

**Janardan Bhagat Shikshan Prasarak Sanstha’s**

**CHANGU KANA THAKUR**

**Arts, Commerce, Science College**

**New Panvel (Autonomous)**

**A PROJECT REPORT ON:**

**“CRYPTOGRAPHY”**

**SUBMITED TO:**

**“UNIVERSITY OF MUMBAI”**

**BY**

**Mr.Aditya Vishwas Gangdhar**

**Under The Guidance Of**

**Prof.Mrs.Gauri Deshpande**

**2019-2020**

**ACKNOWLEDGEMENT**

Presentation, inspiration and motivation have always played a key role in the success of any venture.

I express my sincere thanks to **Prof.(Dr.)V.D.Barhate, Principal,** Changu Kana Thakur Arts, Science and Commerce College, New Panvel.

I pay my deep sense of gratitude to **Prof.Mrs.Pratibha Jadhav (HOD)** of Computer Science Department for their able guidance and support in completing my project.

I would also like to thank our college guide **Prof. Mrs.Gauri Deshpande** whose guidance was like driving force behind the success of my project.

**INDEX**

|  |  |  |
| --- | --- | --- |
| **SR. No.** | **Contents** | **Page No.** |
| **1** | Introduction |  |
| **2** | Requirement Specifications |  |
| **3** | System Design |  |
|  | * Activity Diagram |  |
|  | * Sequence Diagram |  |
| **4** | System implementation(coding) |  |
| **5** | Results |  |
| **6** | Conclusion & Future Scope |  |
| **7** | Plagiarism |  |
| **8** | References |  |

**INRODUCTION**

Cryptography or cryptology is the practice and study of secure communication including third parties, called adversaries. cryptography is about constructing and analyzing protocols that prevent third parties or the public from reading private messages various aspects in information security such as data confidentiality, data integrity, authentication, and non-repudiation are central to modern cryptography

Modern cryptography exists at the intersection of the disciplines of mathematics, computer science, electrical engineering, communication sciences and physics. Cryptography applications include e-commerce, chip-based payment cards, digital currencies, computer passwords and military communications.

**Software and Hardware Requirements:**

**Python\* versions:** 2.7.X, 3.6.X

**Processors:**

Intel® Core™ i5 processor 4300M at 2.60 GHz or 2.59 GHz (1 socket, 2 cores, 2 threads per core), 8 GB of DRAM

Intel® Xeon® processor E5-2698 v3 at 2.30 GHz (2 sockets, 16 cores each, 1 thread per core), 64 GB of DRAM

Intel® Xeon Phi™ processor 7210 at 1.30 GHz (1 socket, 64 cores, 4 threads per core)

**Disk space:** 2 to 3 GB

**Operating systems:** Windows® 10, macOS\*, and Linux

**System Design details and implementation**

**Components of a Cryptosystem** -

The various components of a basic cryptosystem are as follows –

* **Plaintext** -

Plain text is a term used in cryptography that refers to a message before encryption or after decryption. That is, it is a message in a form that is easily readable by humans.

* **Encryption Algorithm** -

It is a mathematical process that produces a ciphertext for any given plaintext and encryption key. It takes plaintext and an encryption key as input and produces a ciphertext.

* **Ciphertext** -

Cipher is an algorithm that is applied to normal text to get the ciphertext. It is the unreadable output of an encryption algorithm. The term "digit" is sometimes used as an alternative term for ciphertext. Ciphertext is not understandable until it is converted to plain text using a key.

* **Encryption Algorithm** -   
  It is a mathematical process, which produces a unique plain text for any given ciphertext and decryption key. It is a cryptographic algorithm that accepts encrypted text and a decryption key as input and generates a simple text. The decoding algorithm essentially reverses the encryption algorithm and is therefore closely related to it.
* **Encryption Key** -

It is a value that is known to the sender. The sender inputs the encryption key into the encryption algorithm along with the plaintext in order to compute the ciphertext.

* **Decryption Key** -

It is a value that is known to the receiver. The decryption key is related to the encryption key, but is not always identical to it. The receiver inputs the decryption key into the decryption algorithm along with the ciphertext in order to compute the plaintext.

