Somaiya Vidyavihar University
K. J. Somaiya College of Engineering
Department of Computer Engineering

Batch:___C3__ Roll No.:_____110_________

Experiment No.

Title: Encryption-Decryption programs using classical cryptography (Playfair cipher, Transposition cipher)

Objective: Implement Playfair cipher and Transposition cipher

Expected Outcome of Experiment:

CO	Outcome
1	Explain various security goals, threats, vulnerabilities and controls

Books/ Journals/ Websites referred:
https://www.geeksforgeeks.org/

https://www.techtarget.com/searchsecurity/definition/cipher

Abstract:-

In cryptology, the discipline concerned with the study of cryptographic algorithms, a cipher is an algorithm for encrypting and decrypting data.

Symmetric key encryption, also called secret key encryption, depends on the use of ciphers, which operate symmetrically. With symmetric encryption algorithms, the same encryption key is applied to data in the same way, whether the objective is to convert plaintext to ciphertext or ciphertext to plaintext. A cipher transforms data by processing the original, plaintext characters or other data into ciphertext. The ciphertext should appear as random data.

Traditionally, ciphers used these two main types of transformation:

Transposition ciphers keep all the original bits of data in a byte but mix their order.

Substitution ciphers replace specific data sequences with other data sequences. For example, one type of substitution would be to transform all bits with a value of 1 to a value of 0, and vice versa.

The data output from either method is called the ciphertext.

Playfair cipher and columnar ciphers are both transposition ciphers.

Related Theory: -

1. **Playfair cipher**

The Playfair cipher was the first practical digraph substitution cipher. The scheme was invented in 1854 by Charles Wheatstone but was named after Lord Playfair who promoted the use of the cipher. In playfair cipher unlike traditional cipher we encrypt a pair of alphabets(digraphs) instead of a single alphabet.

It was used for tactical purposes by British forces in the Second Boer War and in World War I and for the same purpose by the Australians during World War II. This was because Playfair is reasonably fast to use and requires no special equipment.

The Playfair Cipher Encryption Algorithm:

The Algorithm consists of 2 steps:

1. Generate the key Square(5×5):

The key square is a 5×5 grid of alphabets that acts as the key for encrypting the plaintext. Each of the 25 alphabets must be unique and one letter of the alphabet (usually J) is omitted from the table (as the table can hold only 25 alphabets). If the plaintext contains J, then it is replaced by I.

The initial alphabets in the key square are the unique alphabets of the key in the order in which they appear followed by the remaining letters of the alphabet in order.

2. Algorithm to encrypt the plain text:

The plaintext is split into pairs of two letters (digraphs). If there is an odd number of letters, a Z is added to the last letter.

1. Pair cannot be made with same letter. Break the letter in single and add a bogus letter to the previous letter.

2. If the letter is standing alone in the process of pairing, then add an extra bogus letter with the alone letter.

3. If both the letters are in the same column: Take the letter below each one (going back to the top if at the bottom).

4. If both the letters are in the same row: Take the letter to the right of each one (going back to the leftmost if at the rightmost position).

5. If neither of the above rules is true: Form a rectangle with the two letters and take the letters on the horizontal opposite corner of the rectangle.

2. Columnar Cipher

In a transposition cipher, the order of the alphabets is re-arranged to obtain the cipher-text.

The message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order.

Width of the rows and the permutation of the columns are usually defined by a keyword.

For example, the word HACK is of length 4 (so the rows are of length 4), and the permutation is defined by the alphabetical order of the letters in the keyword. In this case, the order would be "3 1 2 4".

Any spare spaces are filled with nulls or left blank or placed by a character (Example: _).

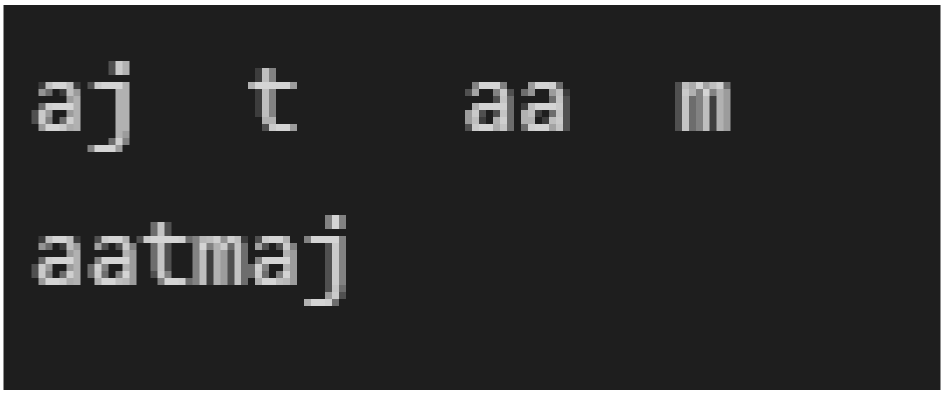
Finally, the message is read off in columns, in the order specified by the keyword.

