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

Ignatius Timothy Manullang / 13517044

Muhammad Rizki Fonna / 13516001

Source Code

main.py

```python
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')]]


window = sg.Window('Cryptography Program', layout)

def enigmaEncryptDecryptInit():
    alphabet= "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    alphabetList = list(alphabet)
    steckerbrettDictionary = {' ': ' '}
    pairsInSteckerbrett = sg.popup_get_text('Input the number of pairs in steckerbrett (min 0)', 'Input the number of pairs in steckerbrett')
    for i in range(int(pairsInSteckerbrett)):
        firstSteckerbrettAlphabet = sg.popup_get_text("Pair #" + str(i+1),"Input the first alphabet in pair: ")
        secondSteckerbrettAlphabet = sg.popup_get_text("Pair #" + str(i+1), "Input the second alphabet in pair: ")
        steckerbrettDictionary[firstSteckerbrettAlphabet.upper()] = secondSteckerbrettAlphabet.upper()
    #print("Input alpha, beta and gamma rotor s")
    alphaRotor = int(sg.popup_get_text("Input Alpha Rotor shift (0-25)", "Input Alpha Rotor shift (0-25)"))
    betaRotor = int(sg.popup_get_text("Input Beta Rotor shift (0-25)", "Input Beta Rotor shift (0-25)"))
    gammaRotor = int(sg.popup_get_text("Input Gamma Rotor shift (0-25)", "Input Gamma Rotor shift (0-25)"))
    return alphabetList, steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor

while True:
    event, values = window.read()
    if event is None or event == 'Exit':
        break
    elif event=='Encrypt':
        if values['Vigenere']:
            if values['FiveChar']:
window['-CIPHERTEXT_ENCRYPT-'].update(vg.wrapFiveCharacters(vg.encryption(values['-PLAINTEXT_ENCRYPT-'], values['-KEY_ENCRYPT-'], False, '', False)))
            else:
window['-CIPHERTEXT_ENCRYPT-'].update(vg.encryption(values['-PLAINTEXT_ENCRYPT-'], values['-KEY_ENCRYPT-'], False, '', False))
        if values['FullVigenere']:
            if values['FiveChar']:
window['-CIPHERTEXT_ENCRYPT-'].update(fvg.wrapFiveCharacters(fvg.encryption(values['-PLAINTEXT_ENCRYPT-'], values['-KEY_ENCRYPT-'], False, '', False)))
            else:
window['-CIPHERTEXT_ENCRYPT-'].update(fvg.encryption(values['-PLAINTEXT_ENCRYPT-'], values['-KEY_ENCRYPT-'], False, '', False))
        if values['RunningKeyVigenere']:
            if values['FiveChar']:
window['-CIPHERTEXT_ENCRYPT-'].update(avg.wrapFiveCharacters(avg.encryption(values['-PLAINTEXT_ENCRYPT-'], values['-KEY_ENCRYPT-'], False, '', False)))
            else:
window['-CIPHERTEXT_ENCRYPT-'].update(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.wrapFiveCharacters(vge.encryptTextExtend
edVigenere(message, key)))
            else:
window['-CIPHERTEXT_ENCRYPT-'].update(vge.encryptTextExtendedVigenere(message, key))
        if values['Playfair']:
            message = values['-PLAINTEXT_ENCRYPT-']
            key = values['-KEY_ENCRYPT-']
            if values['FiveChar']:
window['-CIPHERTEXT_ENCRYPT-'].update(p.wrapFiveCharacters(p.toUpperCase(p.playfairEncrypt(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.myszkowskiTranspositionEncrypt(vg.encryption(values['-PLAINTEXT_ENCRYPT-'], values['-KEY_ENCRYPT-'], False, '', False), values['-KEY_ENCRYPT-'])))
            else:
window['-CIPHERTEXT_ENCRYPT-'].update(se.myszkowskiTranspositionEncrypt(vg.encryption(values['-PLAINTEXT_ENCRYPT-'], values['-KEY_ENCRYPT-'], False, '', False), values['-KEY_ENCRYPT-']))
            if values['Affine']:
                if values['FiveChar']:
window['-CIPHERTEXT_ENCRYPT-'].update(af.wrapFiveCharacters(af.encryption(values['-PLAINTEXT_ENCRYPT-'], values['-KEY_ENCRYPT_M-'], values['-KEY_ENCRYPT_B-'], False, '', False)))
                else:
window['-CIPHERTEXT_ENCRYPT-'].update(af.encryption(values['-PLAINTEXT_ENCRYPT-'], values['-KEY_ENCRYPT_M-'], values['-KEY_ENCRYPT_B-'], False, '', False))
            if values['Hill']:
                message = values['-PLAINTEXT_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 = values['-PLAINTEXT_ENCRYPT-']
                alphabetList, steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor = enigmaEncryptDecryptInit()
                print("gammaRotor = ", gammaRotor)
                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 Text File':
            if values['Vigenere']:
                if values['FiveChar']:
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.encryptTextExtendedVigenere(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.playfairEncrypt(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.myszkowskiTranspositionEncrypt(vg.encryption('', values['-KEY_ENCRYPT-'], True, values['-PATH_SOURCE_ENCRYPT-']), values['-KEY_ENCRYPT-'])))
                else:
                    window['-CIPHERTEXT_ENCRYPT-'].update(se.myszkowskiTranspositionEncrypt(vg.encryption('', 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'])
            if values['ExtendedVigenere']:
                message = e.readTextFromFile(values['-PATH_SOURCE_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 = e.readTextFromFile(values['-PATH_SOURCE_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))))
            if values['SuperEncryption']:
                if values['FiveChar']:
                    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-']))))
                else:
                    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-'])))
            if values['Affine']:
                if values['FiveChar']:
                    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-']))))
                else:
                    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-'])))
            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.write_to_file(values['-PATH_ENCRYPT-'], h.wrapFiveCharacters(resultMessage)))
                else:
                    window['-CIPHERTEXT_ENCRYPT-'].update(h.write_to_file(values['-PATH_ENCRYPT-'], 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.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))))
            if values['SuperEncryption']:
                if values['FiveChar']:
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-']))))
                else:
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-'])))
            if values['Affine']:
                if values['FiveChar']:
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))))
                else:
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)))
            if values['Hill']:
                message = values['-PLAINTEXT_ENCRYPT-']
                key = values['-KEY_ENCRYPT-']
                resultMessage = h.generateHillResultMessage(message, key, encrypt
= True)
                if values['FiveChar']:
window['-CIPHERTEXT_ENCRYPT-'].update(h.write_to_file(values['-PATH_ENCRYPT-'],
h.wrapFiveCharacters(resultMessage)))
                else:
window['-CIPHERTEXT_ENCRYPT-'].update(h.write_to_file(values['-PATH_ENCRYPT-'],
resultMessage))
            if values['Enigma']:
                text = values['-PLAINTEXT_ENCRYPT-']
                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=='Decrypt':
            if values['Vigenere']:
                if values['FiveChar']:
window['-PLAINTEXT_DECRYPT-'].update(vg.wrapFiveCharacters(vg.decryption(values['-
CIPHERTEXT_DECRYPT-'], values['-KEY_DECRYPT-'])))
                else:
window['-PLAINTEXT_DECRYPT-'].update(vg.decryption(values['-CIPHERTEXT_DECRYPT-'],
values['-KEY_DECRYPT-']))
            if values['FullVigenere']:
                if values['FiveChar']:
```

```
window['-PLAINTEXT_DECRYPT-'].update(fvg.wrapFiveCharacters(fvg.decryption(values[
'-CIPHERTEXT_DECRYPT-'], values['-KEY_DECRYPT-'])))
                else:
window['-PLAINTEXT_DECRYPT-'].update(fvg.decryption(values['-CIPHERTEXT_DECRYPT-']
, values['-KEY_DECRYPT-']))
            if values['RunningKeyVigenere']:
                if values['FiveChar']:
window['-PLAINTEXT_DECRYPT-'].update(avg.wrapFiveCharacters(avg.decryption(values[
'-CIPHERTEXT_DECRYPT-'], values['-KEY_DECRYPT-'])))
                else:
window['-PLAINTEXT_DECRYPT-'].update(avg.decryption(values['-CIPHERTEXT_DECRYPT-']
, values['-KEY_DECRYPT-']))
            if values['ExtendedVigenere']:
                message = values['-CIPHERTEXT_DECRYPT-']
                key = values['-KEY_DECRYPT-']
                if values['FiveChar']:
window['-PLAINTEXT_DECRYPT-'].update(vge.wrapFiveCharacters(vge.decryptTextExtende
dVigenere(message, key)))
                else:
window['-PLAINTEXT_DECRYPT-'].update(vge.decryptTextExtendedVigenere(message,
key))
            if values['Playfair']:
                message = values['-CIPHERTEXT_DECRYPT-']
                key = values['-KEY_DECRYPT-']
                if values['FiveChar']:
window['-PLAINTEXT_DECRYPT-'].update(p.wrapFiveCharacters(p.toUpperCase(p.playfair
Decrypt(message, key))))
                else:
window['-PLAINTEXT_DECRYPT-'].update(p.toUpperCase(p.playfairDecrypt(message,
key)))
            if values['SuperEncryption']:
                if values['FiveChar']:
window['-PLAINTEXT_DECRYPT-'].update(vg.wrapFiveCharacters(vg.decryption(se.myszko
wskiTranspositionDecrypt(values['-CIPHERTEXT_DECRYPT-'], values['-KEY_DECRYPT-']),
values['-KEY_DECRYPT-'])))
                else:
window['-PLAINTEXT_DECRYPT-'].update(vg.decryption(se.myszkowskiTranspositionDecry
pt(values['-CIPHERTEXT_DECRYPT-'], values['-KEY_DECRYPT-']),
values['-KEY_DECRYPT-']))
            if values['Affine']:
                if values['FiveChar']:
window['-PLAINTEXT_DECRYPT-'].update(af.wrapFiveCharacters(af.decryption(values['-
CIPHERTEXT_DECRYPT-'], values['-KEY_DECRYPT_M-'], values['-KEY_DECRYPT_B-'],
False, '', False)))
                else:
window['-PLAINTEXT_DECRYPT-'].update(af.decryption(values['-CIPHERTEXT_DECRYPT-'],
values['-KEY_DECRYPT_M-'], values['-KEY_DECRYPT_B-'], False, '', False))
            if values['Hill']:
                message = values['-CIPHERTEXT_DECRYPT-']
                key = values['-KEY_DECRYPT-']
                resultMessage = h.generateHillResultMessage(message, key, encrypt
= False)
                if values['FiveChar']:
                    window['-PLAINTEXT_DECRYPT-'].update(h.wrapFiveCharacters(resultMessage))
                else:
                    window['-PLAINTEXT_DECRYPT-'].update(resultMessage)
            if values['Enigma']:
                text = values['-CIPHERTEXT_DECRYPT-']
                alphabetList, steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor = enigmaEncryptDecryptInit()
                if values['FiveChar']:
                    window['-PLAINTEXT_DECRYPT-'].update(e.wrapFiveCharacters(e.encryptDecrypt(text, steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor, alphabetList)))
                else:
                    window['-PLAINTEXT_DECRYPT-'].update(e.encryptDecrypt(text, steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor, alphabetList))

        elif event=='Decrypt Text File':
            if values['Vigenere']:
                if values['FiveChar']:
                    window['-PLAINTEXT_DECRYPT-'].update(vg.wrapFiveCharacters(vg.decryption('', values['-KEY_DECRYPT-'], True, values['-PATH_SOURCE_DECRYPT-'])))
                else:
                    window['-PLAINTEXT_DECRYPT-'].update(vg.decryption('', values['-KEY_DECRYPT-'], True, values['-PATH_SOURCE_DECRYPT-']))
            if values['FullVigenere']:
                if values['FiveChar']:
                    window['-PLAINTEXT_DECRYPT-'].update(fvg.wrapFiveCharacters(fvg.decryption('', values['-KEY_DECRYPT-'], True, values['-PATH_SOURCE_DECRYPT-'])))
                else:
                    window['-PLAINTEXT_DECRYPT-'].update(fvg.decryption('', values['-KEY_DECRYPT-'], True, values['-PATH_SOURCE_DECRYPT-']))
            if values['RunningKeyVigenere']:
                if values['FiveChar']:
                    window['-PLAINTEXT_DECRYPT-'].update(avg.wrapFiveCharacters(avg.decryption('', values['-KEY_DECRYPT-'], True, values['-PATH_SOURCE_DECRYPT-'])))
                else:
                    window['-PLAINTEXT_DECRYPT-'].update(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.wrapFiveCharacters(vge.decryptTextExtendedVigenere(message, key)))
                else:
                    window['-PLAINTEXT_DECRYPT-'].update(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.wrapFiveCharacters(p.toUpperCase(p.playfairDecrypt(message, key))))
                else:
                    window['-PLAINTEXT_DECRYPT-'].update(p.toUpperCase(p.playfairDecrypt(message, key)))
            if values['SuperEncryption']:
                if values['FiveChar']:
                    window['-PLAINTEXT_DECRYPT-'].update(se.wrapFiveCharacters(se.myszkowskiTranspositionDecrypt(vg.decryption('', values['-KEY_DECRYPT-'], True, values['-PATH_SOURCE_DECRYPT-']), values['-KEY_DECRYPT-'])))
                else:
                    window['-PLAINTEXT_DECRYPT-'].update(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.wrapFiveCharacters(af.decryption('', values['-KEY_DECRYPT_M-'], values['-KEY_DECRYPT_B-'], True, values['-PATH_SOURCE_DECRYPT-'])))
                else:
                    window['-PLAINTEXT_DECRYPT-'].update(af.decryption('', values['-KEY_DECRYPT_M-'], values['-KEY_DECRYPT_B-'], True, values['-PATH_SOURCE_DECRYPT-']))
            if values['Hill']:
                message = e.readTextFromFile(values['-PATH_SOURCE_DECRYPT-'])
                key = values['-KEY_DECRYPT-']
                resultMessage = h.generateHillResultMessage(message, key, encrypt = False)
                if values['FiveChar']:
                    window['-PLAINTEXT_DECRYPT-'].update(h.wrapFiveCharacters(resultMessage))
                else:
                    window['-PLAINTEXT_DECRYPT-'].update(resultMessage)
            if values['Enigma']:
                text = e.readTextFromFile(values['-PATH_SOURCE_DECRYPT-'])
                alphabetList, steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor = enigmaEncryptDecryptInit()
                if values['FiveChar']:
                    window['-PLAINTEXT_DECRYPT-'].update(e.wrapFiveCharacters(e.encryptDecrypt(text, steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor, alphabetList)))
                else:
                    window['-PLAINTEXT_DECRYPT-'].update(e.encryptDecrypt(text, steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor, alphabetList))

        elif event == 'Decrypt File, Output File':
            print("hello")
            if values['ExtendedVigenere']:
                print("hello")
                listOfBytes = []
                with open(values['-PATH_SOURCE_DECRYPT-'], "rb") as f:
                    while (byte := f.read(1)):
                        listOfBytes.append(byte)
                f.close()
                keyword = values['-KEY_DECRYPT-']
                key = vge.generateKey(listOfBytes, keyword)
                decryptedText = vge.decryptByteExtendedVigenere(listOfBytes, key)
                h = open(values['-PATH_DECRYPT-'], "wb+")
                for index in range(len(decryptedText)):
                    h.write(decryptedText[index])
                h.close()

        elif event=='Decrypt from Text File, Output into Text File':
            if values['Vigenere']:
                if values['FiveChar']:
                    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-']))))
                else:
                    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-'])))
            if values['Hill']:
                message = e.readTextFromFile(values['-PATH_SOURCE_DECRYPT-'])
                key = values['-KEY_DECRYPT-']
                resultMessage = h.generateHillResultMessage(message, key, encrypt = False)
                if values['FiveChar']:
                    window['-PLAINTEXT_DECRYPT-'].update(h.write_to_file(values['-PATH_DECRYPT-'], h.wrapFiveCharacters(resultMessage)))
                else:
                    window['-PLAINTEXT_DECRYPT-'].update(h.write_to_file(values['-PATH_DECRYPT-'], resultMessage))
            if values['Enigma']:
                text = e.readTextFromFile(values['-PATH_SOURCE_DECRYPT-'])
                alphabetList, steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor = enigmaEncryptDecryptInit()
                if values['FiveChar']:
                    window['-PLAINTEXT_DECRYPT-'].update(e.write_to_file(values['-PATH_DECRYPT-'], e.wrapFiveCharacters(e.encryptDecrypt(text, steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor, alphabetList))))
                else:
                    window['-PLAINTEXT_DECRYPT-'].update(e.write_to_file(values['-PATH_DECRYPT-'], e.encryptDecrypt(text, steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor, alphabetList)))

        elif event=='Decrypt Output into Text File':
            if values['Vigenere']:
                if values['FiveChar']:
                    window['-PLAINTEXT_DECRYPT-'].update(vg.write_to_file(values['-PATH_DECRYPT-'], vg.wrapFiveCharacters(vg.decryption(values['-CIPHERTEXT_DECRYPT-'], values['-KEY_DECRYPT-']))))
                else:
                    window['-PLAINTEXT_DECRYPT-'].update(vg.write_to_file(values['-PATH_DECRYPT-'], vg.decryption(values['-CIPHERTEXT_DECRYPT-'], values['-KEY_DECRYPT-'])))
            if values['FullVigenere']:
                if values['FiveChar']:
                    window['-PLAINTEXT_DECRYPT-'].update(fvg.write_to_file(values['-PATH_DECRYPT-'], fvg.wrapFiveCharacters(fvg.decryption(values['-CIPHERTEXT_DECRYPT-'], values['-KEY_DECRYPT-']))))
                else:
                    window['-PLAINTEXT_DECRYPT-'].update(fvg.write_to_file(values['-PATH_DECRYPT-'], fvg.decryption(values['-CIPHERTEXT_DECRYPT-'], values['-KEY_DECRYPT-'])))
            if values['RunningKeyVigenere']:
                if values['FiveChar']:
                    window['-PLAINTEXT_DECRYPT-'].update(avg.write_to_file(values['-PATH_DECRYPT-'], avg.wrapFiveCharacters(avg.decryption(values['-CIPHERTEXT_DECRYPT-'], values['-KEY_DECRYPT-']))))
                else:
                    window['-PLAINTEXT_DECRYPT-'].update(avg.write_to_file(values['-PATH_DECRYPT-'],
avg.decryption(values['-CIPHERTEXT_DECRYPT-'], values['-KEY_DECRYPT-'])))
            if values['ExtendedVigenere']:
                message = values['-CIPHERTEXT_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 = values['-CIPHERTEXT_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(vg.write_to_file(values['-PATH_DECRYPT-'], vg.wrapFiveCharacters(vg.decryption(se.myszkowskiTranspositionDecrypt(values['-CIPHERTEXT_DECRYPT-'], values['-KEY_DECRYPT-']), values['-KEY_DECRYPT-']))))
                else:
                    window['-PLAINTEXT_DECRYPT-'].update(vg.write_to_file(values['-PATH_DECRYPT-'], vg.decryption(se.myszkowskiTranspositionDecrypt(values['-CIPHERTEXT_DECRYPT-'], values['-KEY_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['-CIPHERTEXT_DECRYPT-'], values['-KEY_DECRYPT_M-'], values['-KEY_DECRYPT_B-'], False, ''))))
                else:
                    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, '')))
            if values['Hill']:
                message = values['-CIPHERTEXT_DECRYPT-']
                key = values['-KEY_DECRYPT-']
                resultMessage = h.generateHillResultMessage(message, key, encrypt = False)
                if values['FiveChar']:
                    window['-PLAINTEXT_DECRYPT-'].update(h.write_to_file(values['-PATH_DECRYPT-'], h.wrapFiveCharacters(resultMessage)))
                else:
                    window['-PLAINTEXT_DECRYPT-'].update(h.write_to_file(values['-PATH_DECRYPT-'], resultMessage))
            if values['Enigma']:
                text = values['-CIPHERTEXT_DECRYPT-']
                alphabetList, steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor = enigmaEncryptDecryptInit()
                if values['FiveChar']:
                    window['-PLAINTEXT_DECRYPT-'].update(e.write_to_file(values['-PATH_DECRYPT-'],
e.wrapFiveCharacters(e.encryptDecrypt(text, steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor, alphabetList))))
                else:
window['-PLAINTEXT_DECRYPT-'].update(e.write_to_file(values['-PATH_DECRYPT-'], e.encryptDecrypt(text, steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor, alphabetList)))

    window.close()
```

