**Nama : Affan Radiansyah Putra**

**NIM : 222048**

**Kelas : 5TKKO-G**

**Tabel Laporan**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Spesifikasi | Berhasil (√) | Kurang Berhasil(√) | Keterangan |
| 1. | Vigenere Chiper | √ |  | Dapat bekerja dengan baik |
| 2. | Extended Vigenere Chiper | √ |  | Dapat bekerja dengan baik |
| 3. | Playfair Chiper | √ |  | Sedikit kurang di dekripsi |
| 4. | Enigma Chiper | √ |  | Dapat bekrja dengan baik |
| 5. | One-Time Pad | **√** |  | Dapat bekerja dengan baik |

**Source Program**

import tkinter as tk

from tkinter import messagebox, ttk

def vigenere\_cipher\_standard(text, key, mode):

    alphabet\_upper = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

    alphabet\_lower = 'abcdefghijklmnopqrstuvwxyz'

    result = ''

    key = key.upper()

    key\_len = len(key)

    j = 0

    for i, char in enumerate(text):

        if char.isupper():

            shift = alphabet\_upper.index(key[j % key\_len])

            idx = alphabet\_upper.index(char)

            new\_idx = (idx + shift) % 26 if mode == 'encrypt' else (idx - shift) % 26

            result += alphabet\_upper[new\_idx]

            j += 1

        elif char.islower():

            shift = alphabet\_upper.index(key[j % key\_len])

            idx = alphabet\_lower.index(char)

            new\_idx = (idx + shift) % 26 if mode == 'encrypt' else (idx - shift) % 26

            result += alphabet\_lower[new\_idx]

            j += 1

        else:

            result += char

    return result

def vigenere\_cipher\_extended(text, key, mode):

    result = ''

    key\_len = len(key)

    for i, char in enumerate(text):

        shift = ord(key[i % key\_len])

        char\_code = ord(char)

        new\_code = (char\_code + shift) % 256 if mode == 'encrypt' else (char\_code - shift) % 256

        result += chr(new\_code)

    return result

def playfair\_cipher(text, key, mode):

    def create\_matrix(key):

        alphabet = "ABCDEFGHIKLMNOPQRSTUVWXYZ"

        key = ''.join(sorted(set(key.upper()), key=lambda x: key.index(x)))

        matrix = [char for char in key if char in alphabet]

        for char in alphabet:

            if char not in matrix:

                matrix.append(char)

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

    def find\_position(char, matrix):

        for row\_idx, row in enumerate(matrix):

            if char in row:

                return row\_idx, row.index(char)

    def process\_pair(a, b, matrix, mode):

        ra, ca = find\_position(a, matrix)

        rb, cb = find\_position(b, matrix)

        if ra == rb:

            return (matrix[ra][(ca+1) % 5] + matrix[rb][(cb+1) % 5]) if mode == 'encrypt' else (matrix[ra][(ca-1) % 5] + matrix[rb][(cb-1) % 5])

        elif ca == cb:

            return (matrix[(ra+1) % 5][ca] + matrix[(rb+1) % 5][cb]) if mode == 'encrypt' else (matrix[(ra-1) % 5][ca] + matrix[(rb-1) % 5][cb])

        else:

            return matrix[ra][cb] + matrix[rb][ca]

    matrix = create\_matrix(key)

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

    if len(text) % 2 != 0:

        text += 'X'

    result = ''

    for i in range(0, len(text), 2):

        result += process\_pair(text[i], text[i+1], matrix, mode)

    return result

def one\_time\_pad(text, key, mode):

    alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

    result = ''

    key = key.upper()

    key\_len = len(key)

    if len(key) < len(text):

        messagebox.showerror("Error", "Key must be at least as long as the text")

        return ""

    for i, char in enumerate(text):

        if char.isalpha():

            shift = alphabet.index(key[i])

            idx = alphabet.index(char.upper())

            new\_idx = (idx + shift) % 26 if mode == 'encrypt' else (idx - shift) % 26

            result += alphabet[new\_idx] if char.isupper() else alphabet[new\_idx].lower()

        else:

            result += char

    return result

def enigma\_cipher(text, key, mode):

    rotor\_1 = 'EKMFLGDQVZNTOWYHXUSPAIBRCJ'

    rotor\_2 = 'AJDKSIRUXBLHWTMCQGZNPYFVOE'

    rotor\_3 = 'BDFHJLCPRTXVZNYEIWGAKMUSQO'

    reflector = 'YRUHQSLDPXNGOKMIEBFZCWVJAT'

    alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

    rotor\_1\_position = alphabet.index(key[0].upper())

    rotor\_2\_position = alphabet.index(key[1].upper())

    rotor\_3\_position = alphabet.index(key[2].upper())

    result = ''

    def rotate\_rotor(rotor):

        return rotor[1:] + rotor[0]

    for char in text.upper():

        if char not in alphabet:

            result += char

            continue

        rotor\_1 = rotate\_rotor(rotor\_1)

        rotor\_1\_position = (rotor\_1\_position + 1) % 26

        if rotor\_1\_position == 0:

            rotor\_2 = rotate\_rotor(rotor\_2)

