

Laporan Tugas Kecil 1 IF4020 Kriptografi

Semester I Tahun 2020/2021



Ignatius Timothy Manullang / 13517044

Muhammad Rizki Fonna / 13516001

Source Code

main.py

```
1. import PySimpleGUI as sg
2. import vigenere as vg
3. import full_vigenere as fvg
4. import autokey_vigenere as avg
5. import affine as af
6. import vigenereExtended as vge
7. import enigma as e
8. import playfair as p
9. import hill as h
10. import myszkowskiTransposition as se
11.
12. layout = [[sg.Text('Cipher method')],
13.            [sg.Radio('vigenere', "cipherMethod", default=True, size=(10,1),
14.                    key='Vigenere'),
15.              sg.Radio('FullVigenere', "cipherMethod", key='FullVigenere'),
16.              sg.Radio('RunningKeyVigenere', "cipherMethod",
17.                    key='RunningKeyVigenere'),
18.              sg.Radio('ExtendedVigenere', "cipherMethod",
19.                    key='ExtendedVigenere'),
20.              sg.Radio('Playfair', "cipherMethod", key='Playfair'),
21.              sg.Radio('SuperEncryption', "cipherMethod",
22.                    key='SuperEncryption'),
23.              sg.Radio('Affine', "cipherMethod", key='Affine'),
24.              sg.Radio('Hill', "cipherMethod", key='Hill'),
25.              sg.Radio('Enigma', "cipherMethod", key='Enigma')],
26.            [sg.Text('Print method')],
27.            [sg.Radio('NoSpace', "printMethod", default=True, size=(10,1),
28.                    key='NoSpace'),
29.              sg.Radio('FiveChar', "printMethod", key='FiveChar')],
30.            [sg.Text('Encryption')],
31.            [sg.Text('Enter Plaintext:'), sg.Input(key='-PLAINTEXT_ENCRYPT-')],
32.            [sg.Text('Enter key:'), sg.Input(key='-KEY_ENCRYPT-')],
33.            [sg.Text('Enter m key (for Affine):'),
34.              sg.Input(key='-KEY_ENCRYPT_M-')],
35.            [sg.Text('Enter b key (for Affine):'),
36.              sg.Input(key='-KEY_ENCRYPT_B-')],
37.            [sg.Text('Enter Input Filename (Path):'),
38.              sg.Input(key='-PATH_SOURCE_ENCRYPT-')],
39.            [sg.Text('Enter Output Filename (Path):'),
40.              sg.Input(key='-PATH_ENCRYPT-')],
41.            [sg.Text('Ciphertext:'), sg.Text(size=(30,1),
42.                    key='-CIPHERTEXT_ENCRYPT-')],
43.            [sg.Button('Encrypt'), sg.Button('Encrypt Text File'),
44.              sg.Button('Encrypt File, Output File'), sg.Button('Encrypt from Text File, Output
45.                    into Text File'), sg.Button('Encrypt Output into Text File'), sg.Button('Exit')],
46.            [sg.Text('Decryption')],
47.            [sg.Text('Enter Ciphertext:'), sg.Input(key='-CIPHERTEXT_DECRYPT-')],
48.            [sg.Text('Enter key:'), sg.Input(key='-KEY_DECRYPT-')],
49.            [sg.Text('Enter m key (for Affine):'),
50.              sg.Input(key='-KEY_DECRYPT_M-')],
51.            [sg.Text('Enter b key (for Affine):'),
52.              sg.Input(key='-KEY_DECRYPT_B-')],
53.            [sg.Text('Enter Input Filename (Path):'),
54.              sg.Input(key='-PATH_SOURCE_DECRYPT-')],
55.            [sg.Text('Enter Output Filename (Path):'),
56.              sg.Input(key='-PATH_DECRYPT-')],
57.            [sg.Text('Plaintext:'), sg.Text(size=(30,1),
58.                    key='-PLAINTEXT_DECRYPT-')],
59.            [sg.Button('Decrypt'), sg.Button('Decrypt Text File'),
60.              sg.Button('Decrypt File, Output File'), sg.Button('Decrypt from Text File, Output
```

```

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

```

```

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

```

```

126.         else:
127.             window['-CIPHERTEXT_ENCRYPT-'].update(vg.encryption('',
values['-KEY_ENCRYPT-'], True, values['-PATH_SOURCE_ENCRYPT-']))
128.         if values['FullVigenere']:
129.             if values['FiveChar']:
130.                 window['-CIPHERTEXT_ENCRYPT-'].update(fvg.wrapFiveCharacters(fvg.encryption('',
values['-KEY_ENCRYPT-'], True, values['-PATH_SOURCE_ENCRYPT-'])))
131.         else:
132.             window['-CIPHERTEXT_ENCRYPT-'].update(fvg.encryption('',
values['-KEY_ENCRYPT-'], True, values['-PATH_SOURCE_ENCRYPT-']))
133.         if values['RunningKeyVigenere']:
134.             if values['FiveChar']:
135.                 window['-CIPHERTEXT_ENCRYPT-'].update(avg.wrapFiveCharacters(avg.encryption('',
values['-KEY_ENCRYPT-'], True, values['-PATH_SOURCE_ENCRYPT-'])))
136.         else:
137.             window['-CIPHERTEXT_ENCRYPT-'].update(avg.encryption('',
values['-KEY_ENCRYPT-'], True, values['-PATH_SOURCE_ENCRYPT-']))
138.         if values['ExtendedVigenere']:
139.             message = e.readTextFromFile(values['-PATH_SOURCE_ENCRYPT-'])
140.             key = values['-KEY_ENCRYPT-']
141.             if values['FiveChar']:
142.                 window['-CIPHERTEXT_ENCRYPT-'].update(vge.wrapFiveCharacters(vge.encryptTextExtend
edVigenere(message, key)))
143.         else:
144.             window['-CIPHERTEXT_ENCRYPT-'].update(vge.encryptTextExtendedVigenere(message,
key))
145.         if values['Playfair']:
146.             message = e.readTextFromFile(values['-PATH_SOURCE_ENCRYPT-'])
147.             key = values['-KEY_ENCRYPT-']
148.             if values['FiveChar']:
149.                 window['-CIPHERTEXT_ENCRYPT-'].update(p.wrapFiveCharacters(p.toUpperCase(p.playfai
rEncrypt(message, key)))
150.         else:
151.             window['-CIPHERTEXT_ENCRYPT-'].update(p.toUpperCase(p.playfairEncrypt(message,
key)))
152.         if values['SuperEncryption']:
153.             if values['FiveChar']:
154.                 window['-CIPHERTEXT_ENCRYPT-'].update(se.wrapFiveCharacters(se.myszkowskiTransposi
tionEncrypt(vg.encryption('', values['-KEY_ENCRYPT-'], True,
values['-PATH_SOURCE_ENCRYPT-']), values['-KEY_ENCRYPT-'])))
155.         else:
156.             window['-CIPHERTEXT_ENCRYPT-'].update(se.myszkowskiTranspositionEncrypt(vg.encrypt
ion('', values['-KEY_ENCRYPT-'], True, values['-PATH_SOURCE_ENCRYPT-']),
values['-KEY_ENCRYPT-']))
157.         if values['Affine']:
158.             if values['FiveChar']:
159.                 window['-CIPHERTEXT_ENCRYPT-'].update(af.wrapFiveCharacters(af.encryption('',
values['-KEY_ENCRYPT_M-'], values['-KEY_ENCRYPT_B-'], True,
values['-PATH_SOURCE_ENCRYPT-'])))
160.         else:
161.             window['-CIPHERTEXT_ENCRYPT-'].update(af.encryption('',
values['-KEY_ENCRYPT_M-'], values['-KEY_ENCRYPT_B-'], True,
values['-PATH_SOURCE_ENCRYPT-']))
162.         if values['Hill']:
163.             message = e.readTextFromFile(values['-PATH_SOURCE_ENCRYPT-'])
164.             key = values['-KEY_ENCRYPT-']