**Activity Diagram**

Select Algorithm

Select Message

Decrypt(d)

Encrypt(e)

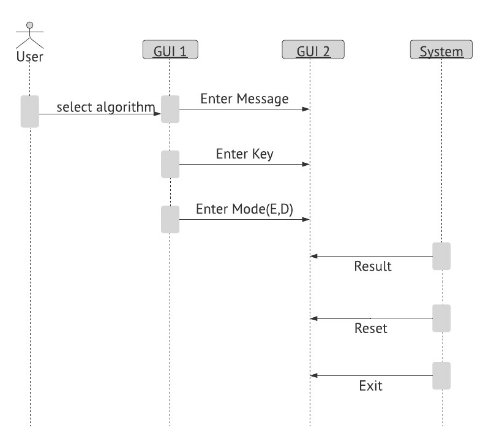
Enter Message

Enter Key

Result

Exit

**Sequence Diagram**



**SYSTEM CODING**

**Coding**

# import tkinter module

from tkinter import \*

from tkinter.font import Font

#import other required modules

import time

import datetime

import math

#creating root object

root = Tk()

root.configure(background='gray12')

root.geometry("800x800")

root.title("cryptography")

font1 = Font(family="Times New Roman",size=50,weight="bold",slant="italic",underline=1)

head = Label(root,text = "Cryptography",bg="gray12",font=font1,fg = "dodger blue", bd = 8, anchor='w')

head.pack(pady=10)

display = Label(root, font = ('Times New Roman', 30, 'bold','italic'),

text = "Substitution Cipher",bg="gray12",

fg = "cyan", bd = 8, anchor='w')

display.pack(pady=10)

def vigenere(): # new window definition

newwin = Toplevel(root)

#defining size of window

newwin.geometry("1200x5000")

newwin.configure(background='gray12')

#placing title of window

newwin.title("vigenere cipher")

#frames

Tops = Frame(newwin,width = 1600, relief = SUNKEN)

Tops.pack(side = TOP)

a1 = Frame(newwin,bg="gray12",width = 800, height = 700, relief = SUNKEN)

a1.pack(side = LEFT)

Msg = StringVar()

key = StringVar()

mode = StringVar()

Result = StringVar()

# ==============================================

# TIME

# ==============================================

localtime = time.asctime(time.localtime(time.time()))

lblInfo = Label(Tops, font = ('Times New Roman', 50, 'bold'),

text = "Vigenère cipher",bg="gray12",

fg = "cyan", bd = 10, anchor='w')

lblInfo.grid(row = 0, column = 0)

lblInfo = Label(Tops, font=('Times New Roman', 20, 'bold'),

text = localtime, fg = "gray12",bg="white" ,

bd = 10, anchor = 'w')

lblInfo.grid(row = 1, column = 0)

#Labels and TextBox

lb1Msg = Label(a1,font=('Times New Roman',16,'bold'),bg="gray12",fg="cyan",

text="MESSAGE:",bd=16,anchor="w")

lb1Msg.grid(row=1,column=0)

txtMsg = Entry(a1, font = ('Times New Roman', 16, 'bold'),

textvariable = Msg, bd = 10, insertwidth = 4,

bg = "light cyan", justify = 'right')

txtMsg.grid(row = 1, column = 1)

lblkey = Label(a1, font = ('Times New Roman', 16, 'bold'),bg="gray12",

text = "KEY:",fg="cyan", bd = 16, anchor = "w")

lblkey.grid(row = 2, column = 0)

txtkey = Entry(a1, font = ('Times New Roman', 16, 'bold'),

textvariable = key, bd = 10, insertwidth = 4,

bg = "light cyan", justify = 'right')

txtkey.grid(row = 2, column = 1)

lblmode = Label(a1, font = ('Times New Roman', 16, 'bold'),

text = "MODE(e for encrypt, d for decrypt):",bg="gray12",

fg="cyan", bd = 16, anchor = "w")

lblmode.grid(row = 3, column = 0)

txtmode = Entry(a1, font = ('Times New Roman', 16, 'bold'),

textvariable = mode, bd = 10, insertwidth = 4,

bg = "light cyan", justify = 'right')

txtmode.grid(row = 3, column = 1)

lblRes = Label(a1, font = ('Times New Roman', 16, 'bold'),bg="gray12",

text = "Result:",fg="cyan", bd = 16, anchor = "w")

lblRes.grid(row = 4, column = 0)

txtRes = Entry(a1, font = ('Times New Roman', 16, 'bold'),

textvariable = Result, bd = 10, insertwidth = 4,

bg = "light cyan", justify = 'right')

txtRes.grid(row = 4, column = 1)

def Ref():

print("Message= ", (Msg.get()))

clear = Msg.get()

k = key.get()

m = mode.get()

if (m == 'e'):

