TUGAS 1

KEAMANAN KOMPUTER/KRIPTOGRAFI

Di susun oleh :

Nama : Nur Fadliyansyah Yahya

Nim : 202006

Kelas : 5TKKO-B Teknik Informatika

FAKULTAS TEKNIK

PROGRAM STUDI TEKNIK INFORMATIKA

UNIVERSITAS DIPA MAKASSAR

2022/2023

DAFTAR ISI

1. Source Code Program

Pembuatan source code di lakukan dalam bahasa python. Pembuatan GUI di lakukan dalam tkinter yang terdapat pada bahasa perograman python. Terdapat 5 file yang ber extention py(python) yang akan di build pada text editor visual studio code.

- Vigenere Cipher.py

```
    from CommonLib import *


def GenerateVigenereKey(key,length):
    # Generate key for Vigenere Cipher according to
desired length
    # Input : key, desired length
    # Output : Vigenere Cipher key with desired length
    if (len(key)>=length): # key is longer than desired
length -> take only the front characters
        return key[0:length]
    else: # key is shorter, duplicate the key
        multiple =  length//len(key)
        remainder = length%len(key)

        return key*multiple + key[0:remainder]

def VigenereEncrypt(plaintext,key):
    # Vigenere Encrypt plaintext with key
    # Input : plaintext, key
    # Output : ciphertext
```

```
    # Prepare the plaintext and key
    prepared_plaintext = PrepareText(plaintext)
    extended_key =
GenerateVigenereKey(PrepareText(key),len(prepared_plainte
xt))

    # Encrypt
    result = ""
    for i in range(len(prepared_plaintext)):
        encrypted_char_num =
(CharToNum(prepared_plaintext[i]) +
CharToNum(extended_key[i]))%26
        result += NumToChar(encrypted_char_num)
        if (i%5==4): # Set ciphertext to blocks of 5
characters
            result += " "

    return result.upper()

def VigenereDecrypt(ciphertext,key):
    # Vigenere Decrypt ciphertext with key
    # Input : ciphertext, key
    # Output : plaintext

    # Prepare the ciphertext and key
    prepared_ciphertext = PrepareText(ciphertext)
```

```
    extended_key =
GenerateVigenereKey(PrepareText(key),len(prepared_ciphert
ext))

    # Decrypt
    result = ""
    for i in range(len(prepared_ciphertext)):
        decrypted_char_num =
(CharToNum(prepared_ciphertext[i]) -
CharToNum(extended_key[i]))%26
        result += NumToChar(decrypted_char_num)

    return result

def GenerateVigenereAutoKey(plaintext,key):
    # Generate auto key for Vigenere Cipher according to
plaintext
    # Input : plaintext, key
    # Output : Auto Key Vigenere
    if (len(key)>=len(plaintext)): # key is longer than
desired length -> take only the front key characters
        return key[0:len(plaintext)]
    else: # key is shorter, extend with plaintext
        return key + plaintext[0:len(plaintext)-len(key)]

def AutoKeyVigenereEncrypt(plaintext,key):
    # Auto Key Vigenere Encrypt plaintext with key
    # Input : plaintext, key
```

```
    # Output : ciphertext

    # Prepare the plaintext and key
    prepared_plaintext = PrepareText(plaintext)
    extended_key =
GenerateVigenereAutoKey(prepared_plaintext,PrepareText(ke
y))

    # Encrypt
    result = ""
    for i in range(len(prepared_plaintext)):
        encrypted_char_num =
(CharToNum(prepared_plaintext[i]) +
CharToNum(extended_key[i]))%26
        result += NumToChar(encrypted_char_num)
        if (i%5==4): # Set ciphertext to blocks of 5
characters
            result += " "

    return result.upper()

def AutoKeyVigenereDecrypt(ciphertext,key):
    # Auto Key Vigenere Decrypt ciphertext with key
    # Input : ciphertext, key
    # Output : plaintext

    # Prepare the ciphertext
    prepared_ciphertext = PrepareText(ciphertext)
```

```
    key = PrepareText(key)

    # Decrypt
    result = ""
    # For the first key-length characters, decrypt with
the key
    for i in range(len(key)):
        decrypted_char_num =
(CharToNum(prepared_ciphertext[i]) -
CharToNum(key[i]))%26
        result += NumToChar(decrypted_char_num)

    # For the next characters, decrypt starting from the
first decrypted character
    for i in range(len(key),len(prepared_ciphertext)):
        decrypted_char_num =
(CharToNum(prepared_ciphertext[i]) - CharToNum(result[i-
len(key)]))%26
        result += NumToChar(decrypted_char_num)

    return result
```

- Extended vigenere cipher.py

```
from VigenereLib import *

def ExtendedEncrypt(plaintext,key):
    # Extended Vigenere Encrypt plaintext with key
    # Input : plaintext, key
    # Output : ciphertext

    # Prepare the key
    extended_key =
GenerateVigenereKey(key,len(plaintext))

    # Encrypt
    result = ""
    for i in range(len(plaintext)):
        encrypted_char_num = (ord(plaintext[i]) +
ord(extended_key[i]))%256
        result += chr(encrypted_char_num)
    return result

def ExtendedDecrypt(ciphertext,key):
    # Extended Vigenere Decrypt ciphertext with key
    # Input : ciphertext, key
    # Output : plaintext

    # Prepare the key
```

```
    extended_key =
GenerateVigenereKey(key,len(ciphertext))
    # Decrypt
    result = ""
    for i in range(len(ciphertext)):
        decrypted_char_num = (ord(ciphertext[i]) -
ord(extended_key[i]))%256
        result += chr(decrypted char num)

    return result
```

- Playfair cipher.py

```
from CommonLib import *


def GeneratePlayfairKeyMatrix(key):
    # Create Playfair Key Matrix
    # Input : key(string)
    # Output : Playfair Key Matrix from key

    result = ""

    # delete non alphabet characters and 'j'
    for c in key.lower():
        if ((c not in result) and (c!='j') and
(c.isalpha())):
            result+=c

    # extend the key with the other characters not in
string according to alphabet order (except 'j')
    for c in range(ord('a'),ord('z')+1):
        if ((chr(c) not in result) and (chr(c)!='j')):
            result+=chr(c)

    # create the matrix
    result_matr = []
    for i in range(5):
        new_array = []
        for j in range(5):
```

```
            new_array.append(result[i*5+j])
        result_matr.append(new_array)

    return result_matr

def PlayfairPlaintextBigram(plaintext):
    # Create bigram from plaintext
    # Input : plaintext
    # Output : array of playfair bigrams

    # Prepare the plaintext
    prepared_plaintext = PrepareText(plaintext)

    # Replace j with i
    replaced_plaintext = ""
    for c in prepared_plaintext:
        if (c=='j'):
            replaced_plaintext += 'i'
        else:
            replaced_plaintext += c

    # Create the bigram array
    bigram_array = []
    i = 0
    while (i<len(replaced_plaintext)):
        if (i==(len(replaced_plaintext)-1)): # for the
last character with no pair, add 'x'
            bigram = replaced_plaintext[i] + 'x'
```

```
            i = i + 1
        elif
(replaced_plaintext[i]==replaced_plaintext[i+1]): # for
bigram with same characters, set 'x' as the first
character pair
            bigram = replaced_plaintext[i] + 'x'
            i = i + 1
        else: # create bigram with the next character
            bigram = replaced_plaintext[i] +
replaced_plaintext[i+1]
            i = i + 2

        bigram_array.append(bigram)

    return bigram_array

def PlayfairCiphertextBigram(ciphertext):
    # Create bigram from ciphertext
    # Input : ciphertext
    # Output : array of playfair bigrams

    # Prepare the ciphertext
    prepared_ciphertext = PrepareText(ciphertext)

    # Create the bigram array
    bigram_array = []
    i = 0
    while (i<len(prepared_ciphertext)):
```

```
        # pair every two characters
        # for playfair ciphertext length, it is always
expected to have even length
        bigram = prepared_ciphertext[i] +
prepared_ciphertext[i+1]
        bigram_array.append(bigram)
        i = i + 2

    return bigram_array

def FindPlayfairIndex(bigram,key_matr):
    # Find bigram index position in key matrix
    # Input : bigram to be searched, key matrix
    # Output : (x,y) index of bigram characters
    found0 = False
    found1 = False
    for i in range(5):
        for j in range(5):
            if (key_matr[i][j]==bigram[0]):
                x0 = j
                y0  =  i
                found0 = True
            if (key_matr[i][j]==bigram[1]):
                x1 = j
                y1  =  i
                found1 = True
            if ((found0) and (found1)):
                break
```

```
        if ((found0) and (found1)):
            break

    return x0,y0,x1,y1

def PlayfairEncrypt(plaintext,key):
    # Playfair Encrypt plaintext with key
    # Input : plaintext, key
    # Output : ciphertext

    # Prepare the bigram and key matrix
    plaintext_bigram = PlayfairPlaintextBigram(plaintext)
    playfair_key = GeneratePlayfairKeyMatrix(key)

    # Encrypt
    encrypted_text = ""
    for bigram in plaintext_bigram:
        x0,y0,x1,y1 =
FindPlayfairIndex(bigram,playfair_key)
        if (x0==x1): # same row -> take the next
characters
            encrypted_bigram = playfair_key[(y0+1)%5][x0]
+ playfair_key[(y1+1)%5][x1]
        elif (y0==y1): # same column -> take the next
characters
            encrypted_bigram = playfair_key[y0][(x0+1)%5]
+ playfair_key[y1][(x1+1)%5]
```

```
        else: # different row and column -> take the
rectangle angle characters
            encrypted_bigram = playfair_key[y0][x1] +
playfair_key[y1][x0]

        encrypted_text += encrypted_bigram

    # Split into blocks of 5 characters
    result = ""
    for i in range(len(encrypted_text)):
        result += encrypted_text[i].upper()
        if (i%5==4):
            result += ' '

    return result

def PlayfairDecrypt(ciphertext,key):
    # Playfair Decrypt ciphertext with key
    # Input : ciphertext, key
    # Output : plaintext

    # Prepare the bigram and key matrix
    ciphertext_bigram =
PlayfairCiphertextBigram(ciphertext)
    playfair_key = GeneratePlayfairKeyMatrix(key)

    # Decrypt
    decrypted_text = ""
```

```
    for bigram in ciphertext_bigram:
        x0,y0,x1,y1 =
FindPlayfairIndex(bigram,playfair_key)
        if (x0==x1): # same row -> take the previous
character
            decrypted_bigram = playfair_key[(y0-1)%5][x0]
+ playfair_key[(y1-1)%5][x1]
        elif (y0==y1): # same column -> take the previous
character
            decrypted_bigram = playfair_key[y0][(x0-1)%5]
+ playfair_key[y1][(x1-1)%5]
        else: # different row and column -> take the
rectangle angle characters
            decrypted_bigram = playfair_key[y0][x1] +
playfair_key[y1][x0]

        decrypted_text += decrypted_bigram

    # Delete the inserted 'x'
    result = ""
    for i in range(len(decrypted_text)):
        if (decrypted_text[i]=='x'):
            if (i==(len(decrypted_text)-1)): # skip the
last 'x' character
                pass
            elif (decrypted_text[i-
1]==decrypted_text[i+1]): # skip the 'x' character
between two same characters
```

```
                pass
            else:
                result += decrypted_text[i]
        else:
            result += decrypted text[i]

    return result
```