```

```

165.         resultMessage = h.generateHillResultMessage(message, key, encrypt
= True)
166.         if values['FiveChar']:
167.             window['-CIPHERTEXT_ENCRYPT-'].update(h.wrapFiveCharacters(resultMessage))
168.         else:
169.             window['-CIPHERTEXT_ENCRYPT-'].update(resultMessage)
170.         if values['Enigma']:
171.             text = e.readTextFromFile(values['-PATH_SOURCE_ENCRYPT-'])
172.             alphabetList, steckerbrettDictionary, alphaRotor, betaRotor,
gammaRotor = enigmaEncryptDecryptInit()
173.             if values['FiveChar']:
174.                 window['-CIPHERTEXT_ENCRYPT-'].update(e.wrapFiveCharacters(e.encryptDecrypt(text,
steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor, alphabetList)))
175.         else:
176.             window['-CIPHERTEXT_ENCRYPT-'].update(e.encryptDecrypt(text,
steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor, alphabetList))
177.
178.         elif event=='Encrypt File, Output File':
179.             if values['ExtendedVigenere']:
180.                 listOfBytes = []
181.                 with open(values['-PATH_SOURCE_ENCRYPT-'], "rb") as f:
182.                     while (byte := f.read(1)):
183.                         listOfBytes.append(byte)
184.                 f.close()
185.                 keyword = values['-KEY_ENCRYPT-']
186.                 key = vge.generateKey(listOfBytes, keyword)
187.                 cipherText = vge.encryptByteExtendedVigenere(listOfBytes, key)
188.                 g = open(values['-PATH_ENCRYPT-'], "wb+")
189.                 for index in range(len(cipherText)):
190.                     g.write(cipherText[index])
191.                 g.close()
192.
193.             elif event=='Encrypt from Text File, Output into Text File':
194.                 if values['Vigenere']:
195.                     if values['FiveChar']:
196.                         window['-CIPHERTEXT_ENCRYPT-'].update(vg.write_to_file(values['PATH_ENCRYPT'],
vg.wrapFiveCharacters(vg.encryption('', values['-KEY_ENCRYPT-'], True,
values['-PATH_SOURCE_ENCRYPT-']))))
197.                     else:
198.                         window['-CIPHERTEXT_ENCRYPT-'].update(vg.write_to_file(values['PATH_ENCRYPT'],
vg.encryption('', values['-KEY_ENCRYPT-'], True,
values['-PATH_SOURCE_ENCRYPT-']))))
199.                 if values['FullVigenere']:
200.                     if values['FiveChar']:
201.                         window['-CIPHERTEXT_ENCRYPT-'].update(fvg.write_to_file(values['PATH_ENCRYPT'],
fvg.wrapFiveCharacters(fvg.encryption('', values['-KEY_ENCRYPT-'], True,
values['-PATH_SOURCE_ENCRYPT-']))))
202.                     else:
203.                         window['-CIPHERTEXT_ENCRYPT-'].update(fvg.write_to_file(values['PATH_ENCRYPT'],
fvg.encryption('', values['-KEY_ENCRYPT-'], True,
values['-PATH_SOURCE_ENCRYPT-']))))
204.                 if values['RunningKeyVigenere']:
205.                     if values['FiveChar']:
206.                         window['-CIPHERTEXT_ENCRYPT-'].update(avg.write_to_file(values['PATH_ENCRYPT'],
avg.wrapFiveCharacters(avg.encryption('', values['-KEY_ENCRYPT-'], True,
values['-PATH_SOURCE_ENCRYPT-']))))
207.                     else:
208.

```

```

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

```

```

244.             if values['FiveChar']:
245.                 window['-CIPHERTEXT_ENCRYPT-'].update(e.write_to_file(values['-PATH_ENCRYPT-'],
                e.wrapFiveCharacters(e.encryptDecrypt(text, steckerbrettDictionary, alphaRotor,
                betaRotor, gammaRotor, alphabetList))))
246.             else:
247.                 window['-CIPHERTEXT_ENCRYPT-'].update(e.write_to_file(values['-PATH_ENCRYPT-'],
                e.encryptDecrypt(text, steckerbrettDictionary, alphaRotor, betaRotor, gammaRotor,
                alphabetList)))
248.
249.         elif event=='Encrypt Output into Text File':
250.             if values['Vigenere']:
251.                 if values['FiveChar']:
252.                     window['-CIPHERTEXT_ENCRYPT-'].update(vg.write_to_file(values['-PATH_ENCRYPT-'],
                    vg.wrapFiveCharacters(vg.encryption(values['-PLAINTEXT_ENCRYPT-'],
                    values['-KEY_ENCRYPT-'], False, '', False))))
253.             else:
254.                 window['-CIPHERTEXT_ENCRYPT-'].update(vg.write_to_file(values['-PATH_ENCRYPT-'],
                    vg.encryption(values['-PLAINTEXT_ENCRYPT-'], values['-KEY_ENCRYPT-'], False, '',
                    False)))
255.             if values['FullVigenere']:
256.                 if values['FiveChar']:
257.                     window['-CIPHERTEXT_ENCRYPT-'].update(fvg.write_to_file(values['-PATH_ENCRYPT-'],
                    fvg.wrapFiveCharacters(fvg.encryption(values['-PLAINTEXT_ENCRYPT-'],
                    values['-KEY_ENCRYPT-'], False, '', False))))
258.             else:
259.                 window['-CIPHERTEXT_ENCRYPT-'].update(fvg.write_to_file(values['-PATH_ENCRYPT-'],
                    fvg.encryption(values['-PLAINTEXT_ENCRYPT-'], values['-KEY_ENCRYPT-'], False, '',
                    False)))
260.             if values['RunningKeyVigenere']:
261.                 if values['FiveChar']:
262.                     window['-CIPHERTEXT_ENCRYPT-'].update(avg.write_to_file(values['-PATH_ENCRYPT-'],
                    avg.wrapFiveCharacters(avg.encryption(values['-PLAINTEXT_ENCRYPT-'],
                    values['-KEY_ENCRYPT-'], False, '', False))))
263.             else:
264.                 window['-CIPHERTEXT_ENCRYPT-'].update(avg.write_to_file(values['-PATH_ENCRYPT-'],
                    avg.encryption(values['-PLAINTEXT_ENCRYPT-'], values['-KEY_ENCRYPT-'], False, '',
                    False)))
265.             if values['ExtendedVigenere']:
266.                 message = values['-PLAINTEXT_ENCRYPT-']
267.                 key = values['-KEY_ENCRYPT-']
268.                 if values['FiveChar']:
269.                     window['-CIPHERTEXT_ENCRYPT-'].update(vge.write_to_file(values['-PATH_ENCRYPT-'],
                    vge.wrapFiveCharacters(vge.encryptTextExtendedVigenere(message, key))))
270.             else:
271.                 window['-CIPHERTEXT_ENCRYPT-'].update(vge.write_to_file(values['-PATH_ENCRYPT-'],
                    vge.encryptTextExtendedVigenere(message, key)))
272.             if values['Playfair']:
273.                 message = values['-PLAINTEXT_ENCRYPT-']
274.                 key = values['-KEY_ENCRYPT-']
275.                 if values['FiveChar']:
276.                     window['-CIPHERTEXT_ENCRYPT-'].update(p.write_to_file(values['-PATH_ENCRYPT-'],
                    p.wrapFiveCharacters(p.toUpperCase(p.playfairEncrypt(message, key))))
277.             else:
278.