Implementation Details:

1. **Keyed Transposition cipher**

```
def transposition(key,string):
    length=len(key)
    matrix = make_matrix_from_key(string,length)
    arranged = arranger(key)
    matrix= list(zip(*matrix))
    matrix_cpy = list(matrix) # shallow copy
    for ctr in range(len(arranged)):
        matrix_cpy[arranged[ctr]] = matrix[ctr]
    ans=""
    for i in matrix_cpy:
        for j in i:
          ans+=j
    return ans


def make_matrix_from_key(string,length):
    answer = []
    if(len(string)!=length**2):
        for i in range(length**2-len(string)):
            string+=" "
    for i in range(length):
        answer.append([])
        for j in range(length):
            answer[i].append(string[length*i+j])

    return answer




def arranger(key):

    sorted_list=sorted(key)

    answer = []

    for i in key:

       answer.append(sorted_list.index(i))

    return answer


def decrypt(key,plaintext):

    length = len(key)

    matrix = make_matrix_from_key(plaintext,length,)

    arranged = arranger(key)


    matrix_cpy = list(matrix) # shallow copy

    for ctr in range(len(arranged)):

        matrix_cpy[ctr] = matrix[arranged[ctr]]

    matrix_cpy= list(zip(*matrix_cpy))

    ans=""

    for i in matrix_cpy:

        for j in i:

          ans+=j

    return ans
```

```python
ciphertext=transposition("hack","aatmaj")

print(ciphertext)

plaintext = decrypt("hack",ciphertext)

print(plaintext)
```

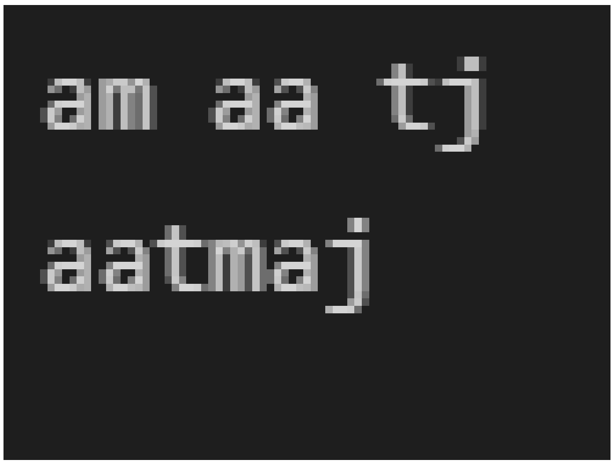


```
aj  t   aa  m
aatmaj
```

2. Keyless transposition cipher

```
def transposition(key,string):

    length=key

    matrix = make_matrix_from_key(string,length)

    arranged = arranger(key)

    matrix= list(zip(*matrix))

    matrix_cpy = list(matrix) # shallow copy

    for ctr in range(len(arranged)):

        matrix_cpy[arranged[ctr]] = matrix[ctr]

    ans=""

    for i in matrix_cpy:

        for j in i:

          ans+=j

    return ans


def make_matrix_from_key(string,length):

    answer = []

    if(len(string)!=length**2):

        for i in range(length**2-len(string)):

            string+=" "

    for i in range(length):

        answer.append([])
        for j in range(length):
            answer[i].append(string[length*i+j])
    return answer


def arranger(keylen):

    return range(keylen)

def decrypt(key,plaintext):
    length = key
    matrix = make_matrix_from_key(plaintext,length,)
    arranged = arranger(key)

    matrix_cpy = list(matrix) # shallow copy
    for ctr in range(len(arranged)):
        matrix_cpy[ctr] = matrix[arranged[ctr]]
    matrix_cpy= list(zip(*matrix_cpy))
    ans=""
    for i in matrix_cpy:
        for j in i:
          ans+=j
    return ans

keylen=3
```

```
ciphertext=transposition(keylen,"aatmaj")

print(ciphertext)

plaintext = decrypt(keylen,ciphertext)

print(plaintext)
```