- Enigma cipher.py

```
from CommonLib import *

def AffineEncrypt(plaintext,multiple,offset):
    # Affine Encrypt
    # Input : plaintext, multiple, offset
    # Output : ciphertext

    # Prepare the plaintext
    prepared_plaintext = PrepareText(plaintext)

    # Encrypt
    result = ""
    for i in range(len(prepared_plaintext)):
        encrypted_char_num =
(multiple*CharToNum(prepared_plaintext[i]) + offset)%26
        result += NumToChar(encrypted_char_num)
        if (i%5==4): # Set ciphertext to blocks of 5
characters
            result += " "

    return result.upper()

def AffineDecrypt(ciphertext,multiple,offset):
    # Affine Decrypt
    # Input : ciphertext, multiple, offset
    # Output : plaintext
```

```
    # Prepare the ciphertext
    prepared_ciphertext = PrepareText(ciphertext)

    # Find modulo inverse
    for i in range(1,26):
        if ((multiple*i)%26==1):
            inverse_modulo = i
            break

    # Decrypt
    result = ""
    for i in range(len(prepared_ciphertext)):
        encrypted_char_num =
inverse_modulo*(CharToNum(prepared_ciphertext[i]) -
offset)%26
        result += NumToChar(encrypted_char_num)

    return result
```

- One-time pad.py

```
from CommonLib import *
from VigenereLib import *

def MatrixFullVigenere(key):
    arr_key = []
    for i in range(len(key)):
        num_key = CharToNum(key[i])
        arr_key.append(i)

    abjadUrut = []
    for i in range(26):
        abjadUrut.append(i)

    X = 0
    tempAbjad = abjadUrut.copy()
    matVig = []
    for i in range(26):
        rowVig = []
        for j in range(25):
            X = X + arr_key[j%len(arr_key)]
            X = X%len(tempAbjad)
            rowVig.append(tempAbjad[X])
            tempAbjad.pop(X)
            X = tempAbjad[0]
        rowVig.append(tempAbjad[0])
        tempAbjad.pop(0)
```

```
        tempAbjad = rowVig.copy()
        matVig.append(rowVig)
    return matVig

def  FullVigenereEncrypt(plaintext,key):
    # Vigenere Encrypt plaintext with key
    # Input : plaintext, key
    # Output : ciphertext

    # Prepare the plaintext, key, and matrix
    prepared_plaintext = PrepareText(plaintext)
    extended_key =
GenerateVigenereKey(key,len(prepared_plaintext))
    matrixVig = MatrixFullVigenere(key)

    # Encrypt
    result = ""
    for i in range(len(prepared_plaintext)):
        num_plaintext = CharToNum(prepared_plaintext[i])
        num_extended_key = CharToNum(extended_key[i])
        encrypted_char_num =
matrixVig[num_extended_key][num_plaintext]
        result += NumToChar(encrypted_char_num)
        if (i%5==4): # Set ciphertext to blocks of 5
characters
            result += " "
    return result.upper()
```

```
def  FullVigenereDecrypt(ciphertext,key):
    # Vigenere Decrypt ciphertext with key
    # Input : ciphertext, key
    # Output : plaintext

    # Prepare the ciphertext, key, and matrix
    prepared_ciphertext = PrepareText(ciphertext)
    extended_key =
GenerateVigenereKey(key,len(prepared_ciphertext))
    matrixVig = MatrixFullVigenere(key)

    # Decrypt
    result = ""
    for i in range(len(prepared_ciphertext)):
        num_ciphertext =
CharToNum(prepared_ciphertext[i])
        num_extended_key = CharToNum(extended_key[i])
        idx =
matrixVig[num_extended_key].index(num_ciphertext)
        decrypted_char_num = idx
        result += NumToChar(decrypted_char_num)

    return result
```

- Componets.py

```
import tkinter as tk
import tkinter.scrolledtext as st

class TextFrame:
    def __init__(self,title,width=50,height=5):
        # Constructor for text frame
        # Components : title and input field (big text)
        # Input :
        #           title(string) for title
        #           width(int) and height(int) for input
field dimensions
        self.frame = tk.Frame()

        self.label =
tk.Label(master=self.frame,text=title)
        self.label.pack()

        self.entry =
st.ScrolledText(master=self.frame,width=width,height=heig
ht)
        self.entry.pack()

class KeyFrame:
    def __init__(self,title,width=30):
        # Constructor for key frame
        # Components : title, entry, randomizer
```

```
        # Input :
        #           title(string) for title
        #           width(int) for input field width
        self.frame = tk.Frame()

        self.label =
tk.Label(master=self.frame,text=title)
        self.label.pack()

        self.entry =
tk.Entry(master=self.frame,width=width)
        self.entry.pack()

        self.random_label =
tk.Label(master=self.frame,text="Randomizer length")
        self.random_label.pack(padx=2,pady=2,side="left",
anchor="center")

        self.random_entry =
tk.Entry(master=self.frame,width=4)
        self.random_entry.pack(padx=2,pady=2,side="left",
anchor="center")

        self.button =
tk.Button(master=self.frame,width=10,text="Randomize")
        self.button.pack(padx=2,pady=2,side="left",anchor
="center")
```

```
class AffineKeyFrame:
    def __init__(self,width=10):
        # Constructor for affine key frame
        # Components : two titles and two input fields
(one line)
        # Input :
        #           width(int) for input field width
        self.frame = tk.Frame()

        self.multiple_label =
tk.Label(master=self.frame,text="Multiple (m)")
        self.multiple_label.pack(padx=2,pady=2,side="left
")

        self.multiple_entry =
tk.Entry(master=self.frame,width=width)
        self.multiple_entry.pack(padx=2,pady=2,side="left
")

        self.offset_label =
tk.Label(master=self.frame,text="Offset (b)")
        self.offset_label.pack(padx=2,pady=2,side="left")

        self.offset_entry =
tk.Entry(master=self.frame,width=width)
        self.offset_entry.pack(padx=2,pady=2,side="left")

class ButtonListFrame:
```

```
    def ___init___(self,title,labels,width=20):
        # Constructor for list of buttons frame
        # Components : title and button list
        # Input :
        #           title(string) for title
        #           labels(list of strings) for button
label
        #           width(int) for button width
        self.frame = tk.Frame()

        self.label =
tk.Label(master=self.frame,text=title)
        self.label.pack()

        self.button_list = []
        for label in labels:
            new_button =
tk.Button(master=self.frame,text=label,width=width)
            new_button.pack(padx=2,pady=2)
            self.button_list.append(new_button)
```

