import tkinter as tk

from tkinter import messagebox

from itertools import cycle

import random

import string

# Helper functions for Vigenere Cipher and Playfair Cipher

# Vigenere Cipher standard (26 huruf alfabet)

def vigenere\_cipher(text, key, mode='encrypt'):

alphabet = string.ascii\_uppercase

key = key.upper()

text = text.upper()

key\_stream = cycle(key)

result = []

for char in text:

if char in alphabet:

text\_pos = alphabet.index(char)

key\_pos = alphabet.index(next(key\_stream))

if mode == 'encrypt':

result.append(alphabet[(text\_pos + key\_pos) % 26])

elif mode == 'decrypt':

result.append(alphabet[(text\_pos - key\_pos) % 26])

else:

result.append(char)

return ''.join(result)

# Extended Vigenere Cipher (256 karakter ASCII)

def extended\_vigenere\_cipher(text, key, mode='encrypt'):

key\_stream = cycle(key)

result = []

for char in text:

char\_code = ord(char)

key\_code = ord(next(key\_stream))

if mode == 'encrypt':

result.append(chr((char\_code + key\_code) % 256))

elif mode == 'decrypt':

result.append(chr((char\_code - key\_code) % 256))

return ''.join(result)

# Playfair Cipher (26 huruf alfabet)

def playfair\_cipher(text, key, mode='encrypt'):

alphabet = "ABCDEFGHIKLMNOPQRSTUVWXYZ"

key = "".join(sorted(set(key.upper()), key=lambda x: key.index(x))) # Remove duplicates while preserving order

matrix = [c for c in key if c in alphabet] + [c for c in alphabet if c not in key]

# Create 5x5 matrix

def create\_matrix():

return [matrix[i:i + 5] for i in range(0, 25, 5)]

matrix = create\_matrix()

def find\_position(char):

for row in range(5):

if char in matrix[row]:

return (row, matrix[row].index(char))

return None

def process\_pair(char1, char2, mode):

row1, col1 = find\_position(char1)

row2, col2 = find\_position(char2)

if row1 == row2:

# Same row, shift to right or left

if mode == 'encrypt':

return matrix[row1][(col1 + 1) % 5] + matrix[row2][(col2 + 1) % 5]

else:

return matrix[row1][(col1 - 1) % 5] + matrix[row2][(col2 - 1) % 5]

elif col1 == col2:

# Same column, shift down or up

if mode == 'encrypt':

return matrix[(row1 + 1) % 5][col1] + matrix[(row2 + 1) % 5][col2]

else:

return matrix[(row1 - 1) % 5][col1] + matrix[(row2 - 1) % 5][col2]

else:

# Rectangle swap

return matrix[row1][col2] + matrix[row2][col1]

text = text.upper().replace("J", "I")

text = ''.join(c for c in text if c in alphabet)

if len(text) % 2 != 0:

text += 'X' # Padding if the length of text is odd

result = []

for i in range(0, len(text), 2):

char1, char2 = text[i], text[i + 1]

if char1 == char2:

char2 = 'X'

result.append(process\_pair(char1, char2, mode))

return ''.join(result)

# Enigma Cipher with 3 rotors (26 huruf alfabet)

class EnigmaMachine:

def \_\_init\_\_(self):

self.rotor1 = "EKMFLGDQVZNTOWYHXUSPAIBRCJ"

self.rotor2 = "AJDKSIRUXBLHWTMCQGZNPYFVOE"

self.rotor3 = "BDFHJLCPRTXVZNYEIWGAKMUSQO"

self.reflector = "YRUHQSLDPXNGOKMIEBFZCWVJAT"

self.rotors = [self.rotor1, self.rotor2, self.rotor3]

def encrypt(self, text):

alphabet = string.ascii\_uppercase

text = text.upper()

result = []

for char in text:

if char in alphabet:

char = self.pass\_rotor\_forward(char)

char = self.reflector[alphabet.index(char)]

char = self.pass\_rotor\_backward(char)

result.append(char)

self.rotate\_rotors()

else:

result.append(char)

return ''.join(result)

def pass\_rotor\_forward(self, char):

alphabet = string.ascii\_uppercase

for rotor in self.rotors:

char = rotor[alphabet.index(char)]

return char

def pass\_rotor\_backward(self, char):

alphabet = string.ascii\_uppercase

for rotor in reversed(self.rotors):

char = alphabet[rotor.index(char)]

return char

def rotate\_rotors(self):

self.rotor1 = self.rotor1[1:] + self.rotor1[0]

