYPT-'].update(fvg.write_to_file(values['-PATH_DECRYPT-'],
    fvg.wrapFiveCharacters(fvg.decryption(values['-CIPHERTEXT_DECRYPT-'],
    values['-KEY_DECRYPT-']))))
499.                else:
500.
    window['-PLAINTEXT_DECRYPT-'].update(fvg.write_to_file(values['-PATH_DECRYPT-'],
    fvg.decryption(values['-CIPHERTEXT_DECRYPT-'], values['-KEY_DECRYPT-'])))
501.            if values['RunningKeyVigenere']:
502.                if values['FiveChar']:
503.
    window['-PLAINTEXT_DECRYPT-'].update(avg.write_to_file(values['-PATH_DECRYPT-'],
    avg.wrapFiveCharacters(avg.decryption(values['-CIPHERTEXT_DECRYPT-'],
    values['-KEY_DECRYPT-']))))
504.                else:
505.
    window['-PLAINTEXT_DECRYPT-'].update(avg.write_to_file(values['-PATH_DECRYPT-'],
```

```
    avg.decryption(values['-CIPHERTEXT_DECRYPT-'], values['-KEY_DECRYPT-'])))
506.            if values['ExtendedVigenere']:
507.                message = values['-CIPHERTEXT_DECRYPT-']
508.                key = values['-KEY_DECRYPT-']
509.                if values['FiveChar']:
510.
    window['-PLAINTEXT_DECRYPT-'].update(vge.write_to_file(values['-PATH_DECRYPT-'],
    vge.wrapFiveCharacters(vge.decryptTextExtendedVigenere(message, key))))
511.                else:
512.
    window['-PLAINTEXT_DECRYPT-'].update(vge.write_to_file(values['-PATH_DECRYPT-'],
    vge.decryptTextExtendedVigenere(message, key)))
513.            if values['Playfair']:
514.                message = values['-CIPHERTEXT_DECRYPT-']
515.                key = values['-KEY_DECRYPT-']
516.                if values['FiveChar']:
517.
    window['-PLAINTEXT_DECRYPT-'].update(p.write_to_file(values['-PATH_DECRYPT-'],
    p.wrapFiveCharacters(p.toUpperCase(p.playfairDecrypt(message, key)))))
518.                else:
519.
    window['-PLAINTEXT_DECRYPT-'].update(p.write_to_file(values['-PATH_DECRYPT-'],
    p.toUpperCase(p.playfairDecrypt(message, key))))
520.            if values['SuperEncryption']:
521.                if values['FiveChar']:
522.
    window['-PLAINTEXT_DECRYPT-'].update(vg.write_to_file(values['-PATH_DECRYPT-'],
    vg.wrapFiveCharacters(vg.decryption(se.myszkowskiTranspositionDecrypt(values['-CIP
    HERTEXT_DECRYPT-'], values['-KEY_DECRYPT-']), values['-KEY_DECRYPT-']))))
523.                else:
524.
    window['-PLAINTEXT_DECRYPT-'].update(vg.write_to_file(values['-PATH_DECRYPT-'],
    vg.decryption(se.myszkowskiTranspositionDecrypt(values['-CIPHERTEXT_DECRYPT-'],
    values['-KEY_DECRYPT-']), values['-KEY_DECRYPT-'])))
525.            if values['Affine']:
526.                if values['FiveChar']:
527.
    window['-PLAINTEXT_DECRYPT-'].update(af.write_to_file(values['-PATH_DECRYPT-'],
    af.wrapFiveCharacters(af.decryption(values['-CIPHERTEXT_DECRYPT-'],
    values['-KEY_DECRYPT_M-'], values['-KEY_DECRYPT_B-'], False, ''))))
528.                else:
529.
    window['-PLAINTEXT_DECRYPT-'].update(af.write_to_file(values['-PATH_DECRYPT-'],
    af.decryption(values['-CIPHERTEXT_DECRYPT-'], values['-KEY_DECRYPT_M-'],
    values['-KEY_DECRYPT_B-'], False, '')))
530.            if values['Hill']:
531.                message = values['-CIPHERTEXT_DECRYPT-']
532.                key = values['-KEY_DECRYPT-']
533.                resultMessage = h.generateHillResultMessage(message, key, encrypt
    = False)
534.                if values['FiveChar']:
535.
    window['-PLAINTEXT_DECRYPT-'].update(h.write_to_file(values['-PATH_DECRYPT-'],
    h.wrapFiveCharacters(resultMessage)))
536.                else:
537.
    window['-PLAINTEXT_DECRYPT-'].update(h.write_to_file(values['-PATH_DECRYPT-'],
    resultMessage))
538.            if values['Enigma']:
539.                text = values['-CIPHERTEXT_DECRYPT-']
540.                alphabetList, steckerbrettDictionary, alphaRotor, betaRotor,
    gammaRotor = enigmaEncryptDecryptInit()
541.                if values['FiveChar']:
542.
    window['-PLAINTEXT_DECRYPT-'].update(e.write_to_file(values['-PATH_DECRYPT-'],
```

```
    e.wrapFiveCharacters(e.encryptDecrypt(text, steckerbrettDictionary, alphaRotor,
    betaRotor, gammaRotor, alphabetList))))
543.                else:
544.
    window['-PLAINTEXT_DECRYPT-'].update(e.write_to_file(values['-PATH_DECRYPT-'],
    e.encryptDecrypt(text, steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor,
    alphabetList)))
545.
546.    window.close()
```

Vigenere.py (vigenere standar)

```
1.  from textwrap import wrap
2.
3.  def write_to_file(path, ciphertext):
4.      file1 = open(path,"w")
5.      file1.write(ciphertext)
6.      file1.close()
7.
8.  def wrapFiveCharacters(message):
9.      messageWrapFive = wrap(message,5)
10.     return ' '.join(messageWrapFive)
11.
12. def encryption(plaintext, key, from_file, path, write_to_file):
13.     #Input plaintext
14.
15.     #plaintext = input("Masukkan plaintext:")
16.
17.     #Input key from user#
18.     #key = input("Masukkan key:")
19.     if (from_file):
20.         file1 = open(path,"r")
21.         plaintext_list = file1.readlines()
22.         seperator=''
23.         file1.close()
24.         plaintext=seperator.join(plaintext_list)
25.
26.     #Convert into lowercase#
27.     plaintext=plaintext.lower()
28.     key=key.lower()
29.
30.     #Remove comma
31.     plaintext = plaintext.replace(',', '')
32.     key = key.replace(',', '')
33.     #print(plaintext)
34.
35.     #Remove space
36.     plaintext = plaintext.replace(' ', '')
37.     key = key.replace(' ', '')
38.
39.     #Remove punctuation
40.     # Define punctuation
41.     punctuations = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''
42.
43.     # Remove punctuation from plaintext
44.     no_punct = ""
45.     for char in plaintext:
46.         if char not in punctuations:
47.             no_punct = no_punct + char
48.     plaintext=no_punct
49.
50.     # Remove punctuation from key
```

```
    no_punct = ""
    for char in key:
        if char not in punctuations:
            no_punct = no_punct + char
    key=no_punct