- CommonLib.py

```
def CharToNum(char):
    # Changes a character to its nominal value, relative
to 'a'
    # Ex : 'a' -> 0, 'b' -> 1, 'c' -> 2, ..., 'z' -> 25
    # Expected input : lowercase character char
    # Output : number(0-25)
    return ord(char.lower()) - ord('a')

def NumToChar(num):
    # Changes a number to the corresponding character
    # Ex : 0 -> 'a', 1 -> 'b', 2 -> 'c', ..., 25 -> 'z'
    # Expected input : number(0-25)
    # Output : lowercase character
    return chr(ord('a')+num%26)

def PrepareText(text):
    # Preparing a text before encrypt/decrypt
    # delete all non-alphabet characters
(space,marks,number,etc.) and turning the alphabets to
lowercase characters
    # Expected input : mixed string
    # Output : alphabet string
    result = ""
    for char in text:
        if (char.isalpha()):
            result += char.lower()
```

```
    return result
```

- GUI.py

```
import tkinter as tk
import tkinter.scrolledtext as st
import tkinter.filedialog as fd
import random
import math
import pickle

from Components import *

from CommonLib import *
from VigenereLib import *
from PlayfairLib import *
from AffineLib import *
from FullVigenereLib import *
from ExtendedLib import *

class GUI:
    def_init_(self,parent):#--
        - init ---#
        self.parent = parent
        parent.title("Kriptografi")
        self.mode = "Vigenere"

        #--- define grid ---#
        parent.columnconfigure([0,1,2,3],weight=1)
```

```
        parent.rowconfigure([0,1,2],weight=1,minsize=100)

        #--- plaintext ---#
        self.plaintext = TextFrame(
            title="Plaintext",
            width=50,
            height=5
        )
        self.plaintext.frame.grid(row=0,column=0,columnsp
an=3)

        #--- encrypt button ---#
        self.encrypt_button =
tk.Button(text="Encrypt",command=self.Encrypt)
        self.encrypt_button.grid(row=1,column=0,padx=10,p
ady=10)

        #--- keyframe ---#
        self.keyframe = KeyFrame(title="Key",width=34)
        self.keyframe.button.bind("<Button-
1>",self.RandomizeKey)
        self.keyframe.frame.grid(row=1,column=1)

        #--- affinekeyframe ---#
        self.affinekeyframe = AffineKeyFrame(width=6)
        # is created only but not packed

        #--- decrypt button ---#
```

```
        self.decrypt_button =
tk.Button(text="Decrypt",command=self.Decrypt)
        self.decrypt_button.grid(row=1,column=2,padx=10,p
ady=10)

        #--- ciphertext ---#
        self.ciphertext = TextFrame(
            title="Ciphertext",
            width=50,
            height=5
        )
        self.ciphertext.frame.grid(row=2,column=0,columns
pan=3)

        #--- cipher method button list ---#
        cipher_method_list = ["Vigenere","Full
Vigenere","Auto-Key Vigenere","Extended
Vigenere","Playfair","Affine"]
        self.cipher_method_frame = ButtonListFrame(
            title = "Method : Vigenere",
            labels = cipher_method_list,
            width = 25
        )
        for button in
self.cipher_method_frame.button_list:
            button.bind("<Button-1>",lambda
event,mode=button["text"]: self.ChangeMode(event,mode))
```

```
        self.cipher_method_frame.frame.grid(row=0,column=
3,rowspan=3,sticky="ns")


        #--- file frame ---#
        file_method_list = ["Open Plaintext
TextFile","Open Ciphertext TextFile","Save Plaintext to
TextFile","Save Ciphertext to TextFile","Binary File"]
        self.file_frame = ButtonListFrame(
            title = "File",
            labels = file_method_list,
            width = 25
        )
        self.file_frame.button_list[0].bind("<Button-
1>",lambda event,text="plaintext":
self.OpenFileText(event,text))
        self.file_frame.button_list[1].bind("<Button-
1>",lambda event,text="ciphertext":
self.OpenFileText(event,text))
        self.file_frame.button_list[2].bind("<Button-
1>",lambda event,text="plaintext":
self.SaveFileText(event,text))
        self.file_frame.button_list[3].bind("<Button-
1>",lambda event,text="ciphertext":
self.SaveFileText(event,text))
        self.file_frame.button_list[4].bind("<Button-
1>",self.BinaryFileWindow)
        self.file_frame.frame.grid(row=2,column=3)
```

```
    def ChangeMode(self,event,mode):
        # Event handler when button in cipher button list
is pressed
        # Change cipher mode and the text
        if (mode=="Affine" and self.mode!="Affine"): #
Change to Affine from not Affine, change the frame
            self.keyframe.frame.grid_forget()
            self.affinekeyframe.frame.grid(row=1,column=1
)
        elif (mode!="Affine" and self.mode=="Affine"): #
Clicked not Affine from Affine, change the frame
            self.affinekeyframe.frame.grid_forget()
            self.keyframe.frame.grid(row=1,column=1)

        self.mode = mode
        self.cipher_method_frame.label["text"] = "Method
: " + mode

    def RandomizeKey(self,event):
        # Create randomized key according to input length

        # Take input length
        length = self.keyframe.random_entry.get()

        if (len(length)==0): # No length is inputed
            self.AlertWindow("Please insert randomizer
length")
```

```
        elif (not length.isnumeric()): # Inputted length
is not a number
            self.AlertWindow("Please insert randomizer
length in number")
        else:
            # Generate random string
            randomizer = ""
            for i in range(int(length)):
                randomizer +=
NumToChar(random.randint(0,26))

            self.keyframe.entry.delete(0,tk.END)
            self.keyframe.entry.insert(0,randomizer)


    def Encrypt(self):
        # Event handler when encrypt button is pressed
        # Encrypt plaintext and key

        if (self.mode!="Affine"): # for methods other
than Affine
            # Take the plaintext and key from the field
            plaintext =
self.plaintext.entry.get("1.0",tk.END)[:-1]
            key = self.keyframe.entry.get()

            # Check for validity
            if (len(plaintext)==0): # Empty plaintext
```

```
                self.AlertWindow("Please insert
plaintext")
            elif (len(key)==0): # Empty key
                self.AlertWindow("Please insert key")
            else:
                # Encrypt
                if (self.mode=="Vigenere"): # Vigenere
                    ciphertext =
VigenereEncrypt(plaintext,key)
                elif (self.mode=="Full Vigenere"): # Full
Vigenere
                    ciphertext =
FullVigenereEncrypt(plaintext,key)
                elif (self.mode=="Auto-Key Vigenere"): #
Auto Key
                    ciphertext =
AutoKeyVigenereEncrypt(plaintext,key)
                elif (self.mode=="Extended Vigenere"): #
Extended
                    ciphertext =
ExtendedEncrypt(plaintext,key)
                elif (self.mode=="Playfair"): # Playfair
                    ciphertext =
PlayfairEncrypt(plaintext,key)

                # Insert into ciphertext field
                self.ciphertext.entry.delete("1.0",tk.END
)
```

```
                self.ciphertext.entry.insert("1.0",cipher
text)

        else: # Affine
            # Take the plaintext and parameters from the
field
            plaintext =
self.plaintext.entry.get("1.0",tk.END)[:-1]
            multiple =
self.affinekeyframe.multiple_entry.get()
            offset =
self.affinekeyframe.offset_entry.get()

            # Check for validity
            if (len(plaintext)==0): # Empty plaintext
                self.AlertWindow("Please insert
plaintext")
            elif (not multiple.isnumeric() or not
offset.isnumeric()): # Non numeric multiple and offset
                self.AlertWindow("Multiple and offset is
a number")
            else:
                # Encrypt
                multiple = int(multiple)
                offset = int(offset)
                if (math.gcd(multiple,26)!=1):
                    self.AlertWindow("Multiple is not
relative prime of 26")
```

```
                else:
                    # Insert into ciphertext field
                    ciphertext =
AffineEncrypt(plaintext,multiple,offset)
                    self.ciphertext.entry.delete("1.0",tk
.END)
                    self.ciphertext.entry.insert("1.0",ci
phertext)


    def Decrypt(self):
        # Event handler when decrypt button is pressed
        # Decrypt ciphertext and key

        if (self.mode!="Affine"): # for methods other
than Affine
            # Take the ciphertext and key from the field
            key = self.keyframe.entry.get()
            ciphertext =
self.ciphertext.entry.get("1.0",tk.END)[:-1]

            # Check for validity
            if (len(ciphertext)==0): # Empty ciphertext