Vigenere.py (vigenere standar)

```
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
```

```python
    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(',', '')

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

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

        # 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



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

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

            sisa = len(ciphertext)-len(key)

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


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

Autokey_vigenere.py

```python
#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)

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

        sisa = len(plaintext)-len(key)

        for i in range(sisa):
            key+=plaintext[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(',', '')

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

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

        # 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)

```

```python
    #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(' ', '')
```

```python

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

        # 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



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

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

            sisa = len(ciphertext)-len(key)

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


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

        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)

        """
       f = open("full_vigenere_matrix.txt", "r")
       f1=f.readlines()
       for x in f1:
           alphabet_shuffled.append(x)
       """

```

```python
        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

```python
#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)
```

```python
    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)

    ciphertext=toUpperCase(ciphertext)

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

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

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

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

        #Alphabet list for conversion from number to character#
        alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
        alphabet = list(alphabet)
        key_m=int(key_m)
        key_b=int(key_b)
        #Find the inverse of key_m

        key_m_inverse = getModularInverse(key_m, 26)

        plaintext=''
        for i in range(len(ciphertext)):
            plainnumber=(key_m_inverse*(alphabet.index(ciphertext[i])-key_b))%26

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

Hill.py

```
# Hill Cipher

from textwrap import wrap

def generateMessageVector(message, n):
    messageVector = [[0] for i in range(n)]
    for i in range(n):
        messageVector[i][0] = ord(message[i]) % 65
    return messageVector

def generateKeyMatrix(key, n):
    keyMatrix = [[0] * n for i in range(n)]
    k = 0
    for i in range(n):
        for j in range(n):
            print(key[k])
            keyMatrix[i][j] = ord(key[k]) % 65
            k += 1
    return keyMatrix

```

