Pandit Deendayal Energy University

School of Technology

Information Security Lab

B.Tech-Computer Science & Engineering (Sem-V)

PATEL VEDANT H.

19BCP138

DIVISION – 2

Lab 7 Assignment

❖ **Aim:** Study and Implement program for Hill Cipher.

❖ **Introduction:**

Hill Cipher in cryptography was invented and developed in 1929 by Lester S. Hill, a renowned American mathematician. Hill Cipher is Digraphic but can expand to multiply any size of letters, adding more complexity and reliability for better use. Hill cipher is a polygraphic substitution cipher based on linear algebra. Each letter is represented by a number modulo 26. Often the simple scheme A = 0, B = 1, …, Z = 25 is used, but this is not an essential feature of the cipher.

Encryption: To encrypt a message, each block of n letters (considered as an n-component vector) is multiplied by an invertible n × n matrix, against modulus 26.

Decryption: To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption. The matrix used for encryption is the cipher key, and it should be chosen randomly from the set of invertible n × n matrices (modulo 26).

❖ **Program:**

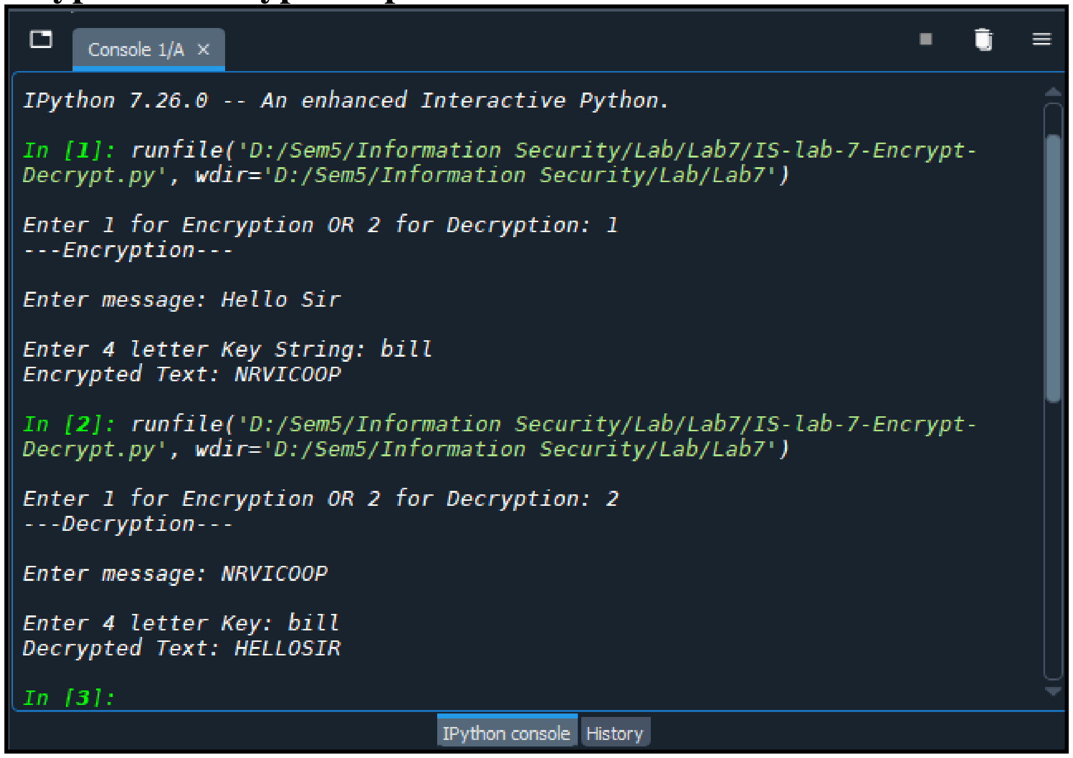
➢ **Encrypt and Decrypt Python Program:**

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 30 23:05:53 2021

@author: vedpa
"""

import sys
import numpy as np

def cipher_encryption():
    msg = input("Enter message: ").upper()
    msg = msg.replace(" ", "")

    # if message length is odd number, append 0 at the end
    len_chk = 0
    if len(msg) % 2 != 0:
        msg += "0"
        len_chk = 1