                self.AlertWindow("Please insert
ciphertext")
            elif (len(key)==0): # Empty key
                self.AlertWindow("Please insert key")
            else:
```

```
                # Decrypt
                if (self.mode=="Vigenere"): # Vigenere
                    plaintext =
VigenereDecrypt(ciphertext,key)
                elif (self.mode=="Full Vigenere"): # Full
Vigenere
                    plaintext =
FullVigenereDecrypt(ciphertext,key)
                elif (self.mode=="Auto-Key Vigenere"): #
Auto Key
                    plaintext =
AutoKeyVigenereDecrypt(ciphertext,key)
                elif (self.mode=="Extended Vigenere"): #
Extended
                    plaintext =
ExtendedDecrypt(ciphertext,key)
                elif (self.mode=="Playfair"): # Playfair
                    plaintext =
PlayfairDecrypt(ciphertext,key)

                # Insert into plaintext field
                self.plaintext.entry.delete("1.0",tk.END)
                self.plaintext.entry.insert("1.0",plainte
xt)

        else: # Affine
            # Take the plaintext and parameters from the
field
```

```
            ciphertext =
self.ciphertext.entry.get("1.0",tk.END)[:-1]
            multiple =
self.affinekeyframe.multiple_entry.get()
            offset =
self.affinekeyframe.offset_entry.get()

            # Check for validity
            if (len(ciphertext)==0): # Empty plaintext
                self.AlertWindow("Please insert
plaintext")
            elif (not multiple.isnumeric() or not
offset.isnumeric()): # Non numeric multiple and offset
                self.AlertWindow("Multiple and offset is
a number")
            else:
                # Decrypt
                multiple = int(multiple)
                offset = int(offset)
                if (math.gcd(multiple,26)!=1):
                    self.AlertWindow("Multiple and is not
relative prime of 26")
                else:
                    # Insert into ciphertext field
                    plaintext =
AffineDecrypt(ciphertext,multiple,offset)
                    self.plaintext.entry.delete("1.0",tk.
END)
```

```
                    self.plaintext.entry.insert("1.0",pla
intext)

    def OpenFileText(self,event,text):
        # Open file using open file dialog

        # Take filename
        filename = fd.askopenfilename(
            initialdir = "/",
            title = "Select " + text + " file",
            filetypes = [("Text files (.txt)","*.txt")]
        )

        if (filename!=""): # If filename is chosen
            file = open(filename,"rt")
            content = file.read()

            file.close()

            if (text=="plaintext"): # For plaintext,
insert to plaintext field
                self.plaintext.entry.delete("1.0",tk.END)
                self.plaintext.entry.insert("1.0",content
)
            elif (text=="ciphertext"): # For ciphertext,
insert to ciphertext field
                self.ciphertext.entry.delete("1.0",tk.END
)
```

```
                self.ciphertext.entry.insert("1.0",conten
t)

        return "break"

    def SaveFileText(self,event,text):
        # Save file using save file dialog

        # Take filename
        filename = fd.asksaveasfilename(
            initialdir = "/",
            title = "Select " + text + " file",
            filetypes = [("Text files (.txt)","*.txt")],
            defaultextension = [("Text files
(.txt)","*.txt")]
        )

        if (filename!=""): # If file name is chosen
            file = open(filename,"wt")
            if (text=="plaintext"): # For plaintext,
insert the plaintext
                plaintext =
self.plaintext.entry.get("1.0",tk.END)[:-1]
                file.write(plaintext)
            elif (text=="ciphertext"): # For ciphertext,
insert the ciphertext
                ciphertext =
self.ciphertext.entry.get("1.0",tk.END)[:-1]
```

```
                file.write(ciphertext)

            file.close()

        return "break"

    def AlertWindow(self,text):
        # Create new window for alert
        # Components : label with input text and dismiss
button
        alert_window = tk.Toplevel(self.parent)
        alert_window.title("Alert")

        tk.Label(master=alert_window,text=text).pack(padx
=120,pady=20)
        tk.Button(master=alert_window,text="OK",width=10,
command=lambda:alert_window.destroy()).pack(pady=10)

        alert_window.grab_set()

    def BinaryFileWindow(self,event):
        # Create new window for file encrypt/decrypt
        # Components : label, key entry, and buttons
        new_window = tk.Toplevel(self.parent)
        new_window.title("Binary File")

        self.binary_file = ""
```

```
        self.file_label =
tk.Label(master=new_window,text="File : " +
self.binary_file)
        self.file_label.grid(row=0,column=0,columnspan=2,
sticky="we",padx=120,pady=2)
        self.key_label =
tk.Label(master=new_window,text="Key :")
        self.key_label.grid(row=1,column=0,columnspan=2,p
ady=2)
        self.key_entry =
tk.Entry(master=new_window,width=15)
        self.key_entry.grid(row=2,column=0,columnspan=2,p
ady=2)
        tk.Button(master=new_window,text="Choose
File",width=20,command=self.ChooseFileBinary).grid(row=3,
column=0,columnspan=2,pady=2)
        tk.Button(master=new_window,text="Encrypt and
Save",width=20,command=self.SaveEncryptedFile).grid(row=4
,column=0,columnspan=2,pady=2)
        tk.Button(master=new_window,text="Decrypt and
Save",width=20,command=self.SaveDecryptedFile).grid(row=5
,column=0,columnspan=2,pady=2)
        tk.Button(master=new_window,text="Unselect
File",width=20,command=self.UnselectFile).grid(row=6,colu
mn=0,columnspan=2,pady=2)


        new_window.grab_set()
```

```
    def ChooseFileBinary(self):
        # Take filename
        filename = fd.askopenfilename(
            initialdir = "/",
            title = "Select  file",
            filetypes = [("Text files
(.txt)","*.txt"),("Binary files (.bin)","*.bin"),("All
files","*.*")],
        )

        if (filename!=""):
            self.file_label["text"] = "File : " +
filename
            self.binary_file = filename

    def SaveEncryptedFile(self):
        # buka file di self.binary_file
        if (self.binary_file==""):
            self.AlertWindow("Please choose a file")
        else:
            #encrypt
            key = self.key_entry.get()
            if (len(key)==0):
                self.AlertWindow("Please insert key")
            else:
                file = open(self.binary_file,"r")
                readPlaintext = file.read()
```

```
                encripChipertext =
ExtendedEncrypt(readPlaintext, key)
                file.close()
                # save
                filename = fd.asksaveasfilename(
                    initialdir = "/",
                    title = "Save file",
                    filetypes = [("Binary files
(.bin)","*.bin"),("All files","*.*")],
                    defaultextension = [("Binary files
(.bin)","*.bin"),("All files","*.*")]
                )
                if (filename!=""):
                    output_file = open(filename, "wb")
                    pickle.dump(encripChipertext,
output_file)
                    output_file.close()


    def SaveDecryptedFile(self):
        # buka file di self.binary_file
        if (self.binary_file==""):
            self.AlertWindow("Please choose a file")
        else:
            # decrypt
            key = self.key_entry.get()
            if (len(key)==0):
                self.AlertWindow("Please insert key")
```

```
            else:
                if (self.binary_file[-4:]==".bin"):
                    file = open(self.binary_file,"rb")
                    readChipertext = pickle.load(file)
                    decripPlaintext =
ExtendedDecrypt(readChipertext, key)
                    file.close()
                    # sama kayak atasnya
                    filename = fd.asksaveasfilename(
                        initialdir = "/",
                        title = "Save file",
                        filetypes = [("Text files
(.txt)","*.txt"),("All files","*.*")],
                        defaultextension = [("Text files
(.txt)","*.txt"),("All files","*.*")]
                    )
                    if (filename!=""):
                        output_file = open(filename, "w")
                        output_file.write(decripPlaintext
)
                        output_file.close()
                else:
                    self.AlertWindow("Ekstensi file yang
mau didekripsi harus .bin")

    def UnselectFile(self):
        self.binary_file = ""
```

```
        self.file_label["text"] = "File : " +
self.binary_file
```