```python
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

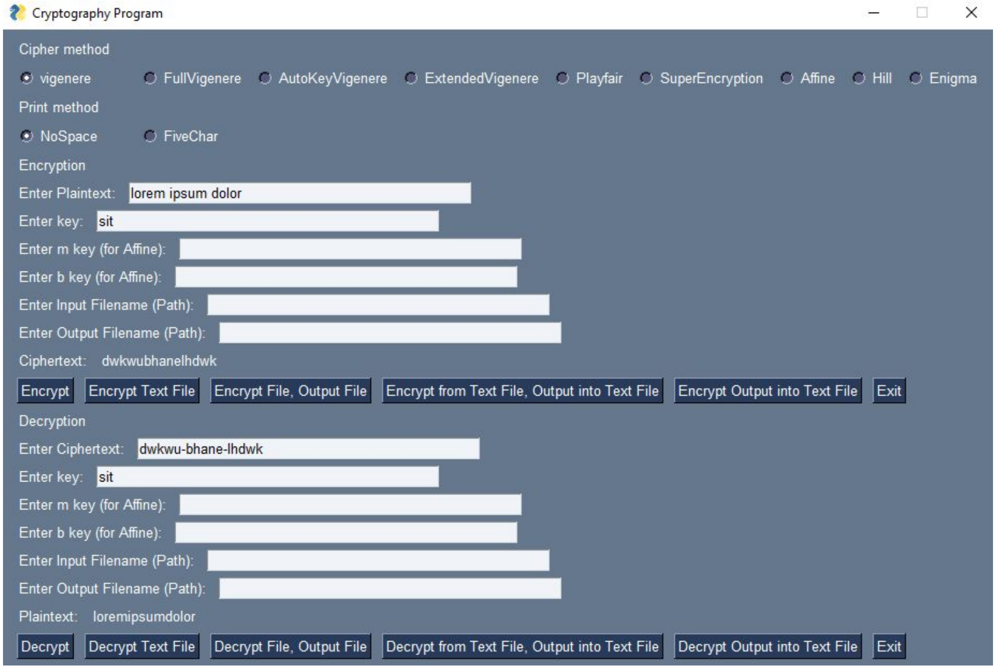
```
#enigma

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)
```

```python
    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.



Cryptography Program

Cipher method

● vigenere ○ FullVigenere ○ AutoKeyVigenere ○ ExtendedVigenere ○ Playfair ○ SuperEncryption ○ Affine ○ Hill ○ Enigma

Print method

● NoSpace ○ FiveChar

Encryption

Enter Plaintext: lorem ipsum dolor

Enter key: sit

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

Enter Output Filename (Path):

Ciphertext: dwkwubhanelhdwk

Decryption

Enter Ciphertext: dwkwu-bhane-lhdwk

Enter key: sit

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

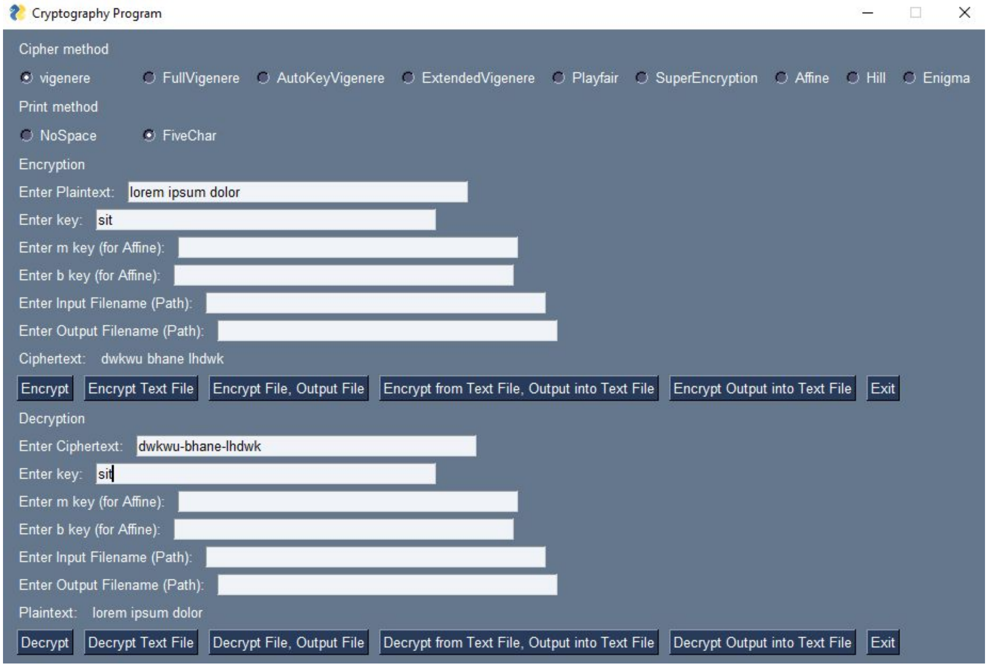
Enter Output Filename (Path):

Plaintext: loremipsumdolor

Encrypt
Encrypt Text File
Encrypt File, Output File
Encrypt from Text File, Output into Text File
Encrypt Output into Text File
Exit
Decrypt
Decrypt Text File
Decrypt File, Output File
Decrypt from Text File, Output into Text File
Decrypt Output into Text File
Exit

b. Five Char

i.



Cryptography Program

Cipher method

● vigenere ○ FullVigenere ○ AutoKeyVigenere ○ ExtendedVigenere ○ Playfair ○ SuperEncryption ○ Affine ○ Hill ○ Enigma

Print method

○ NoSpace ● FiveChar

Encryption

Enter Plaintext: lorem ipsum dolor

Enter key: sit

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

Enter Output Filename (Path):

Ciphertext: dwkwu bhane lhdwk

Decryption

Enter Ciphertext: dwkwu-bhane-lhdwk

Enter key: sit

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

Enter Output Filename (Path):

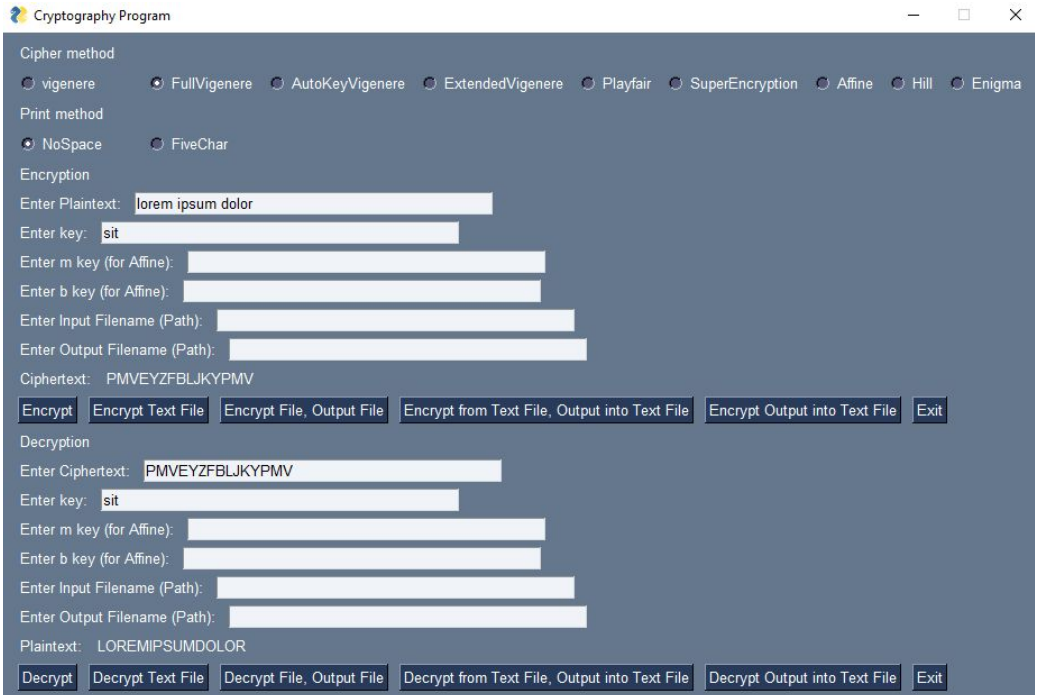
Plaintext: lorem ipsum dolor

Encrypt
Encrypt Text File
Encrypt File, Output File
Encrypt from Text File, Output into Text File
Encrypt Output into Text File
Exit
Decrypt
Decrypt Text File
Decrypt File, Output File
Decrypt from Text File, Output into Text File
Decrypt Output into Text File
Exit

2. Full Vigenere

a. No Space

i.


Cryptography Program
Encrypt
Encrypt Text File
Encrypt File, Output File
Encrypt from Text File, Output into Text File
Encrypt Output into Text File
Exit
Decrypt
Decrypt Text File
Decrypt File, Output File
Decrypt from Text File, Output into Text File
Decrypt Output into Text File
Exit

Cipher method

○ vigenere ● FullVigenere ○ AutoKeyVigenere ○ ExtendedVigenere ○ Playfair ○ SuperEncryption ○ Affine ○ Hill ○ Enigma

Print method

● NoSpace ○ FiveChar

Encryption

Enter Plaintext: lorem ipsum dolor

Enter key: sit

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

Enter Output Filename (Path):

Ciphertext: PMVEYZFBLJKYPMV

Decryption

Enter Ciphertext: PMVEYZFBLJKYPMV

Enter key: sit

Enter m key (for Affine):

Enter b key (for Affine):

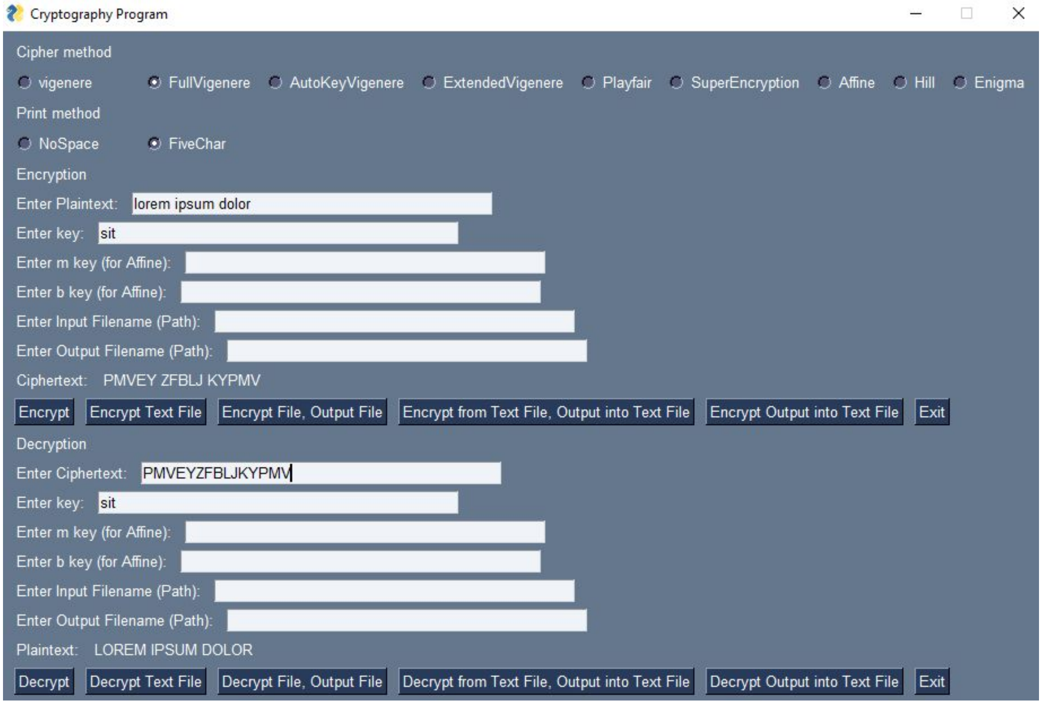
Enter Input Filename (Path):

Enter Output Filename (Path):

Plaintext: LOREMIPSUMDOLOR

b. Five Char

i.


Cryptography Program
Encrypt
Encrypt Text File
Encrypt File, Output File
Encrypt from Text File, Output into Text File
Encrypt Output into Text File
Exit
Decrypt
Decrypt Text File
Decrypt File, Output File
Decrypt from Text File, Output into Text File
Decrypt Output into Text File
Exit

Cipher method