```



```

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

```

```

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

```

```

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

```

```

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

```

```

438.         else:
439.             window['-PLAINTEXT_DECRYPT-'].update(vg.write_to_file(values['-PATH_DECRYPT-'],
vg.decryption('', values['-KEY_DECRYPT-'], True,
values['-PATH_SOURCE_DECRYPT-'])))
440.             if values['FullVigenere']:
441.                 if values['FiveChar']:
442.                     window['-PLAINTEXT_DECRYPT-'].update(fvg.write_to_file(values['-PATH_DECRYPT-'],
fvg.wrapFiveCharacters(fvg.decryption('', values['-KEY_DECRYPT-'], True,
values['-PATH_SOURCE_DECRYPT-']))))
443.             else:
444.                 window['-PLAINTEXT_DECRYPT-'].update(fvg.write_to_file(values['-PATH_DECRYPT-'],
fvg.decryption('', values['-KEY_DECRYPT-'], True,
values['-PATH_SOURCE_DECRYPT-'])))
445.                 if values['RunningKeyVigenere']:
446.                     if values['FiveChar']:
447.                         window['-PLAINTEXT_DECRYPT-'].update(avg.write_to_file(values['-PATH_DECRYPT-'],
avg.wrapFiveCharacters(avg.decryption('', values['-KEY_DECRYPT-'], True,
values['-PATH_SOURCE_DECRYPT-']))))
448.             else:
449.                 window['-PLAINTEXT_DECRYPT-'].update(avg.write_to_file(values['-PATH_DECRYPT-'],
avg.decryption('', values['-KEY_DECRYPT-'], True,
values['-PATH_SOURCE_DECRYPT-'])))
450.                 if values['ExtendedVigenere']:
451.                     message = e.readTextFromFile(values['-PATH_SOURCE_DECRYPT-'])
452.                     key = values['-KEY_DECRYPT-']
453.                     if values['FiveChar']:
454.                         window['-PLAINTEXT_DECRYPT-'].update(vge.write_to_file(values['-PATH_DECRYPT-'],
vge.wrapFiveCharacters(vge.decryptTextExtendedVigenere(message, key))))
455.             else:
456.                 window['-PLAINTEXT_DECRYPT-'].update(vge.write_to_file(values['-PATH_DECRYPT-'],
vge.decryptTextExtendedVigenere(message, key)))
457.                 if values['Playfair']:
458.                     message = e.readTextFromFile(values['-PATH_SOURCE_DECRYPT-'])
459.                     key = values['-KEY_DECRYPT-']
460.                     if values['FiveChar']:
461.                         window['-PLAINTEXT_DECRYPT-'].update(p.write_to_file(values['-PATH_DECRYPT-'],
p.wrapFiveCharacters(p.toUpperCase(p.playfairDecrypt(message, key))))
462.             else:
463.                 window['-PLAINTEXT_DECRYPT-'].update(p.write_to_file(values['-PATH_DECRYPT-'],
p.toUpperCase(p.playfairDecrypt(message, key)))
464.                 if values['SuperEncryption']:
465.                     if values['FiveChar']:
466.                         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-']))))
467.             else:
468.                 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-'])))
469.                 if values['Affine']:
470.                     if values['FiveChar']:
471.                         window['-PLAINTEXT_DECRYPT-'].update(af.write_to_file(values['-PATH_DECRYPT-'],

```

```

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

```

```

        avg.decryption(values['-CIPHERTEXT_DECRYPT-'], values['-KEY_DECRYPT-']))
506.         if values['ExtendedVigenere']:
507.             message = values['-CIPHERTEXT_DECRYPT-']
508.             key = values['-KEY_DECRYPT-']
509.             if values['FiveChar']:
510.
511.                 window['-PLAINTEXT_DECRYPT-'].update(vge.write_to_file(values['-PATH_DECRYPT-'],
512. vge.wrapFiveCharacters(vge.decryptTextExtendedVigenere(message, key))))
513.             else:
514.
515.                 window['-PLAINTEXT_DECRYPT-'].update(vge.write_to_file(values['-PATH_DECRYPT-'],
516. vge.decryptTextExtendedVigenere(message, key)))
517.             if values['Playfair']:
518.                 message = values['-CIPHERTEXT_DECRYPT-']
519.                 key = values['-KEY_DECRYPT-']
520.                 if values['FiveChar']:
521.
522.                     window['-PLAINTEXT_DECRYPT-'].update(p.write_to_file(values['-PATH_DECRYPT-'],
523. p.wrapFiveCharacters(p.toUpperCase(p.playfairDecrypt(message, key))))
524.                 else:
525.
526.                     window['-PLAINTEXT_DECRYPT-'].update(p.write_to_file(values['-PATH_DECRYPT-'],
527. p.toUpperCase(p.playfairDecrypt(message, key))))
528.             if values['SuperEncryption']:
529.                 if values['FiveChar']:
530.
531.                     window['-PLAINTEXT_DECRYPT-'].update(vg.write_to_file(values['-PATH_DECRYPT-'],
532. vg.wrapFiveCharacters(vg.decryption(se.myszkowskiTranspositionDecrypt(values['-CIP
533. HERTEXT_DECRYPT-'], values['-KEY_DECRYPT-']), values['-KEY_DECRYPT-'])))
534.                 else:
535.
536.                     window['-PLAINTEXT_DECRYPT-'].update(vg.write_to_file(values['-PATH_DECRYPT-'],
537. vg.decryption(se.myszkowskiTranspositionDecrypt(values['-CIPHERTEXT_DECRYPT-'],
538. values['-KEY_DECRYPT-']), values['-KEY_DECRYPT-'])))
539.             if values['Affine']:
540.                 if values['FiveChar']:
541.
542.                     window['-PLAINTEXT_DECRYPT-'].update(af.write_to_file(values['-PATH_DECRYPT-'],
543. af.wrapFiveCharacters(af.decryption(values['-CIPHERTEXT_DECRYPT-'],
544. values['-KEY_DECRYPT_M-'], values['-KEY_DECRYPT_B-'], False, ''))))
545.                 else:
546.
547.                     window['-PLAINTEXT_DECRYPT-'].update(af.write_to_file(values['-PATH_DECRYPT-'],
548. af.decryption(values['-CIPHERTEXT_DECRYPT-'], values['-KEY_DECRYPT_M-'],
549. values['-KEY_DECRYPT_B-'], False, '')))
550.             if values['Hill']:
551.                 message = values['-CIPHERTEXT_DECRYPT-']
552.                 key = values['-KEY_DECRYPT-']
553.                 resultMessage = h.generateHillResultMessage(message, key, encrypt
554. = False)
555.                 if values['FiveChar']:
556.
557.                     window['-PLAINTEXT_DECRYPT-'].update(h.write_to_file(values['-PATH_DECRYPT-'],
558. h.wrapFiveCharacters(resultMessage)))
559.                 else:
560.
561.                     window['-PLAINTEXT_DECRYPT-'].update(h.write_to_file(values['-PATH_DECRYPT-'],
562. resultMessage))
563.             if values['Enigma']:
564.                 text = values['-CIPHERTEXT_DECRYPT-']
565.                 alphabetList, steckerbrettDictionary, alphaRotor, betaRotor,
566. gammaRotor = enigmaEncryptDecryptInit()
567.                 if values['FiveChar']:
568.
569.                     window['-PLAINTEXT_DECRYPT-'].update(e.write_to_file(values['-PATH_DECRYPT-'],

```



```

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

```

Vigenere.py (vigenere standar)

```

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

```



```

51. no_punct = ""
52. for char in key:
53.     if char not in punctuations:
54.         no_punct = no_punct + char
55. key=no_punct
56.
57. #Compare plaintext and key's Length#
58. if (len(key)<len(plaintext)):
59.     real_key=key
60.     times = len(plaintext)//len(key)
61.
62.     for i in range(times-1):
63.         key+=real_key
64.
65.     sisa = len(plaintext)-len(key)
66.
67.     for i in range(sisa):
68.         key+=real_key[i]
69.
70. #If key>plaintext#
71. elif (len(key)>len(plaintext)):
72.     key=key[:len(plaintext)]
73.
74. print(key)
75.
76. #Alphabet List for conversion from number to character#
77. alphabet = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',
'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', \
78.             'w', 'x', 'y', 'z']
79. ciphertext=''
80. for i in range(len(plaintext)):
81.     # if (i%5==0) and (i!=0):
82.     # ciphertext+='-'
83.     ciphernumber=(alphabet.index(plaintext[i])+alphabet.index(key[i]))%26
84.     #print(ciphernumber, end=', ')
85.     ciphertext+=alphabet[ciphernumber]
86.     #print(ciphertext)
87. return ciphertext
88.
89.
90. def decryption(ciphertext, key, from_file = False, path = ''):
91.     #Input ciphertext
92.     if (from_file):
93.         file1 = open(path,"r")
94.         ciphertext_list = file1.readlines()
95.         seperator=' '
96.         file1.close()
97.         ciphertext=seperator.join(ciphertext_list)
98.
99.     #ciphertext = input("Masukkan ciphertext:")
100.
101.     print(ciphertext[0])
102.
103.     #Input key from user#
104.     #key = input("Masukkan key:")
105.
106.     #print(key)
107.
108.     #Convert into lowercase#
109.     ciphertext=ciphertext.lower()
110.     key=key.lower()
111.
112.     #Remove comma
113.     ciphertext = ciphertext.replace(',',' ')
114.     key = key.replace(',',' ')

