K. S. Institute of Technology

Department of Computer Science & Engineering

Mini Project Report

Academic Year:	2019-20(Even)		
Course Name(Code):	Cryptography Network Security & Cyber Law (17CS61)		
Mini Project Title:	Implementation of One Time Pad (aka Vernam Cipher) Algorithm		
Group No:	07		
Group Members:	Divya Yashaswi Kanney	1KS17CS023	
Semester/Section	VI/A		

1. Description of the mini project:

- The aim of the project is to implement the Vernam Cipher algorithm to encrypt plain text into cipher text with the help of a key and to decrypt cipher text into plain text with the help of a key.
- **Front end:** Tkinter Python
- **Back end:** Python

- **Working of the mini project:** The mini project contains a python file named 'OneTimePad.py'.

- **Prerequisites:**
  1. Python (Download)
  2. Tkinter module

- **Steps to run:**
  1. Launch the terminal or the command prompt
  2. Change the directory to the directory containing the files of the mini project
  3. Type **python OneTimePad.py**
  4. Select 'ENCRYPTION' or 'DECRYPTION' to navigate and perform respective operation

2. Description of the algorithm:

In cryptography, the one-time pad (OTP) is an encryption technique that cannot be cracked, but requires the use of a one-time pre-shared key the same size as, or longer than, the message being sent. In this technique, a plaintext is paired with a random secret key (also referred to as a one-time pad). Then, each bit or character of the plaintext is encrypted by combining it with the corresponding bit or character from the pad using modular addition. If the key is (1) truly random, (2) at least as long as the plaintext, (3) never reused in whole or in part, and (4) kept completely secret, then the resulting ciphertext will be impossible to decrypt or break.It has also been proven that any cipher with the property of perfect secrecy must use keys with effectively the same requirements as OTP keys.[3] Digital versions of one-time pad ciphers have been used by nations for critical diplomatic and military communication, but the problems of secure key distribution have made them impractical for most applications.

Example:

```
      H       E       L       L       O  message
   7 (H)   4 (E)  11 (L)  11 (L)  14 (O) message
+ 23 (X)  12 (M)   2 (C)  10 (K)  11 (L) key
= 30      16      13      21      25     message + key
=  4 (E)  16 (Q)  13 (N)  21 (V)  25 (Z) (message + key) mod 26
      E       Q       N       V       Z  → ciphertext
```

```
      E       Q       N       V       Z  ciphertext
    4 (E)  16 (Q)  13 (N)  21 (V)  25 (Z) ciphertext
-  23 (X)  12 (M)   2 (C)  10 (K)  11 (L) key
= -19       4      11      11      14     ciphertext - key
=   7 (H)   4 (E)  11 (L)  11 (L)  14 (O) ciphertext - key (mod 26)
      H       E       L       L       O  → message
```

3. Algorithm:

Encryption:

1. Assign a number to each character of the plain-text and the key according to alphabetical order.

2. Add both the number (Corresponding plain-text character number and Key character number).

3. Subtract the number from 26 if the added number is greater than 26, if it isn't then leave it.

Decryption:

For the Decryption apply the just reverse process of encryption.

$$\text{ciphertext} = \text{plaintext} \oplus \text{key}$$

$$\text{ciphertext} \oplus \text{key} = (\text{plaintext} \oplus \text{key}) \oplus \text{key} = \text{plaintext} \oplus (\text{key} \oplus \text{key}) = \text{plaintext}$$

4. Implementation of algorithm(code):

```
from tkinter import *
import random
import string
from typing import List

one_time_pad = Tk()
one_time_pad.title("One Time Pad")
one_time_pad.minsize(500, 500)

firstLabel = Label(one_time_pad, text="One Time Pad")
firstLabel.grid(row=0, column=250)

msg_label = Label(one_time_pad, text="Enter your Message:")
msg_label.grid(row=30, column=150)
msg = Entry(one_time_pad)
msg.grid(row=30, column=250, columnspan=2)

dict_count_uppercase = {
    0: 'A',
    1: 'B',
    2: 'C',
    3: 'D',
    4: 'E',
    5: 'F',
    6: 'G',
    7: 'H',
    8: 'I',
    9: 'J',
    10: 'K',
    11: 'L',
    12: 'M',
    13: 'N',
    14: 'O',
    15: 'P',
    16: 'Q',
    17: 'R',
    18: 'S',
    19: 'T',
    20: 'U',
    21: 'V',
    22: 'W',
    23: 'X',
```