○ vigenere ● FullVigenere ○ AutoKeyVigenere ○ ExtendedVigenere ○ Playfair ○ SuperEncryption ○ Affine ○ Hill ○ Enigma

Print method

○ NoSpace ● FiveChar

Encryption

Enter Plaintext: lorem ipsum dolor

Enter key: sit

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

Enter Output Filename (Path):

Ciphertext: PMVEY ZFBLJ KYPMV

Decryption

Enter Ciphertext: PMVEYZFBLJKYPMV

Enter key: sit

Enter m key (for Affine):

Enter b key (for Affine):

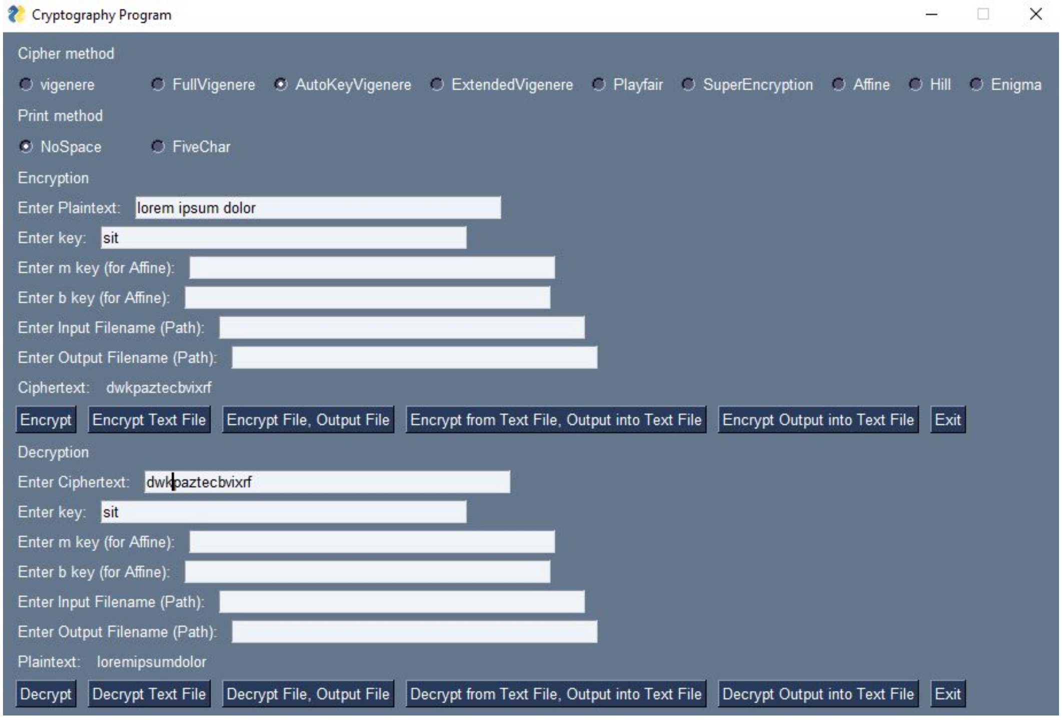
Enter Input Filename (Path):

Enter Output Filename (Path):

Plaintext: LOREM IPSUM DOLOR

3. Auto Key Vigenere

a. No Space


Cryptography Program

Cipher method

○ vigenere ○ FullVigenere ● AutoKeyVigenere ○ ExtendedVigenere ○ Playfair ○ SuperEncryption ○ Affine ○ Hill ○ Enigma

Print method

● NoSpace ○ FiveChar

Encryption

Enter Plaintext: lorem ipsum dolor

Enter key: sit

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

Enter Output Filename (Path):

Ciphertext: dwkpaztecbvixrf

Encrypt
Encrypt Text File
Encrypt File, Output File
Encrypt from Text File, Output into Text File
Encrypt Output into Text File
Exit

Decryption

Enter Ciphertext: dwkpaztecbvixrf

Enter key: sit

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

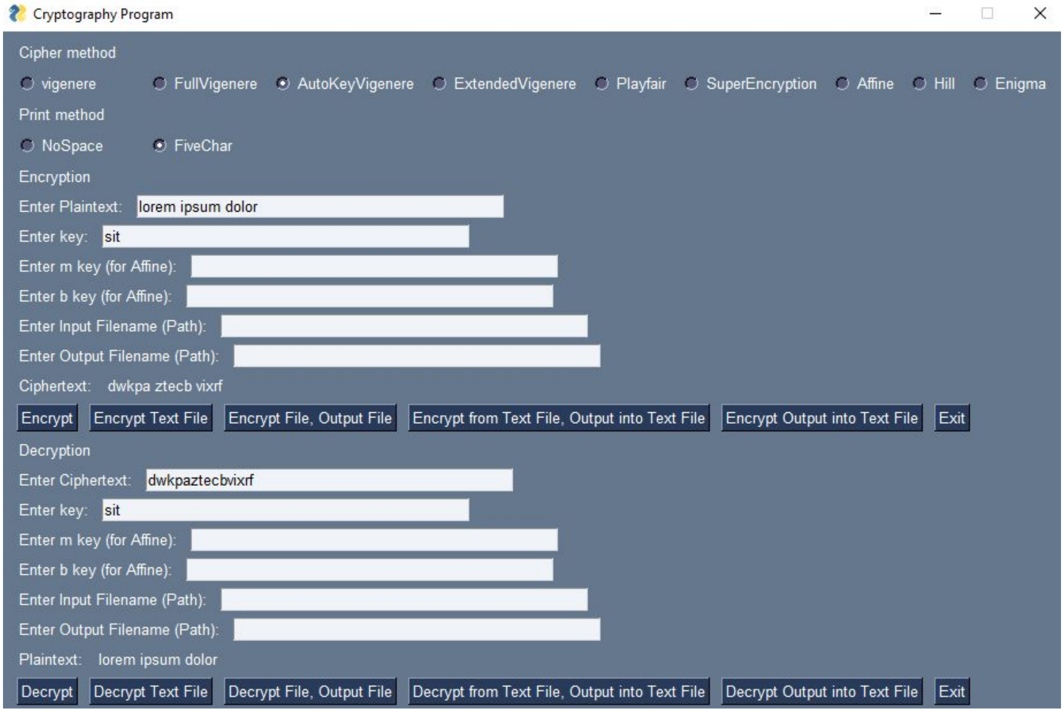
Enter Output Filename (Path):

Plaintext: loremipsumdolor

Decrypt
Decrypt Text File
Decrypt File, Output File
Decrypt from Text File, Output into Text File
Decrypt Output into Text File
Exit

i.

b. Five Char


Cryptography Program

Cipher method

○ vigenere ○ FullVigenere ● AutoKeyVigenere ○ ExtendedVigenere ○ Playfair ○ SuperEncryption ○ Affine ○ Hill ○ Enigma

Print method

○ NoSpace ● FiveChar

Encryption

Enter Plaintext: lorem ipsum dolor

Enter key: sit

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

Enter Output Filename (Path):

Ciphertext: dwkpa ztecb vixrf

Encrypt
Encrypt Text File
Encrypt File, Output File
Encrypt from Text File, Output into Text File
Encrypt Output into Text File
Exit

Decryption

Enter Ciphertext: dwkpaztecbvixrf

Enter key: sit

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

Enter Output Filename (Path):

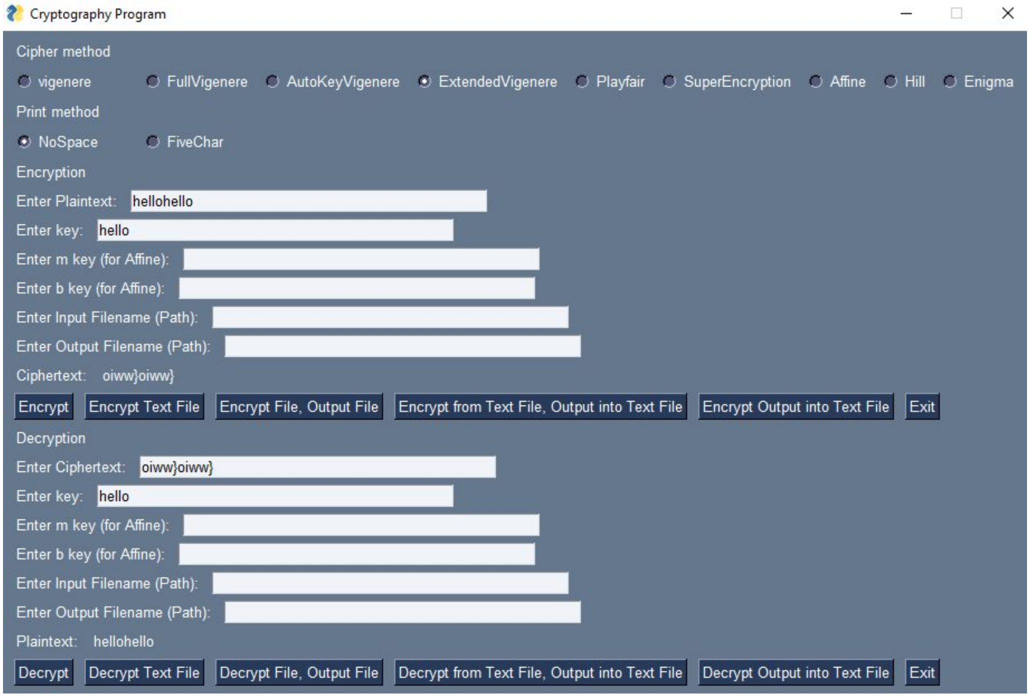
Plaintext: lorem ipsum dolor

Decrypt
Decrypt Text File
Decrypt File, Output File
Decrypt from Text File, Output into Text File
Decrypt Output into Text File
Exit

i.

4. Extended Vigenere

   a. No Space

i. 

Cipher method

○ vigenere ○ FullVigenere ○ AutoKeyVigenere ● ExtendedVigenere ○ Playfair ○ SuperEncryption ○ Affine ○ Hill ○ Enigma

Print method

● NoSpace ○ FiveChar

Encryption

Enter Plaintext: hellohello

Enter key: hello

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

Enter Output Filename (Path):

Ciphertext: oiww}oiww}

Encrypt
Encrypt Text File
Encrypt File, Output File
Encrypt from Text File, Output into Text File
Encrypt Output into Text File
Exit

Decryption

Enter Ciphertext: oiww}oiww}

Enter key: hello

Enter m key (for Affine):

Enter b key (for Affine):

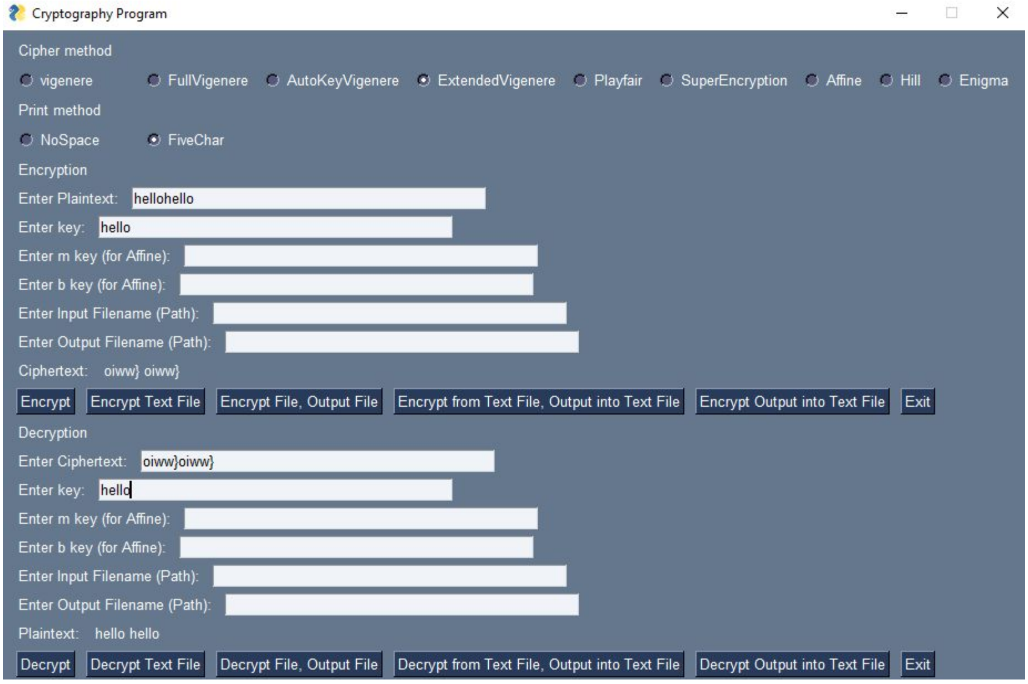
Enter Input Filename (Path):

Enter Output Filename (Path):

Plaintext: hellohello

Decrypt
Decrypt Text File
Decrypt File, Output File
Decrypt from Text File, Output into Text File
Decrypt Output into Text File
Exit

b. Five Char

i. 

Cipher method