main.py

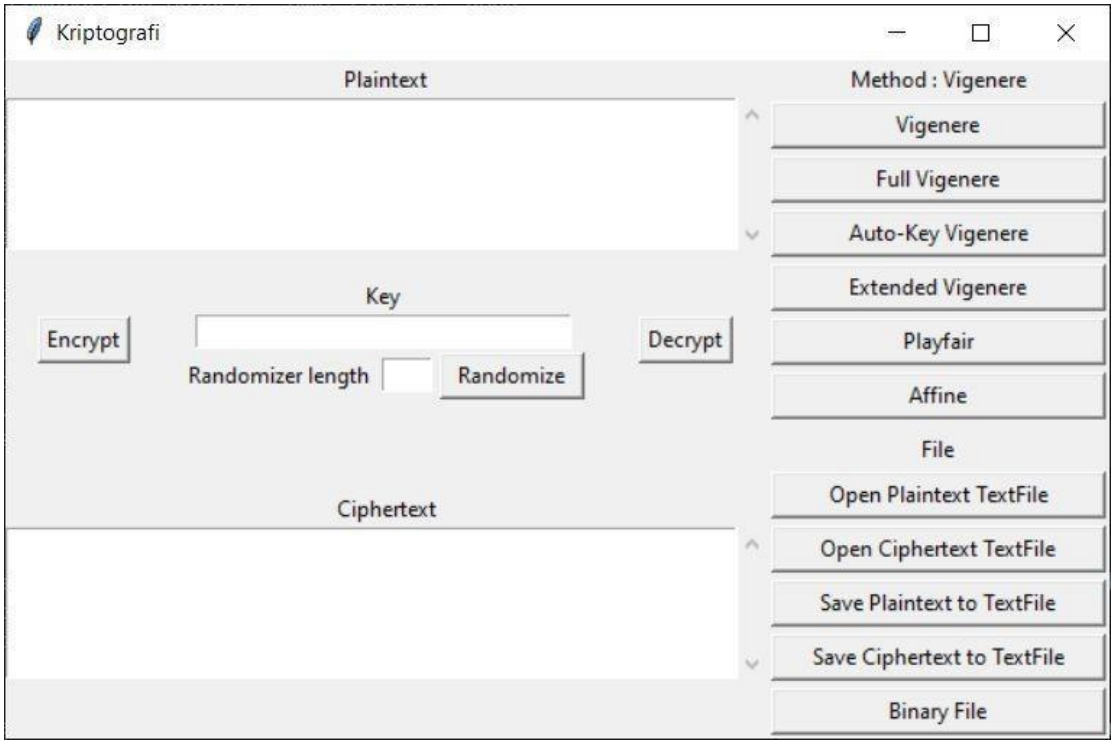
```
from GUI import *


window = tk.Tk()
GUI(window)
window.mainloop()
```

2. Tampilan antarmuka program (print screen)

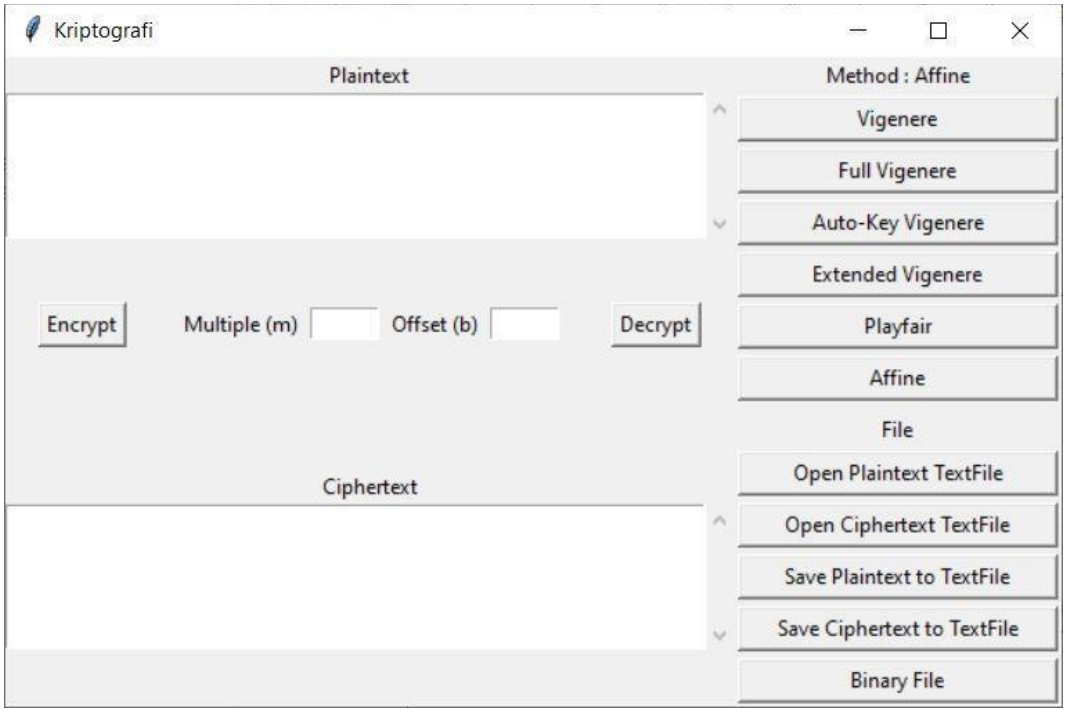
Antarmuka program dibuat dengan menggunakan tkinter pada dengan komponen pada Components.py dan digabungkan pada GUI.py.

Berikut adalah tampilan dari antarmuka program yang dibuat.



Gambar 1.1 Tampilan Antarmuka Program

Untuk metode Affine Cipher yang tidak menggunakan key, tampilan dari KeyFrame akan berubah ketika dipilih metode Affine Cipher. Pada tampilan ini, diminta input parameter m dan b untuk melakukan Affine Cipher seperti pada gambar berikut.



Gambar 1.2 Tampilan Antarmuka Program pada Affine Cipher

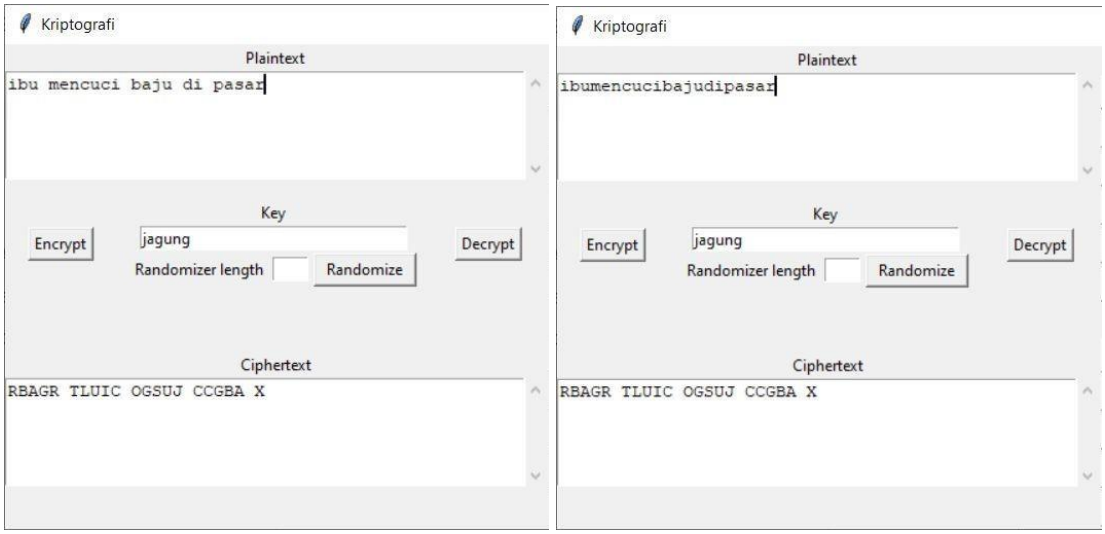
Untuk handling file binary, ketika button “Binary File” di pojok kanan bawah ditekanakan muncul window sebagai berikut.

