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 Hill 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 Hill 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 'HillCipher.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 HillCipher.py**
  4. Select 'ENCRYPTION' or 'DECRYPTION' to navigate and perform respective operation

2. Description of the algorithm:

In classical cryptography, the Hill cipher is a polygraphic substitution cipher based on linear algebra. Invented by Lester S. Hill in 1929, it was the first polygraphic cipher in which it was practical (though barely) to operate on more than three symbols at once. The following discussion assumes an elementary knowledge of matrices.

Each letter is represented by a number modulo 26. Though this is not an essential feature of the cipher, this simple scheme is often used:

Letter A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Number 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

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. 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). The cipher can, of course, be adapted to an alphabet with any number of letters; all arithmetic just needs to be done modulo the number of letters instead of modulo 26.

3. Algorithm:

$$\begin{pmatrix} C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} K_{11} K_{12} K_{13} \\ K_{21} K_{22} K_{23} \\ K_{31} K_{32} K_{33} \end{pmatrix} \begin{pmatrix} P_1 \\ P_2 \\ P_3 \end{pmatrix} \bmod 26$$

$$C = KP \bmod 26$$

$$P = K^{-1} C \bmod 26 = KK^{-1} P = P$$

Example:

Encryption

We have to encrypt the message 'ACT' (n=3).The key is 'GYBNQKURP' which can be written as the nxn matrix:

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix}$$

The message 'ACT' is written as vector:

$$\begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix}$$

The enciphered vector is given as:

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix} = \begin{bmatrix} 67 \\ 222 \\ 319 \end{bmatrix} \equiv \begin{bmatrix} 15 \\ 14 \\ 7 \end{bmatrix} \pmod{26}$$

which corresponds to ciphertext of 'POH'

Decryption

To decrypt the message, we turn the ciphertext back into a vector, then simply multiply by the inverse matrix of the key matrix (IFKVIVVMI in letters).The inverse of the matrix used in the previous example is:

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix}^{-1} \equiv \begin{bmatrix} 8 & 5 & 10 \\ 21 & 8 & 21 \\ 21 & 12 & 8 \end{bmatrix} \pmod{26}$$

For the previous Ciphertext 'POH':

$$\begin{bmatrix} 8 & 5 & 10 \\ 21 & 8 & 21 \\ 21 & 12 & 8 \end{bmatrix} \begin{bmatrix} 15 \\ 14 \\ 7 \end{bmatrix} \equiv \begin{bmatrix} 260 \\ 574 \\ 539 \end{bmatrix} \equiv \begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix} \pmod{26}$$

which gives us back 'ACT'.

4. Implementation of algorithm(code):

.
.

. 2

. 2 1

. 2 2

. 1

. 2

1
2
3
4

1
11
12
13
14
1
1
1
1
1
2
21

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

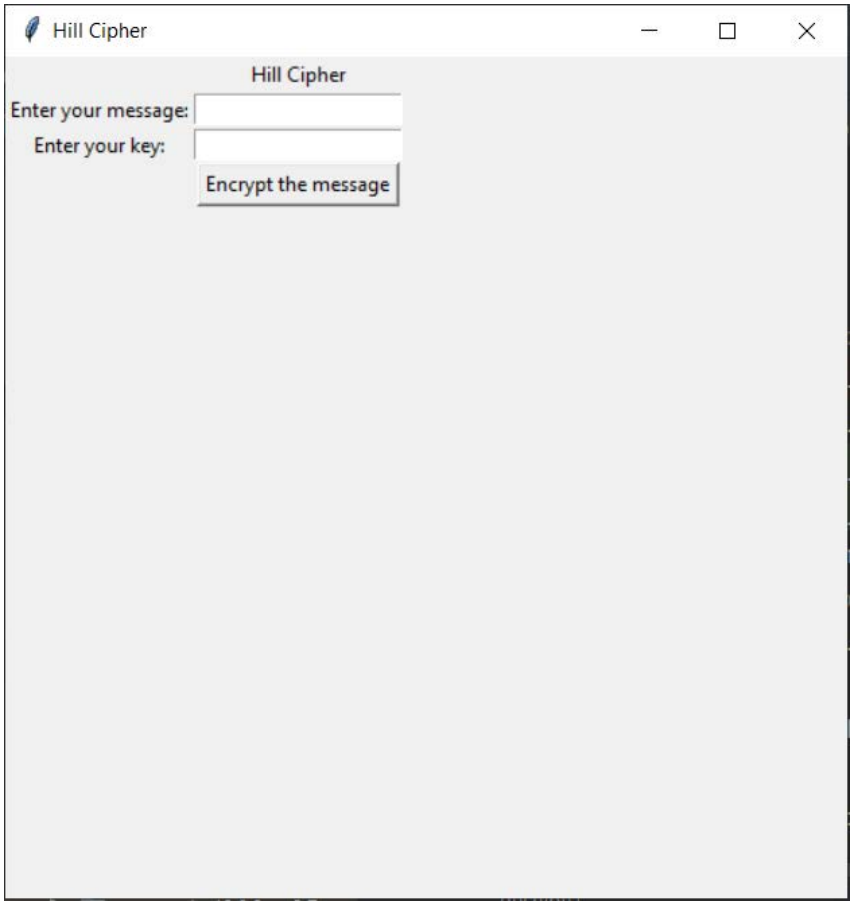
```
22
23
24
2

    1
    2
    3
    4

    1
   11
    12
    13
    14
   1
    1
   1
   1
   1
    2
   21
    22
   23
   24
   2

1
2
3
4
```

