Nama : Muhammad Rafif Ramadhansyah

Program Studi : Teknik Informatika - S1

NIM : 4611422008

Mata Kuliah : Kriptografi

Dosen Pengampu : Dr. Alamsyah, S.Si., M.Kom.

1. Source Code Program

```
import tkinter as tk
from tkinter import filedialog, messagebox
import numpy as np

class CipherGUI:
    def __init__(self, master):
        self.master = master
        master.title("Program Cipher Desktop")
        master.geometry("600x400")

        self.cipher_types = ["Vigenere Cipher", "Playfair Cipher", "Hill Cipher"]
        self.current_cipher = tk.StringVar()
        self.current_cipher.set(self.cipher_types[0])

        self.create_widgets()

    def create_widgets(self):
        # Cipher selection
        tk.Label(self.master, text="Pilih Jenis Cipher:").grid(row=0, column=0, sticky="w",
padx=10, pady=5)
        tk.OptionMenu(self.master, self.current_cipher, *self.cipher_types,
command=self.on_cipher_change).grid(row=0, column=1, sticky="w", padx=10,
pady=5)

        # Input text
        tk.Label(self.master, text="Masukkan Plaintext:").grid(row=1, column=0,
sticky="w", padx=10, pady=5)
        self.input_text = tk.Text(self.master, height=5, width=50)
        self.input_text.grid(row=1, column=1, padx=10, pady=5)
```

```
        # Key input
        self.key_label = tk.Label(self.master, text="Key (minimal 12 karakter):")
        self.key_label.grid(row=2, column=0, sticky="w", padx=10, pady=5)
        self.key_entry = tk.Entry(self.master, width=50)
        self.key_entry.grid(row=2, column=1, padx=10, pady=5)

        # Hill Cipher matrix input
        self.matrix_frame = tk.Frame(self.master)
        self.matrix_frame.grid(row=3, column=1, padx=10, pady=5)
        self.matrix_entries = []
        for i in range(2):
            row_entries = []
            for j in range(2):
                entry = tk.Entry(self.matrix_frame, width=5)
                entry.grid(row=i, column=j, padx=2, pady=2)
                row_entries.append(entry)
            self.matrix_entries.append(row_entries)
        self.matrix_frame.grid_remove()  # Hide initially

        # Buttons
        tk.Button(self.master, text="Unggah File", command=self.upload_file).grid(row=4,
column=0, sticky="w", padx=10, pady=5)
        tk.Button(self.master, text="Enkripsi", command=self.encrypt).grid(row=4,
column=1, sticky="w", padx=10, pady=5)
        tk.Button(self.master, text="Dekripsi", command=self.decrypt).grid(row=4,
column=1, sticky="e", padx=10, pady=5)

        # Output text
        tk.Label(self.master, text="Hasil:").grid(row=5, column=0, sticky="w", padx=10,
pady=5)
        self.output_text = tk.Text(self.master, height=5, width=50)
        self.output_text.grid(row=5, column=1, padx=10, pady=5)

    def on_cipher_change(self, *args):
        if self.current_cipher.get() == "Hill Cipher":
            self.key_label.config(text="Matrix Key (2x2):")
            self.key_entry.grid_remove()
            self.matrix_frame.grid()
        else:
```

```
            self.key_label.config(text="Key (minimal 12 karakter):")
            self.key_entry.grid()
            self.matrix_frame.grid_remove()

    def upload_file(self):
        file_path = filedialog.askopenfilename(filetypes=[("Text files", "*.txt")])
        if file_path:
            with open(file_path, 'r') as file:
                content = file.read()
                self.input_text.delete(1.0, tk.END)
                self.input_text.insert(tk.END, content)

    def get_hill_matrix(self):
        return [[int(self.matrix_entries[i][j].get()) for j in range(2)] for i in range(2)]

    def encrypt(self):
        plaintext = self.input_text.get(1.0, tk.END).strip()
        cipher_type = self.current_cipher.get()

        if cipher_type == "Hill Cipher":
            key = self.get_hill_matrix()
        else:
            key = self.key_entry.get()
            if len(key) < 12:
                messagebox.showerror("Error", "Key setidaknya harus memiliki panjang 12
karakter.")
                return

        if cipher_type == "Vigenere Cipher":
            ciphertext = self.vigenere_cipher(plaintext, key, mode='encrypt')
        elif cipher_type == "Playfair Cipher":
            ciphertext = self.playfair_cipher(plaintext, key, mode='encrypt')
        elif cipher_type == "Hill Cipher":
            ciphertext = self.hill_cipher(plaintext, key, mode='encrypt')

        self.output_text.delete(1.0, tk.END)
        self.output_text.insert(tk.END, ciphertext)

    def decrypt(self):
        ciphertext = self.input_text.get(1.0, tk.END).strip()