Result.set(encode(k, clear))

else:

Result.set(decode(k, clear))

# exit function

def qExit():

root.destroy()

# Function to reset the window

def Reset():

Msg.set("")

key.set("")

mode.set("")

Result.set("")

# Show message button

btnTotal = Button(a1, padx = 16, pady = 8, bd = 10, fg = "cyan",

font = ('Times New Roman', 16, 'bold'), width = 10,

text = "Show Message", bg = "light cyan",

command = Ref).grid(row = 9, column = 0)

# Reset button

btnReset = Button(a1, padx = 16, pady = 8, bd = 10,

fg = "cyan", font = ('Times New Roman', 16, 'bold'),

width = 10, text = "Reset", bg = "light cyan",

command = Reset).grid(row = 9, column = 1)

# Exit button

btnExit = Button(a1, padx = 16, pady = 8, bd = 10,

fg = "cyan", font = ('Times New Roman', 16, 'bold'),

width = 10, text = "Exit", bg = "light cyan",

command = qExit).grid(row =9, column = 2)

# Vigenère cipher

import base64

# Function to encode

def encode(key, clear):

enc = []

for i in range(len(clear)):

key\_c = key[i % len(key)]

enc\_c = chr((ord(clear[i]) +

ord(key\_c)) % 256)

enc.append(enc\_c)

return base64.urlsafe\_b64encode("".join(enc).encode()).decode()

# Function to decode

def decode(key, enc):

dec = []

enc = base64.urlsafe\_b64decode(enc).decode()

for i in range(len(enc)):

key\_c = key[i % len(key)]

dec\_c = chr((256 + ord(enc[i]) -

ord(key\_c)) % 256)

dec.append(dec\_c)

return "".join(dec)

def caeser():

newwin1 = Toplevel(root)

#defining size of window

newwin1.geometry("1200x5000")

newwin1.configure(background='gray12')

#placing title of window

newwin1.title("Ceaser Cipher")

Tops1 = Frame(newwin1,width = 1600, relief = SUNKEN)

Tops1.pack(side = TOP)

a2 = Frame(newwin1,bg="gray12",width = 800, height = 700, relief = SUNKEN)

a2.pack(side = LEFT)

# ==============================================

# TIME

# ==============================================

localtime = time.asctime(time.localtime(time.time()))

lblInfo = Label(Tops1, font = ('Times New Roman', 50, 'bold'),

text = "Caeser cipher",bg="gray12",

fg = "cyan", bd = 10, anchor='w')

lblInfo.grid(row = 0, column = 0)

lblInfo = Label(Tops1, font=('Times New Roman', 20, 'bold'),

text = localtime, fg = "gray12",bg="white" ,

bd = 10, anchor = 'w')

lblInfo.grid(row = 1, column = 0)

Msg = StringVar()

key = IntVar()

mode = StringVar()

Result = StringVar()

#Labels and TextBox

lb1Msg = Label(a2,font=('Times New Roman',16,'bold'),bg="gray12",fg="cyan",

text="MESSAGE:",bd=16,anchor="w")

lb1Msg.grid(row=1,column=0)

txtMsg = Entry(a2, font = ('Times New Roman', 16, 'bold'),

textvariable = Msg, bd = 10, insertwidth = 4,

bg = "light cyan", justify = 'right')

txtMsg.grid(row = 1, column = 1)

lblkey = Label(a2, font = ('Times New Roman', 16, 'bold'),bg="gray12",

text = "KEY:",fg="cyan", bd = 16, anchor = "w")

lblkey.grid(row = 2, column = 0)

txtkey = Entry(a2, font = ('Times New Roman', 16, 'bold'),

textvariable = key, bd = 10, insertwidth = 4,

bg = "light cyan", justify = 'right')