# One-time pad (26 huruf alfabet)

def one\_time\_pad(text, pad):

alphabet = string.ascii\_uppercase

text = text.upper()

pad = pad.upper()

result = []

for t\_char, p\_char in zip(text, pad):

if t\_char in alphabet and p\_char in alphabet:

t\_index = alphabet.index(t\_char)

p\_index = alphabet.index(p\_char)

result.append(alphabet[(t\_index + p\_index) % 26])

else:

result.append(t\_char)

return ''.join(result)

# GUI Application

class CipherApp:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Cipher Application")

self.label = tk.Label(root, text="Choose Cipher:")

self.label.pack()

self.options = ["Vigenere Cipher", "Extended Vigenere Cipher", "Playfair Cipher", "Enigma Cipher", "One-time Pad"]

self.cipher\_var = tk.StringVar(value=self.options[0])

self.dropdown = tk.OptionMenu(root, self.cipher\_var, \*self.options)

self.dropdown.pack()

self.text\_label = tk.Label(root, text="Text:")

self.text\_label.pack()

self.text\_entry = tk.Entry(root, width=50)

self.text\_entry.pack()

self.key\_label = tk.Label(root, text="Key/Pad:")

self.key\_label.pack()

self.key\_entry = tk.Entry(root, width=50)

self.key\_entry.pack()

self.result\_label = tk.Label(root, text="Result:")

self.result\_label.pack()

self.result\_text = tk.Text(root, height=10, width=50)

self.result\_text.pack()

self.encrypt\_button = tk.Button(root, text="Encrypt", command=self.encrypt\_text)

self.encrypt\_button.pack()

self.decrypt\_button = tk.Button(root, text="Decrypt", command=self.decrypt\_text)

self.decrypt\_button.pack()

def encrypt\_text(self):

text = self.text\_entry.get()

key = self.key\_entry.get()

cipher\_type = self.cipher\_var.get()

if cipher\_type == "Vigenere Cipher":

result = vigenere\_cipher(text, key, mode='encrypt')

elif cipher\_type == "Extended Vigenere Cipher":

result = extended\_vigenere\_cipher(text, key, mode='encrypt')

elif cipher\_type == "Playfair Cipher":

result = playfair\_cipher(text, key, mode='encrypt')

elif cipher\_type == "Enigma Cipher":

enigma = EnigmaMachine()

result = enigma.encrypt(text)

elif cipher\_type == "One-time Pad":

result = one\_time\_pad(text, key)

self.result\_text.delete(1.0, tk.END)

self.result\_text.insert(tk.END, result)

def decrypt\_text(self):

text = self.text\_entry.get()

key = self.key\_entry.get()

cipher\_type = self.cipher\_var.get()

if cipher\_type == "Vigenere Cipher":

result = vigenere\_cipher(text, key, mode='decrypt')

elif cipher\_type == "Extended Vigenere Cipher":

result = extended\_vigenere\_cipher(text, key, mode='decrypt')

elif cipher\_type == "Playfair Cipher":

result = playfair\_cipher(text, key, mode='decrypt')

elif cipher\_type == "Enigma Cipher":

messagebox.showinfo("Info", "Enigma Cipher tidak dapat di-decrypt dengan key saat ini")

result = "Decryption not supported"

elif cipher\_type == "One-time Pad":

messagebox.showinfo("Info", "One-time Pad memerlukan pad yang sama untuk dekripsi")

result = "Decryption not supported"

self.result\_text.delete(1.0, tk.END)

self.result\_text.insert(tk.END, result)

# Main Application

root = tk.Tk()

app = CipherApp(root)

root.mainloop()