4. Implementation of algorithm(code):
Continued...

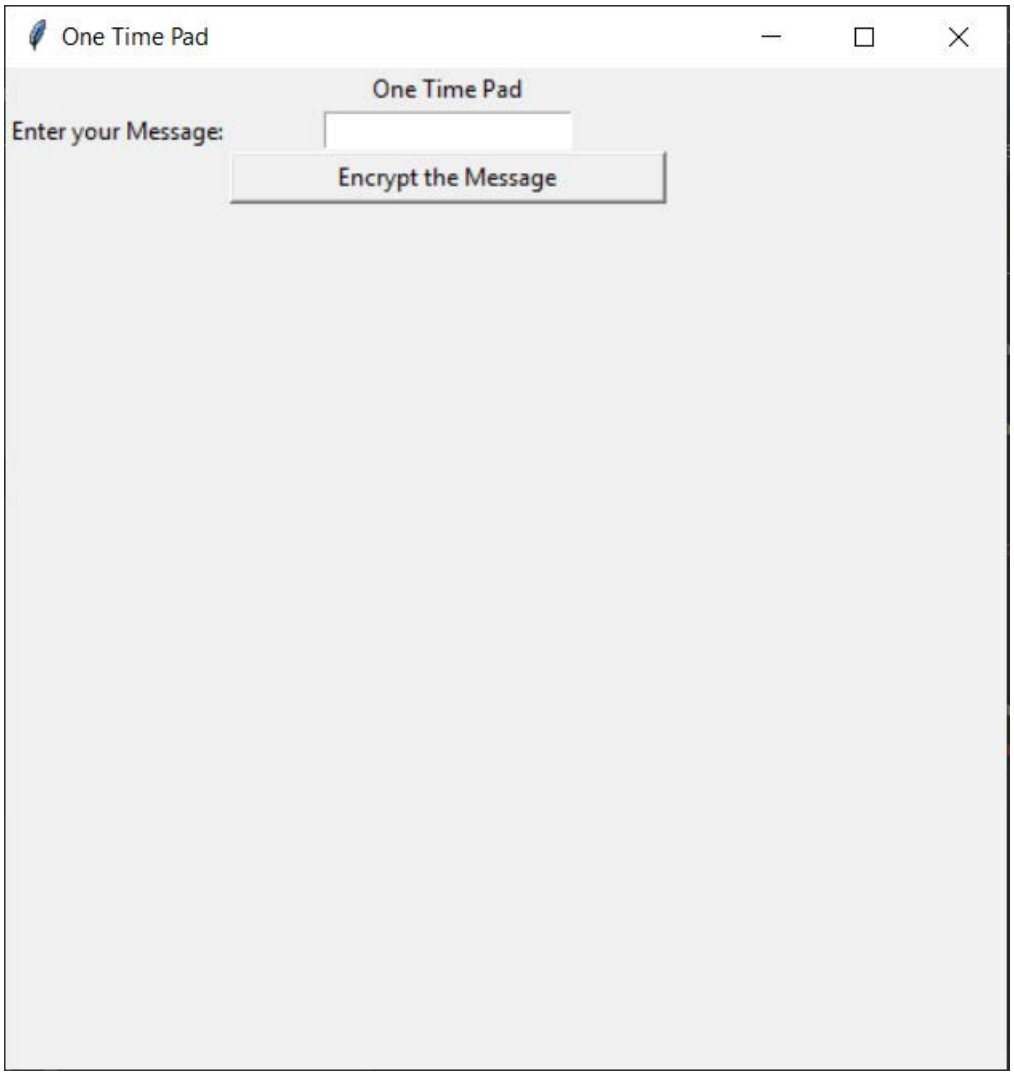
```
    24: 'Y',
    25: 'Z',
    'A': 0,
    'B': 1,
    'C': 2,
    'D': 3,
    'E': 4,
    'F': 5,
    'G': 6,
    'H': 7,
    'I': 8,
    'J': 9,
    'K': 10,
    'L': 11,
    'M': 12,
    'N': 13,
    'O': 14,
    'P': 15,
    'Q': 16,
    'R': 17,
    'S': 18,
    'T': 19,
    'U': 20,
    'V': 21,
    'W': 22,
    'X': 23,
    'Y': 24,
    'Z': 25
 }

 dict_count_lowercase = {
    0: 'a',
    1: 'b',
    2: 'c',
    3: 'd',
    4: 'e',
    5: 'f',
    6: 'g',
    7: 'h',
    8: 'i',
    9: 'j',
    10: 'k',
```