    #Compare plaintext and key's length#
    if (len(key)<len(plaintext)):
        real_key=key
        times = len(plaintext)//len(key)

        for i in range(times-1):
            key+=real_key

        sisa = len(plaintext)-len(key)

        for i in range(sisa):
            key+=real_key[i]

    #If key>plaintext#
    elif (len(key)>len(plaintext)):
        key=key[:len(plaintext)]

    print(key)

    #Alphabet list for conversion from number to character#
    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']
    ciphertext=''
    for i in range(len(plaintext)):
        # if (i%5==0) and (i!=0):
        #     ciphertext+='-'
        ciphernumber=(alphabet.index(plaintext[i])+alphabet.index(key[i]))%26
        #print(ciphernumber, end=', ')
        ciphertext+=alphabet[ciphernumber]
        #print(ciphertext)
    return ciphertext


def decryption(ciphertext, key, from_file = False, path = ''):
    #Input ciphertext
    if (from_file):
        file1 = open(path,"r")
        ciphertext_list = file1.readlines()
        seperator=''
        file1.close()
        ciphertext=seperator.join(ciphertext_list)

    #ciphertext = input("Masukkan ciphertext:")

        print(ciphertext[0])

        #Input key from user#
        #key = input("Masukkan key:")

        #print(key)

        #Convert into lowercase#
        ciphertext=ciphertext.lower()
        key=key.lower()

        #Remove comma
        ciphertext = ciphertext.replace(',', '')
        key = key.replace(',', '')
```

```
115.
116.         #Remove space
117.         ciphertext = ciphertext.replace(' ', '')
118.         key = key.replace(' ', '')
119.
120.         #Remove punctuation
121.         # Define punctuation
122.         punctuations = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''
123.
124.         # Remove punctuation from ciphertext
125.         no_punct = ""
126.         for char in ciphertext:
127.             if char not in punctuations:
128.                 no_punct = no_punct + char
129.         ciphertext=no_punct
130.
131.         # Remove punctuation from key
132.         no_punct = ""
133.         for char in key:
134.             if char not in punctuations:
135.                 no_punct = no_punct + char
136.         key=no_punct
137.
138.
139.
140.         #Compare plaintext and key's length#
141.         if (len(key)<len(ciphertext)):
142.             real_key=key
143.             times = len(ciphertext)//len(key)
144.             print(len(ciphertext))
145.
146.             for i in range(times-1):
147.                 key+=real_key
148.
149.             sisa = len(ciphertext)-len(key)
150.
151.             for i in range(sisa):
152.                 key+=real_key[i]
153.
154.
155.         #If key>plaintext#
156.         elif (len(key)>len(ciphertext)):
157.             key=key[:len(ciphertext)]
158.
159.         print(key)
160.
161.         #Alphabet list for conversion from number to character#
162.         alphabet = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l',
    'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',\
163.                 'w', 'x', 'y', 'z']
164.
165.
166.         plaintext=''
167.         for i in range(len(ciphertext)):
168.             #alphabet_cycle=itertools.cycle(alphabet)
169.             #print(key[i])
170.             plainnumber=(alphabet.index(ciphertext[i])-alphabet.index(key[i]))%26
171.
172.
173.             plaintext+=alphabet[plainnumber]
174.             #print(ciphertext)
175.         return plaintext
```

Autokey_vigenere.py

```
#autokey_vigenere

from textwrap import wrap

def write_to_file(path, ciphertext):
    file1 = open(path,"w")
    file1.write(ciphertext)
    file1.close()

def wrapFiveCharacters(message):
    messageWrapFive = wrap(message,5)
    return ' '.join(messageWrapFive)

def encryption(plaintext, key, from_file, path, write_to_file):
    #Input plaintext

    #plaintext = input("Masukkan plaintext:")

    #Input key from user#
    #key = input("Masukkan key:")
    if (from_file):
        file1 = open(path,"r")
        plaintext_list = file1.readlines()
        seperator=''
        file1.close()
        plaintext=seperator.join(plaintext_list)

    #Convert into lowercase#
    plaintext=plaintext.lower()
    key=key.lower()

    #Remove comma
    plaintext = plaintext.replace(',', '')
    key = key.replace(',', '')
    #print(plaintext)

    #Remove space
    plaintext = plaintext.replace(' ', '')
    key = key.replace(' ', '')

    #Remove punctuation
    # Define punctuation
    punctuations = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''

    # Remove punctuation from plaintext
    no_punct = ""
    for char in plaintext:
        if char not in punctuations:
            no_punct = no_punct + char
    plaintext=no_punct

    # Remove punctuation from key
    no_punct = ""
    for char in key:
        if char not in punctuations:
            no_punct = no_punct + char
    key=no_punct

    #Compare plaintext and key's length#
    if (len(key)<len(plaintext)):
        """
       real_key=key
       times = len(plaintext)//len(key)
```

```
64.
65.         for i in range(times-1):
66.             key+=real_key
67.         """
68.
69.          sisa = len(plaintext)-len(key)
70.
71.          for i in range(sisa):
72.              key+=plaintext[i]
73.
74.     #If key>plaintext#
75.     elif (len(key)>len(plaintext)):
76.         key=key[:len(plaintext)]
77.
78.     print(key)
79.
80.     #Alphabet list for conversion from number to character#
81.     alphabet = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',
    'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',\
82.              'w', 'x', 'y', 'z']
83.     ciphertext=''
84.     for i in range(len(plaintext)):
85.         # if (i%5==0) and (i!=0):
86.         #     ciphertext+='-'
87.         ciphernumber=(alphabet.index(plaintext[i])+alphabet.index(key[i]))%26
88.         #print(ciphernumber, end=', ')
89.         ciphertext+=alphabet[ciphernumber]
90.         #print(ciphertext)
91.     return ciphertext
92.
93.
94. def decryption(ciphertext, key, from_file = False, path = ''):
95.     #Input ciphertext
96.     if (from_file):
97.         file1 = open(path,"r")
98.         ciphertext_list = file1.readlines()
99.         seperator=''
100.             file1.close()
101.             ciphertext=seperator.join(ciphertext_list)
102.
103.         #ciphertext = input("Masukkan ciphertext:")
104.
105.         print(ciphertext[0])
106.
107.         #Input key from user#
108.         #key = input("Masukkan key:")
109.
110.         #print(key)
111.
112.         #Convert into lowercase#
113.         ciphertext=ciphertext.lower()
114.         key=key.lower()
115.
116.         #Remove comma
117.         ciphertext = ciphertext.replace(',', '')
118.         key = key.replace(',', '')
119.
120.         #Remove space
121.         ciphertext = ciphertext.replace(' ', '')
122.         key = key.replace(' ', '')
123.
124.         #Remove punctuation
125.         # Define punctuation
126.         punctuations = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''
127.
```

```
        # Remove punctuation from ciphertext
        no_punct = ""
        for char in ciphertext:
            if char not in punctuations:
                no_punct = no_punct + char
        ciphertext=no_punct

        # Remove punctuation from key
        no_punct = ""
        for char in key:
            if char not in punctuations:
                no_punct = no_punct + char
        key=no_punct