Gambar 1.3 Tampilan Antarmuka Program untuk Binary File

3. Contoh plainteks dan cipherteks (text, gambar, file database, audio, video)

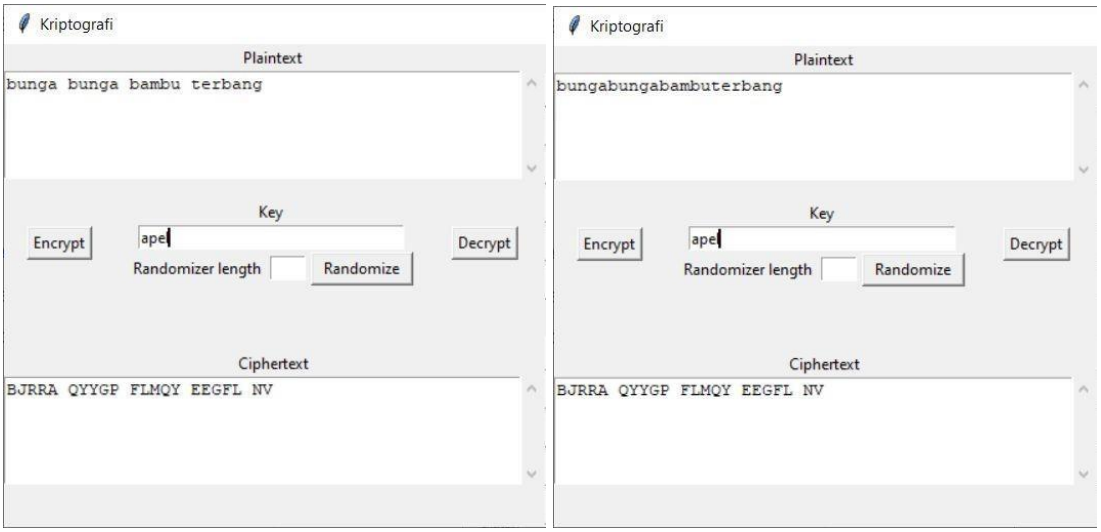
Pengujian Vigenere Cipher

Pada pengujian pertama, digunakan plaintext “ibu mencuci baju di pasar” dengan key“jagung” yang dienkripsi dengan Vigenere Cipher kemudian didekripsi lagi.



Gambar 1.4 Pengujian Vigenere Cipher 1

Selanjutnya, digunakan input plaintext “bunga bunga bambu terbang” dengan key“apel” yang memberikan hasil sebagai berikut.

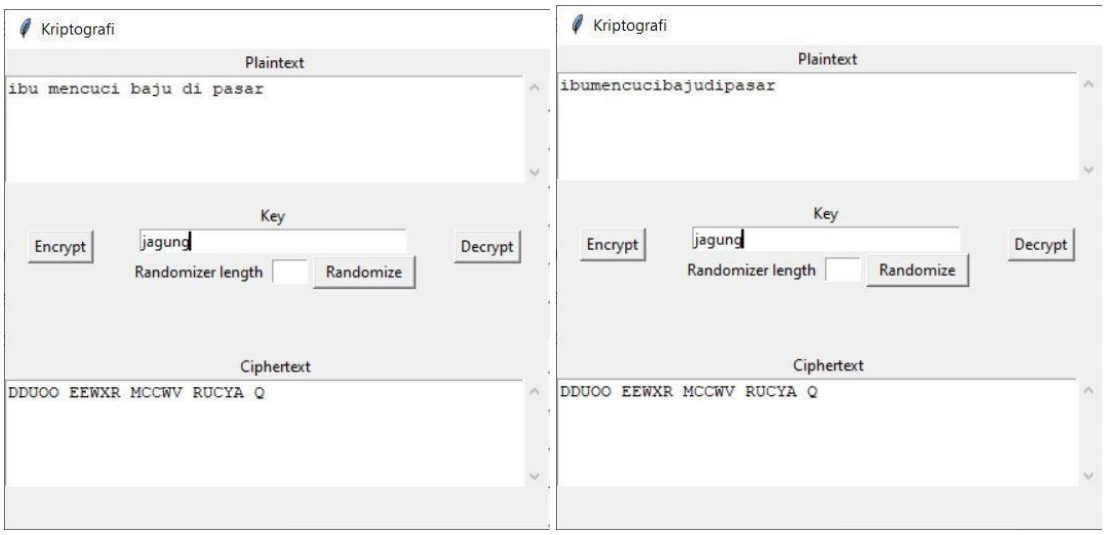


Gambar 5. Pengujian Vigenere Cipher 2

Dari hasil yang diperoleh, terlihat bahwa proses enkripsi-dekripsi yang terjadi berjalan dengan baik, hanya saja terjadi kehilangan informasi pada teks selain alfabet (seperti spasi). Hal ini sudah sesuai dengan spesifikasi yang diharapkan.

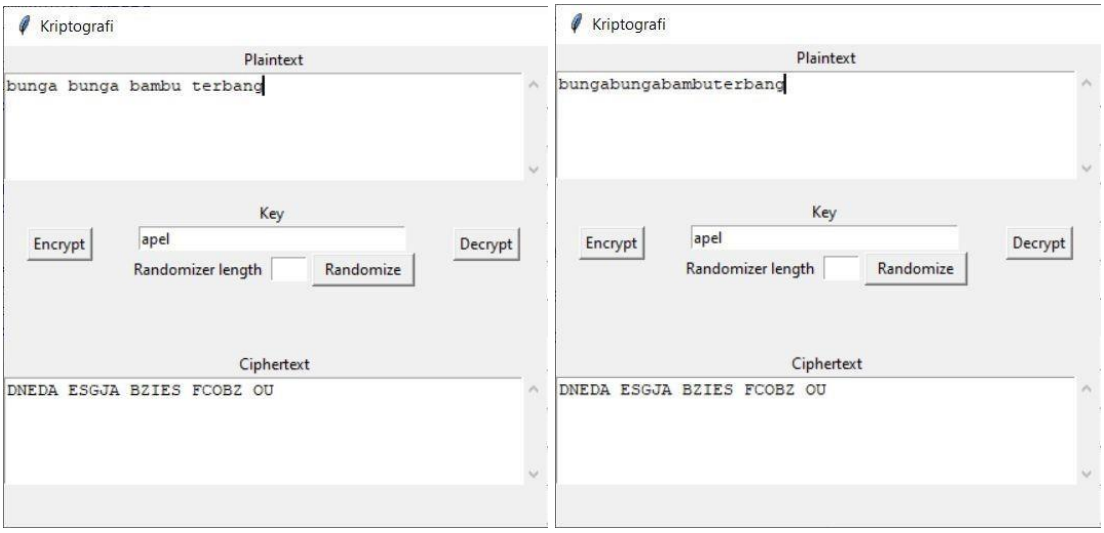
Pengujian Full Vigenere Cipher

Pada pengujian pertama, digunakan plaintext “ibu mencuci baju di pasar” dengan key“jagung” yang dienkripsi dengan Full Vigenere Cipher kemudian didekripsi lagi.



Gambar 1.6 Pengujian Full Vigenere Cipher 1

Selanjutnya, digunakan input plaintext “bunga bunga bambu terbang” dengan key“apel” yang memberikan hasil sebagai berikut.

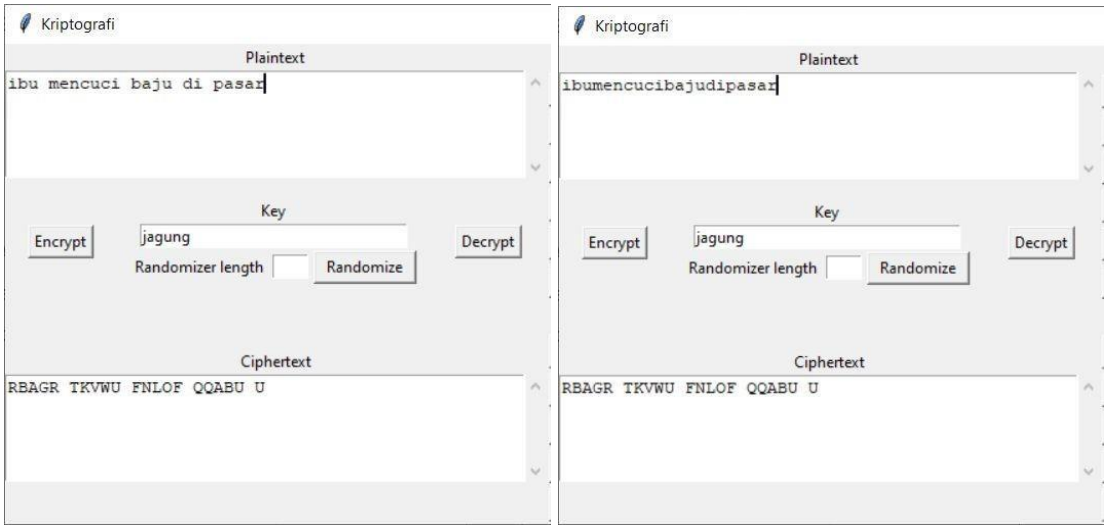


Gambar 1.7 Pengujian Full Vigenere Cipher 2

Dari hasil yang diperoleh, terlihat bahwa proses enkripsi-dekripsi yang terjadi berjalan dengan baik, hanya saja terjadi kehilangan informasi pada teks selain alfabet (seperti spasi). Hal ini sudah sesuai dengan spesifikasi yang diharapkan.