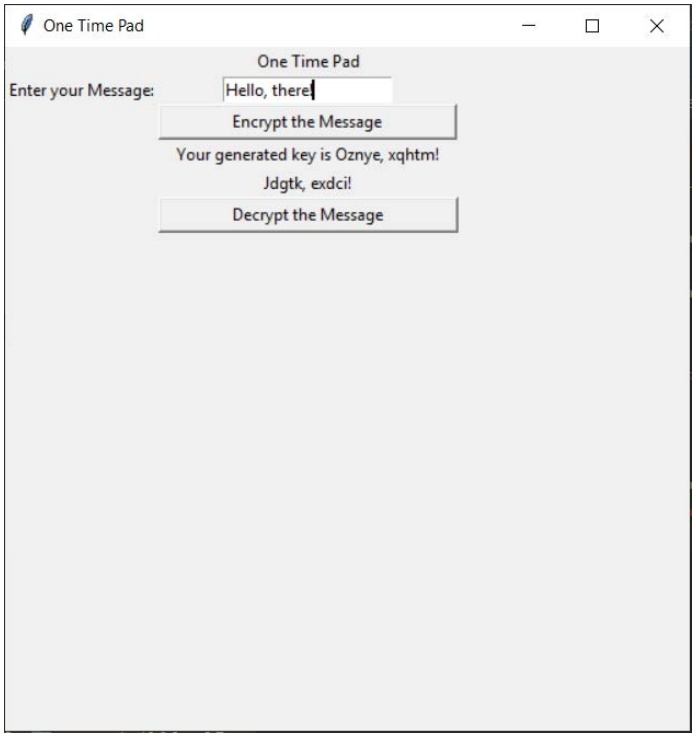
```
    12: 'm',
    13: 'n',
    14: 'o',
    15: 'p',
    16: 'q',
    17: 'r',
    18: 's',
    19: 't',
    20: 'u',
    21: 'v',
    22: 'w',
    23: 'x',
    24: 'y',
    25: 'z',
    'a': 0,
    'b': 1,
    'c': 2,
    'd': 3,
    'e': 4,
    'f': 5,
    'g': 6,
    'h': 7,
    'i': 8,
    'j': 9,
    'k': 10,
    'l': 11,
    'm': 12,
    'n': 13,
    'o': 14,
    'p': 15,
    'q': 16,
    'r': 17,
    's': 18,
    't': 19,
    'u': 20,
    'v': 21,
    'w': 22,
    'x': 23,
    'y': 24,
    'z': 25
}


def generate_key(message):
    lower_letters = string.ascii_lowercase
    upper_letters = string.ascii_uppercase
    key = ''
    for i in message:
        if i.isalpha():
            if i.isupper():
                key += random.choice(upper_letters)
            else:
                key += random.choice(lower_letters)
        else:
            key += i
    return key
```