```

```

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

```

```

1. #autokey_vigenere
2.
3. from textwrap import wrap
4.
5. def write_to_file(path, ciphertext):
6.     file1 = open(path, "w")
7.     file1.write(ciphertext)
8.     file1.close()
9.
10. def wrapFiveCharacters(message):
11.     messageWrapFive = wrap(message, 5)
12.     return ' '.join(messageWrapFive)
13.
14. def encryption(plaintext, key, from_file, path, write_to_file):
15.     #Input plaintext
16.
17.     #plaintext = input("Masukkan plaintext:")
18.
19.     #Input key from user#
20.     #key = input("Masukkan key:")
21.     if (from_file):
22.         file1 = open(path, "r")
23.         plaintext_list = file1.readlines()
24.         seperator=' '
25.         file1.close()
26.         plaintext=seperator.join(plaintext_list)
27.
28.     #Convert into lowercase#
29.     plaintext=plaintext.lower()
30.     key=key.lower()
31.
32.     #Remove comma
33.     plaintext = plaintext.replace(',', ' ')
34.     key = key.replace(',', ' ')
35.     #print(plaintext)
36.
37.     #Remove space
38.     plaintext = plaintext.replace(' ', '')
39.     key = key.replace(' ', '')
40.
41.     #Remove punctuation
42.     # Define punctuation
43.     punctuations = '!"()-[]{};:'"\<>./?@$%^&*~_''''
44.
45.     # Remove punctuation from plaintext
46.     no_punct = ""
47.     for char in plaintext:
48.         if char not in punctuations:
49.             no_punct = no_punct + char
50.     plaintext=no_punct
51.
52.     # Remove punctuation from key
53.     no_punct = ""
54.     for char in key:
55.         if char not in punctuations:
56.             no_punct = no_punct + char
57.     key=no_punct
58.
59.     #Compare plaintext and key's Length#
60.     if (len(key)<len(plaintext)):
61.         """"
62.         real_key=key
63.         times = len(plaintext)//len(key)

```

```

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

```

```

128.     # Remove punctuation from ciphertext
129.     no_punct = ""
130.     for char in ciphertext:
131.         if char not in punctuations:
132.             no_punct = no_punct + char
133.     ciphertext=no_punct
134.
135.     # Remove punctuation from key
136.     no_punct = ""
137.     for char in key:
138.         if char not in punctuations:
139.             no_punct = no_punct + char
140.     key=no_punct
141.
142.     #Make the real key
143.     #Alphabet list for conversion from number to character#
144.     alphabet = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l',
'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',\
145.                 'w', 'x', 'y', 'z']
146.
147.
148.     plaintext=''
149.     for i in range(len(ciphertext)):
150.         #alphabet_cycle=itertools.cycle(alphabet)
151.         #print(key[i])
152.         plainnumber=(alphabet.index(ciphertext[i])-alphabet.index(key[i]))%26
153.
154.
155.         plaintext+=alphabet[plainnumber]
156.         key+=alphabet[plainnumber]
157.         #print(ciphertext)
158.     print(key)
159.
160.
161.     #Alphabet list for conversion from number to character#
162.     alphabet = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l',
'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',\
163.                 'w', 'x', 'y', 'z']
164.
165.
166.     plaintext=''
167.     for i in range(len(ciphertext)):
168.         #alphabet_cycle=itertools.cycle(alphabet)
169.         #print(key[i])
170.         plainnumber=(alphabet.index(ciphertext[i])-alphabet.index(key[i]))%26
171.
172.
173.         plaintext+=alphabet[plainnumber]
174.         #print(ciphertext)
175.     return plaintext

```

full_vigenere.py

```

1. #full vigenere cipher
2.
3. import random
4. import pickle
5. from textwrap import wrap
6.
7. def write_to_file(path, ciphertext):
8.     file1 = open(path,"w")

```

```

9.     file1.write(ciphertext)
10.    file1.close()
11.
12.    def wrapFiveCharacters(message):
13.        messageWrapFive = wrap(message,5)
14.        return ' '.join(messageWrapFive)
15.
16.    def encryption(plaintext, key, from_file, path, write_to_file):
17.        if (from_file):
18.            file1 = open(path,"r")
19.            plaintext_list = file1.readlines()
20.            seperator=' '
21.            file1.close()
22.            plaintext=seperator.join(plaintext_list)
23.
24.            #Convert into lowercase#
25.            plaintext= toUpperCase(plaintext)
26.            key=toUpperCase(key)
27.
28.            #Remove comma
29.            plaintext = plaintext.replace(',',' ')
30.            key = key.replace(',',' ')
31.            #print(plaintext)
32.
33.            #Remove space
34.            plaintext = plaintext.replace(' ','')
35.            key = key.replace(' ','')
36.
37.            #Remove punctuation
38.            # Define punctuation
39.            punctuations = '!()-[]{};:\"\, <> . / ? @ # $ % ^ & * _ ~ '''
40.
41.            # Remove punctuation from plaintext
42.            no_punct = ""
43.            for char in plaintext:
44.                if char not in punctuations:
45.                    no_punct = no_punct + char
46.            plaintext=no_punct
47.
48.            # Remove punctuation from key
49.            no_punct = ""
50.            for char in key:
51.                if char not in punctuations:
52.                    no_punct = no_punct + char
53.            key=no_punct
54.
55.            #Compare plaintext and key's Length#
56.            if (len(key)<len(plaintext)):
57.                real_key=key
58.                times = len(plaintext)//len(key)
59.
60.                for i in range(times-1):
61.                    key+=real_key
62.
63.                sisa = len(plaintext)-len(key)
64.
65.                for i in range(sisa):
66.                    key+=real_key[i]
67.
68.            #If key>plaintext#
69.            elif (len(key)>len(plaintext)):
70.                key=key[:len(plaintext)]
71.
72.            print(key)
73.

```

```

74.     #Alphabet list for conversion from number to character#
75.     alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
76.     alphabet = list(alphabet)
77.
78.     # define an empty list
79.     alphabet_shuffled = []
80.
81.     # open file and read the content in a list
82.     with open('full_vigenere_matrix.txt', 'r') as filehandle:
83.         for line in filehandle:
84.             # remove linebreak which is the last character of the string
85.             currentAlphabet = line[:-1]
86.
87.             # add item to the list
88.             alphabet_shuffled.append(currentAlphabet)
89.
90.     print(alphabet_shuffled[0][0])
91.     ciphertext=''
92.     for i in range(len(plaintext)):
93.         ciphernumber_x=alphabet.index(plaintext[i])
94.         ciphernumber_y=alphabet.index(key[i])
95.         print(ciphernumber_x)
96.         print(ciphernumber_y)
97.         ciphertext+=alphabet_shuffled[ciphernumber_x][ciphernumber_y]
98.         print(ciphertext)
99.
100.    return ciphertext
101.
102.    def write_to_file(path, ciphertext):
103.        file1 = open(path,"w")
104.        file1.write(ciphertext)
105.        file1.close()
106.
107.    def toUpperCase(text):
108.        return "".join(filter(str.isupper, text.upper()))
109.
110.    def decryption(ciphertext, key, from_file = False, path = ''):
111.        #Input ciphertext
112.        if (from_file):
113.            file1 = open(path,"r")
114.            ciphertext_list = file1.readlines()
115.            seperator=''
116.            file1.close()
117.            ciphertext=seperator.join(ciphertext_list)
118.
119.            #ciphertext = input("Masukkan ciphertext:")
120.            print("Ciphertext:", end="")
121.            print(ciphertext)
122.
123.            #Input key from user#
124.            #key = input("Masukkan key:")
125.
126.            #print(key)
127.
128.            #Convert into lowercase#
129.            ciphertext=toUpperCase(ciphertext)
130.            key=toUpperCase(key)
131.
132.            #Remove comma
133.            ciphertext = ciphertext.replace(',', '')
134.            key = key.replace(',', '')
135.
136.            #Remove space
137.            ciphertext = ciphertext.replace(' ', '')
138.            key = key.replace(' ', '')

```

```

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

```



```

204.     plaintext=''
205.
206.     for i in range(len(ciphertext)):
207.         a = alphabet.index(key[i])
208.         for j in range(0,26):
209.             if (alphabet_shuffled[j][a]==ciphertext[i]):
210.                 break
211.         plainnumber=j
212.         plaintext+=alphabet[plainnumber]
213.
214.
215.
216.     return plaintext
217.