        #Make the real key
        #Alphabet list for conversion from number to character#
        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']


        plaintext=''
        for i in range(len(ciphertext)):
            #alphabet_cycle=itertools.cycle(alphabet)
            #print(key[i])
            plainnumber=(alphabet.index(ciphertext[i])-alphabet.index(key[i]))%26


            plaintext+=alphabet[plainnumber]
            key+=alphabet[plainnumber]
            #print(ciphertext)
        print(key)


        #Alphabet list for conversion from number to character#
        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']


        plaintext=''
        for i in range(len(ciphertext)):
            #alphabet_cycle=itertools.cycle(alphabet)
            #print(key[i])
            plainnumber=(alphabet.index(ciphertext[i])-alphabet.index(key[i]))%26


            plaintext+=alphabet[plainnumber]
            #print(ciphertext)
        return plaintext
```

full_vigenere.py

```
#full vigenere cipher

import random
import pickle
from textwrap import wrap

def write_to_file(path, ciphertext):
    file1 = open(path,"w")
```

```
    file1.write(ciphertext)
    file1.close()

def wrapFiveCharacters(message):
    messageWrapFive = wrap(message,5)
    return ' '.join(messageWrapFive)

def encryption(plaintext, key, from_file, path, write_to_file):
    if (from_file):
        file1 = open(path,"r")
        plaintext_list = file1.readlines()
        seperator=''
        file1.close()
        plaintext=seperator.join(plaintext_list)

    #Convert into lowercase#
    plaintext= toUpperCase(plaintext)
    key=toUpperCase(key)

    #Remove comma
    plaintext = plaintext.replace(',', '')
    key = key.replace(',', '')
    #print(plaintext)

    #Remove space
    plaintext = plaintext.replace(' ', '')
    key = key.replace(' ', '')

    #Remove punctuation
    # Define punctuation
    punctuations = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''

    # Remove punctuation from plaintext
    no_punct = ""
    for char in plaintext:
        if char not in punctuations:
            no_punct = no_punct + char
    plaintext=no_punct

    # Remove punctuation from key
    no_punct = ""
    for char in key:
        if char not in punctuations:
            no_punct = no_punct + char
    key=no_punct

    #Compare plaintext and key's length#
    if (len(key)<len(plaintext)):
        real_key=key
        times = len(plaintext)//len(key)

        for i in range(times-1):
            key+=real_key

        sisa = len(plaintext)-len(key)

        for i in range(sisa):
            key+=real_key[i]

    #If key>plaintext#
    elif (len(key)>len(plaintext)):
        key=key[:len(plaintext)]

    print(key)

```

```
    #Alphabet list for conversion from number to character#
    alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    alphabet = list(alphabet)

    # define an empty list
    alphabet_shuffled = []

    # open file and read the content in a list
    with open('full_vigenere_matrix.txt', 'r') as filehandle:
        for line in filehandle:
            # remove linebreak which is the last character of the string
            currentAlphabet = line[:-1]

            # add item to the list
            alphabet_shuffled.append(currentAlphabet)

    print(alphabet_shuffled[0][0])
    ciphertext=''
    for i in range(len(plaintext)):
        ciphernumber_x=alphabet.index(plaintext[i])
        ciphernumber_y=alphabet.index(key[i])
        print(ciphernumber_x)
        print(ciphernumber_y)
        ciphertext+=alphabet_shuffled[ciphernumber_x][ciphernumber_y]
        print(ciphertext)

        return ciphertext

    def write_to_file(path, ciphertext):
        file1 = open(path,"w")
        file1.write(ciphertext)
        file1.close()

    def toUpperCase(text):
        return "".join(filter(str.isupper, text.upper()))

    def decryption(ciphertext, key, from_file = False, path = ''):
        #Input ciphertext
        if (from_file):
            file1 = open(path,"r")
            ciphertext_list = file1.readlines()
            seperator=''
            file1.close()
            ciphertext=seperator.join(ciphertext_list)

        #ciphertext = input("Masukkan ciphertext:")
        print("Ciphertext:", end="")
        print(ciphertext)

        #Input key from user#
        #key = input("Masukkan key:")

        #print(key)

        #Convert into lowercase#
        ciphertext=toUpperCase(ciphertext)
        key=toUpperCase(key)

        #Remove comma
        ciphertext = ciphertext.replace(',', '')
        key = key.replace(',', '')

        #Remove space
        ciphertext = ciphertext.replace(' ', '')
        key = key.replace(' ', '')
```

```
139.
140.         #Remove punctuation
141.         # Define punctuation
142.         punctuations = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''
143.
144.         # Remove punctuation from ciphertext
145.         no_punct = ""
146.         for char in ciphertext:
147.             if char not in punctuations:
148.                 no_punct = no_punct + char
149.         ciphertext=no_punct
150.
151.         # Remove punctuation from key
152.         no_punct = ""
153.         for char in key:
154.             if char not in punctuations:
155.                 no_punct = no_punct + char
156.         key=no_punct
157.
158.
159.
160.         #Compare plaintext and key's length#
161.         if (len(key)<len(ciphertext)):
162.             real_key=key
163.             times = len(ciphertext)//len(key)
164.             print(len(ciphertext))
165.
166.             for i in range(times-1):
167.                 key+=real_key
168.
169.             sisa = len(ciphertext)-len(key)
170.
171.             for i in range(sisa):
172.                 key+=real_key[i]
173.
174.
175.         #If key>plaintext#
176.         elif (len(key)>len(ciphertext)):
177.             key=key[:len(ciphertext)]
178.
179.         print(key)
180.
181.         #Alphabet list for conversion from number to character#
182.         alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
183.         alphabet = list(alphabet)
184.
185.         # define an empty list
186.         alphabet_shuffled = []
187.
188.         # open file and read the content in a list
189.         with open('full_vigenere_matrix.txt', 'r') as filehandle:
190.             for line in filehandle:
191.                 # remove linebreak which is the last character of the string
192.                 currentAlphabet = line[:-1]
193.
194.                 # add item to the list
195.                 alphabet_shuffled.append(currentAlphabet)
196.
197.         """
198.        f = open("full_vigenere_matrix.txt", "r")
199.        f1=f.readlines()
200.        for x in f1:
201.            alphabet_shuffled.append(x)
202.        """
203.
```