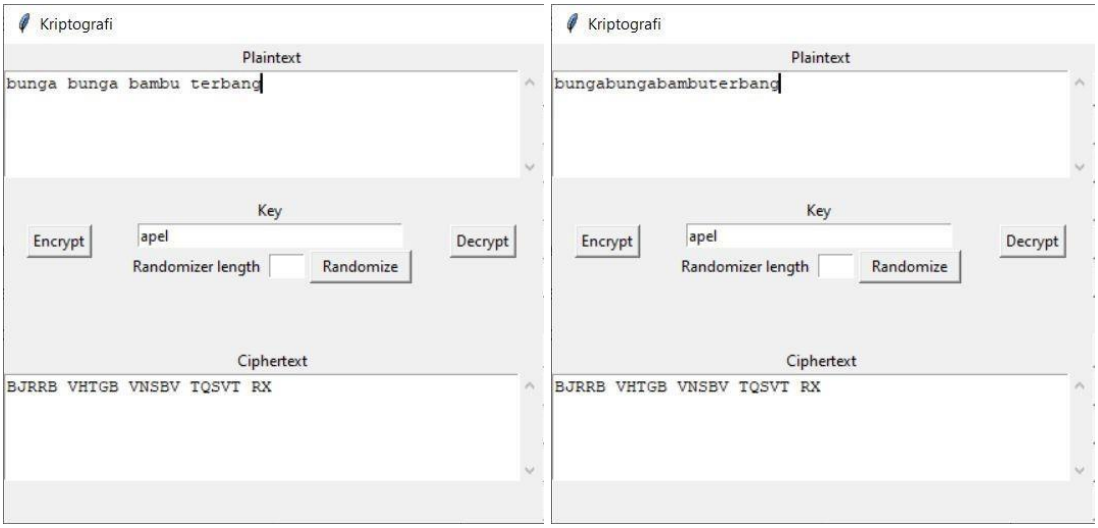
Pengujian Auto Key Vigenere Cipher

Pada pengujian pertama, digunakan plaintext “ibu mencuci baju di pasar” dengan key “jagung” yang dienkripsi dengan Auto Key Vigenere Cipher kemudian didekripsi lagi.



Gambar 1.8 Pengujian Auto Key Vigenere Cipher 1

Selanjutnya, digunakan input plaintext “bunga bunga bambu terbang” dengan key“apel” yang memberikan hasil sebagai berikut.

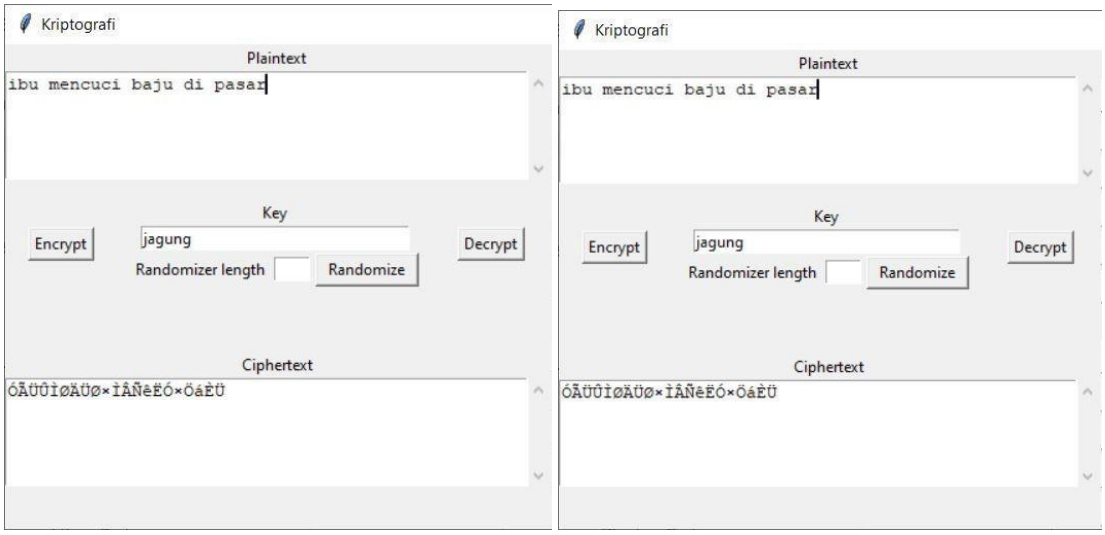


Gambar 1.9 Pengujian Auto Key Vigenere Cipher 2

Dari hasil yang diperoleh, terlihat bahwa proses enkripsi-dekripsi yang terjadi berjalan dengan baik, hanya saja terjadi kehilangan informasi pada teks selain alfabet (seperti spasi). Hal ini sudah sesuai dengan spesifikasi yang diharapkan.

Pengujian Extended Vigenere Cipher

Untuk pengujian Extended Vigenere Cipher pertama dilakukan dengan melakukan input pada GUI program yang dibuat, dengan mengisi bagian plaintext dan keykemudian melakukan enkripsi-dekripsi seperti biasa. Digunakan plaintext “ibu mencuci baju di pasar” dengan key “jagung” yang dienkripsi dengan Extended Vigenere Cipher kemudian didekripsi lagi.



Gambar 2.1 Pengujian Extended Vigenere Cipher 1

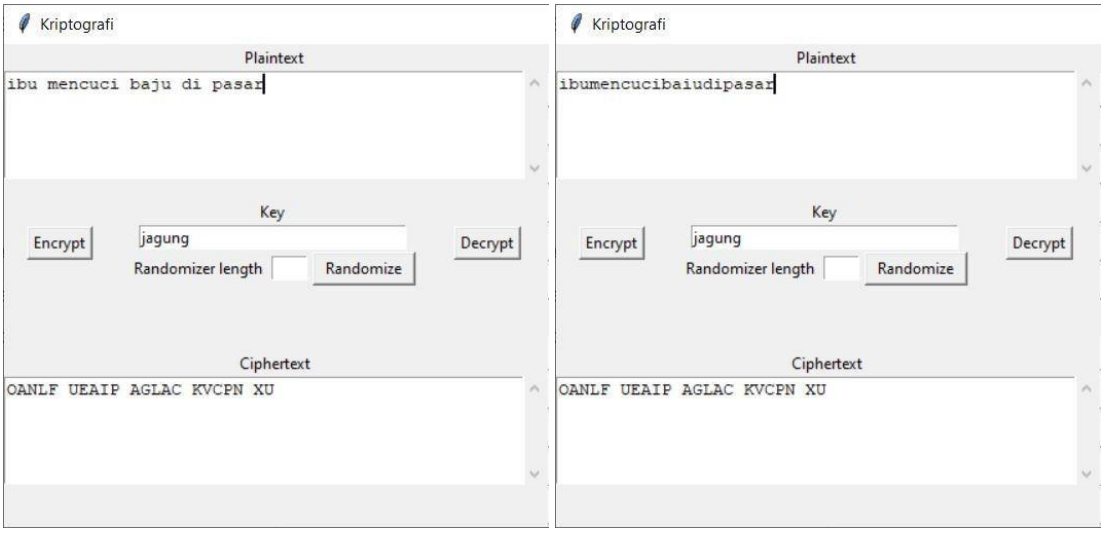
Selanjutnya, dilakukan pengujian pada file binary. Digunakan plaintext yang sama yaitu “ibu mencuci baju di pasar” dengan key “jagung” yang dienkripsi denganExtended Vigenere Cipher kemudian didekripsi lagi.

Gambar 2.2 Pengujian Extended Vigenere Cipher 2

Dari hasil yang diperoleh, terlihat bahwa proses enkripsi-dekripsi yang terjadi berjalan cukup baik, masih terdapat sedikit masalah pada enkripsi-dekripsi biasa dimana terdapat 32 karakter ASCII hasil enkripsi yang tidak dapat ditampilkan (*not printable*) sehingga memengaruhi proses dekripsi (rentan salah). Keunggulan chiper ini adalah proses enkripsi tidak menghilangkan karakter selain abjad, sehingga karakter seperti spasi dan tanda baca dapat terbaca (dekripsi) dengan baik.