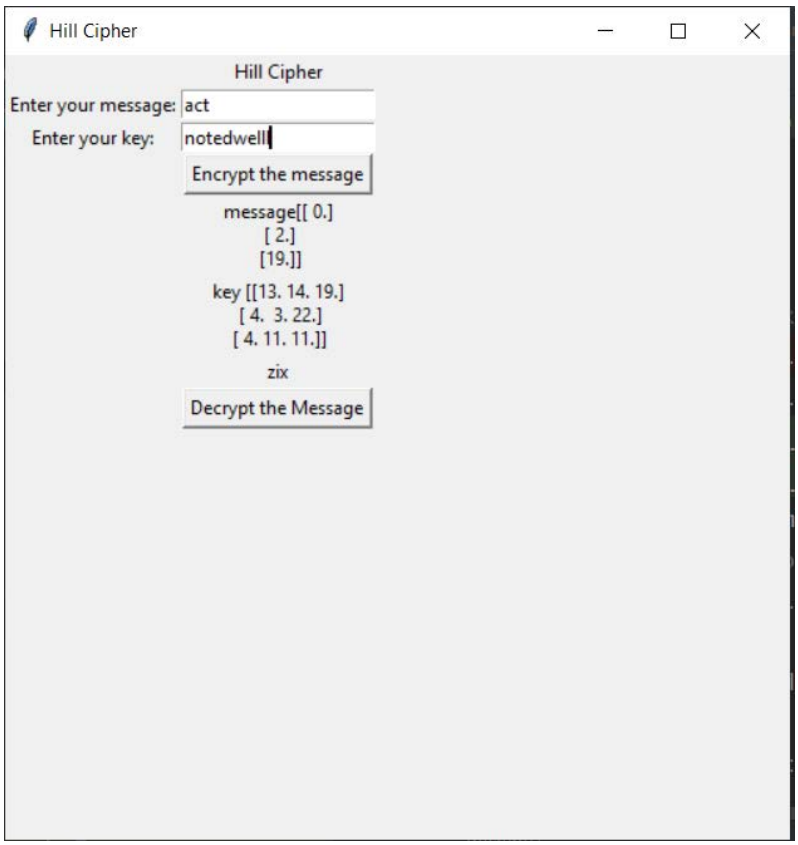
```
    9: 'j',
    10: 'k',
    11: 'l',
    12: 'm',
    13: 'n',
    14: 'o',
    15: 'p',
    16: 'q',
    17: 'r',
    18: 's',
    19: 't',
    20: 'u',
    21: 'v',
    22: 'w',
    23: 'x',
    24: 'y',
    25: 'z',
    'a': 0,
    'b': 1,
    'c': 2,
    'd': 3,
    'e': 4,
    'f': 5,
    'g': 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_code(key, n):
    key_matrix = np.zeros((n, n), dtype=str)
    key_coded = np.zeros((n, n))
    k = 0
    for i in range(n):
        for j in range(n):
            if k == len(key):
                k = 0
            key_matrix[i, j] = key[k]
```