```
        plaintext=''

        for i in range(len(ciphertext)):
            a = alphabet.index(key[i])
            for j in range(0,26):
                if (alphabet_shuffled[j][a]==ciphertext[i]):
                    break
            plainnumber=j
            plaintext+=alphabet[plainnumber]



        return plaintext

```

vigenereExtended.py

```
#vigenere Extended
from textwrap import wrap

def generateKey(string, key):
    key = list(key)
    if len(string) == len(key):
        return(key)
    else:
        for i in range(len(string) - len(key)):
            key.append(key[i % len(key)])
    return("" . join(key))

def encryptByteExtendedVigenere(bytePlainText, key):
    result = [[0] for i in range(len(bytePlainText))]
    key = key.strip().upper()
    keyIndex = 0
    keylength = len(key)
    for i in range(len(bytePlainText)):
        keyIndex = keyIndex % keylength
        shift = ord(key[keyIndex]) - 65
        result[i] = bytes([(ord(bytePlainText[i]) + shift) % 256])
        keyIndex+=1
    return result

def decryptByteExtendedVigenere(byteCipherText, key):
    result = [[0] for i in range(len(byteCipherText))]
    key = key.strip().upper()
    keyIndex = 0
    keylength = len(key)
    for i in range(len(byteCipherText)):
        keyIndex = keyIndex % keylength
        shift = ord(key[keyIndex]) - 65
        result[i] = bytes([(ord(byteCipherText[i]) + 256 - shift) % 256])
        keyIndex+=1
    return result

def encryptTextExtendedVigenere(plaintext, key):
    result = [[0] for i in range(len(plaintext))]
    key = key.strip().upper()
    keyIndex = 0
    keylength = len(key)
    for i in range(len(plaintext)):
        keyIndex = keyIndex % keylength
        shift = ord(key[keyIndex]) - 65
        result[i] = chr((ord(plaintext[i]) + shift) % 256)
```

```
        keyIndex+=1
    return ''.join(i for i in result)

def decryptTextExtendedVigenere(cipherText, key):
    result = [[0] for i in range(len(cipherText))]
    key = key.strip().upper()
    keyIndex = 0
    keylength = len(key)
    for i in range(len(cipherText)):
        keyIndex = keyIndex % keylength
        shift = ord(key[keyIndex]) - 65
        result[i] = chr((ord(cipherText[i]) + 256 - shift) % 256)
        keyIndex+=1
    return ''.join(i for i in result)

def wrapFiveCharacters(message):
    messageWrapFive = wrap(message,5)
    return ' '.join(messageWrapFive)

def write_to_file(path, text):
    file1 = open(path,"w+")
    file1.write(text)
    file1.close()

def readTextFromFile(path):
    file1 = open(path, "r")
    data = file1.read()
    file1.close()
    return data
```

Playfair.py

```
#playfair
from textwrap import wrap

def generateUniqueAlphabetList(seq):
    seen = {}
    return [seen.setdefault(x, x) for x in seq if x not in seen]

def partition(seq, n):
    return [seq[i : i + n] for i in range(0, len(seq), n)]

def toUpperCase(text):
    return "".join(filter(str.isupper, text.upper()))

def splitMessageToDigraphs(message):
    messageList=[]
    for e in message:
        messageList.append(e)
    for i in range(len(messageList)):
        if " " in messageList:
            messageList.remove(" ")
    i=0
    for e in range(len(messageList)//2):
        if messageList[i]==messageList[i+1]:
            messageList.insert(i+1,'X')
        i=i+2
    if len(messageList)%2==1:
        messageList.append("X")
    i=0
    digraphList=[]
    for x in range(1,len(messageList)//2+1):
```

```
        digraphList.append(messageList[i:i+2])
        i=i+2
    return digraphList

def formatMessage(message, replaceFrom, replaceTo):
        message = toUpperCase(message)
        formattedMessage = message.replace(replaceFrom, replaceTo)
        return formattedMessage

def generateEncryptionDictionary(matrix):
    encryptionDictionary = {}
    for row in matrix:
        for i in range(5):
            for j in range(5):
                if i != j:
                    encryptionDictionary[row[i] + row[j]] = row[(i + 1) % 5] +
row[(j + 1) % 5]
    for c in zip(*matrix):
        for i in range(5):
            for j in range(5):
                if i != j:
                    encryptionDictionary[c[i] + c[j]] = c[(i + 1) % 5] + c[(j + 1)
% 5]
    for i in range(5):
        for j in range(5):
            for k in range(5):
                for l in range(5):
                    if i != k and j != l:
                        encryptionDictionary[matrix[i][j] + matrix[k][l]] =
matrix[i][l] + matrix[k][j]
    return encryptionDictionary

def playfairEncrypt(message, key, replaceFrom = 'J', replaceTo = None):
    if replaceTo is None:
        replaceTo = 'I' if replaceFrom == 'J' else ''
    alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    digraphList = splitMessageToDigraphs(formatMessage(message, replaceFrom,
replaceTo))
    uniqueFormattedMatrix = partition(generateUniqueAlphabetList(formatMessage(key
+ alphabet, replaceFrom, replaceTo)), 5)
    encryptionDictionary = generateEncryptionDictionary(uniqueFormattedMatrix)
    cipheredMessage = " ".join(encryptionDictionary[a + b] for a, b in
digraphList)
    return cipheredMessage
    #return " ".join(enc[a + (b if b else 'X')] for a, b in lst)

def playfairDecrypt(message, key, replaceFrom = 'J', replaceTo = 'I'):
    if replaceTo is None:
        replaceTo = 'I' if replaceFrom == 'J' else ''
    alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    lst = splitMessageToDigraphs(formatMessage(message, replaceFrom, replaceTo))
    uniqueFormattedMatrix = partition(generateUniqueAlphabetList(formatMessage(key
+ alphabet, replaceFrom, replaceTo)), 5)
    encryptionDictionary = generateEncryptionDictionary(uniqueFormattedMatrix)
    decryptionDictionary = dict((value, key) for key, value in
encryptionDictionary.items())
    decipheredMessage = " ".join(decryptionDictionary[p] for p in
partition(formatMessage(message, replaceFrom, replaceTo), 2))
    return decipheredMessage

def wrapFiveCharacters(message):
    messageWrapFive = wrap(message,5)
    return ' '.join(messageWrapFive)

def write_to_file(path, text):
```

```
    file1 = open(path,"w+")
    file1.write(text)
    file1.close()

def readTextFromFile(path):
    file1 = open(path, "r")
    data = file1.read()
    file1.close()
    return data
```

myszkowskiTransposition.py

```
#SuperEncryption using vigenere.py + Myszkowski transposition cipher (this file)

#myszkowski
import math
from textwrap import wrap

def write_to_file(path, ciphertext):
    file1 = open(path,"w")
    file1.write(ciphertext)
    file1.close()

def readTextFromFile(path):
    file1 = open(path, "r")
    data = file1.read()
    file1.close()
    return data

def wrapFiveCharacters(message):
    messageWrapFive = wrap(message,5)
    return ' '.join(messageWrapFive)

def toUpperCase(text):
    return "".join(filter(str.isupper, text.upper()))

def generateKeyList(key):
    alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    keyList = []
    duplicateFound = True
    for i in key:
        duplicateFound = True
        while duplicateFound:
            if i in keyList:
                i = alphabet[(alphabet.index(i) + 1) % 26]
            else:
                duplicateFound = False
        keyList.append(i)
    return keyList

def myszkowskiTranspositionEncrypt(message, key):
    message = toUpperCase(message)
    key = toUpperCase(key)
    encryptedMessage = ""
    keyIndex = 0
    messageLength = float(len(message))
    messageList = list(message)
    keyList = generateKeyList(key)
    key = ''.join(keyList)
    sortedKeyList = sorted(keyList)
    column = len(key)
```