            rotor\_2\_position = (rotor\_2\_position + 1) % 26

            if rotor\_2\_position == 0:

                rotor\_3 = rotate\_rotor(rotor\_3)

                rotor\_3\_position = (rotor\_3\_position + 1) % 26

        idx = alphabet.index(char)

        idx = alphabet.index(rotor\_1[idx])

        idx = alphabet.index(rotor\_2[idx])

        idx = alphabet.index(rotor\_3[idx])

        idx = alphabet.index(reflector[idx])

        idx = rotor\_3.index(alphabet[idx])

        idx = rotor\_2.index(alphabet[idx])

        idx = rotor\_1.index(alphabet[idx])

        result += alphabet[idx]

    return result

def process\_cipher():

    text = input\_text.get("1.0", "end-1c")

    key = key\_entry.get()

    cipher\_type = cipher\_var.get()

    mode = mode\_var.get()

    if cipher\_type == "Vigenere Standard":

        result = vigenere\_cipher\_standard(text, key, mode)

    elif cipher\_type == "Vigenere Extended":

        result = vigenere\_cipher\_extended(text, key, mode)

    elif cipher\_type == "Playfair":

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

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

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

    elif cipher\_type == "Enigma":

        result = enigma\_cipher(text, key, mode)

    else:

        messagebox.showerror("Error", "Unsupported cipher type")

        return

    output\_text.delete("1.0", "end")

    output\_text.insert("1.0", result)

root = tk.Tk()

root.title("Cipher GUI")

root.geometry("600x600")

root.config(bg="#f0f5fc")

header = tk.Label(root, text="Cipher Encryption & Decryption", font=("Arial", 16, "bold"), bg="#4a7a8c", fg="white")

header.pack(fill="x")

frame = tk.Frame(root, bg="#e0efff")

frame.pack(expand=True, fill="both", padx=20, pady=20)

tk.Label(frame, text="Input Text:", font=("Arial", 12), bg="#e0efff").grid(row=0, column=0, sticky="w", pady=5)

input\_text = tk.Text(frame, height=5, width=50, font=("Arial", 10))

input\_text.grid(row=1, column=0, columnspan=2, pady=5)

tk.Label(frame, text="Key:", font=("Arial", 12), bg="#e0efff").grid(row=2, column=0, sticky="w", pady=5)

key\_entry = tk.Entry(frame, font=("Arial", 10))

key\_entry.grid(row=3, column=0, columnspan=2, pady=5)

cipher\_var = tk.StringVar(value="Vigenere Standard")

cipher\_menu = ttk.Combobox(frame, textvariable=cipher\_var, values=["Vigenere Standard", "Vigenere Extended", "Playfair", "One-Time Pad", "Enigma"], font=("Arial", 10))

cipher\_menu.grid(row=5, column=0, columnspan=2, pady=5)

mode\_var = tk.StringVar(value="encrypt")

mode\_menu = ttk.Combobox(frame, textvariable=mode\_var, values=["encrypt", "decrypt"], font=("Arial", 10))

mode\_menu.grid(row=7, column=0, columnspan=2, pady=5)

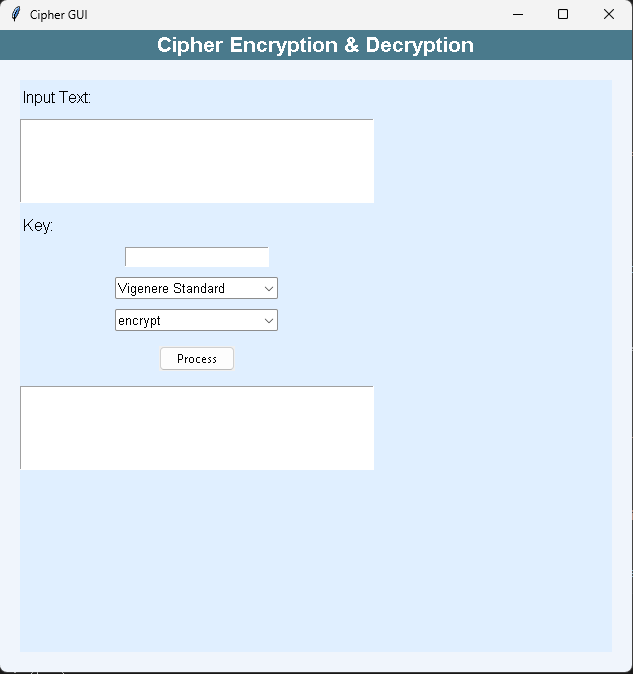
ttk.Button(frame, text="Process", command=process\_cipher).grid(row=8, column=0, columnspan=2, pady=10)

output\_text = tk.Text(frame, height=5, width=50, font=("Arial", 10))

output\_text.grid(row=10, column=0, columnspan=2, pady=5)

root.mainloop()

**Tampilan Antarmuka**

****

**Contoh Plainteks dan Chiperteks**

****

Teks yang diinput diatas merupakan sebuah plainteks setelah dilakukan pengenkripsian melalui metode vigenere standard maka terbentuklah chiperteks.