|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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. |  |  |  |  |  |

**INDEX**

**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]

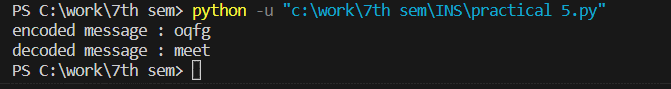
join\_decoding\_string = ''.join(int\_to\_chat(single\_decode\_list))

return join\_decoding\_string

print("encoded message :",encoding\_hill\_cipher("Meet"))

print("decoded message :",decoding\_hill\_cipher(encoding\_hill\_cipher("Meet")))

output:



**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 :")

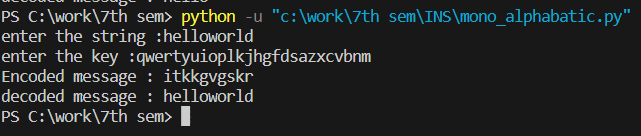
for i in key\_string:

key.append(i)

print("Encoded message :", encoding\_mono\_alphabetic(string, key))

print("decoded message :", decoding\_mono\_alphabetic(encoding\_mono\_alphabetic(string, key)))

Output:



**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)