```
    row = int(math.ceil(messageLength / column))
    fill_null = int((row * column) - messageLength)
    messageList.extend('*' * fill_null)
    matrix = [messageList[i: i + column]
              for i in range(0, len(messageList), column)]
    for _ in range(column):
        curr_idx = key.index(sortedKeyList[keyIndex])
        encryptedMessage += ''.join([row[curr_idx] for row in matrix])
        keyIndex += 1
    return encryptedMessage

def myszkowskiTranspositionDecrypt(message, key):
    message = toUpperCase(message)
    key = toUpperCase(key)
    decryptedMessage = ""
    keyIndex = 0
    messageIndex = 0
    messageLength = float(len(message))
    messageList = list(message)
    column = len(key)
    row = int(math.ceil(messageLength / column))
    keyList = generateKeyList(key)
    key = ''.join(keyList)
    sortedKeyList = sorted(keyList)
    decrytedMessageList = []
    for _ in range(row):
        decrytedMessageList += [[None] * column]
    for _ in range(column):
        curr_idx = key.index(sortedKeyList[keyIndex])
        for j in range(row):
            decrytedMessageList[j][curr_idx] = messageList[messageIndex]
            messageIndex += 1
        keyIndex += 1
    decryptedMessage = ''.join(sum(decrytedMessageList, []))
    additionalTemp = decryptedMessage.count('*')
    if additionalTemp > 0:
        return decryptedMessage[: -additionalTemp]
    return decryptedMessage

```

Affine.py

```
#affine

from textwrap import wrap

def write_to_file(path, ciphertext):
    file1 = open(path,"w")
    file1.write(ciphertext)
    file1.close()

def wrapFiveCharacters(message):
    messageWrapFive = wrap(message,5)
    return ' '.join(messageWrapFive)

def toUpperCase(text):
    return "".join(filter(str.isupper, text.upper()))

def extendedGCDEuclideanAlgorithm(a, b):
    if a == 0 :
        return b, 0, 1
    greatestCommonDenominator, x1, y1 = extendedGCDEuclideanAlgorithm(b%a, a)
```

```
    x = y1 - (b//a) * x1
    y = x1
    return greatestCommonDenominator, x, y

def getModularInverse(a, m):
    greatestCommonDenominator, x, y = extendedGCDEuclideanAlgorithm(a, m)
    if greatestCommonDenominator != 1:
        return None
    else:
        return x % m

def encryption(plaintext, key_m, key_b, from_file, path, write_to_file):
    if (from_file):
        file1 = open(path,"r")
        plaintext_list = file1.readlines()
        seperator=''
        file1.close()
        plaintext=seperator.join(plaintext_list)

    plaintext=toUpperCase(plaintext)

    #Remove comma
    plaintext = plaintext.replace(',', '')
    #key = key.replace(',', '')
    #print(plaintext)

    #Remove space
    plaintext = plaintext.replace(' ', '')
    #key = key.replace(' ', '')

    #Remove punctuation
    # Define punctuation
    punctuations = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''

    # Remove punctuation from plaintext
    no_punct = ""
    for char in plaintext:
        if char not in punctuations:
            no_punct = no_punct + char
    plaintext=no_punct

    #Alphabet list for conversion from number to character#
    alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    alphabet = list(alphabet)
    ciphertext=''
    for i in range(len(plaintext)):
        # if (i%5==0) and (i!=0):
        #     ciphertext+='-'
        ciphernumber=(int(key_m)*alphabet.index(plaintext[i])+int(key_b))%26
        #ciphernumber=(alphabet.index(plaintext[i])+alphabet.index(key[i]))%26
        #print(ciphernumber, end=', ')
        ciphertext+=alphabet[ciphernumber]
        #print(ciphertext)
    return ciphertext


def decryption(ciphertext, key_m, key_b, from_file = False, path = '',
write_to_file = False):
    #Input plaintext
    if (from_file):
        file1 = open(path,"r")
        ciphertext_list = file1.readlines()
        seperator=''
        file1.close()
        ciphertext=seperator.join(ciphertext_list)
```

```
85.
86.     ciphertext=toUpperCase(ciphertext)
87.
88.     #Remove comma
89.     ciphertext = ciphertext.replace(',', '')
90.     #key = key.replace(',', '')
91.
92.     #Remove space
93.     ciphertext = ciphertext.replace(' ', '')
94.     #key = key.replace(' ', '')
95.
96.     #Remove punctuation
97.     # Define punctuation
98.     punctuations = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''
99.
100.        # Remove punctuation from ciphertext
101.        no_punct = ""
102.        for char in ciphertext:
103.            if char not in punctuations:
104.                no_punct = no_punct + char
105.        ciphertext=no_punct
106.
107.        #Alphabet list for conversion from number to character#
108.        alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
109.        alphabet = list(alphabet)
110.        key_m=int(key_m)
111.        key_b=int(key_b)
112.        #Find the inverse of key_m
113.
114.        key_m_inverse = getModularInverse(key_m, 26)
115.
116.        plaintext=''
117.        for i in range(len(ciphertext)):
118.            plainnumber=(key_m_inverse*(alphabet.index(ciphertext[i])-key_b))%26
119.
120.            plaintext+=alphabet[plainnumber]
121.            print(ciphertext)
122.        return plaintext
```

Hill.py

```
1.  # Hill Cipher
2.
3.  from textwrap import wrap
4.
5.  def generateMessageVector(message, n):
6.      messageVector = [[0] for i in range(n)]
7.      for i in range(n):
8.          messageVector[i][0] = ord(message[i]) % 65
9.      return messageVector
10.
11. def generateKeyMatrix(key, n):
12.     keyMatrix = [[0] * n for i in range(n)]
13.     k = 0
14.     for i in range(n):
15.         for j in range(n):
16.             print(key[k])
17.             keyMatrix[i][j] = ord(key[k]) % 65
18.             k += 1
19.     return keyMatrix
20.
```

```
def generateCipherMatrix(message, keyMatrix, messageVector, n):
    cipherMatrix = [[0] for i in range(n)]
    for i in range(n):
        for j in range(1):
            cipherMatrix[i][j] = 0
            for x in range(n):
                cipherMatrix[i][j] += (keyMatrix[i][x] * messageVector[x][j])
            cipherMatrix[i][j] = cipherMatrix[i][j] % 26
    return cipherMatrix

def generateInverseMatrix(matrix):
    determinant = round(calculateDeterminant(matrix))
    inverseMatrix = [[0 for j in range(len(matrix))] for i in range(len(matrix))]
    for i in range(len(matrix)):
        inverseMatrix[i][i] = 1
    for focusDiagonal in range(len(matrix)):
        focusDiagonalScaler = 1.0 / matrix[focusDiagonal][focusDiagonal]
        for j in range(len(matrix)):
            matrix[focusDiagonal][j] *= focusDiagonalScaler
            inverseMatrix[focusDiagonal][j] *= focusDiagonalScaler
        for i in list(range(len(matrix)))[0:focusDiagonal] +
list(range(len(matrix)))[focusDiagonal+1:]:
            currentRowScaler = matrix[i][focusDiagonal]
            for j in range(len(matrix)):
                matrix[i][j] = matrix[i][j] - currentRowScaler *
matrix[focusDiagonal][j]
                inverseMatrix[i][j] = inverseMatrix[i][j] - currentRowScaler *
inverseMatrix[focusDiagonal][j]
    for x in range(len(inverseMatrix)):
        for y in range(len(inverseMatrix[0])):
            inverseMatrix[x][y] = round(inverseMatrix[x][y]*determinant)
    for x in range(len(inverseMatrix)):
        for y in range(len(inverseMatrix[0])):
            inverseMatrix[x][y] = inverseMatrix[x][y] * (determinant % 26)
    return inverseMatrix