```

vigenereExtended.py

```

1.  #vigenere Extended
2.  from textwrap import wrap
3.
4.  def generateKey(string, key):
5.      key = list(key)
6.      if len(string) == len(key):
7.          return(key)
8.      else:
9.          for i in range(len(string) - len(key)):
10.             key.append(key[i % len(key)])
11.     return("".join(key))
12.
13. def encryptByteExtendedVigenere(bytePlainText, key):
14.     result = [[0] for i in range(len(bytePlainText))]
15.     key = key.strip().upper()
16.     keyIndex = 0
17.     keylength = len(key)
18.     for i in range(len(bytePlainText)):
19.         keyIndex = keyIndex % keylength
20.         shift = ord(key[keyIndex]) - 65
21.         result[i] = bytes([(ord(bytePlainText[i]) + shift) % 256])
22.         keyIndex+=1
23.     return result
24.
25. def decryptByteExtendedVigenere(byteCipherText, key):
26.     result = [[0] for i in range(len(byteCipherText))]
27.     key = key.strip().upper()
28.     keyIndex = 0
29.     keylength = len(key)
30.     for i in range(len(byteCipherText)):
31.         keyIndex = keyIndex % keylength
32.         shift = ord(key[keyIndex]) - 65
33.         result[i] = bytes([(ord(byteCipherText[i]) + 256 - shift) % 256])
34.         keyIndex+=1
35.     return result
36.
37. def encryptTextExtendedVigenere(plaintext, key):
38.     result = [[0] for i in range(len(plaintext))]
39.     key = key.strip().upper()
40.     keyIndex = 0
41.     keylength = len(key)
42.     for i in range(len(plaintext)):
43.         keyIndex = keyIndex % keylength
44.         shift = ord(key[keyIndex]) - 65
45.         result[i] = chr((ord(plaintext[i]) + shift) % 256)

```

```

46.         keyIndex+=1
47.     return ''.join(i for i in result)
48.
49. def decryptTextExtendedVigenere(cipherText, key):
50.     result = [[0] for i in range(len(cipherText))]
51.     key = key.strip().upper()
52.     keyIndex = 0
53.     keylength = len(key)
54.     for i in range(len(cipherText)):
55.         keyIndex = keyIndex % keylength
56.         shift = ord(key[keyIndex]) - 65
57.         result[i] = chr((ord(cipherText[i]) + 256 - shift) % 256)
58.         keyIndex+=1
59.     return ''.join(i for i in result)
60.
61. def wrapFiveCharacters(message):
62.     messageWrapFive = wrap(message,5)
63.     return ' '.join(messageWrapFive)
64.
65. def write_to_file(path, text):
66.     file1 = open(path, "w+")
67.     file1.write(text)
68.     file1.close()
69.
70. def readTextFromFile(path):
71.     file1 = open(path, "r")
72.     data = file1.read()
73.     file1.close()
74.     return data

```

Playfair.py

```

1.  #playfair
2.  from textwrap import wrap
3.
4.  def generateUniqueAlphabetList(seq):
5.      seen = {}
6.      return [seen.setdefault(x, x) for x in seq if x not in seen]
7.
8.  def partition(seq, n):
9.      return [seq[i : i + n] for i in range(0, len(seq), n)]
10.
11. def toUpperCase(text):
12.     return "".join(filter(str.isupper, text.upper()))
13.
14. def splitMessageToDigraphs(message):
15.     messageList=[]
16.     for e in message:
17.         messageList.append(e)
18.     for i in range(len(messageList)):
19.         if " " in messageList:
20.             messageList.remove(" ")
21.     i=0
22.     for e in range(len(messageList)//2):
23.         if messageList[i]==messageList[i+1]:
24.             messageList.insert(i+1,'X')
25.         i=i+2
26.     if len(messageList)%2==1:
27.         messageList.append("X")
28.     i=0
29.     digraphList=[]
30.     for x in range(1,len(messageList)//2+1):

```

```

31.         digraphList.append(messageList[i:i+2])
32.         i=i+2
33.     return digraphList
34.
35. def formatMessage(message, replaceFrom, replaceTo):
36.     message = toUpperCase(message)
37.     formattedMessage = message.replace(replaceFrom, replaceTo)
38.     return formattedMessage
39.
40. def generateEncryptionDictionary(matrix):
41.     encryptionDictionary = {}
42.     for row in matrix:
43.         for i in range(5):
44.             for j in range(5):
45.                 if i != j:
46.                     encryptionDictionary[row[i] + row[j]] = row[(i + 1) % 5] +
row[(j + 1) % 5]
47.             for c in zip(*matrix):
48.                 for i in range(5):
49.                     for j in range(5):
50.                         if i != j:
51.                             encryptionDictionary[c[i] + c[j]] = c[(i + 1) % 5] + c[(j + 1)
% 5]
52.             for i in range(5):
53.                 for j in range(5):
54.                     for k in range(5):
55.                         for l in range(5):
56.                             if i != k and j != l:
57.                                 encryptionDictionary[matrix[i][j] + matrix[k][l]] =
matrix[i][l] + matrix[k][j]
58.     return encryptionDictionary
59.
60. def playfairEncrypt(message, key, replaceFrom = 'J', replaceTo = None):
61.     if replaceTo is None:
62.         replaceTo = 'I' if replaceFrom == 'J' else ''
63.     alphabet = "ABCDEFGHGIJKLMNOPQRSTUVWXYZ"
64.     digraphList = splitMessageToDigraphs(formatMessage(message, replaceFrom,
replaceTo))
65.     uniqueFormattedMatrix = partition(generateUniqueAlphabetList(formatMessage(key
+ alphabet, replaceFrom, replaceTo)), 5)
66.     encryptionDictionary = generateEncryptionDictionary(uniqueFormattedMatrix)
67.     cipheredMessage = " ".join(encryptionDictionary[a + b] for a, b in
digraphList)
68.     return cipheredMessage
69.     #return " ".join(enc[a + (b if b else 'X')] for a, b in lst)
70.
71. def playfairDecrypt(message, key, replaceFrom = 'J', replaceTo = 'I'):
72.     if replaceTo is None:
73.         replaceTo = 'I' if replaceFrom == 'J' else ''
74.     alphabet = "ABCDEFGHGIJKLMNOPQRSTUVWXYZ"
75.     lst = splitMessageToDigraphs(formatMessage(message, replaceFrom, replaceTo))
76.     uniqueFormattedMatrix = partition(generateUniqueAlphabetList(formatMessage(key
+ alphabet, replaceFrom, replaceTo)), 5)
77.     encryptionDictionary = generateEncryptionDictionary(uniqueFormattedMatrix)
78.     decryptionDictionary = dict((value, key) for key, value in
encryptionDictionary.items())
79.     decipheredMessage = " ".join(decryptionDictionary[p] for p in
partition(formatMessage(message, replaceFrom, replaceTo), 2))
80.     return decipheredMessage
81.
82. def wrapFiveCharacters(message):
83.     messageWrapFive = wrap(message, 5)
84.     return ' '.join(messageWrapFive)
85.
86. def write_to_file(path, text):

```

```

87.     file1 = open(path, "w+")
88.     file1.write(text)
89.     file1.close()
90.
91. def readTextFromFile(path):
92.     file1 = open(path, "r")
93.     data = file1.read()
94.     file1.close()
95.     return data