```

```
        cipher_type = self.current_cipher.get()

        if cipher_type == "Hill Cipher":
            key = self.get_hill_matrix()
        else:
            key = self.key_entry.get()
            if len(key) < 12:
                messagebox.showerror("Error", "Key setidaknya harus memiliki panjang 12
karakter.")
                return

        if cipher_type == "Vigenere Cipher":
            plaintext = self.vigenere_cipher(ciphertext, key, mode='decrypt')
        elif cipher_type == "Playfair Cipher":
            plaintext = self.playfair_cipher(ciphertext, key, mode='decrypt')
        elif cipher_type == "Hill Cipher":
            plaintext = self.hill_cipher(ciphertext, key, mode='decrypt')

        self.output_text.delete(1.0, tk.END)
        self.output_text.insert(tk.END, plaintext)

    def vigenere_cipher(self, text, key, mode='encrypt'):
        result = []
        key_length = len(key)
        text = ''.join(filter(str.isalpha, text.upper()))
        key = ''.join(filter(str.isalpha, key.upper()))

        for i, char in enumerate(text):
            key_char = key[i % key_length]
            if mode == 'encrypt':
                result.append(chr((ord(char) + ord(key_char) - 2 * ord('A')) % 26 + ord('A')))
            else:
                result.append(chr((ord(char) - ord(key_char) + 26) % 26 + ord('A')))

        return ''.join(result)

    def playfair_cipher(self, text, key, mode='encrypt'):
        def create_matrix(key):
            alphabet = 'ABCDEFGHIKLMNOPQRSTUVWXYZ'  # I and J are treated as one
letter
```

```
    key = ''.join(dict.fromkeys(key.upper().replace('J', 'I') + alphabet))
    return [list(key[i:i+5]) for i in range(0, 25, 5)]

def find_position(matrix, char):
    for i, row in enumerate(matrix):
        if char in row:
            return i, row.index(char)
    return None

def prepare_text(text):
    text = ''.join(filter(str.isalpha, text.upper().replace('J', 'I')))
    prepared = []
    i = 0
    while i < len(text):
        if i == len(text) - 1 or text[i] == text[i+1]:
            prepared.append(text[i] + 'X')
            i += 1
        else:
            prepared.append(text[i:i+2])
            i += 2
    return prepared

matrix = create_matrix(key)
text_pairs = prepare_text(text)
result = []

for pair in text_pairs:
    char1, char2 = pair
    row1, col1 = find_position(matrix, char1)
    row2, col2 = find_position(matrix, char2)

    if row1 == row2:
        if mode == 'encrypt':
            result.append(matrix[row1][(col1+1)%5] + matrix[row2][(col2+1)%5])
        else:
            result.append(matrix[row1][(col1-1)%5] + matrix[row2][(col2-1)%5])
    elif col1 == col2:
        if mode == 'encrypt':
            result.append(matrix[(row1+1)%5][col1] + matrix[(row2+1)%5][col2])
        else:
```

```
                result.append(matrix[(row1-1)%5][col1] + matrix[(row2-1)%5][col2])
            else:
                result.append(matrix[row1][col2] + matrix[row2][col1])

        return ' '.join(result)

    def hill_cipher(self, text, key, mode='encrypt'):
        def matrix_mod_inv(matrix, modulus):
            det = int(np.round(np.linalg.det(matrix)))
            det_inv = pow(det % modulus, -1, modulus)
            adjoint = np.array([[matrix[1,1], -matrix[0,1]],
                                [-matrix[1,0], matrix[0,0]]])
            return (det_inv * adjoint % modulus).astype(int)

        key_matrix = np.array(key)
        text = ''.join(filter(str.isalpha, text.upper()))
        if len(text) % 2 != 0:
            text += 'X'

        result = []
        for i in range(0, len(text), 2):
            pair = np.array([ord(text[i]) - 65, ord(text[i+1]) - 65])
            if mode == 'encrypt':
                encrypted = np.dot(key_matrix, pair) % 26
            else:
                inv_key = matrix_mod_inv(key_matrix, 26)
                encrypted = np.dot(inv_key, pair) % 26
            result.extend([chr(int(c) + 65) for c in encrypted])

        return ''.join(result)

root = tk.Tk()
gui = CipherGUI(root)
root.mainloop()
```

2. Deskripsi Singkat Program

Program ini merupakan sebuah aplikasi GUI desktop yang dibangun menggunakan Python dan Tkinter. Program ini memungkinkan pengguna untuk mengenkripsi dan mendekripsi teks menggunakan tiga jenis cipher: Vigenere Cipher, Playfair Cipher, dan

Hill Cipher. Pengguna dapat memasukkan plaintext, memilih jenis cipher, dan mendapatkan ciphertext yang sesuai.

3. Link GitHub

GitHub Repository: https://github.com/raaapiiip/program-cipher-kriptografi.git