def generateZeroMatrix(rows, cols):
    zeroMatrix = []
    while len(zeroMatrix) < rows:
        zeroMatrix.append([])
        while len(zeroMatrix[-1]) < cols:
            zeroMatrix[-1].append(0.0)
    return zeroMatrix

def generateMatrixCopy(matrix):
    MatrixCopy = generateZeroMatrix(len(matrix), len(matrix[0]))
    for i in range(len(matrix)):
        for j in range(len(matrix[0])):
            MatrixCopy[i][j] = matrix[i][j]
    return MatrixCopy

def calculateDeterminant(matrix):
    if len(matrix) == 1 and len(matrix[0]) == 1:
        return matrix[0][0]
    matrixCopy = generateMatrixCopy(matrix)
    for focusDiagonal in range(len(matrix)):
        if matrixCopy[focusDiagonal][focusDiagonal] == 0:
            matrixCopy[focusDiagonal][focusDiagonal] = 1.0e-18
        for i in range(focusDiagonal+1, len(matrix)):
            currentRowScaler = matrixCopy[i][focusDiagonal] /
matrixCopy[focusDiagonal][focusDiagonal]
            for j in range(len(matrix)):
                matrixCopy[i][j] = matrixCopy[i][j] - currentRowScaler *
matrixCopy[focusDiagonal][j]
    product = 1.0
```

```
    for i in range(len(matrix)):
        product *= matrixCopy[i][i]
    return product

def generateDecipherMatrix(message, keyMatrix, messageVector, n):
    decipherMatrix = [[0] for i in range(n)]
    for i in range(n):
        for j in range(1):
            decipherMatrix[i][j] = 0
            for x in range(n):
                decipherMatrix[i][j] += (keyMatrix[i][x] * messageVector[x][j])
            decipherMatrix[i][j] = decipherMatrix[i][j] % 26
    return decipherMatrix

def HillCipherEncryption(message, key):
    n = 3
    keyMatrix = generateKeyMatrix(key, n)
    messageVector = generateMessageVector(message, n)
    cipherMatrix = generateCipherMatrix(messageVector, keyMatrix, messageVector,
n)
        CipherText = []
        for i in range(n):
            CipherText.append(chr(cipherMatrix[i][0] + 65))
        CipheredText = "".join(CipherText)
        return CipheredText

    def HillCipherDecryption(message, key):
        n = len(message)
        keyMatrix = generateKeyMatrix(key, n)
        invertedKeyMatrix = generateInverseMatrix(keyMatrix)
        decipherMessageVector = generateMessageVector(message, n)
        decipherMatrix = generateDecipherMatrix(message, invertedKeyMatrix,
decipherMessageVector, n)
        DecipherText = []
        for i in range(n):
            DecipherText.append(chr(round(decipherMatrix[i][0] + 65)))
        DecipheredText = "".join(DecipherText)
        return DecipheredText

    def wrapFiveCharacters(message):
        messageWrapFive = wrap(message,5)
        return ' '.join(messageWrapFive)

    def toUpperCase(text):
        return "".join(filter(str.isupper, text.upper()))

    def write_to_file(path, text):
        file1 = open(path,"w+")
        file1.write(text)
        file1.close()

    def readTextFromFile(path):
        file1 = open(path, "r")
        data = file1.read()
        file1.close()
        return data

    def generateHillResultMessage(message, key, encrypt):
        resultList = []
        message = toUpperCase(message)
        message = wrap(message, 3)
        if (len(key)<9):
            real_key=key
            times = 9//len(key)
            for i in range(times-1):
```

```
                key+=real_key
            sisa = 9-len(key)
            for i in range(sisa):
                key+=real_key[i]
        #If key>plaintext#
        elif (len(key)>9):
            key=key[:9]
        for messagePart in message:
            add = 0
            if len(messagePart) < 3:
                add = 3 - len(messagePart)
                for i in range(add):
                    messagePart = ''.join([messagePart, 'X'])
            if encrypt:
                cipherText = HillCipherEncryption(messagePart, key)
                resultList.append(cipherText)
            else:
                decipherText = HillCipherDecryption(messagePart, key)
                resultList.append(decipherText)
        return ''.join(resultList)
```

Enigma.py

```
#enigma

from textwrap import wrap

def permutate(rotor, alphabetList):
    newAlphabet = ''.join(alphabetList)
    newAlphabetList = list(newAlphabet)
    for i in range(rotor):
        newAlphabetList.insert(0, newAlphabetList[-1])
        newAlphabetList.pop(-1)
    return newAlphabetList

def inversePermutation(rotor, alphabetList):
    newAlphabet = ''.join(alphabetList)
    newAlphabetList = list(newAlphabet)
    for i in range(rotor):
        newAlphabetList.append(newAlphabetList[0])
        newAlphabetList.pop(0)
    return newAlphabetList

def turnRotors(alphaRotor, betaRotor, gammaRotor, alphabetList):
    alphaRotor += 1
    if alphaRotor % len(alphabetList) == 0:
        betaRotor += 1
        alphaRotor = 0
    if betaRotor % len(alphabetList) == 0 and alphaRotor % len(alphabetList) != 0
and betaRotor >= (len(alphabetList) - 1):
        gammaRotor += 1
        betaRotor = 1
    return alphaRotor, betaRotor, gammaRotor

def encryptDecrypt(text, steckerbrettDictionary, alphaRotor, betaRotor,
gammaRotor, alphabetList):
    for letter in list(steckerbrettDictionary.keys()):
        if letter in alphabetList:
            alphabetList.remove(letter)
            alphabetList.remove(steckerbrettDictionary[letter])
            steckerbrettDictionary.update({steckerbrettDictionary[letter]:letter})
    print("steckerbrettDictionary = ", steckerbrettDictionary)
```