txtkey.grid(row = 2, column = 1)

lblmode = Label(a2, font = ('Times New Roman', 16, 'bold'),

text = "MODE(E for encrypt, D for decrypt):",bg="gray12",

fg="cyan", bd = 16, anchor = "w")

lblmode.grid(row = 3, column = 0)

txtmode = Entry(a2, font = ('Times New Roman', 16, 'bold'),

textvariable = mode, bd = 10, insertwidth = 4,

bg = "light cyan", justify = 'right')

txtmode.grid(row = 3, column = 1)

lblRes = Label(a2, font = ('Times New Roman', 16, 'bold'),bg="gray12",

text = "Result:",fg="cyan", bd = 16, anchor = "w")

lblRes.grid(row = 4, column = 0)

txtRes = Entry(a2, font = ('Times New Roman', 16, 'bold'),

textvariable = Result, bd = 10, insertwidth = 4,

bg = "light cyan", justify = 'right')

txtRes.grid(row = 4, column = 1)

def Ref():

print("Message= ", (Msg.get()))

text = Msg.get()

k = key.get()

print(k)

m = mode.get()

if (m == 'E' or m == 'e'):

s = 4

Result.set(encdec(k, text))

elif(m == 'D' or m == 'd'):

s = 26-k

Result.set(encdec(k, text))

else:

print("wrong choice")

print("Shift key=",str(s))

# exit function

def qExit():

root.destroy()

# Function to reset the window

def Reset():

Msg.set("")

key.set("")

mode.set("")

Result.set("")

# Show message button

btnTotal = Button(a2, padx = 16, pady = 8, bd = 10, fg = "cyan",

font = ('Times New Roman', 16, 'bold'), width = 10,

text = "Show Message", bg = "light cyan",

command = Ref).grid(row = 9, column = 0)

# Reset button

btnReset = Button(a2, padx = 16, pady = 8, bd = 10,

fg = "cyan", font = ('Times New Roman', 16, 'bold'),

width = 10, text = "Reset", bg = "light cyan",

command = Reset).grid(row = 9, column = 1)

# Exit button

btnExit = Button(a2, padx = 16, pady = 8, bd = 10,

fg = "cyan", font = ('Times New Roman', 16, 'bold'),

width = 10, text = "Exit", bg = "light cyan",

command = qExit).grid(row =9, column = 2)

def encdec(s, text):

result = " "

for i in range(len(text)):

char=text[i]

print(char)

if(char.isupper()):

result=chr((ord(char)+s-65)%26+65)

print(result)

else:

result+=chr((ord(char)+s-97)%26+97)

return result

def vernam():

newwin3 = Toplevel(root)

#defining size of window

newwin3.geometry("1200x5000")

newwin3.configure(background='gray12')

#placing title of window

newwin3.title("Vernam Cipher")

Tops3 = Frame(newwin3,width = 1600, relief = SUNKEN)

Tops3.pack(side = TOP)

a4 = Frame(newwin3,bg="gray12",width = 800, height = 700, relief = SUNKEN)

a4.pack(side = LEFT)

# ==============================================

# TIME

# ==============================================

localtime = time.asctime(time.localtime(time.time()))

lblInfo = Label(Tops3, font = ('Times New Roman', 50, 'bold'),

text = "Vernam cipher",bg="gray12",

fg = "cyan", bd = 10, anchor='w')

lblInfo.grid(row = 0, column = 0)

lblInfo = Label(Tops3, font=('Times New Roman', 20, 'bold'),

text = localtime, fg = "gray12",bg="white" ,

bd = 10, anchor = 'w')

lblInfo.grid(row = 1, column = 0)

Msg = StringVar()

key = StringVar()

mode = StringVar()

Result = StringVar()