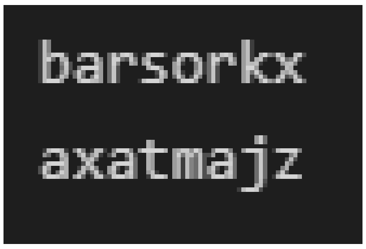
```
am aa tj

aatmaj
```

3. Playfair cipher

```
def playfair(key, plaintext):

    plaintext = plaintext.replace("i", "j")

    matrix = make_matrix_from_key(key)

    pairs = make_pairs(plaintext)

    answer = crypt(pairs, matrix)

    ans=""

    for i in answer:

        for j in i:

          ans+=j

    return ans
```

```python
    # print(matrix)



def crypt(pairs, matrix):
    answer = []
    for pair in pairs:
        answer.append(crypt_pair(pair, matrix))

    return answer



def crypt_pair(pair, matrix):
    # print(matrix)
    first = matrix.index(pair[0])
    second = matrix.index(pair[1])
    div_1, mod_1 = divmod(first, 5)
    div_2, mod_2 = divmod(second, 5)

    # check if both are in the same row
    if div_1 == div_2:
        mod_1 = (mod_1 + 1) % 5
        mod_2 = (mod_2 + 1) % 5

    elif mod_1 == mod_2:  # column same
        div_1 = (div_1 + 1) % 5
        div_2 = (div_2 + 1) % 5


    else:

        mod_1,mod_2 = mod_2,mod_1

    first = div_1*5+mod_1

    second = div_2*5+mod_2

    return (matrix[first], matrix[second])




def make_pairs(plaintext):

    # plaintext+='z'

    pairs = []

    i = 0

    while i <= len(plaintext) - 1:

        if i == len(plaintext) - 1:

            pairs.append((plaintext[i], "z"))

            break


        if plaintext[i] != plaintext[i + 1]:

            pairs.append((plaintext[i], plaintext[i + 1]))

            i += 1

        else:

            pairs.append((plaintext[i], "x"))

        i += 1
```

```python
    return pairs



def make_matrix_from_key(key):
    length = 5
    alphabets = [
        "a",
        "b",
        "c",
        "d",
        "e",
        "f",
        "g",
        "h",
        "j",
        "k",
        "l",
        "m",
        "n",
        "o",
        "p",
        "q",
        "r",
        "s",
        "t",
        "u",

        "v",

        "w",

        "x",

        "y",

        "z",

    ]

    for i in alphabets:

        if i not in key:

            key += i

    answer = []

    if len(key) != length**2:

        for i in range(length**2 - len(key)):

            key += " "

    for i in range(length):

        # answer.append([])

        for j in range(length):

            answer.append(key[length * i + j])

    return answer


def deplayfair(key,plaintext):

    matrix = make_matrix_from_key(key)

    pairs = make_pairs(plaintext)


    answer = decrypt(pairs, matrix)
    ans=""
    for i in answer:
        for j in i:
          ans+=j
    return ans



def decrypt(pairs, matrix):
    answer = []
    for pair in pairs:
        answer.append(decrypt_pair(pair, matrix))

    return answer

def decrypt_pair(pair, matrix):
    # print(matrix)
    first = matrix.index(pair[0])
    second = matrix.index(pair[1])
    div_1, mod_1 = divmod(first, 5)
    div_2, mod_2 = divmod(second, 5)

    # check if both are in the same row
    if div_1 == div_2:
        mod_1 = (mod_1 - 1) % 5
        mod_2 = (mod_2 - 1) % 5
    elif mod_1 == mod_2:  # column same
        div_1 = (div_1 - 1) % 5
        div_2 = (div_2 - 1) % 5

    else:
        mod_1,mod_2 = mod_2,mod_1
    first = div_1*5+mod_1
    second = div_2*5+mod_2
    return (matrix[first], matrix[second])

key = "monarchy"
ciphertext = playfair(key, "aatmaj")
print(ciphertext)

plaintext = deplayfair(key, ciphertext)
print(plaintext)
```



```
barsorkx
axatmajz
```