○ vigenere ○ FullVigenere ○ AutoKeyVigenere ● ExtendedVigenere ○ Playfair ○ SuperEncryption ○ Affine ○ Hill ○ Enigma

Print method

○ NoSpace ● FiveChar

Encryption

Enter Plaintext: hellohello

Enter key: hello

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

Enter Output Filename (Path):

Ciphertext: oiww} oiww}

Encrypt
Encrypt Text File
Encrypt File, Output File
Encrypt from Text File, Output into Text File
Encrypt Output into Text File
Exit

Decryption

Enter Ciphertext: oiww}oiww}

Enter key: hello

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

Enter Output Filename (Path):

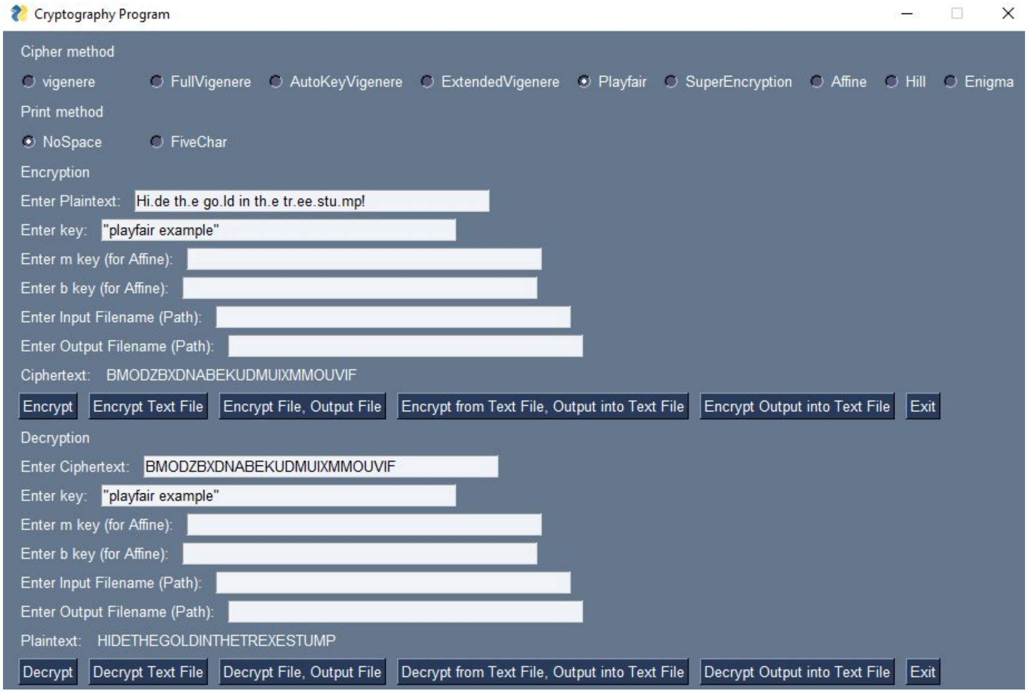
Plaintext: hello hello

Decrypt
Decrypt Text File
Decrypt File, Output File
Decrypt from Text File, Output into Text File
Decrypt Output into Text File
Exit

5. Playfair

a. No Space

i.



Cryptography Program

Cipher method

○ vigenere ○ FullVigenere ○ AutoKeyVigenere ○ ExtendedVigenere ● Playfair ○ SuperEncryption ○ Affine ○ Hill ○ Enigma

Print method

● NoSpace ○ FiveChar

Encryption

Enter Plaintext: Hi.de th.e go.ld in th.e tr.ee.stu.mp!

Enter key: "playfair example"

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

Enter Output Filename (Path):

Ciphertext: BMODZBXDNABEKUDMUIXMMOUVIF

Decryption

Enter Ciphertext: BMODZBXDNABEKUDMUIXMMOUVIF

Enter key: "playfair example"

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

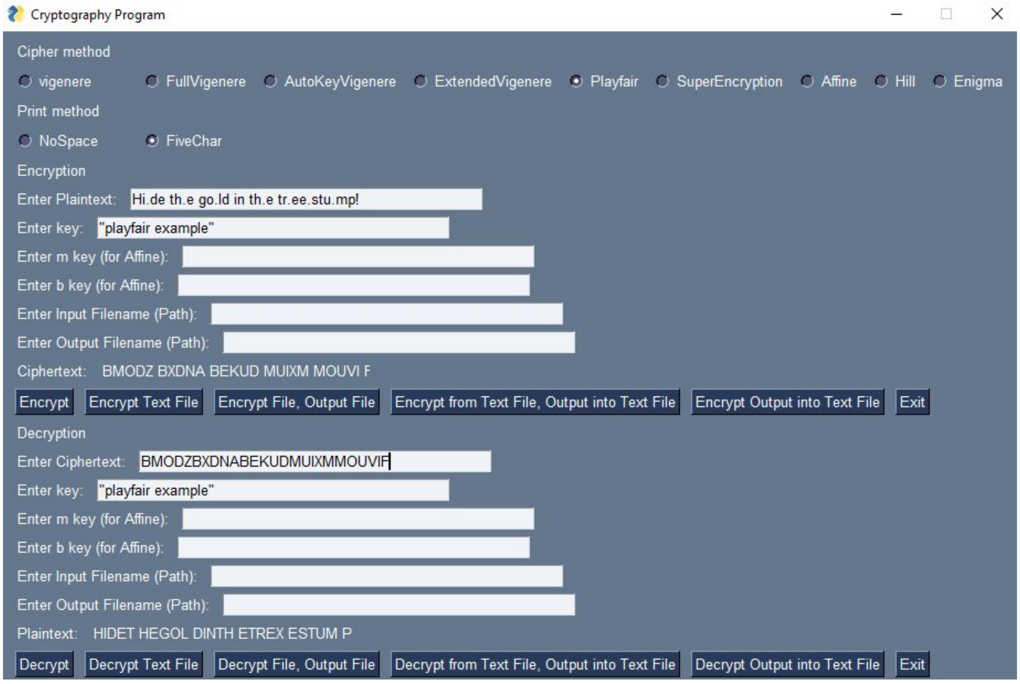
Enter Output Filename (Path):

Plaintext: HIDETHEGOLDINTHETREXESTUMP

Encrypt
Encrypt Text File
Encrypt File, Output File
Encrypt from Text File, Output into Text File
Encrypt Output into Text File
Exit
Decrypt
Decrypt Text File
Decrypt File, Output File
Decrypt from Text File, Output into Text File
Decrypt Output into Text File
Exit

b. Five Char

i.



Cryptography Program

Cipher method

○ vigenere ○ FullVigenere ○ AutoKeyVigenere ○ ExtendedVigenere ● Playfair ○ SuperEncryption ○ Affine ○ Hill ○ Enigma

Print method

○ NoSpace ● FiveChar

Encryption

Enter Plaintext: Hi.de th.e go.ld in th.e tr.ee.stu.mp!

Enter key: "playfair example"

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

Enter Output Filename (Path):

Ciphertext: BMODZ BXDNA BEKUD MUIXM MOUVI F

Decryption

Enter Ciphertext: BMODZBXDNABEKUDMUIXMMOUVIF

Enter key: "playfair example"

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

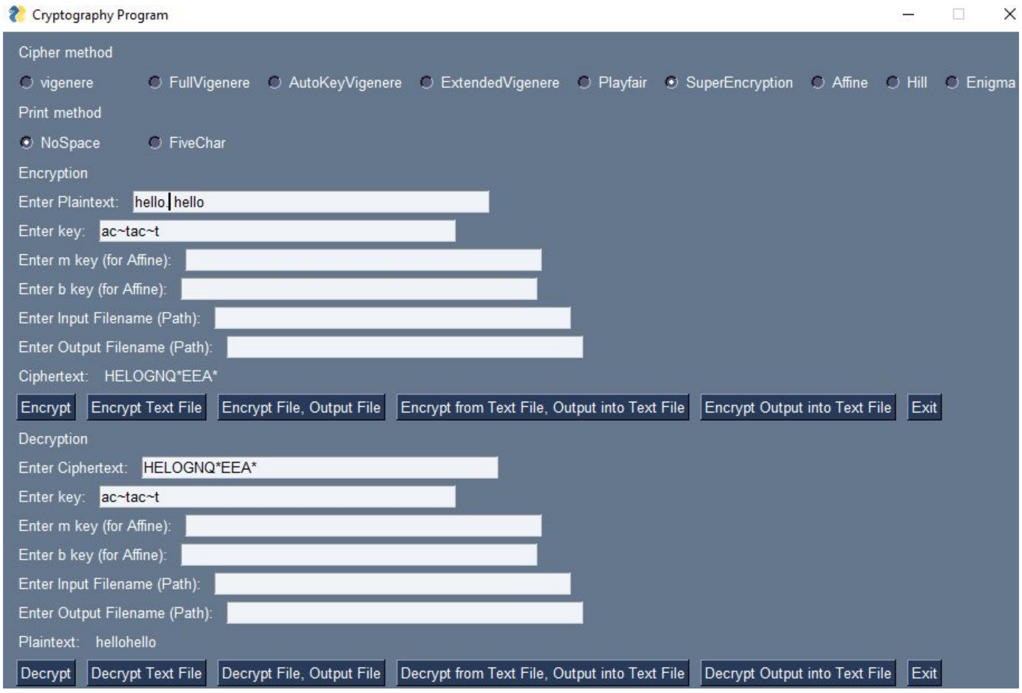
Enter Output Filename (Path):

Plaintext: HIDET HEGOL DINTH ETREX ESTUM P

Encrypt
Encrypt Text File
Encrypt File, Output File
Encrypt from Text File, Output into Text File
Encrypt Output into Text File
Exit
Decrypt
Decrypt Text File
Decrypt File, Output File
Decrypt from Text File, Output into Text File
Decrypt Output into Text File
Exit

6. SuperEncryption (Vigenere Standard + Myszkowski Transposition Cipher)

a. No Space

i. 

Cryptography Program
Encrypt
Encrypt Text File
Encrypt File, Output File
Encrypt from Text File, Output into Text File
Encrypt Output into Text File
Exit
Decrypt
Decrypt Text File
Decrypt File, Output File
Decrypt from Text File, Output into Text File
Decrypt Output into Text File
Exit

Cipher method

○ vigenere ○ FullVigenere ○ AutoKeyVigenere ○ ExtendedVigenere ○ Playfair ● SuperEncryption ○ Affine ○ Hill ○ Enigma

Print method

● NoSpace ○ FiveChar

Encryption