```
    reflector = [letter for letter in reversed(alphabetList)]
    print("reflector = ", reflector)
    resultText = []
    upperCaseText = toUpperCase(text)
    upperCaseText.split()
    for letter in upperCaseText:
        if letter in steckerbrettDictionary:
            resultText.append(steckerbrettDictionary[letter])
            alphaRotor, betaRotor, gammaRotor = turnRotors(alphaRotor, betaRotor,
gammaRotor, alphabetList)
        else:
            print("alphaRotor = ", alphaRotor)
            print("alphabetList = ", alphabetList)
            newList = permutate((alphaRotor), alphabetList)
            tempLetter = newList[alphabetList.index(letter)]
            #tempLetter = permutate((alphaRotor),
alphabetList)[alphabetList.index(letter)]
            print(newList)
            print(tempLetter)
            tempLetter = permutate((betaRotor),
alphabetList)[alphabetList.index(tempLetter)]
            tempLetter = permutate((gammaRotor),
alphabetList)[alphabetList.index(tempLetter)]
            tempLetter = reflector[alphabetList.index(tempLetter)]
            tempLetter = inversePermutation((gammaRotor),
alphabetList)[alphabetList.index(tempLetter)]
            tempLetter = inversePermutation((betaRotor),
alphabetList)[alphabetList.index(tempLetter)]
            tempLetter = inversePermutation((alphaRotor),
alphabetList)[alphabetList.index(tempLetter)]
            resultText.append(tempLetter)
            alphaRotor, betaRotor, gammaRotor = turnRotors(alphaRotor, betaRotor,
gammaRotor, alphabetList)
    return ''.join(resultText)

def readTextFromFile(path):
    file1 = open(path, "r")
    data = file1.read()
    file1.close()
    return data

def write_to_file(path, text):
    file1 = open(path,"w+")
    file1.write(text)
    file1.close()

def wrapFiveCharacters(message):
    messageWrapFive = wrap(message,5)
    return ' '.join(messageWrapFive)