#Labels and TextBox

lb1Msg = Label(a4,font=('Times New Roman',16,'bold'),bg="gray12",fg="cyan",

text="MESSAGE:",bd=16,anchor="w")

lb1Msg.grid(row=1,column=0)

txtMsg = Entry(a4, font = ('Times New Roman', 16, 'bold'),

textvariable = Msg, bd = 10, insertwidth = 4,

bg = "light cyan", justify = 'right')

txtMsg.grid(row = 1, column = 1)

lblkey = Label(a4, font = ('Times New Roman', 16, 'bold'),bg="gray12",

text = "KEY:",fg="cyan", bd = 16, anchor = "w")

lblkey.grid(row = 2, column = 0)

txtkey = Entry(a4, font = ('Times New Roman', 16, 'bold'),

textvariable = key, bd = 10, insertwidth = 4,

bg = "light cyan", justify = 'right')

txtkey.grid(row = 2, column = 1)

lblmode = Label(a4, font = ('Times New Roman', 16, 'bold'),

text = "MODE(e for encrypt, d for decrypt):",bg="gray12",

fg="cyan", bd = 16, anchor = "w")

lblmode.grid(row = 3, column = 0)

txtmode = Entry(a4, font = ('Times New Roman', 16, 'bold'),

textvariable = mode, bd = 10, insertwidth = 4,

bg = "light cyan", justify = 'right')

txtmode.grid(row = 3, column = 1)

lblRes = Label(a4, font = ('Times New Roman', 16, 'bold'),bg="gray12",

text = "Result:",fg="cyan", bd = 16, anchor = "w")

lblRes.grid(row = 4, column = 0)

txtRes = Entry(a4, font = ('Times New Roman', 16, 'bold'),

textvariable = Result, bd = 10, insertwidth = 4,

bg = "light cyan", justify = 'right')

txtRes.grid(row = 4, column = 1)

def Ref():

print("message=",(Msg.get()))

msg1=Msg.get()

k=key.get()

m1=mode.get()

if (m1=='e'):

Result.set(encrypt1(k, msg1))

elif(m1=='d'):

Result.set(decrypt1(k, msg1))

else:

print("wrong choice")

#exit function

def qExit():

root.destroy()

# Function to reset the window

def Reset():

Msg.set("")

key.set("")

mode.set("")

Result.set("")

# Show message button

btnTotal = Button(a4, padx = 16, pady = 8, bd = 10, fg = "cyan",

font = ('Times New Roman', 16, 'bold'), width = 10,

text = "Show Message", bg = "light cyan",

command = Ref).grid(row = 9, column = 0)

# Reset button

btnReset = Button(a4, padx = 16, pady = 8, bd = 10,

fg = "cyan", font = ('Times New Roman', 16, 'bold'),

width = 10, text = "Reset", bg = "light cyan",

command = Reset).grid(row = 9, column = 1)

# Exit button

btnExit = Button(a4, padx = 16, pady = 8, bd = 10,

fg = "cyan", font = ('Times New Roman', 16, 'bold'),

width = 10, text = "Exit", bg = "light cyan",

command = qExit).grid(row =9, column = 2)

def encrypt1(key,msg1):

msg1=str(msg1)

m1=msg1.upper().replace("","")

encrypt=""

for i in range(len(m1)):

letter=ord(m1[i])-65

s=key.upper().replace("","")

l=ord(s[i])-65

letter=(letter+i)%25

letter+=65

encrypt=encrypt+chr(letter)

return encrypt

def decrypt1(key,msg1):

msg1=str(msg1)

m1=msg1.upper().replace("","")

decrypt=""

for i in range(len(m1)):

letter=ord(m1[i])-65

s=key.upper().replace("","")

l=ord(s[i])-65

letter=(letter-i)%25

if letter<=0:

letter=letter+25

letter+=65

else:

letter+=65

decrypt=decrypt+chr(letter)

return decrypt

def rail():

newwin4 = Toplevel(root)

#defining size of window

newwin4.geometry("1200x5000")

newwin4.configure(background='gray12')

#placing title of window

newwin4.title("Rail Fence Cipher")

Tops4 = Frame(newwin4,width = 1600, relief = SUNKEN)

Tops4.pack(side = TOP)

a5 = Frame(newwin4,bg="gray12",width = 800, height = 700, relief = SUNKEN)

a5.pack(side = LEFT)