Enter Plaintext: hello. hello

Enter key: ac~tac~t

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

Enter Output Filename (Path):

Ciphertext: HELOGNQ*EEA*

Decryption

Enter Ciphertext: HELOGNQ*EEA*

Enter key: ac~tac~t

Enter m key (for Affine):

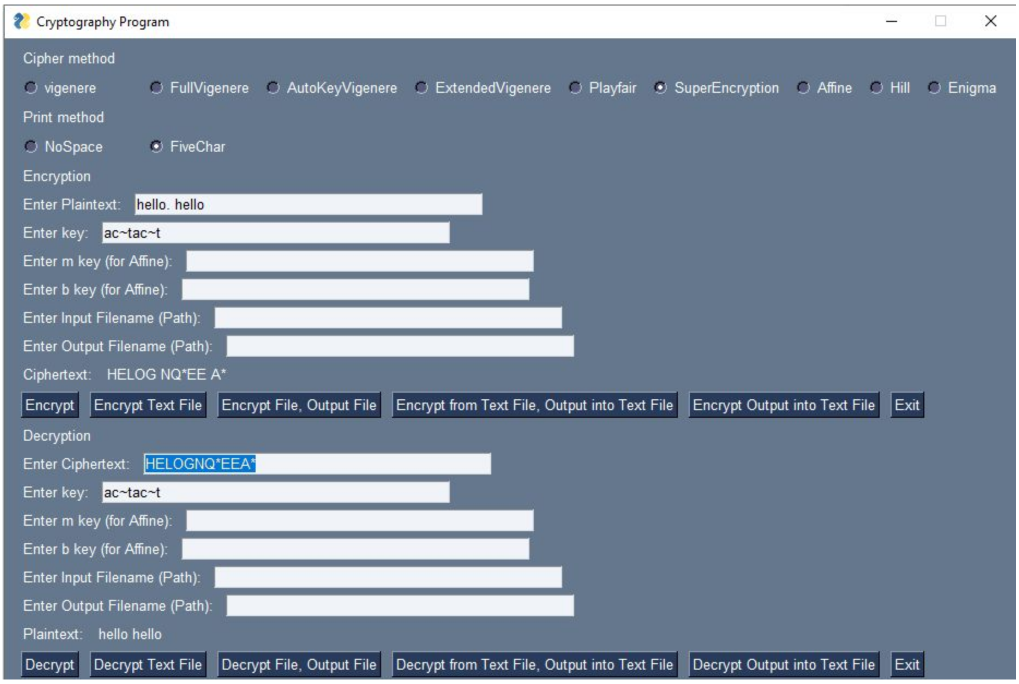
Enter b key (for Affine):

Enter Input Filename (Path):

Enter Output Filename (Path):

Plaintext: hellohello

b. Five Char

i. 

Cryptography Program
Encrypt
Encrypt Text File
Encrypt File, Output File
Encrypt from Text File, Output into Text File
Encrypt Output into Text File
Exit
Decrypt
Decrypt Text File
Decrypt File, Output File
Decrypt from Text File, Output into Text File
Decrypt Output into Text File
Exit

Cipher method

○ vigenere ○ FullVigenere ○ AutoKeyVigenere ○ ExtendedVigenere ○ Playfair ● SuperEncryption ○ Affine ○ Hill ○ Enigma

Print method

○ NoSpace ● FiveChar

Encryption

Enter Plaintext: hello. hello

Enter key: ac~tac~t

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

Enter Output Filename (Path):

Ciphertext: HELOG NQ*EE A*

Decryption

Enter Ciphertext: HELOGNQ*EEA*

Enter key: ac~tac~t

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

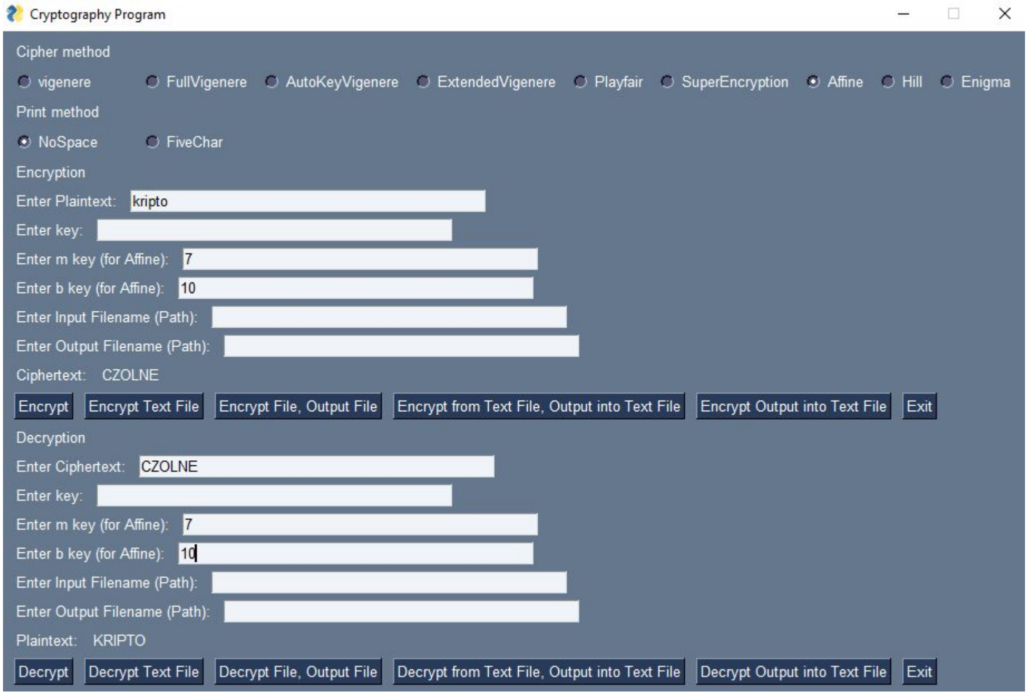
Enter Output Filename (Path):

Plaintext: hello hello

7. Affine

a. No Space

i.



Cipher method

○ vigenere ○ FullVigenere ○ AutoKeyVigenere ○ ExtendedVigenere ○ Playfair ○ SuperEncryption ● Affine ○ Hill ○ Enigma

Print method

● NoSpace ○ FiveChar

Encryption

Enter Plaintext: kripto

Enter key:

Enter m key (for Affine): 7

Enter b key (for Affine): 10

Enter Input Filename (Path):

Enter Output Filename (Path):

Ciphertext: CZOLNE

Encrypt
Encrypt Text File
Encrypt File, Output File
Encrypt from Text File, Output into Text File
Encrypt Output into Text File
Exit

Decryption

Enter Ciphertext: CZOLNE

Enter key:

Enter m key (for Affine): 7

Enter b key (for Affine): 10

Enter Input Filename (Path):

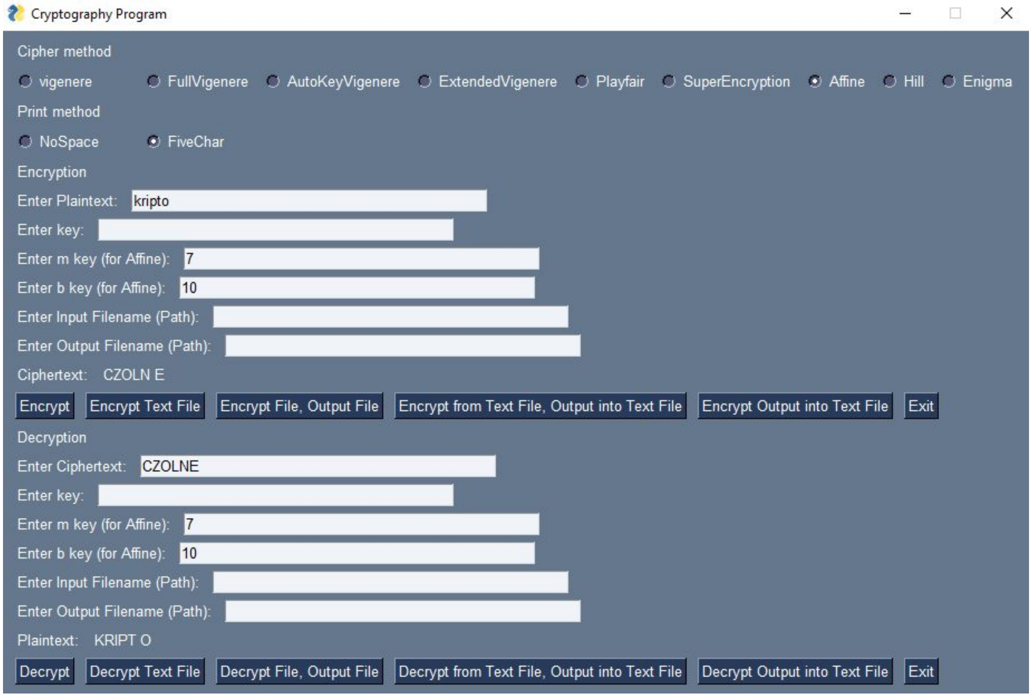
Enter Output Filename (Path):

Plaintext: KRIPTO

Decrypt
Decrypt Text File
Decrypt File, Output File
Decrypt from Text File, Output into Text File
Decrypt Output into Text File
Exit

b. Five Char

i.



Cipher method

○ vigenere ○ FullVigenere ○ AutoKeyVigenere ○ ExtendedVigenere ○ Playfair ○ SuperEncryption ● Affine ○ Hill ○ Enigma

Print method

○ NoSpace ● FiveChar

Encryption

Enter Plaintext: kripto

Enter key:

Enter m key (for Affine): 7

Enter b key (for Affine): 10

Enter Input Filename (Path):

Enter Output Filename (Path):

Ciphertext: CZOLN E

Encrypt
Encrypt Text File
Encrypt File, Output File
Encrypt from Text File, Output into Text File
Encrypt Output into Text File
Exit

Decryption

Enter Ciphertext: CZOLNE

Enter key:

Enter m key (for Affine): 7

Enter b key (for Affine): 10

Enter Input Filename (Path):

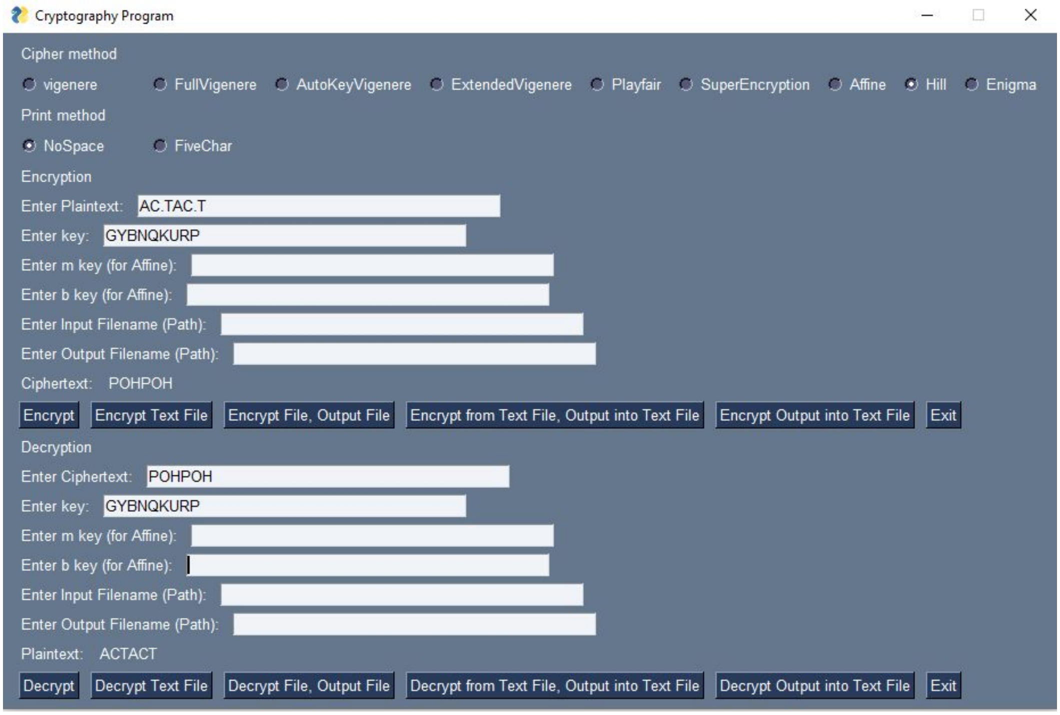
Enter Output Filename (Path):

Plaintext: KRIPT O

Decrypt
Decrypt Text File
Decrypt File, Output File
Decrypt from Text File, Output into Text File
Decrypt Output into Text File
Exit

8. Hill

a. No Space

i. 

Cryptography Program

Cipher method

○ vigenere ○ FullVigenere ○ AutoKeyVigenere ○ ExtendedVigenere ○ Playfair ○ SuperEncryption ○ Affine ● Hill ○ Enigma

Print method

● NoSpace ○ FiveChar

Encryption

Enter Plaintext: AC.TAC.T