```

myszkowskiTransposition.py

```

1.  #SuperEncryption using vigenere.py + Myszkowski transposition cipher (this file)
2.
3.  #myszkowski
4.  import math
5.  from textwrap import wrap
6.
7.  def write_to_file(path, ciphertext):
8.      file1 = open(path, "w")
9.      file1.write(ciphertext)
10.     file1.close()
11.
12. def readTextFromFile(path):
13.     file1 = open(path, "r")
14.     data = file1.read()
15.     file1.close()
16.     return data
17.
18. def wrapFiveCharacters(message):
19.     messageWrapFive = wrap(message, 5)
20.     return ' '.join(messageWrapFive)
21.
22. def toUpperCase(text):
23.     return "".join(filter(str.isupper, text.upper()))
24.
25. def generateKeyList(key):
26.     alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
27.     keyList = []
28.     duplicateFound = True
29.     for i in key:
30.         duplicateFound = True
31.         while duplicateFound:
32.             if i in keyList:
33.                 i = alphabet[(alphabet.index(i) + 1) % 26]
34.             else:
35.                 duplicateFound = False
36.             keyList.append(i)
37.     return keyList
38.
39. def myszkowskiTranspositionEncrypt(message, key):
40.     message = toUpperCase(message)
41.     key = toUpperCase(key)
42.     encryptedMessage = ""
43.     keyIndex = 0
44.     messageLength = float(len(message))
45.     messageList = list(message)
46.     keyList = generateKeyList(key)
47.     key = ' '.join(keyList)
48.     sortedKeyList = sorted(keyList)
49.     column = len(key)

```

```

50. row = int(math.ceil(messageLength / column))
51. fill_null = int((row * column) - messageLength)
52. messageList.extend('*' * fill_null)
53. matrix = [messageList[i: i + column]
54.             for i in range(0, len(messageList), column)]
55. for _ in range(column):
56.     curr_idx = key.index(sortedKeyList[keyIndex])
57.     encryptedMessage += ''.join([row[curr_idx] for row in matrix])
58.     keyIndex += 1
59. return encryptedMessage
60.
61. def myszkowskiTranspositionDecrypt(message, key):
62.     message = toUpperCase(message)
63.     key = toUpperCase(key)
64.     decryptedMessage = ""
65.     keyIndex = 0
66.     messageIndex = 0
67.     messageLength = float(len(message))
68.     messageList = list(message)
69.     column = len(key)
70.     row = int(math.ceil(messageLength / column))
71.     keyList = generateKeyList(key)
72.     key = ''.join(keyList)
73.     sortedKeyList = sorted(keyList)
74.     decryptedMessageList = []
75.     for _ in range(row):
76.         decryptedMessageList += [[None] * column]
77.     for _ in range(column):
78.         curr_idx = key.index(sortedKeyList[keyIndex])
79.         for j in range(row):
80.             decryptedMessageList[j][curr_idx] = messageList[messageIndex]
81.             messageIndex += 1
82.         keyIndex += 1
83.     decryptedMessage = ''.join(sum(decryptedMessageList, []))
84.     additionalTemp = decryptedMessage.count('*')
85.     if additionalTemp > 0:
86.         return decryptedMessage[: -additionalTemp]
87.     return decryptedMessage
88.

```

Affine.py

```

1. #affine
2.
3. from textwrap import wrap
4.
5. def write_to_file(path, ciphertext):
6.     file1 = open(path, "w")
7.     file1.write(ciphertext)
8.     file1.close()
9.
10. def wrapFiveCharacters(message):
11.     messageWrapFive = wrap(message, 5)
12.     return ' '.join(messageWrapFive)
13.
14. def toUpperCase(text):
15.     return "".join(filter(str.isupper, text.upper()))
16.
17. def extendedGCDEuclideanAlgorithm(a, b):
18.     if a == 0 :
19.         return b, 0, 1
20.     greatestCommonDenominator, x1, y1 = extendedGCDEuclideanAlgorithm(b%a, a)

```

```

21.     x = y1 - (b//a) * x1
22.     y = x1
23.     return greatestCommonDenominator, x, y
24.
25. def getModularInverse(a, m):
26.     greatestCommonDenominator, x, y = extendedGCDEuclideanAlgorithm(a, m)
27.     if greatestCommonDenominator != 1:
28.         return None
29.     else:
30.         return x % m
31.
32. def encryption(plaintext, key_m, key_b, from_file, path, write_to_file):
33.     if (from_file):
34.         file1 = open(path, "r")
35.         plaintext_list = file1.readlines()
36.         seperator = ''
37.         file1.close()
38.         plaintext = seperator.join(plaintext_list)
39.
40.     plaintext = toUpperCase(plaintext)
41.
42.     #Remove comma
43.     plaintext = plaintext.replace(',', ' ')
44.     #key = key.replace(',', '')
45.     #print(plaintext)
46.
47.     #Remove space
48.     plaintext = plaintext.replace(' ', '')
49.     #key = key.replace(' ', '')
50.
51.     #Remove punctuation
52.     # Define punctuation
53.     punctuations = '!"()-[]{};:'"\<>./?@#$$%^&*~_''''
54.
55.     # Remove punctuation from plaintext
56.     no_punct = ""
57.     for char in plaintext:
58.         if char not in punctuations:
59.             no_punct = no_punct + char
60.     plaintext = no_punct
61.
62.     #Alphabet list for conversion from number to character#
63.     alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
64.     alphabet = list(alphabet)
65.     ciphertext = ''
66.     for i in range(len(plaintext)):
67.         # if (i%5==0) and (i!=0):
68.         #     ciphertext+='-'
69.         ciphernumber = (int(key_m)*alphabet.index(plaintext[i])+int(key_b))%26
70.         #ciphernumber=(alphabet.index(plaintext[i])+alphabet.index(key[i]))%26
71.         #print(ciphernumber, end=', ')
72.         ciphertext += alphabet[ciphernumber]
73.         #print(ciphertext)
74.     return ciphertext
75.
76.
77. def decryption(ciphertext, key_m, key_b, from_file = False, path = '',
78.     write_to_file = False):
79.     #Input plaintext
80.     if (from_file):
81.         file1 = open(path, "r")
82.         ciphertext_list = file1.readlines()
83.         seperator = ''
84.         file1.close()
85.         ciphertext = seperator.join(ciphertext_list)

```

```

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

```

Hill.py

```

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

```

```

21. def generateCipherMatrix(message, keyMatrix, messageVector, n):
22.     cipherMatrix = [[0] for i in range(n)]
23.     for i in range(n):
24.         for j in range(1):
25.             cipherMatrix[i][j] = 0
26.             for x in range(n):
27.                 cipherMatrix[i][j] += (keyMatrix[i][x] * messageVector[x][j])
28.             cipherMatrix[i][j] = cipherMatrix[i][j] % 26
29.     return cipherMatrix
30.
31. def generateInverseMatrix(matrix):
32.     determinant = round(calculateDeterminant(matrix))
33.     inverseMatrix = [[0 for j in range(len(matrix))] for i in range(len(matrix))]
34.     for i in range(len(matrix)):
35.         inverseMatrix[i][i] = 1
36.         for focusDiagonal in range(len(matrix)):
37.             focusDiagonalScaler = 1.0 / matrix[focusDiagonal][focusDiagonal]
38.             for j in range(len(matrix)):
39.                 matrix[focusDiagonal][j] *= focusDiagonalScaler
40.                 inverseMatrix[focusDiagonal][j] *= focusDiagonalScaler
41.             for i in list(range(len(matrix))[0:focusDiagonal] +
list(range(len(matrix))[focusDiagonal+1:]):
42.                 currentRowScaler = matrix[i][focusDiagonal]
43.                 for j in range(len(matrix)):
44.                     matrix[i][j] = matrix[i][j] - currentRowScaler *
matrix[focusDiagonal][j]
45.                     inverseMatrix[i][j] = inverseMatrix[i][j] - currentRowScaler *
inverseMatrix[focusDiagonal][j]
46.             for x in range(len(inverseMatrix)):
47.                 for y in range(len(inverseMatrix[0])):
48.                     inverseMatrix[x][y] = round(inverseMatrix[x][y]*determinant)
49.             for x in range(len(inverseMatrix)):
50.                 for y in range(len(inverseMatrix[0])):
51.                     inverseMatrix[x][y] = inverseMatrix[x][y] * (determinant % 26)
52.     return inverseMatrix
53.
54. def generateZeroMatrix(rows, cols):
55.     zeroMatrix = []
56.     while len(zeroMatrix) < rows:
57.         zeroMatrix.append([])
58.         while len(zeroMatrix[-1]) < cols:
59.             zeroMatrix[-1].append(0.0)
60.     return zeroMatrix
61.
62. def generateMatrixCopy(matrix):
63.     MatrixCopy = generateZeroMatrix(len(matrix), len(matrix[0]))
64.     for i in range(len(matrix)):
65.         for j in range(len(matrix[0])):
66.             MatrixCopy[i][j] = matrix[i][j]
67.     return MatrixCopy
68.
69. def calculateDeterminant(matrix):
70.     if len(matrix) == 1 and len(matrix[0]) == 1:
71.         return matrix[0][0]
72.     matrixCopy = generateMatrixCopy(matrix)
73.     for focusDiagonal in range(len(matrix)):
74.         if matrixCopy[focusDiagonal][focusDiagonal] == 0:
75.             matrixCopy[focusDiagonal][focusDiagonal] = 1.0e-18
76.         for i in range(focusDiagonal+1, len(matrix)):
77.             currentRowScaler = matrixCopy[i][focusDiagonal] /
matrixCopy[focusDiagonal][focusDiagonal]
78.             for j in range(len(matrix)):
79.                 matrixCopy[i][j] = matrixCopy[i][j] - currentRowScaler *
matrixCopy[focusDiagonal][j]
80.     product = 1.0