    # msg to matrices
    row = 2
    col = int(len(msg)/2)
    msg2d = np.zeros((row, col), dtype=int)

    itr1 = 0
    itr2 = 0
    for i in range(len(msg)):
        if i % 2 == 0:
            msg2d[0][itr1] = int(ord(msg[i])-65)
            itr1 += 1
        else:
            msg2d[1][itr2] = int(ord(msg[i])-65)
            itr2 += 1
    # for

    key = input("Enter 4 letter Key String: ").upper()
    key = key.replace(" ", "")

    # key to 2x2
    key2d = np.zeros((2, 2), dtype=int)
    itr3 = 0
    for i in range(2):
        for j in range(2):
            key2d[i][j] = ord(key[itr3])-65
            itr3 += 1
    # checking validity of the key
    # finding determinant
    deter = key2d[0][0] * key2d[1][1] - key2d[0][1] * key2d[1][0]
    deter = deter % 26

    # finding multiplicative inverse
    mul_inv = -1
    for i in range(26):
        temp_inv = deter * i
        if temp_inv % 26 == 1:
            mul_inv = i
            break
        else:
            continue
    # for

    if mul_inv == -1:
        print("Invalid key")
        sys.exit()
    # if

    encryp_text = ""
    itr_count = int(len(msg)/2)
    if len_chk == 0:
        for i in range(itr_count):
            temp1 = msg2d[0][i] * key2d[0][0] + msg2d[1][i] * key2d[0][1]
            encryp_text += chr((temp1 % 26) + 65)
            temp2 = msg2d[0][i] * key2d[1][0] + msg2d[1][i] * key2d[1][1]
            encryp_text += chr((temp2 % 26) + 65)
        # for
    else:
        for i in range(itr_count-1):
            temp1 = msg2d[0][i] * key2d[0][0] + msg2d[1][i] * key2d[0][1]
            encryp_text += chr((temp1 % 26) + 65)
            temp2 = msg2d[0][i] * key2d[1][0] + msg2d[1][i] * key2d[1][1]
            encryp_text += chr((temp2 % 26) + 65)
        # for
    # if else
    print("Encrypted Text: {}".format(encryp_text))

def cipher_decryption():
    msg = input("Enter message: ").upper()
    msg = msg.replace(" ", "")

    # if message length is odd number, append 0 at the end
    len_chk = 0
    if len(msg) % 2 != 0:
        msg += "0"
        len_chk = 1
```

```
# msg to matrices
row = 2
col = int(len(msg) / 2)
msg2d = np.zeros((row, col), dtype=int)

itr1 = 0
itr2 = 0
for i in range(len(msg)):
    if i % 2 == 0:
        msg2d[0][itr1] = int(ord(msg[i]) - 65)
        itr1 += 1
    else:
        msg2d[1][itr2] = int(ord(msg[i]) - 65)
        itr2 += 1
# for

key = input("Enter 4 letter Key: ").upper()
key = key.replace(" ", "")

# key to 2x2
key2d = np.zeros((2, 2), dtype=int)
itr3 = 0
for i in range(2):
    for j in range(2):
        key2d[i][j] = ord(key[itr3]) - 65
        itr3 += 1

# finding determinant
deter = key2d[0][0] * key2d[1][1] - key2d[0][1] * key2d[1][0]
deter = deter % 26

# finding multiplicative inverse
mul_inv = -1
for i in range(26):
    temp_inv = deter * i
    if temp_inv % 26 == 1:
        mul_inv = i
        break
    else:
        continue
# for

# Adjacent matrix
# swapping
key2d[0][0], key2d[1][1] = key2d[1][1], key2d[0][0]

# changing signs
key2d[0][1] *= -1
key2d[1][0] *= -1
```

```
    key2d[0][1] = key2d[0][1] % 26
    key2d[1][0] = key2d[1][0] % 26

    # multiplying multiplicative inverse with adjacent matrix
    for i in range(2):
        for j in range(2):
            key2d[i][j] *= mul_inv

    # modulo
    for i in range(2):
        for j in range(2):
            key2d[i][j] = key2d[i][j] % 26

    # cipher to plain
    decryp_text = ""
    itr_count = int(len(msg) / 2)
    if len_chk == 0:
        for i in range(itr_count):
            temp1 = msg2d[0][i] * key2d[0][0] + msg2d[1][i] * key2d[0][1]
            decryp_text += chr((temp1 % 26) + 65)
            temp2 = msg2d[0][i] * key2d[1][0] + msg2d[1][i] * key2d[1][1]
            decryp_text += chr((temp2 % 26) + 65)
            # for
    else:
        for i in range(itr_count - 1):
            temp1 = msg2d[0][i] * key2d[0][0] + msg2d[1][i] * key2d[0][1]
            decryp_text += chr((temp1 % 26) + 65)
            temp2 = msg2d[0][i] * key2d[1][0] + msg2d[1][i] * key2d[1][1]
            decryp_text += chr((temp2 % 26) + 65)
            # for
    # if else