Enter key: GYBNQKURP

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

Enter Output Filename (Path):

Ciphertext: POHPOH

Decryption

Enter Ciphertext: POHPOH

Enter key: GYBNQKURP

Enter m key (for Affine):

Enter b key (for Affine):

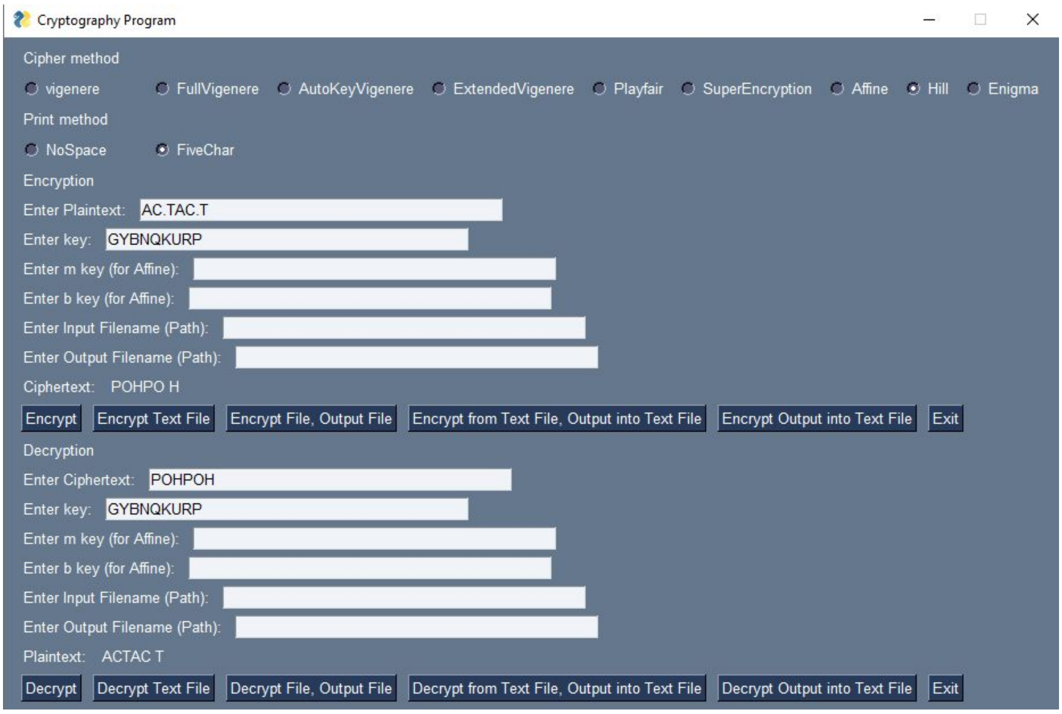
Enter Input Filename (Path):

Enter Output Filename (Path):

Plaintext: ACTACT

Encrypt
Encrypt Text File
Encrypt File, Output File
Encrypt from Text File, Output into Text File
Encrypt Output into Text File
Exit
Decrypt
Decrypt Text File
Decrypt File, Output File
Decrypt from Text File, Output into Text File
Decrypt Output into Text File
Exit

b. Five Char

i. 

Cryptography Program

Cipher method

○ vigenere ○ FullVigenere ○ AutoKeyVigenere ○ ExtendedVigenere ○ Playfair ○ SuperEncryption ○ Affine ● Hill ○ Enigma

Print method

○ NoSpace ● FiveChar

Encryption

Enter Plaintext: AC.TAC.T

Enter key: GYBNQKURP

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

Enter Output Filename (Path):

Ciphertext: POHPO H

Decryption

Enter Ciphertext: POHPOH

Enter key: GYBNQKURP

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

Enter Output Filename (Path):

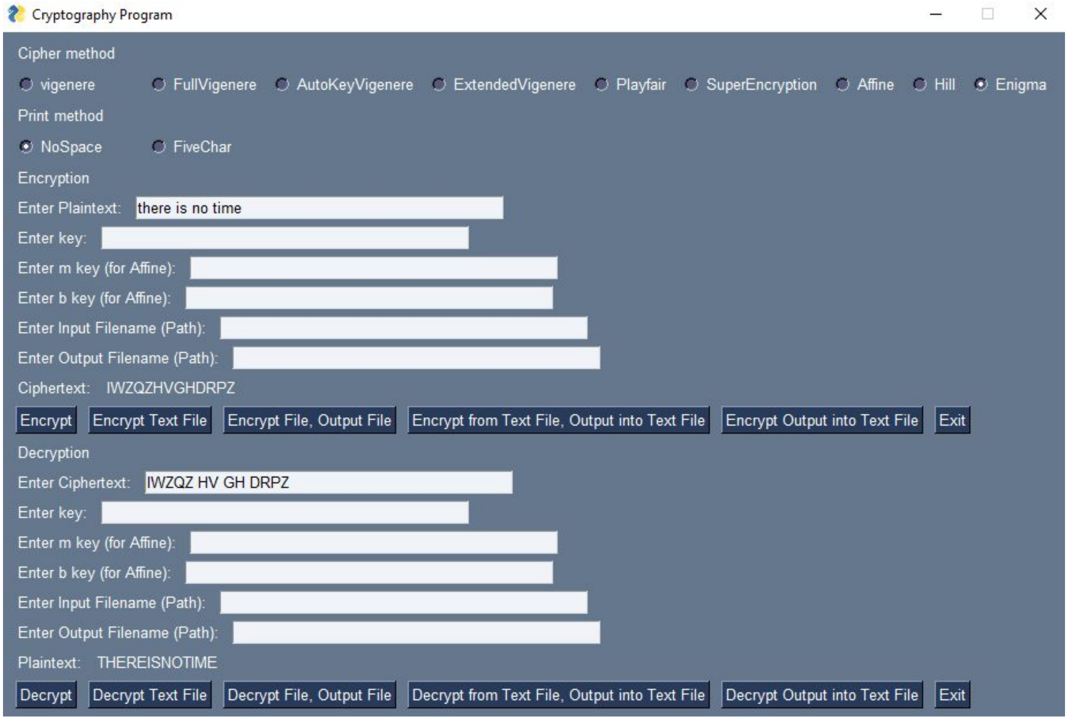
Plaintext: ACTAC T

Encrypt
Encrypt Text File
Encrypt File, Output File
Encrypt from Text File, Output into Text File
Encrypt Output into Text File
Exit
Decrypt
Decrypt Text File
Decrypt File, Output File
Decrypt from Text File, Output into Text File
Decrypt Output into Text File
Exit

9. Enigma

   a. No Space

i.


Cryptography Program

Cipher method

○ vigenere ○ FullVigenere ○ AutoKeyVigenere ○ ExtendedVigenere ○ Playfair ○ SuperEncryption ○ Affine ○ Hill ● Enigma

Print method

● NoSpace ○ FiveChar

Encryption

Enter Plaintext: there is no time

Enter key:

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

Enter Output Filename (Path):

Ciphertext: IWZQZHVGHDRPZ

Encrypt
Encrypt Text File
Encrypt File, Output File
Encrypt from Text File, Output into Text File
Encrypt Output into Text File
Exit

Decryption

Enter Ciphertext: IWZQZ HV GH DRPZ

Enter key:

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

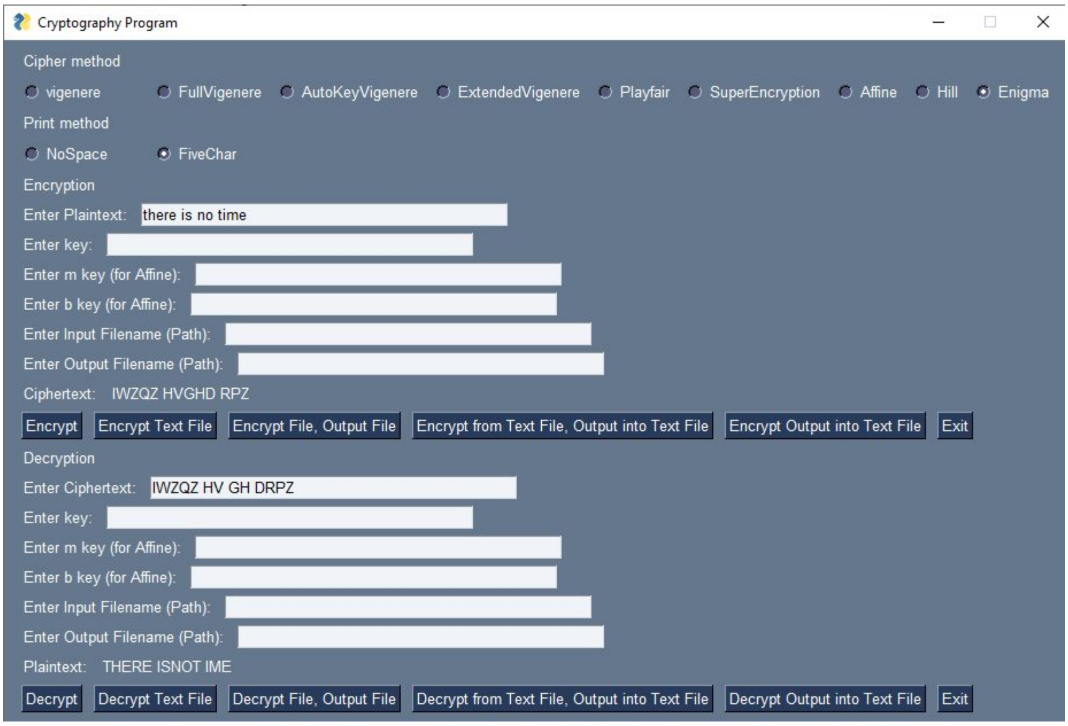
Enter Output Filename (Path):

Plaintext: THEREISNOTIME

Decrypt
Decrypt Text File
Decrypt File, Output File
Decrypt from Text File, Output into Text File
Decrypt Output into Text File
Exit

b. Five Char

i.


Cryptography Program

Cipher method

○ vigenere ○ FullVigenere ○ AutoKeyVigenere ○ ExtendedVigenere ○ Playfair ○ SuperEncryption ○ Affine ○ Hill ● Enigma

Print method

○ NoSpace ● FiveChar

Encryption

Enter Plaintext: there is no time

Enter key:

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

Enter Output Filename (Path):

Ciphertext: IWZQZ HVGHD RPZ

Encrypt
Encrypt Text File
Encrypt File, Output File
Encrypt from Text File, Output into Text File
Encrypt Output into Text File
Exit

Decryption

Enter Ciphertext: IWZQZ HV GH DRPZ

Enter key:

Enter m key (for Affine):

Enter b key (for Affine):

Enter Input Filename (Path):

Enter Output Filename (Path):

Plaintext: THERE ISNOT IME

Decrypt
Decrypt Text File
Decrypt File, Output File
Decrypt from Text File, Output into Text File
Decrypt Output into Text File
Exit

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		