INDEX

SR. No.	TITLE	Page No.	Performance date	Assessment date	Marks	Sign
1	Implement Caesar cipher encryption-decryption.					
2	Implement Monoalphabetic cipher encryption-decryption.					
3	Implement Playfair cipher encryption-decryption.					
4	Implement Polyalphabetic cipher encryption-decryption.					
5	Implement Hill cipher encryption-decryption.					
6	Implement Simple Transposition encryption-decryption.					
7	Implement One time pad encryption-decryption.					
8	Implement Diffi-Hellmen Key exchange Method.					
9	Implement RSA encryption-decryption algorithm.					
10	Demonstrate working of Digital Signature using Cryptool.					

PRACTICAL 5

AIM: Implement Hill cipher encryption-decryption.

Code:

```
def multiply_lists(two_d_list, one_d_list):
    result = [[two_d_list[i][j] * one_d_list[j]
               for j in range(len(one_d_list))] for i in range(len(two_d_list))]
    return result


def char_to_int(text):
    l1 = []
    for char in text:
        if char.isalpha():
            if char.isupper():
                l1.append(ord(char) - 65)
            else:
                l1.append(ord(char) - 97)
    return l1
def int_to_chat(number_list):
    l1 = []
    for integer in number_list:
        l1.append(chr(integer + 97))
    return l1
def encoding_hill_cipher(text):
    single_encode_list = char_to_int(text)
    encode = []
    key = [[3, 1], [5, 2]]
    for i in range(0, 4, 2):
        l2 = []
```

```
        l2.append(single_encode_list[i])
        l2.append(single_encode_list[i + 1])
        x1 = multiply_lists(key, l2)
        x2 = []
        i = 0
        x2.append(x1[i][i] + x1[i][i + 1])
        x2.append(x1[i + 1][i] + x1[i + 1][i + 1])
        x3 = []
        x3.append(x2[i] % 26)
        x3.append(x2[i + 1] % 26)
        encode.append(x3)

    single_encode_list = [i for sublist in encode for i in sublist]
    join_encoding_string = ''.join(int_to_chat(single_encode_list))

    return join_encoding_string

def decoding_hill_cipher(text):
    single_decode_list = char_to_int(text)
    decoding_key = [[2, -1], [-5, 3]]
    decode = []

    for i in range(0, 4, 2):
        l2 = []
        l2.append(single_decode_list[i])
        l2.append(single_decode_list[i + 1])

        x1 = multiply_lists(decoding_key, l2)
```

```
        x2 = []
        i = 0
        x2.append(x1[i][i] + x1[i][i + 1])
        x2.append(x1[i + 1][i] + x1[i + 1][i + 1])
        x3 = []
        x3.append(x2[i] % 26)
        x3.append(x2[i + 1] % 26)
        decode.append(x3)

    single_decode_list = [i for sublist in decode for i in sublist]
    join_decoding_string = ''.join(int_to_chat(single_decode_list))

    return join_decoding_string

print("encoded message :",encoding_hill_cipher("Meet"))
print("decoded message :",decoding_hill_cipher(encoding_hill_cipher("Meet")))
```

output:

```
PS C:\work\7th sem> python -u "c:\work\7th sem\INS\practical 5.py"
encoded message : oqfg
decoded message : meet
PS C:\work\7th sem>
```

PRACTICAL 2

AIM: Implement Monoalphabetic cipher encryption-decryption

Code:

```
def char_to_int(text):
    l1 = []
    for char in text:
        if char.isalpha():
            if char.isupper():
                l1.append(ord(char) - 65)
            else:
                l1.append(ord(char) - 97)
    return l1


def encoding_mono_alphabetic(text, key):
    string_list = []
    for i in string:
        string_list.append(i)

    encoding_mono = []
    int_sting_list = char_to_int(string_list)
    for i in int_sting_list:
        encoding_mono.append(key[i])

    single_encode_list = [i for sublist in encoding_mono for i in sublist]
    join_encoding_string = ''.join((single_encode_list))

    return join_encoding_string

def decoding_mono_alphabetic(text):
```

```
    l1 = []
    alphabet = ['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']


    for i in text:
        l1.append(i)
    l2 = char_to_int(l1)
    decoded_string = []
    for i in text:


        index1 = key.index(i)
        decoded_string.append(alphabet[index1])


    single_decode_list = [i for sublist in decoded_string for i in sublist]
    join_decoding_string = ''.join((single_decode_list))
    return join_decoding_string
string = input("enter the string :")
key = []
key_string = input("enter the key :")
for i in key_string:
    key.append(i)
print("Encoded message :", encoding_mono_alphabetic(string, key))
print("decoded message :", decoding_mono_alphabetic(encoding_mono_alphabetic(string,
key)))
```