# ==============================================

# TIME

# ==============================================

localtime = time.asctime(time.localtime(time.time()))

lblInfo = Label(Tops4, font = ('Times New Roman', 50, 'bold'),

text = "Rail Fence Cipher",bg="gray12",

fg = "cyan", bd = 10, anchor='w')

lblInfo.grid(row = 0, column = 0)

lblInfo = Label(Tops4, font=('Times New Roman', 20, 'bold'),

text = localtime, fg = "gray12",bg="white" ,

bd = 10, anchor = 'w')

lblInfo.grid(row = 1, column = 0)

Msg = StringVar()

key = IntVar()

mode = StringVar()

Result = StringVar()

#Labels and TextBox

lb1Msg = Label(a5,font=('Times New Roman',16,'bold'),bg="gray12",fg="cyan",

text="MESSAGE:",bd=16,anchor="w")

lb1Msg.grid(row=1,column=0)

txtMsg = Entry(a5, font = ('Times New Roman', 16, 'bold'),

textvariable = Msg, bd = 10, insertwidth = 4,

bg = "light cyan", justify = 'right')

txtMsg.grid(row = 1, column = 1)

lblkey = Label(a5, font = ('Times New Roman', 16, 'bold'),bg="gray12",

text = "Layer:",fg="cyan", bd = 16, anchor = "w")

lblkey.grid(row = 2, column = 0)

txtkey = Entry(a5, font = ('Times New Roman', 16, 'bold'),

textvariable = key, bd = 10, insertwidth = 4,

bg = "light cyan", justify = 'right')

txtkey.grid(row = 2, column = 1)

lblmode = Label(a5, font = ('Times New Roman', 16, 'bold'),

text = "Encrypt:",bg="gray12",

fg="cyan", bd = 16, anchor = "w")

lblmode.grid(row = 3, column = 0)

txtmode = Entry(a5, font = ('Times New Roman', 16, 'bold'),

textvariable = mode, bd = 10, insertwidth = 4,

bg = "light cyan", justify = 'right')

txtmode.grid(row = 3, column = 1)

lblRes = Label(a5, font = ('Times New Roman', 16, 'bold'),bg="gray12",

text = "Result:",fg="cyan", bd = 16, anchor = "w")

lblRes.grid(row = 4, column = 0)

txtRes = Entry(a5, font = ('Times New Roman', 16, 'bold'),

textvariable = Result, bd = 10, insertwidth = 4,

bg = "light cyan", justify = 'right')

txtRes.grid(row = 4, column = 1)

def Ref():

print("Message=:",(Msg.get()))

plain\_text = Msg.get()

layer = key.get()

m = mode.get()

if (m == 'e' or m == 'E'):

Result.set(encrypt(layer,plain\_text))

else:

print("wrong choice")

#exit function

def qExit():

root.destroy()

# Function to reset the window

def Reset():

Msg.set("")

key.set("")

mode.set("")

Result.set("")

def encrypt(layers,plain\_text):

plain\_text=plain\_text.replace("","")

plain\_text=plain\_text.upper()

rail=[""]\*layers

layer=0

for character in plain\_text:

rail[layer]+=character

if layer>=layers-1:

layer=0

else:

layer+=1

cipher="".join(rail)

return cipher

# Show message button

btnTotal = Button(a5, padx = 16, pady = 8, bd = 10, fg = "cyan",

font = ('Times New Roman', 16, 'bold'), width = 10,

text = "Show Message", bg = "light cyan",

command = Ref).grid(row = 9, column = 0)

# Reset button

btnReset = Button(a5, padx = 16, pady = 8, bd = 10,

fg = "cyan", font = ('Times New Roman', 16, 'bold'),

width = 10, text = "Reset", bg = "light cyan",

command = Reset).grid(row = 9, column = 1)

# Exit button

btnExit = Button(a5, padx = 16, pady = 8, bd = 10,

fg = "cyan", font = ('Times New Roman', 16, 'bold'),

width = 10, text = "Exit", bg = "light cyan",

command = qExit).grid(row =9, column = 2)

def columnar():

newwin5 = Toplevel(root)

#defining size of window

newwin5.geometry("1200x5000")

newwin5.configure(background='gray12')

#placing title of window

newwin5.title("Simple Columnar Cipher")

Tops5 = Frame(newwin5,width = 1600, relief = SUNKEN)

Tops5.pack(side = TOP)

a6 = Frame(newwin5,bg="gray12",width = 800, height = 700, relief = SUNKEN)

a6.pack(side = LEFT)