```



```

81.     for i in range(len(matrix)):
82.         product *= matrixCopy[i][i]
83.     return product
84.
85. def generateDecipherMatrix(message, keyMatrix, messageVector, n):
86.     decipherMatrix = [[0] for i in range(n)]
87.     for i in range(n):
88.         for j in range(1):
89.             decipherMatrix[i][j] = 0
90.             for x in range(n):
91.                 decipherMatrix[i][j] += (keyMatrix[i][x] * messageVector[x][j])
92.             decipherMatrix[i][j] = decipherMatrix[i][j] % 26
93.     return decipherMatrix
94.
95. def HillCipherEncryption(message, key):
96.     n = 3
97.     keyMatrix = generateKeyMatrix(key, n)
98.     messageVector = generateMessageVector(message, n)
99.     cipherMatrix = generateCipherMatrix(messageVector, keyMatrix, messageVector,
n)
100.     CipherText = []
101.     for i in range(n):
102.         CipherText.append(chr(cipherMatrix[i][0] + 65))
103.     Ciphertext = "".join(CipherText)
104.     return Ciphertext
105.
106. def HillCipherDecryption(message, key):
107.     n = len(message)
108.     keyMatrix = generateKeyMatrix(key, n)
109.     invertedKeyMatrix = generateInverseMatrix(keyMatrix)
110.     decipherMessageVector = generateMessageVector(message, n)
111.     decipherMatrix = generateDecipherMatrix(message, invertedKeyMatrix,
decipherMessageVector, n)
112.     DecipherText = []
113.     for i in range(n):
114.         DecipherText.append(chr(round(decipherMatrix[i][0] + 65)))
115.     DecipheredText = "".join(DecipherText)
116.     return DecipheredText
117.
118. def wrapFiveCharacters(message):
119.     messageWrapFive = wrap(message, 5)
120.     return ' '.join(messageWrapFive)
121.
122. def toUpperCase(text):
123.     return "".join(filter(str.isupper, text.upper()))
124.
125. def write_to_file(path, text):
126.     file1 = open(path, "w+")
127.     file1.write(text)
128.     file1.close()
129.
130. def readTextFromFile(path):
131.     file1 = open(path, "r")
132.     data = file1.read()
133.     file1.close()
134.     return data
135.
136. def generateHillResultMessage(message, key, encrypt):
137.     resultList = []
138.     message = toUpperCase(message)
139.     message = wrap(message, 3)
140.     if (len(key) < 9):
141.         real_key = key
142.         times = 9 // len(key)
143.         for i in range(times - 1):

```

```

144.         key+=real_key
145.         sisa = 9-len(key)
146.         for i in range(sisa):
147.             key+=real_key[i]
148.         #If key>plaintext#
149.         elif (len(key)>9):
150.             key=key[:9]
151.         for messagePart in message:
152.             add = 0
153.             if len(messagePart) < 3:
154.                 add = 3 - len(messagePart)
155.                 for i in range(add):
156.                     messagePart = ''.join([messagePart, 'X'])
157.             if encrypt:
158.                 cipherText = HillCipherEncryption(messagePart, key)
159.                 resultList.append(cipherText)
160.             else:
161.                 decipherText = HillCipherDecryption(messagePart, key)
162.                 resultList.append(decipherText)
163.         return ''.join(resultList)

```

Enigma.py

```

1.  #enigma
2.
3.  from textwrap import wrap
4.
5.  def permute(rotor, alphabetList):
6.      newAlphabet = ''.join(alphabetList)
7.      newAlphabetList = list(newAlphabet)
8.      for i in range(rotor):
9.          newAlphabetList.insert(0, newAlphabetList[-1])
10.         newAlphabetList.pop(-1)
11.     return newAlphabetList
12.
13. def inversePermutation(rotor, alphabetList):
14.     newAlphabet = ''.join(alphabetList)
15.     newAlphabetList = list(newAlphabet)
16.     for i in range(rotor):
17.         newAlphabetList.append(newAlphabetList[0])
18.         newAlphabetList.pop(0)
19.     return newAlphabetList
20.
21. def turnRotors(alphaRotor, betaRotor, gammaRotor, alphabetList):
22.     alphaRotor += 1
23.     if alphaRotor % len(alphabetList) == 0:
24.         betaRotor += 1
25.         alphaRotor = 0
26.     if betaRotor % len(alphabetList) == 0 and alphaRotor % len(alphabetList) != 0
27. and betaRotor >= (len(alphabetList) - 1):
28.         gammaRotor += 1
29.         betaRotor = 1
30.     return alphaRotor, betaRotor, gammaRotor
31.
32. def encryptDecrypt(text, steckerbrettDictionary, alphaRotor, betaRotor,
33. gammaRotor, alphabetList):
34.     for letter in list(steckerbrettDictionary.keys()):
35.         if letter in alphabetList:
36.             alphabetList.remove(letter)
37.             alphabetList.remove(steckerbrettDictionary[letter])
38.             steckerbrettDictionary.update({steckerbrettDictionary[letter]:letter})
39.     print("steckerbrettDictionary = ", steckerbrettDictionary)

```

```

38.     reflector = [letter for letter in reversed(alphabetList)]
39.     print("reflector = ", reflector)
40.     resultText = []
41.     upperCaseText = toUpperCase(text)
42.     upperCaseText.split()
43.     for letter in upperCaseText:
44.         if letter in steckerbrettDictionary:
45.             resultText.append(steckerbrettDictionary[letter])
46.             alphaRotor, betaRotor, gammaRotor = turnRotors(alphaRotor, betaRotor,
gammaRotor, alphabetList)
47.         else:
48.             print("alphaRotor = ", alphaRotor)
49.             print("alphabetList = ", alphabetList)
50.             newList = permute((alphaRotor), alphabetList)
51.             tempLetter = newList[alphabetList.index(letter)]
52.             #tempLetter = permute((alphaRotor),
alphabetList)[alphabetList.index(letter)]
53.             print(newList)
54.             print(tempLetter)
55.             tempLetter = permute((betaRotor),
alphabetList)[alphabetList.index(tempLetter)]
56.             tempLetter = permute((gammaRotor),
alphabetList)[alphabetList.index(tempLetter)]
57.             tempLetter = reflector[alphabetList.index(tempLetter)]
58.             tempLetter = inversePermutation((gammaRotor),
alphabetList)[alphabetList.index(tempLetter)]
59.             tempLetter = inversePermutation((betaRotor),
alphabetList)[alphabetList.index(tempLetter)]
60.             tempLetter = inversePermutation((alphaRotor),
alphabetList)[alphabetList.index(tempLetter)]
61.             resultText.append(tempLetter)
62.             alphaRotor, betaRotor, gammaRotor = turnRotors(alphaRotor, betaRotor,
gammaRotor, alphabetList)
63.     return ''.join(resultText)
64.
65. def readTextFromFile(path):
66.     file1 = open(path, "r")
67.     data = file1.read()
68.     file1.close()
69.     return data
70.
71. def write_to_file(path, text):
72.     file1 = open(path, "w+")
73.     file1.write(text)
74.     file1.close()
75.
76. def wrapFiveCharacters(message):
77.     messageWrapFive = wrap(message, 5)
78.     return ' '.join(messageWrapFive)
79.
80.
81. def toUpperCase(text):
82.     return "".join(filter(str.isupper, text.upper()))

```

Screenshot Program

1. Vigenere Standard
 - a. No Space

Cryptography Program

Cipher method
☒ vigenere ☐ FullVigenere ☐ AutoKeyVigenere ☐ ExtendedVigenere ☐ Playfair ☐ SuperEncryption ☐ Affine ☐ Hill ☐ Enigma

Print method
☒ NoSpace ☐ FiveChar

Encryption
Enter Plaintext:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Ciphertext:

Decryption
Enter Ciphertext:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Plaintext:

i.

b. Five Char

Cryptography Program

Cipher method
☒ vigenere ☐ FullVigenere ☐ AutoKeyVigenere ☐ ExtendedVigenere ☐ Playfair ☐ SuperEncryption ☐ Affine ☐ Hill ☐ Enigma

Print method
☐ NoSpace ☒ FiveChar

Encryption
Enter Plaintext:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Ciphertext:

Decryption
Enter Ciphertext:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Plaintext:

i.