def toUpperCase(text):
    return "".join(filter(str.isupper, text.upper()))
```

Screenshot Program

1. Vigenere Standard
    a. No Space

i.

b. Five Char

i.

2. Full Vigenere
    a. No Space

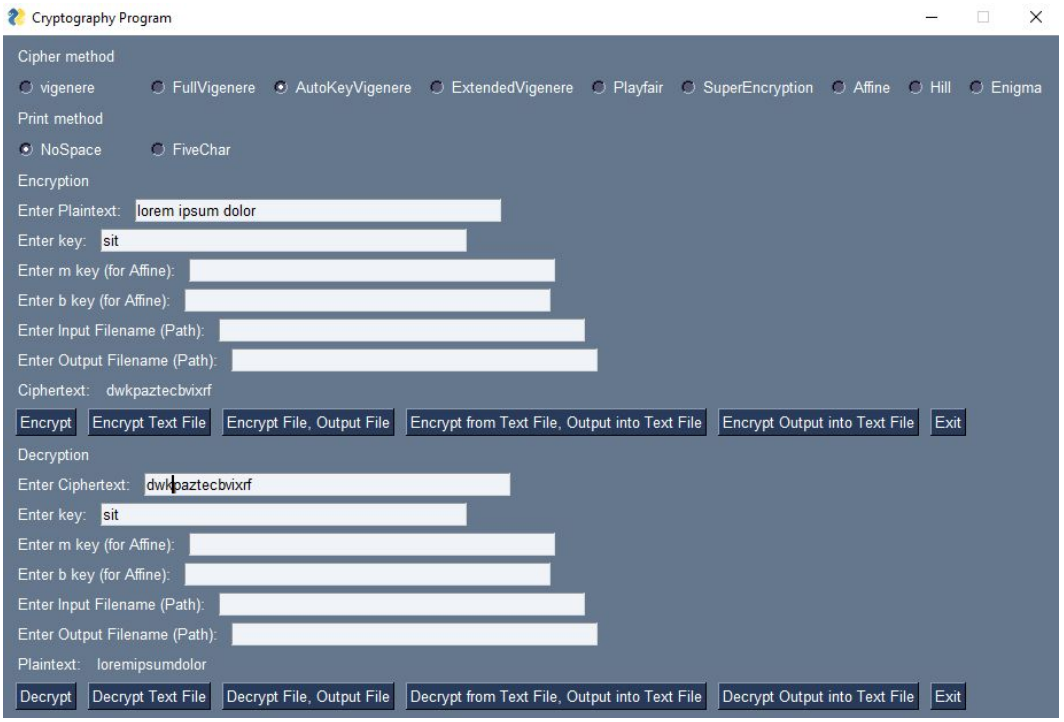
i.

b. Five Char

i.

3. Auto Key Vigenere
   a. No Space

i.



b. Five Char

i.

4. Extended Vigenere
    a. No Space

i.

b. Five Char

i.

5. Playfair
    a. No Space

i.

b. Five Char

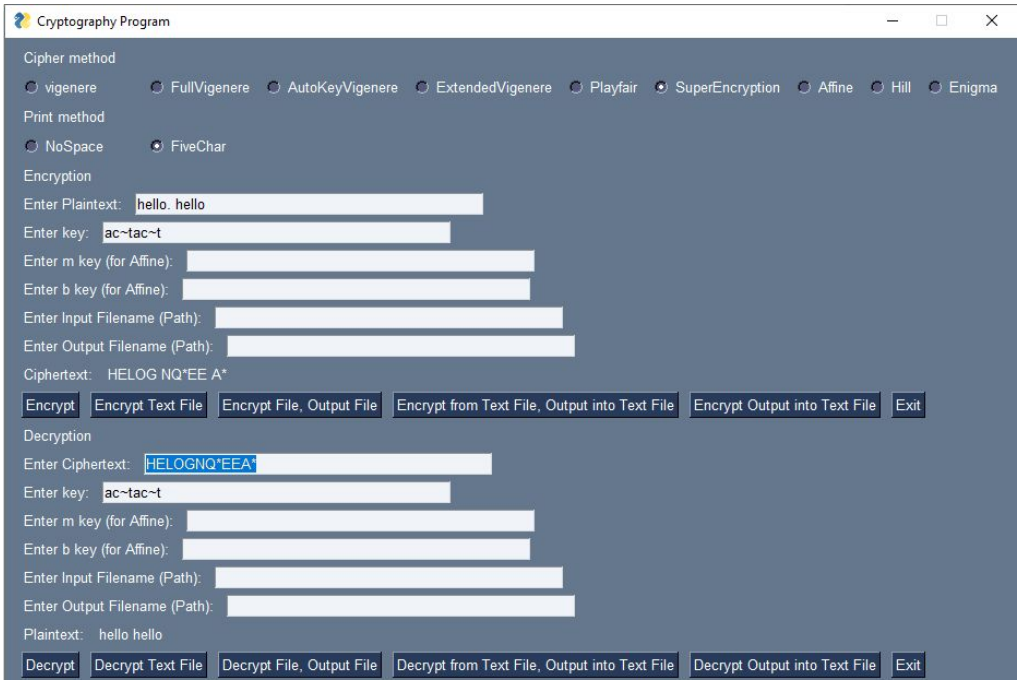
i.

6. SuperEncryption (Vigenere Standard + Myszkowski Transposition Cipher)
   a. No Space

i.

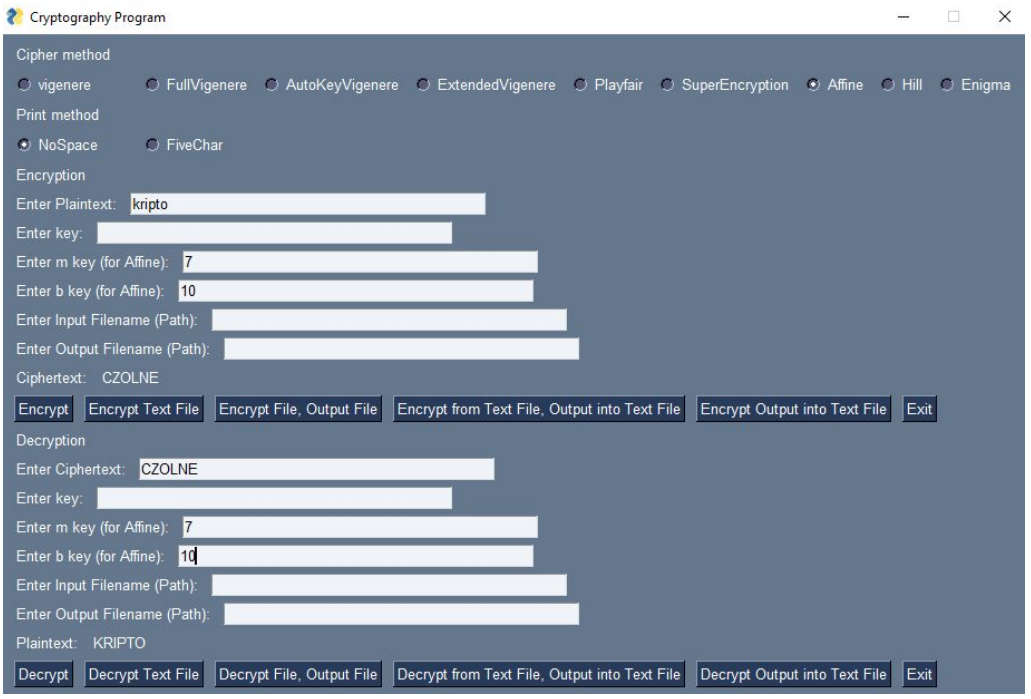
b. Five Char

i.



7. Affine
   a. No Space

i.



b. Five Char

i.

8. Hill
    a. No Space

i.

b. Five Char

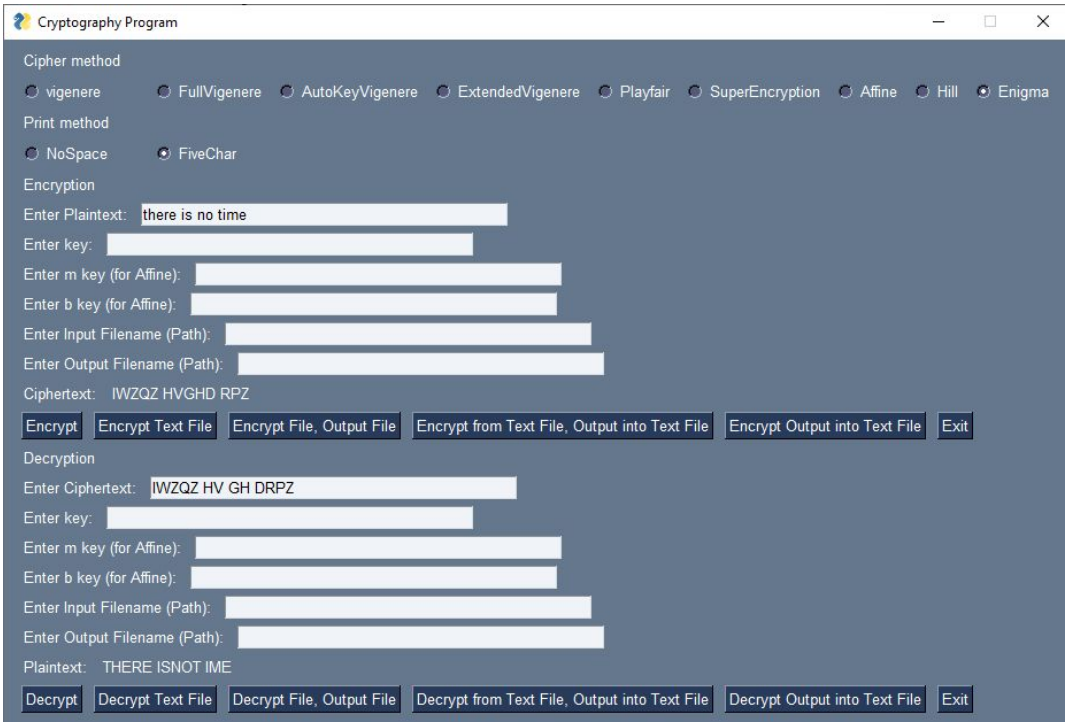
i.

9. Enigma
    a. No Space

i.

b. Five Char

i.



Sample Plaintext, Key, Ciphertext

1. Vigenere
    a. plaintext : lorem ipsum dolor
    b. key : sit
    c. ciphertext : dwkwu-bhane-lhdwk

2. Full Vigenere
    a. plaintext : Lorem ipsum dolor
    b. key : sit
    c. ciphertext : PMVEYZFBLJKYPMV
3. Auto Key Vigenere
    a. plaintext : Lorem ipsum dolor
    b. key : sit
    c. ciphertext : dwkpa-ztecb-vixrf
4. Extended Vigenere
    a. Plaintext: hellohello
    b. Key: hello
    c. Ciphertext: oiww}oiww}
5. Playfair
    a. Plaintext: Hi.de th.e go.ld in th.e tr.ee.stu.mp!
    b. key = "playfair example"
    c. Ciphertext: BMODZBXDNABEKUDMUIXMMOUVIF
6. SuperEncryption
    a. Plaintext: hello. hello
    b. Key: ac~tac~t
    c. Ciphertext: HELOGNQ*EEA*
7. Affine
    a. Affine
    b. plaintext : kripto
    c. key (m) : 7
    d. key (b) : 10
    e. ciphertext : czoln-e
8. Hill
    a. Plaintext: AC.TAC.T
    b. Key: GYBNQKURP
    c. Ciphertext: POHPOH
9. Enigma
    a. Plaintext: there is no time
    b. No. of steckerbrett pairs: 2
    c. Pair #1
    d. First alphabet: B
    e. Second alphabet: A
    f. Pair #1
    g. First alphabet: E
    h. Second alphabet: Z
    i. Alpha Rotor: 3
    j. Beta Rotor: 17
    k. Gamma Rotor: 24
    l. Ciphertext: IWZQZ HV GH DRPZ

Link Github

https://github.com/muhammadrizkifonna/Tugas_Kecil_1_IF4020_Kriptografi_13516001_13517044

How to run

- Open directory in terminal or cmd
- Type in terminal or cmd

```
python3 main.py
```

No	Spek	Berhasil (v)	Kurang Berhasil (v)	Keterangan
1	Vigenere Standard	v		
2	Full Vigenere Cipher	v		
3	Auto-Key Vigenere Cipher	v		
4	Extended Vigenere Cipher	v		
5	Playfair Cipher	v		
6	Super Enkripsi	v		
7	Affine Cipher	v		
8	Hill Cipher (matriks 3x3)	v		
9	Bonus: Enigma Cipher	v		