    print("Decrypted Text: {}".format(decryp_text))

def main():
    choice = int(input("Enter 1 for Encryption OR 2 for Decryption: "))
    if choice == 1:
        print("---Encryption---")
        cipher_encryption()
    elif choice == 2:
        print("---Decryption---")
        cipher_decryption()
    else:
        print("Invalid Choice")

if __name__ == "__main__":
    main()
```

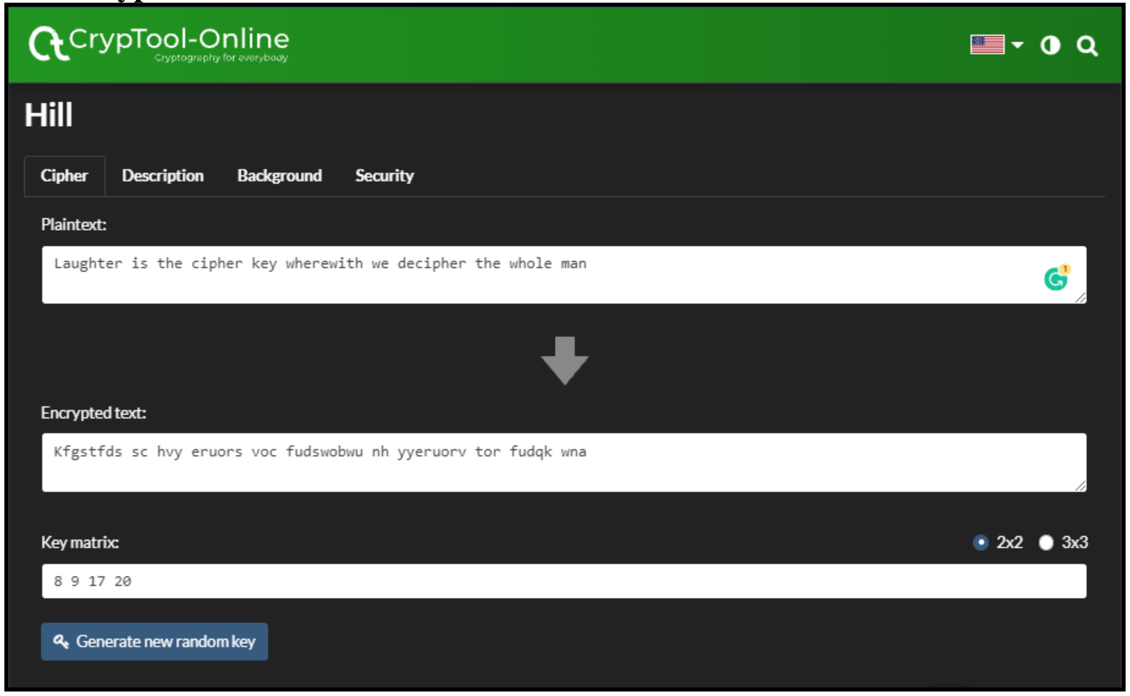
➢ Encrypt and Decrypt Output:



```
IPython 7.26.0 -- An enhanced Interactive Python.

In [1]: runfile('D:/Sem5/Information Security/Lab/Lab7/IS-lab-7-Encrypt-
Decrypt.py', wdir='D:/Sem5/Information Security/Lab/Lab7')

Enter 1 for Encryption OR 2 for Decryption: 1
---Encryption---

Enter message: Hello Sir

Enter 4 letter Key String: bill
Encrypted Text: NRVICOOP

In [2]: runfile('D:/Sem5/Information Security/Lab/Lab7/IS-lab-7-Encrypt-
Decrypt.py', wdir='D:/Sem5/Information Security/Lab/Lab7')

Enter 1 for Encryption OR 2 for Decryption: 2
---Decryption---

Enter message: NRVICOOP

Enter 4 letter Key: bill
Decrypted Text: HELLOSIR

In [3]:
```

➢ CrypTool Online Encrypt and Decrypt Output:

- **Encryption:**



Plaintext:

```
Laughter is the cipher key wherewith we decipher the whole man
```

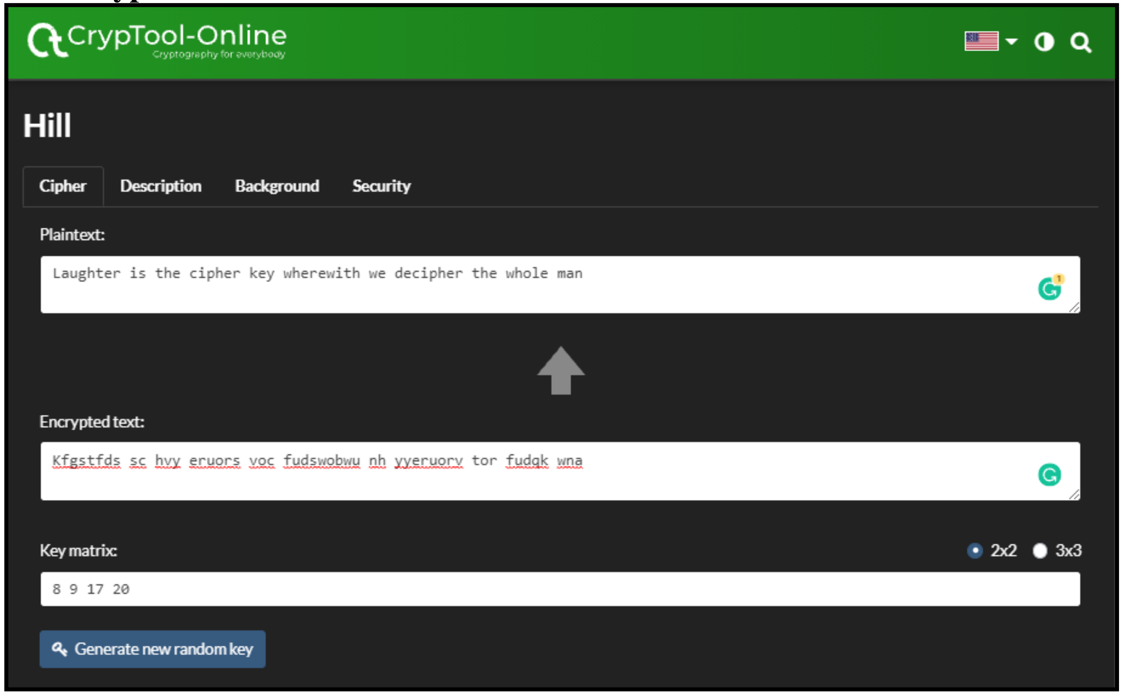
Encrypted text:

```
Kfgstfds sc hvy eruors voc fudswobwu nh yyeruorv tor fudqk wna
```

Key matrix: [✓ 2x2] 3x3

```
8 9 17 20
```

- **Decryption:**



Plaintext:

Laughter is the cipher key wherewith we decipher the whole man

Encrypted text:

Kfgstfds sc hvy eruors voc fudswobwu nh yyeruory tor fudqk wna

Key matrix: [✓ 2x2] 3x3

8 9 17 20

❖ Cryptanalysis:

When attempting to crack a Hill cipher, frequency analysis will be practically useless, especially as the size of the key block increases. For very long cipher texts, frequency analysis may be useful when applied to bigrams (for a 2 by 2 hill cipher), but for short cipher texts, this will not be practical. For a guide on how to break Hill ciphers with a crib, see Cryptanalysis of the Hill Cipher. The basic Hill cipher is vulnerable to a known-plaintext attack, however, (if you know the plaintext and corresponding cipher text the key can be recovered) because it is completely linear. An opponent who intercepts several plaintext/cipher text character pairs can set up a linear system that can (usually) be easily solved; if it happens that this system is indeterminate, it is only necessary to add a few more plaintexts/cipher text pairs. The known cipher text attack is the best one to try when trying to break the hill cipher, if no sections of the plaintext are known, guesses can be made.

❖ Applications:

The protection of valuable data in a multimedia system is one of today’s most challenging tasks for information technology. Hill cipher belongs in the PolyGram substitution case of ciphers and gives an inexpensive, easy, and robust tool for multimedia security.

❖ Reference:

- ➢ http://practicalcryptography.com/ciphers/hill-cipher/
- ➢ https://tutorialspoint.dev/computer-science/advanced-computer-subjects/hill-cipher
- ➢ https://www.jigsawacademy.com/blogs/cyber-security/hill-cipher/