2. Full Vigenere

a. No Space

Cryptography Program

Cipher method
☒ vigenere ☒ FullVigenere ☐ AutoKeyVigenere ☐ ExtendedVigenere ☐ Playfair ☐ SuperEncryption ☐ Affine ☐ Hill ☐ Enigma

Print method
☒ NoSpace ☐ FiveChar

Encryption
Enter Plaintext:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Ciphertext: PMVEYZFBLJKYPMV

Decryption
Enter Ciphertext:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Plaintext: LOREMIPSUMDOLOR

i.

b. Five Char

Cryptography Program

Cipher method
☐ vigenere ☒ FullVigenere ☐ AutoKeyVigenere ☐ ExtendedVigenere ☐ Playfair ☐ SuperEncryption ☐ Affine ☐ Hill ☐ Enigma

Print method
☐ NoSpace ☒ FiveChar

Encryption
Enter Plaintext:
Enter key:
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:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Plaintext: LOREM IPSUM DOLOR

i.

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:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Ciphertext:

Decryption
Enter Ciphertext:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Plaintext:

i.

b. Five Char

Cryptography Program

Cipher method
☐ vigenere ☐ FullVigenere ☒ AutoKeyVigenere ☐ ExtendedVigenere ☐ Playfair ☐ SuperEncryption ☐ Affine ☐ Hill ☐ Enigma

Print method
☐ NoSpace ☒ FiveChar

Encryption
Enter Plaintext:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Ciphertext:

Decryption
Enter Ciphertext:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Plaintext:

i.

4. Extended Vigenere

a. No Space

Cryptography Program

Cipher method
☒ vigenere ☐ FullVigenere ☐ AutoKeyVigenere ☒ ExtendedVigenere ☐ Playfair ☐ SuperEncryption ☐ Affine ☐ Hill ☐ Enigma

Print method
☒ NoSpace ☐ FiveChar

Encryption
Enter Plaintext:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Ciphertext:

Decryption
Enter Ciphertext:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Plaintext:

i.

b. Five Char

Cryptography Program

Cipher method
☐ vigenere ☐ FullVigenere ☐ AutoKeyVigenere ☒ ExtendedVigenere ☐ Playfair ☐ SuperEncryption ☐ Affine ☐ Hill ☐ Enigma

Print method
☐ NoSpace ☒ FiveChar

Encryption
Enter Plaintext:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Ciphertext:

Decryption
Enter Ciphertext:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Plaintext:

i.

5. Playfair

a. No Space

Cryptography Program

Cipher method
☒ vigenere ☐ FullVigenere ☐ AutoKeyVigenere ☐ ExtendedVigenere ☒ Playfair ☐ SuperEncryption ☐ Affine ☐ Hill ☐ Enigma

Print method
☒ NoSpace ☐ FiveChar

Encryption
Enter Plaintext:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Ciphertext: BMODZBXDNABEKUDMUIXMMOUVIF

Decryption
Enter Ciphertext:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Plaintext: HIDETHEGOLDINTHETREXESTUMP

i.

b. Five Char

Cryptography Program

Cipher method
☐ vigenere ☐ FullVigenere ☐ AutoKeyVigenere ☐ ExtendedVigenere ☒ Playfair ☐ SuperEncryption ☐ Affine ☐ Hill ☐ Enigma

Print method
☐ NoSpace ☒ FiveChar

Encryption
Enter Plaintext:
Enter key:
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:
Enter key:
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

i.

6. SuperEncryption (Vigenere Standard + Myszkowski Transposition Cipher)

a. No Space

Cryptography Program

Cipher method
☐ vigenere ☐ FullVigenere ☐ AutoKeyVigenere ☐ ExtendedVigenere ☐ Playfair ☒ SuperEncryption ☐ Affine ☐ Hill ☐ Enigma

Print method
☒ NoSpace ☐ FiveChar

Encryption
Enter Plaintext:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Ciphertext: HELOGNQ*EEA*

Decryption
Enter Ciphertext:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Plaintext: hellohello

i.

b. Five Char

Cryptography Program

Cipher method
☐ vigenere ☐ FullVigenere ☐ AutoKeyVigenere ☐ ExtendedVigenere ☐ Playfair ☒ SuperEncryption ☐ Affine ☐ Hill ☐ Enigma

Print method
☐ NoSpace ☒ FiveChar

Encryption
Enter Plaintext:
Enter key:
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:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Plaintext: hello hello

i.

7. Affine

a. No Space

Cryptography Program

Cipher method
☐ vigenere ☐ FullVigenere ☐ AutoKeyVigenere ☐ ExtendedVigenere ☐ Playfair ☐ SuperEncryption ☒ Affine ☐ Hill ☐ Enigma

Print method
☒ NoSpace ☐ FiveChar

Encryption
Enter Plaintext:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Ciphertext: CZOLNE

Decryption
Enter Ciphertext:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Plaintext: KRIPTO

i.

b. Five Char

Cryptography Program

Cipher method
☐ vigenere ☐ FullVigenere ☐ AutoKeyVigenere ☐ ExtendedVigenere ☐ Playfair ☐ SuperEncryption ☒ Affine ☐ Hill ☐ Enigma

Print method
☐ NoSpace ☒ FiveChar

Encryption
Enter Plaintext:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Ciphertext: CZOLN E

Decryption
Enter Ciphertext:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Plaintext: KRIPT O

i.

8. Hill

a. No Space

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

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: POHPOH
Enter key: GYBNQKURP
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Plaintext: ACTACT

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: 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

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: 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

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

i.

9. Enigma

a. No Space

Cryptography Program

Cipher method
☒ vigenere ☐ FullVigenere ☐ AutoKeyVigenere ☐ ExtendedVigenere ☐ Playfair ☐ SuperEncryption ☐ Affine ☐ Hill ☒ Enigma

Print method
☒ NoSpace ☐ FiveChar

Encryption
Enter Plaintext:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Ciphertext: IWZQZHVGHDRPZ

Decryption
Enter Ciphertext:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Plaintext: THEREISNOTIME

i.

b. Five Char

Cryptography Program

Cipher method
☐ vigenere ☒ FullVigenere ☐ AutoKeyVigenere ☐ ExtendedVigenere ☐ Playfair ☐ SuperEncryption ☐ Affine ☐ Hill ☒ Enigma

Print method
☐ NoSpace ☒ FiveChar

Encryption
Enter Plaintext:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Ciphertext: IWZQZ HVGHD RPZ

Decryption
Enter Ciphertext:
Enter key:
Enter m key (for Affine):
Enter b key (for Affine):
Enter Input Filename (Path):
Enter Output Filename (Path):
Plaintext: THERE ISNOT IME

i.

Sample Plaintext, Key, Ciphertext

1. Vigenere

- plaintext : lorem ipsum dolor
- key : sit
- 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: BMODZBXDNABEKUDMUIXMMOUIVIF
6. SuperEncryption
 - a. Plaintext: hello. hello
 - b. Key: ac~tac~t
 - c. Ciphertext: HELOGNQ*EEA*
7. Affine
 - a. Affine
 - b. plaintext : krypto
 - 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		