Output:

```
PS C:\work\7th sem> python -u "c:\work\7th sem\INS\mono_alphabatic.py"
enter the string :helloworld
enter the key :qwertyuioplkjhgfdsazxcvbnm
Encoded message : itkkgvgskr
decoded message : helloworld
PS C:\work\7th sem>
```

PRACTICAL 3

AIM: Implement Playfair cipher encryption-decryption.

Code:

```
def print_matrix(m):
    print("\nMatrix:")
    for i in m:
        print(i)



def not_in_matrix(key, m):
    for i in m:
        for j in i:
            if key == j:
                return False
    return True



def get_index(key, m):
    for i, k1 in enumerate(m):
        for j, k2 in enumerate(k1):
            if key == k2:
                return [i, j]
    return [-1, -1]



key = "monarchy"
text = "instrument"

print("\nText:", text)
```

```
print("Key:", key)

alpha = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'k', 'l', 'm',
         'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']

matrix = [[' ', ' ', ' ', ' ', ' '], [' ', ' ', ' ', ' ', ' '],
          [' ', ' ', ' ', ' ', ' '], [' ', ' ', ' ', ' ', ' '], [' ', ' ', ' ', ' ', ' ']]


i = j = k = 0

while k < len(key):
    if not_in_matrix(key[k], matrix):
        if j == 5:
            i += 1
            j = 0
        matrix[i][j] = key[k]
        j += 1
    k += 1

for a in alpha:
    if not_in_matrix(a, matrix):
        if j == 5:
            i += 1
            j = 0
        matrix[i][j] = a
        j += 1

print_matrix(matrix)
```

```
split = []
i = 0

while i < len(text):
    s = text[i:i+2]

    if len(s) == 1:
        s += "z"

    if s[0] == s[1]:
        split.append(s[0]+"x")
        i += 1
    else:
        split.append(s)
        i += 2

encoded = []
for i in split:
    v1 = i[0]
    v2 = i[1]

    i1 = get_index(v1, matrix)
    i2 = get_index(v2, matrix)

    v1_i = i1[0]
    v1_j = i1[1]
    v2_i = i2[0]
    v2_j = i2[1]
```

```
    if v1_i == v2_i:
        encoded.append(matrix[v1_i][(v1_j+1) % 5] + matrix[v2_i][(v2_j+1) % 5])

    elif v1_j == v2_j:
        encoded.append(matrix[(v1_i+1) % 5][v1_j] + matrix[(v2_i+1) % 5][v2_j])

    else:
        encoded.append(matrix[v1_i][v2_j] + matrix[v2_i][v1_j])

encoded = "".join(encoded)
print("Encoded: ", encoded)
split = []
i = 0

while i < len(encoded):
    s = encoded[i:i+2]

    split.append(s)
    i += 2

decoded = []

for i in split:
    v1 = i[0]
    v2 = i[1]

    i1 = get_index(v1, matrix)
    i2 = get_index(v2, matrix)
```

```
    v1_i = i1[0]
    v1_j = i1[1]
    v2_i = i2[0]
    v2_j = i2[1]

    if v1_i == v2_i:
        decoded.append(matrix[v1_i][(v1_j-1) % 5] + matrix[v2_i][(v2_j-1) % 5])

    elif v1_j == v2_j:
        decoded.append(matrix[(v1_i-1) % 5][v1_j] + matrix[(v2_i-1) % 5][v2_j])

    else:
        decoded.append((matrix[v2_i][v1_j] + matrix[v1_i][v2_j])[::-1])

decoded = "".join(decoded)
print("\nDecoded:", decoded)
```

```
PS C:\work\7th sem> python -u "c:\work\7th sem\INS\practical 3.py"

Text: instrument
Key: monarchy

Matrix:
['m', 'o', 'n', 'a', 'r']
['c', 'h', 'y', 'b', 'd']
['e', 'f', 'g', 'i', 'k']
['l', 'p', 'q', 's', 't']
['u', 'v', 'w', 'x', 'z']
Encoded:  gatlmzclrq

Decoded: instrument
```