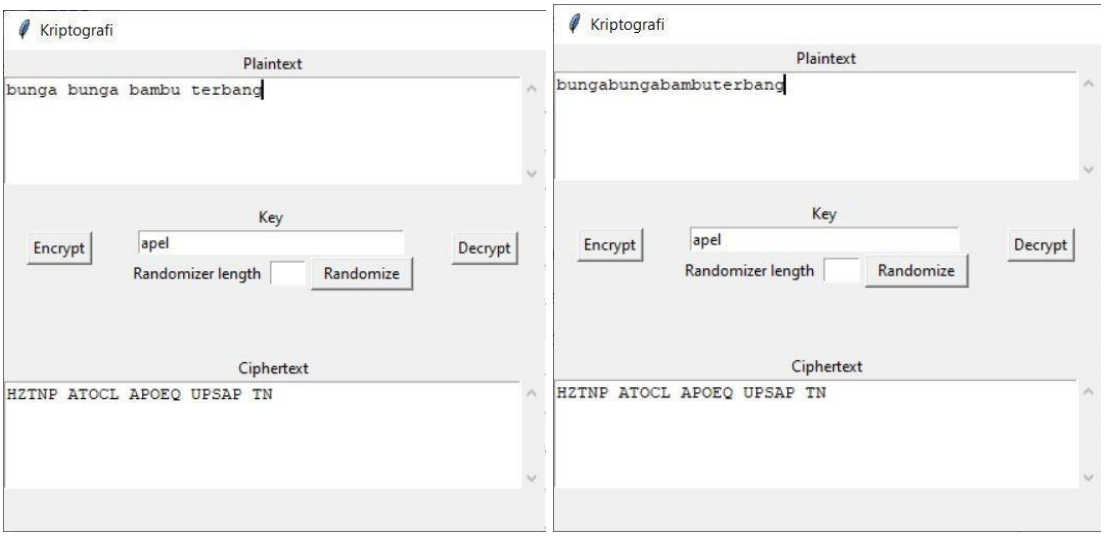
Pengujian Playfair Cipher

Pada pengujian pertama, digunakan plaintext “ibu mencuci baju di pasar” dengan key“jagung” yang dienkripsi dengan Playfair Cipher kemudian didekripsi lagi.



Gambar 2.3 Pengujian Playfair Cipher 1

Selanjutnya, digunakan input plaintext “bunga bunga bambu terbang” dengan key“apel” yang memberikan hasil sebagai berikut.

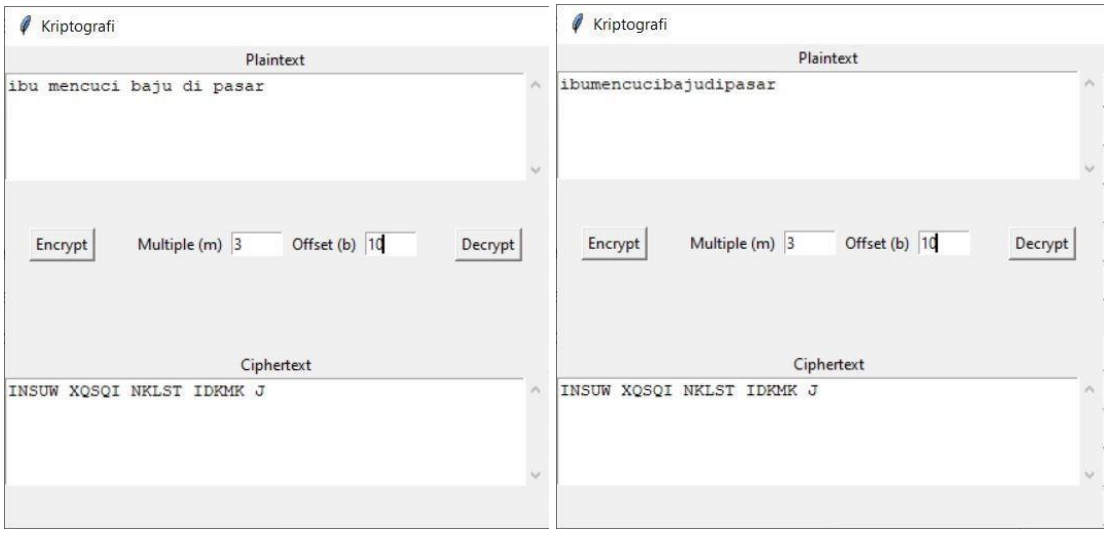


Gambar 2.4 Pengujian Playfair Cipher 2

Dari hasil yang diperoleh, terlihat bahwa proses enkripsi-dekripsi yang terjadi berjalan dengan baik, hanya saja terjadi kehilangan informasi pada teks selain alfabet (seperti spasi), dan kehilangan informasi mengenai huruf ‘j’ pada plainteks (diganti dengan huruf ‘i’). Hal ini sudah sesuai dengan spesifikasi yang diharapkan.

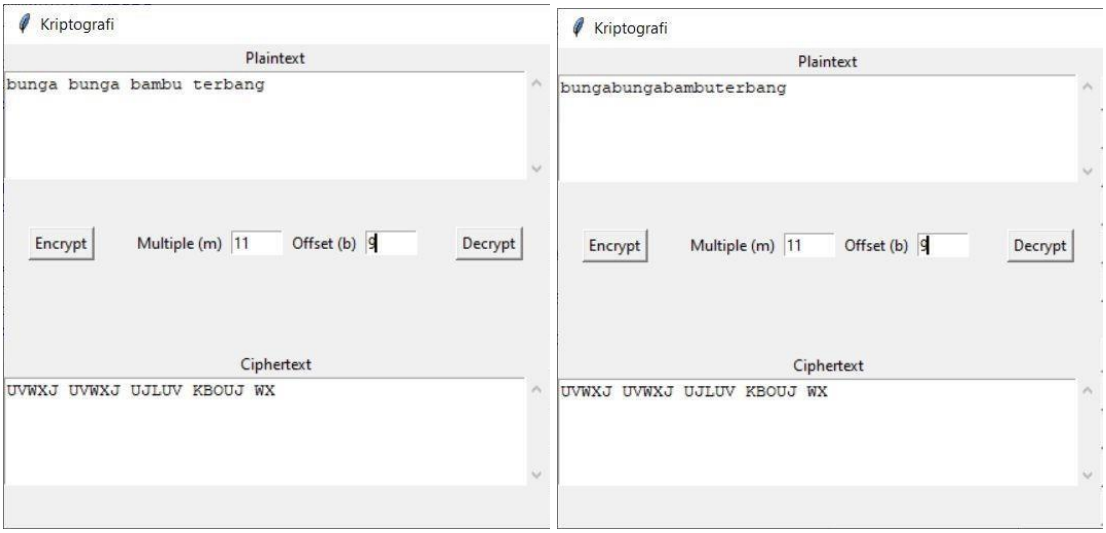
Pengujian Affine Cipher

Pada pengujian pertama, digunakan plaintext “ibu mencuci baju di pasar” dengan m =3 dan b = 10 yang dienkripsi dengan Affine Cipher kemudian didekripsi lagi.



Gambar 2.5. Pengujian Affine Cipher 1

Selanjutnya, digunakan input plaintext “bunga bunga bambu terbang” dengan m = 11dan b = 9 yang memberikan hasil sebagai berikut.



Gambar 2.6 Pengujian Affine Cipher 2

Dari hasil yang diperoleh, terlihat bahwa proses enkripsi-dekripsi yang terjadi berjalan dengan baik, hanya saja terjadi kehilangan informasi pada teks selain alfabet (seperti spasi). Hal ini sudah sesuai dengan spesifikasi yang diharapkan.

4. Keterangan Keberhasilan Program

No	Spek	Berhasil (V)	Kurang berhasil (V)	Keterangan
1	Vigenere Cipher	V		
2	Enigma Cipher	V		
3	Auto-Key Vigenere Cipher	V		
4	Extended Vigenere Cipher		V	Terdapat 32 karakter ASCII yang tidak dapat ditampilkan, membuat proses dekripsi rentan salah (dapat ditingkatkan melalui penyimpanan enkripsi dengan *file of bytes*), untuk file binary ekstensi hanya .bin
5	Playfair Cipher	V		
6	Bonus: Affine Cipher	V		