# ==============================================

# TIME

# ==============================================

localtime = time.asctime(time.localtime(time.time()))

lblInfo = Label(Tops5, font = ('Times New Roman', 50, 'bold'),

text = "Simple Columnar cipher",bg="gray12",

fg = "cyan", bd = 10, anchor='w')

lblInfo.grid(row = 0, column = 0)

lblInfo = Label(Tops5, font=('Times New Roman', 20, 'bold'),

text = localtime, fg = "gray12",bg="white" ,

bd = 10, anchor = 'w')

lblInfo.grid(row = 1, column = 0)

Msg = StringVar()

key = StringVar()

mode = StringVar()

Result = StringVar()

#Labels and TextBox

lb1Msg = Label(a6,font=('Times New Roman',16,'bold'),bg="gray12",fg="cyan",

text="MESSAGE:",bd=16,anchor="w")

lb1Msg.grid(row=1,column=0)

txtMsg = Entry(a6, font = ('Times New Roman', 16, 'bold'),

textvariable = Msg, bd = 10, insertwidth = 4,

bg = "light cyan", justify = 'right')

txtMsg.grid(row = 1, column = 1)

lblkey = Label(a6, font = ('Times New Roman', 16, 'bold'),bg="gray12",

text = "KEY:",fg="cyan", bd = 16, anchor = "w")

lblkey.grid(row = 2, column = 0)

txtkey = Entry(a6, font = ('Times New Roman', 16, 'bold'),

textvariable = key, bd = 10, insertwidth = 4,

bg = "light cyan", justify = 'right')

txtkey.grid(row = 2, column = 1)

lblmode = Label(a6, font = ('Times New Roman', 16, 'bold'),

text = "MODE(e for encrypt, d for decrypt):",bg="gray12",

fg="cyan", bd = 16, anchor = "w")

lblmode.grid(row = 3, column = 0)

txtmode = Entry(a6, font = ('Times New Roman', 16, 'bold'),

textvariable = mode, bd = 10, insertwidth = 4,

bg = "light cyan", justify = 'right')

txtmode.grid(row = 3, column = 1)

lblRes = Label(a6, font = ('Times New Roman', 16, 'bold'),bg="gray12",

text = "Result:",fg="cyan", bd = 16, anchor = "w")

lblRes.grid(row = 4, column = 0)

txtRes = Entry(a6, font = ('Times New Roman', 16, 'bold'),

textvariable = Result, bd = 10, insertwidth = 4,

bg = "light cyan", justify = 'right')

txtRes.grid(row = 4, column = 1)

def Ref():

print("Message= ", (Msg.get()))

msg = Msg.get()

k = key.get()

m = mode.get()

if (m == 'e' or m=='E'):

Result.set(encrypt(k, msg))

else:

Result.set(decrypt(k, msg))

#exit function

def qExit():

root.destroy()

# Function to reset the window

def Reset():

Msg.set("")

key.set("")

mode.set("")

Result.set("")

# Show message button

btnTotal = Button(a6, padx = 16, pady = 8, bd = 10, fg = "cyan",

font = ('Times New Roman', 16, 'bold'), width = 10,

text = "Show Message", bg = "light cyan",

command = Ref).grid(row = 9, column = 0)

# Reset button

btnReset = Button(a6, padx = 16, pady = 8, bd = 10,

fg = "cyan", font = ('Times New Roman', 16, 'bold'),

width = 10, text = "Reset", bg = "light cyan",

command = Reset).grid(row = 9, column = 1)

# Exit button

btnExit = Button(a6, padx = 16, pady = 8, bd = 10,

fg = "cyan", font = ('Times New Roman', 16, 'bold'),

width = 10, text = "Exit", bg = "light cyan",

command = qExit).grid(row =9, column = 2)

def encrypt(key,msg):

result=""

# track key indices

k\_indx = 0

msg\_len = float(len(msg))

print(msg\_len)

msg\_lst = list(msg)

print(msg\_lst)

key\_lst = sorted(list(key))

print(key\_lst)

# calculate column of the matrix

col = len(key)

print(col)

# calculate maximum row of the matrix

row = int(math.ceil(msg\_len / col))

print(row)