Somaiya Vidyavihar University
K. J. Somaiya College of Engineering
Department of Computer Engineering

POSTLAB

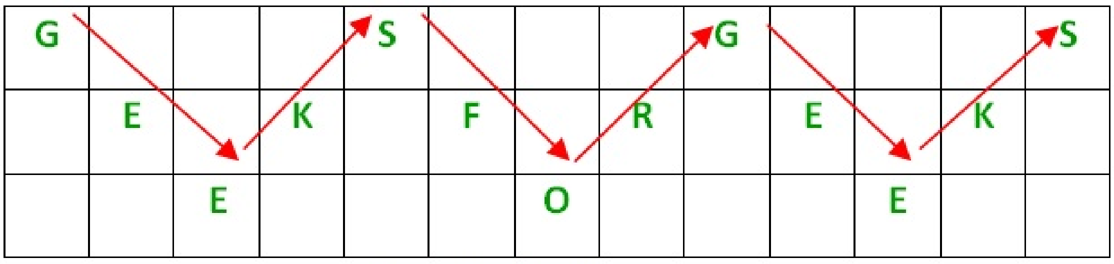
1.1 Write the points of difference between mono-alphabetic cipher and poly-alphabetic cipher.

Mono-alphabetic Cipher:
Each letter in the plaintext is consistently replaced by a fixed letter in the ciphertext.
The substitution remains constant throughout the encryption process.
It is vulnerable to frequency analysis because identical plaintext letters always map to the same ciphertext letter.
Examples include Caesar cipher and Atbash cipher.

Poly-alphabetic Cipher:
Each letter in the plaintext can be mapped to different letters in the ciphertext depending on its position.
Multiple substitution alphabets or keys are used during the encryption process.
It is more secure against frequency analysis due to variable mappings.
Examples include Vigenère cipher and Playfair cipher.

1.2 Explain the working of a rail-fence cipher with the help of an example.
Encryption

In a transposition cipher, the order of the alphabets is re-arranged to obtain the cipher-text. In the rail fence cipher, the plain-text is written downwards and diagonally on successive rails of an imaginary fence.
When we reach the bottom rail, we traverse upwards moving diagonally, after reaching the top rail, the direction is changed again. Thus the alphabets of the message are written in a zig-zag manner.
After each alphabet has been written, the individual rows are combined to obtain the cipher-text. For example, if the message is “GeeksforGeeks” and the number of rails = 3 then cipher is prepared as:



1.3 Discuss any three applications of cryptography.

Secure Communication:
Cryptography is widely used to ensure the confidentiality and integrity of communication over networks. It secures data transmission in various forms, including emails, online transactions, and instant messaging.
Data Encryption:
Cryptography is employed to encrypt sensitive data at rest, such as stored files or databases. This ensures that even if unauthorized access occurs, the data remains unreadable without the proper decryption key.
Digital Signatures:
Cryptography is used for creating digital signatures, which authenticate the origin and integrity of digital messages or documents. This is crucial for verifying the authenticity of electronic transactions and documents in fields like finance and legal documentation.

1.4 Distinguish between substitution and transposition ciphers.

Substitution Ciphers:

In substitution ciphers, each element in the plaintext (usually a letter) is replaced with another element (letter or symbol) in the ciphertext.

The substitution remains consistent throughout the encryption process.

Examples include Caesar cipher, Atbash cipher, and the more complex mono-alphabetic and poly-alphabetic ciphers.

Transposition Ciphers:

Transposition ciphers involve rearranging the positions of elements in the plaintext, rather than substituting them.

The order of the elements is changed based on a specific system or key.