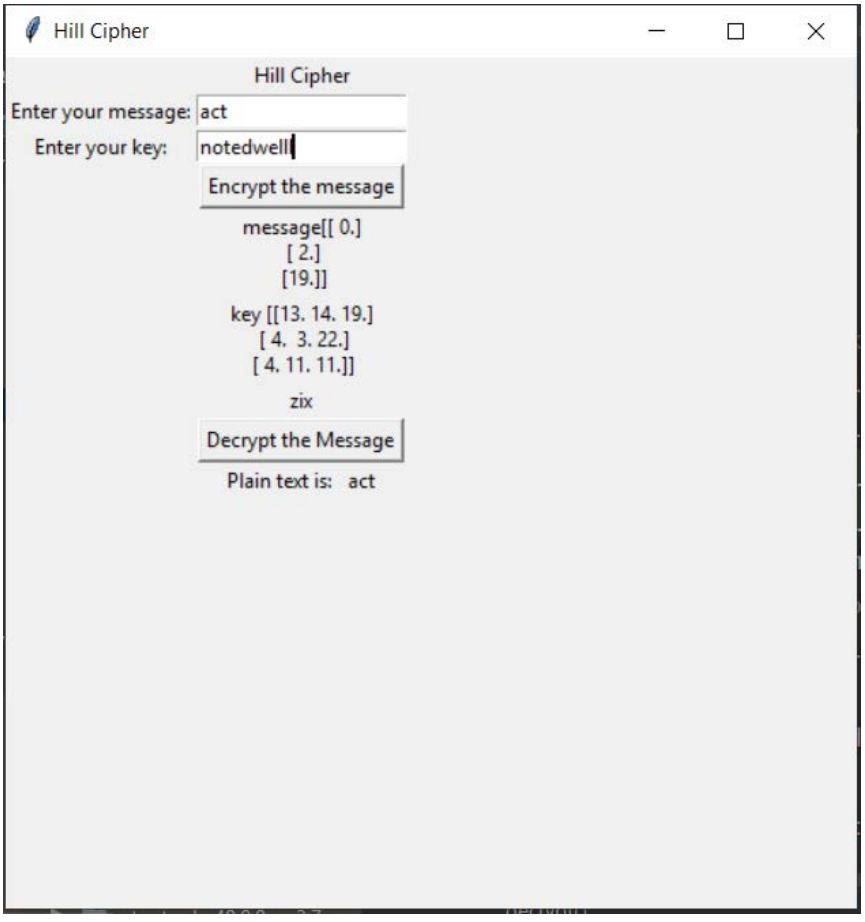
```
            key_coded[i, j] = dict_count_lowercase.get(key[k])

            k += 1
    generated_key_label = Label(hill_cipher, text="key " + str(key_coded))
    generated_key_label.grid(column=250)
    return key_matrix, key_coded


def generate_msg_code(message):
    msg_coded = np.zeros((len(message), 1))
    for i in range(len(message)):
        if message[i].isalpha():
            if message[i].isupper():
                msg_coded[i] = dict_count_uppercase.get(message[i])
            else:
                msg_coded[i] = dict_count_lowercase.get(message[i])
        else:
            msg_coded[i] = message[i]

    label = Label(hill_cipher, text="message" + str(msg_coded))
    label.grid(column=250)
    return msg_coded


def encrypt():
    message = msg.get()
    key = key_entry.get()
    msg_coded = generate_msg_code(message)
    key_matrix, key_coded = generate_key_code(key, len(message))
    det = np.linalg.det(key_coded)
    if det == 0 or det % 13 == 0 or det % 2 == 0:
        error_label = Label(hill_cipher, text="Your key is not valid. Try again.")
        error_label.grid(column=250)
    cipher_coded = key_coded.dot(msg_coded) % 26
    cipher_text = ''
    for i in cipher_coded:
        for j in i:
            cipher_text += dict_count_lowercase.get(j)

    cipher_label = Label(hill_cipher, text=cipher_text)
    cipher_label.grid(column=250)

    decrypt_button = Button(hill_cipher, text="Decrypt the Message",
                            command=lambda: decrypt(cipher_text, cipher_coded, key_matrix, key_coded))
    decrypt_button.grid(column=250)


def get_multiplicative_inverse(key_coded):
    det = np.linalg.det(key_coded)
    det = int(det % 26)
    x = 0
```

```
    for i in range(26):
        if (i * det) % 26 == 1:
            x = i
            break

    return x


def decrypt(cipher_text, cipher_coded, key_matrix, key_coded):
    key_inverse = np.linalg.inv(key_coded)
    key_inverse = np.linalg.det(key_coded) * get_multiplicative_inverse(key_coded) * key_inverse
    key_inverse = key_inverse.astype(int)
    key_inverse = key_inverse % 26
    plain_coded = np.matmul(key_inverse, cipher_coded) % 26  # key_inverse.dot(cipher_coded) % 26
    # plain_coded = np.remainder(plain_coded, module).flatten()
    plain_coded = plain_coded.astype(int)
    plain_text = ''
    for i in plain_coded:
        for j in i:
            plain_text += dict_count_lowercase.get(j)

    plain_label = Label(hill_cipher, text="Plain text is:   " + plain_text)
    plain_label.grid(column=250)


encrypt_button = Button(hill_cipher, text="Encrypt the message", command=encrypt)
encrypt_button.grid(column=250)

hill_cipher.mainloop()
```

5.Snapshots of output:




Hill Cipher
Hill Cipher
Enter your message: act
Enter your key: notedwell
Encrypt the message
message[[0.]
[2.]
[19.]]
key [[13. 14. 19.]
[4. 3. 22.]
[4. 11. 11.]]
zix
Decrypt the Message


Hill Cipher
Hill Cipher
Enter your message: act
Enter your key: notedwell
Encrypt the message
message[[0.]
[2.]
[19.]]
key [[13. 14. 19.]
[4. 3. 22.]
[4. 11. 11.]]
zix
Decrypt the Message
Plain text is: act

6. Conclusion:

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

7. References:

- https://www.geeksforgeeks.org/hill-cipher/
- 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/Hill_cipher