# add the padding character '\_' in empty

# the empty cell of the matix

fill\_null = int((row \* col) - msg\_len)

msg\_lst.extend('\_' \* fill\_null)

print(fill\_null)

# create Matrix and insert message and

# padding characters row-wise

matrix = [msg\_lst[i: i + col]

for i in range(0, len(msg\_lst), col)]

print(matrix)

# read matrix column-wise using key

for \_ in range(col):

#print(\_)

curr\_idx = key.index(key\_lst[k\_indx])

result += ''.join([row[curr\_idx]

for row in matrix])

k\_indx += 1

return result

def decrypt(key,msg):

cipher=""

# track key indices

k\_indx = 0

# track msg indices

msg\_indx = 0

msg\_len = float(len(msg))

msg\_lst = list(msg)

# calculate column of the matrix

col = len(key)

# calculate maximum row of the matrix

row = int(math.ceil(msg\_len / col))

# convert key into list and sort

# alphabetically so we can access

# each character by its alphabetical position.

key\_lst = sorted(list(key))

# create an empty matrix to

# store deciphered message

dec\_cipher = []

for \_ in range(row):

dec\_cipher += [[None] \* col]

# Arrange the matrix column wise according

# to permutation order by adding into new matrix

for \_ in range(col):

curr\_idx = key.index(key\_lst[k\_indx])

for j in range(row):

dec\_cipher[j][curr\_idx] = msg\_lst[msg\_indx]

msg\_indx += 1

k\_indx += 1

# convert decrypted msg matrix into a string

try:

msg = ''.join(sum(dec\_cipher, []))

except TypeError:

raise TypeError("This program cannot",

"handle repeating words.")

null\_count = msg.count('\_')

if null\_count > 0:

return msg[: -null\_count]

return msg

button1 =Button(root, padx=6,pady=2,bd=5,fg="light cyan",

font = ('Times New Roman',17,'bold','italic'),width=17,

text ="Vigenere Cipher",bg="gray12", command =vigenere) #command linked

button1.pack(pady=5)

button2 =Button(root, padx=6,pady=2,bd=5,fg="light cyan",

font = ('Times New Roman',17,'bold','italic'),width=17,

text ="Caeser Cipher",bg="gray12",command=caeser) #command linked

button2.pack(pady=5)

button4 =Button(root, padx=6,pady=2,bd=5,fg="light cyan",

font = ('Times New Roman',17,'bold','italic'),width=17,

text ="vernam cipher",bg="gray12",command=vernam) #command linked

button4.pack(pady=5)

display1 = Label(root, font = ('Times New Roman', 30, 'bold','italic'),

text = "Transposition Cipher",bg="gray12",

fg = "cyan", bd = 8, anchor='w')

display1.pack(pady=10)

button5 =Button(root, padx=6,pady=2,bd=5,fg="light cyan",

font = ('Times New Roman',17,'bold','italic'),width=17,

text ="Rail Fence Cipher",bg="gray12", command =rail) #command linked

button5.pack(pady=5)

button6 =Button(root, padx=6,pady=2,bd=5,fg="light cyan",

font = ('Times New Roman',17,'bold','italic'),width=17,

text ="Simple Columnar Cipher",bg="gray12",command=columnar) #command linked

button6.pack(pady=5)

# keep window alive

root.mainloop()

**Result:**







****

****

****

**Conclusion and Future Scope:**

**Conclusion:**

Internet security is improving. The growing use of the Internet for commercial purposes improves the technology implemented to protect financial transactions. The extension of core technologies to protect multicast communications is possible and can be expected to be implemented as multicast is distributed.

This will help Hackers to encrypt or decrypt the passwords.

**Future scope:**

It helps to provide secrecy in transmission, secrecy in storage, authentication of identity, credentialing system, digital signatures, and secure multiparty computation.

More Cryptography techniques will further be added.

**Plagiarism Report**

**REFERENCES**

**Reference:**

* Programming Python, 4th Edition by Mark Lutz Publisher: O'Reilly Media, Inc. Release Date: December 2010
* <https://www.geeksforgeeks.org/>
* [https://stackoverflow.com](https://stackoverflow.com/)
* <https://www.tutorialspoint.com/cryptography/index.htm>