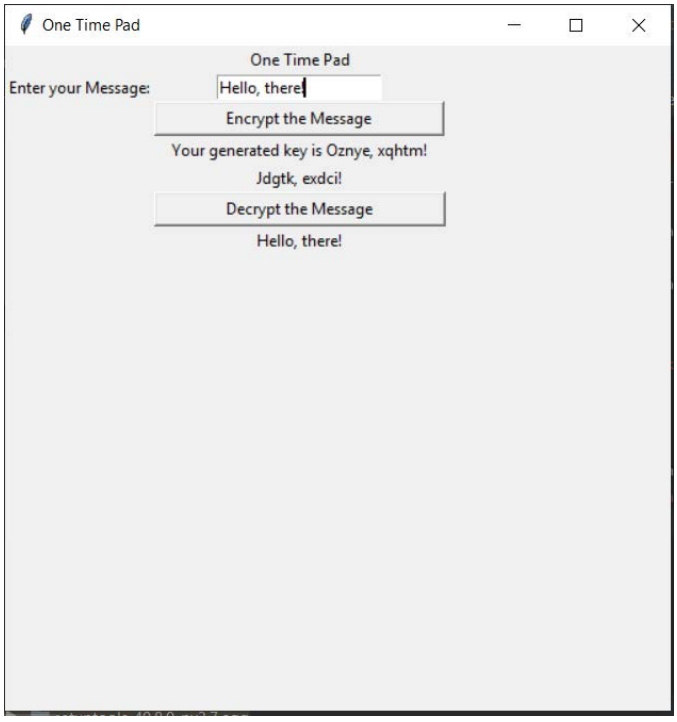
```
def xor(msg_coded, key_coded):
    cipher_coded: List[int] = []
    for i, j in zip(msg_coded, key_coded):
        # x = int(i, 2)
        # y = int(j, 2)
        if type(i) == str and type(j) == str:
            cipher_coded.append(i)
            continue
        z = int(i ^ j)
        cipher_coded.append(z)
    return cipher_coded


def encrypt():
    message = msg.get()
    key = generate_key(message)
    key_label = Label(one_time_pad, text="Your generated key is " + key)
    key_label.grid(column=250)
    global dict_count_lowercase
    global dict_count_uppercase
    msg_coded, key_coded = [], []
    for i, j in zip(message, key):
        if i.isalpha() and j.isalpha():
            if i.islower() and j.islower():
                msg_coded.append(int(dict_count_lowercase.get(i)))
                key_coded.append(int(dict_count_lowercase.get(j)))
            else:
                msg_coded.append(int(dict_count_uppercase.get(i)))
                key_coded.append(int(dict_count_uppercase.get(j)))
        else:
            msg_coded.append(i)
            key_coded.append(j)
    cipher_coded = xor(msg_coded, key_coded)
    cipher_coded_mod = []
    for i in cipher_coded:
        if type(i) == int:
            cipher_coded_mod.append(i % 26)
        else:
            cipher_coded_mod.append(i)
    cipher_text = ""
    for i, j in zip(cipher_coded_mod, message):
        if j.islower():
            cipher_text += dict_count_lowercase.get(i)
        elif j.isupper():
            cipher_text += dict_count_uppercase.get(i)
        else:
            cipher_text += i
    label = Label(one_time_pad, text=cipher_text)
    label.grid(column=250)
    decrypt_button = Button(one_time_pad, text="Decrypt the Message", padx=50,
                            command=lambda: decrypt(key_coded, cipher_coded, cipher_text))
    decrypt_button.grid(column=250)
```

```
def decrypt(key_coded, cipher_coded, cipher_text):
    plain_coded = xor(cipher_coded, key_coded)
    plain_text = ""
    for i, j in zip(plain_coded, cipher_text):
        if j.islower():
            plain_text += dict_count_lowercase.get(i)
        elif j.isupper():
            plain_text += dict_count_uppercase.get(i)
        else:
            plain_text += i
    label = Label(one_time_pad, text=plain_text)
    label.grid(column=250)


encrypt_button = Button(one_time_pad, text="Encrypt the Message", padx=50, command=encrypt)
encrypt_button.grid(row=250, column=250)
one_time_pad.mainloop()
```

5.Snapshots of output:







6. Conclusion:

The project works successfully to encrypt plain texts to cipher texts and decrypt the cipher text to plain text using One Time Pad algorithm.

7. References:

- https://www.geeksforgeeks.org/vernam-cipher-in-cryptography/
- https://www.geeksforgeeks.org/python-tkinter-grid_location-and-grid_size-method/
- https://www.delftstack.com/howto/python-tkinter/how-to-pass-arguments-to-tkinter-button-command/
- https://www.youtube.com/watch?v=YXPyB4XeYLA&t=2800s
- https://en.wikipedia.org/wiki/One-time_pad