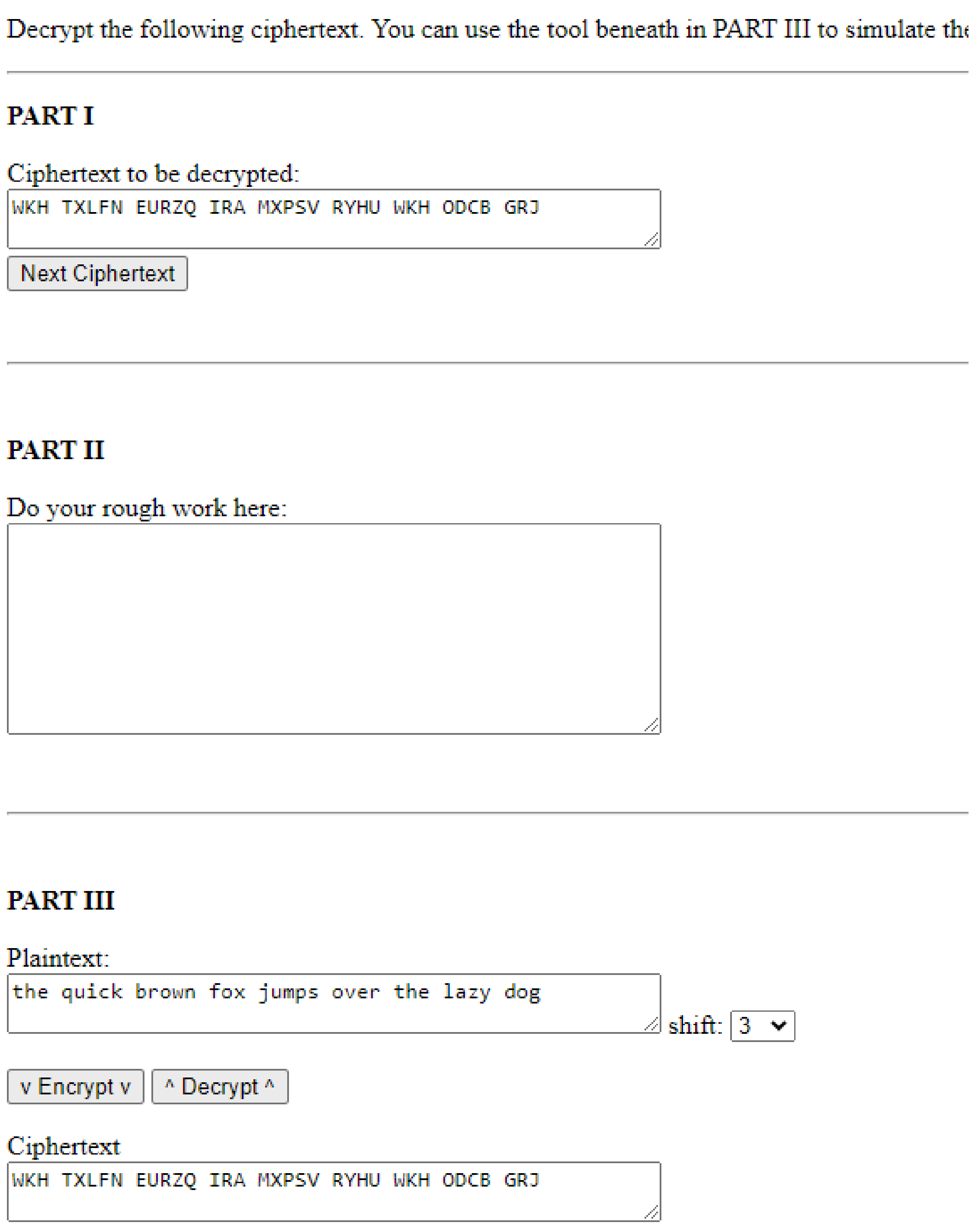
Examples include Rail-fence cipher, Columnar transposition cipher, and Route cipher.

Conclusion:-

Thus we have understood how various encryption algorithms work. We have implemented the playfair cipher and the columnar transposition cipher. We also have understood the working behind caesar cipher. These cipher techniques were used to protect the confidential messages a long time ago. These are very basic techniques and can be hacked easily.

Somaiya Vidyavihar University
K. J. Somaiya College of Engineering
Department of Computer Engineering

VLAB:-



Decrypt the following ciphertext. You can use the tool beneath in PART III to simulate the

PART I

Ciphertext to be decrypted:

```
WKH TXLFN EURZQ IRA MXPSV RYHU WKH ODCB GRJ
```

Next Ciphertext

PART II

Do your rough work here:

PART III

Plaintext:

```
the quick brown fox jumps over the lazy dog
```

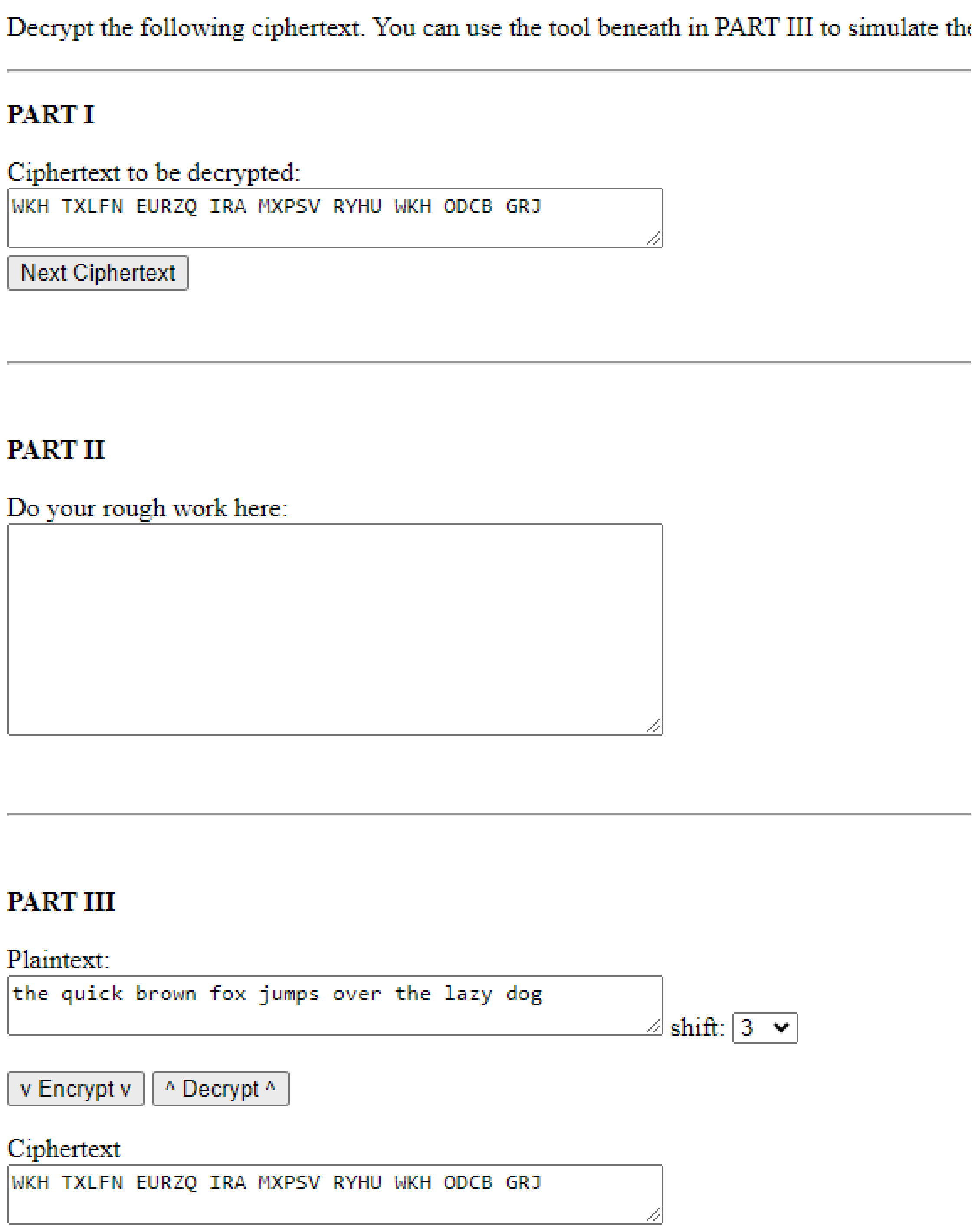
shift: 3

v Encrypt v ^ Decrypt ^

Ciphertext

```
WKH TXLFN EURZQ IRA MXPSV RYHU WKH ODCB GRJ
```

.



Decrypt the following ciphertext. You can use the tool beneath in PART III to simulate the

PART I

Ciphertext to be decrypted:

```
WKH TXLFN EURZQ IRA MXPSV RYHU WKH ODCB GRJ
```

PART II

Do your rough work here:

PART III

Plaintext:

```
the quick brown fox jumps over the lazy dog
```

shift: 3

Ciphertext

```
WKH TXLFN EURZQ IRA MXPSV RYHU WKH ODCB GRJ
```

Somaiya Vidyavihar University
K. J. Somaiya College of Engineering
Department of Computer Engineering