Course Name:	Applied Cryptography (16U3263)	Semester:	V
Date of Performance:	14 / 07 / 25	DIV/ Batch No:	AC2
Student Name:	Aaryan Sharma	Roll No:	16010123012

Experiment No: 1

Title: Encryption-Decryption programs using classical cryptography

Aim and Objective of the Experiment:

To write a program to convert plain text into cipher text using Caesar cipher and Transposition cipher

COs to be achieved:

CO1: Explain the fundamentals of Information Security and cryptography

Books/ Journals/ Websites referred:

1. Stallings, W., Cryptography and Network Security: Principles and Practice, Second edition, Person Education
2. "Caesar Cipher in cryptography", https://www.geeksforgeeks.org/caesar-cipher-in-cryptography/, last retrieved on Aug 01, 2023
3. "PlayFair Cipher in cryptography": https://www.geeksforgeeks.org/playfair-cipher-with-examples/, last retrieved on Aug 01, 2023
4. "Transposition cipher in cryptology, "https://www.britannica.com/topic/transposition-cipher, last retrieved on Aug 01, 2023

Theory:

Transposition and substitution are the two types of cyphers employed in traditional cryptography. Substitution ciphers encrypt plaintext by changing the plaintext one piece at a time, while Transposition ciphers encrypt plaintext by moving small pieces of the message around. Anagrams are a primitive transposition cipher.

Substitutional Cipher:

. **Caesar Cipher**

The earliest known, and the simplest, use of a substitution cipher was by Julius Caesar.
The Caesar cipher involves replacing each letter of the alphabet with the letter standing three places further down the alphabet.
For example,

```
plain:  meet me after the toga party
cipher: PHHW PH DIWHU WKH WRJD SDUWB
```

Note that the alphabet is wrapped around, so that the letter following Z is A.
We can define the transformation by listing all possibilities, as follows:

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

Let us assign a numerical equivalent to each letter:

a	b	c	d	e	f	g	h	i	j	k	l	m
0	1	2	3	4	5	6	7	8	9	10	11	12

n	o	p	q	r	s	t	u	v	w	x	y	z
13	14	15	16	17	18	19	20	21	22	23	24	25

The algorithm can be expressed as follows. For each plaintext letter p, substitute the ciphertext letter C:

$C = E(3, p) = (p + 3) \bmod 26$

A shift may be of any amount, so the general Caesar algorithm is

$$C = E(k, p) = (p + k) \bmod 26$$

where k takes on a value in the range 1 to 25.

The decryption algorithm is simply

$$p = D(k, C) = (C - k) \bmod 26$$

If it is known that a given cipher text is a Caesar cipher, then a brute-force cryptanalysis is easily performed. Simply try all the 25 possible keys.

A. **Play fair 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.

The Playfair Cipher Encryption 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.

e.g. consider Key= Monarchy

KeySquare:

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

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(bogus character) is added to the last letter.

Example 1:

PlainText: "instruments"

After Split: 'in' 'st' 'ru' 'me' 'nt' 'sz'
Example 2: (With ‘x’ as the bogus letter.)
Plain Text: “hello”
After Split: ‘he’ ‘lx’ ‘lo’

Rules for Encryption:

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

For example:

```
Diagraph: "me"
Encrypted Text: cl
Encryption:
  m -> c
  e -> l
```

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

2. **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).

For example:

Diagraph: "st"
Encrypted Text: tl
Encryption:
s -> t
t -> l

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

3. **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.

For example:

Diagraph: "nt"
Encrypted Text: rq

Encryption:
n -> r
t -> q

For example:

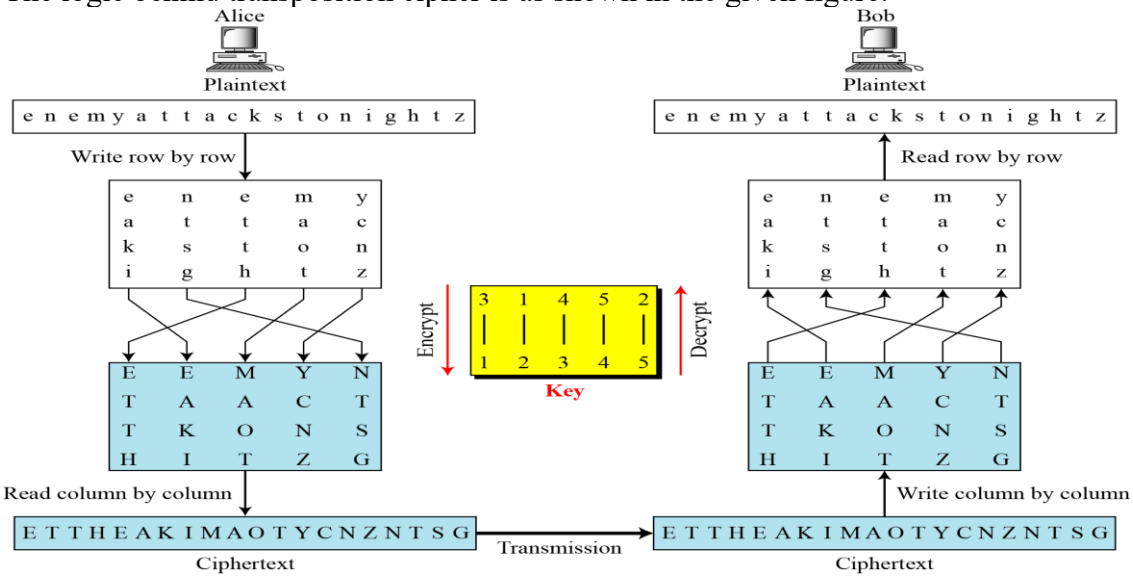
Plain Text: "instrumentsz"
Encrypted Text: gatlmzclrqtx
Encryption:
i -> g
n -> a
s -> t
t -> l
r -> m
u -> z
m -> c
e -> l
n -> r
t -> q
s -> t
z -> x

Decryption in Playfair Cipher:

Transposition cipher

A. **Columnar Cipher:**

The logic behind transposition cipher is as shown in the given figure:



B. **Rail Fence Cipher:**

Example: We encipher NOTHING IS AS IT SEEMS by first writing it on two lines in a zig-zag pattern (or rail fence). The ciphertext is produced by transcribing the first row followed by the second row.

```
N     T     I     G     S     S     T     E     M
   O     H     N     I     A     I     S     E     S
```

Ciphertext: NTIGS STEMO HNIAI SES.
To decrypt, we write half the letters on one line, half on the second. (Note that if there are an odd number of letters, we include the "middle" letter on the top line.)

Code:

1. **Playfair without key**

```
def removeSpaces(plain):

  n = len(plain)

  temp = ""

  for i in range(n):

    if plain[i] != ' ':

      temp += plain[i]

  return temp



def toLowerCase(string):

  return string.lower()



def generateKeyTable(keyT):

  keyT.clear()

  for i in range(5):

    keyT.append([0]*5)
```

```
  alphabet = "abcdefghiklmnopqrstuvwxyz"

  row = 0
  col = 0
  for char in alphabet:
    keyT[row][col] = char
    col += 1
    if col == 5:
      row += 1
      col = 0

def search(keyT, a, b, arr):
  if a == 'j':
    a = 'i'
  if b == 'j':
    b = 'i'

  for i in range(5):
    for j in range(5):
      if keyT[i][j] == a:
        arr[0] = i
```

```
        arr[1] = j
      elif keyT[i][j] == b:
        arr[2] = i
        arr[3] = j


def prepare(string):
  result = ""
  i = 0
  while i < len(string):
    result += string[i]
    if i + 1 < len(string) and string[i] == string[i + 1]:
      result += 'x'
    i += 1


  if len(result) % 2 != 0:
    result += 'x'
  return result


def encrypt(string, keyT):
  n = len(string)
  arr = [0]*4
```

```
  result = list(string)
  for i in range(0, n, 2):
    search(keyT, result[i], result[i+1], arr)

    if arr[0] == arr[2]:
      result[i] = keyT[arr[0]][(arr[1] + 1) % 5]
      result[i+1] = keyT[arr[0]][(arr[3] + 1) % 5]
    elif arr[1] == arr[3]:
      result[i] = keyT[(arr[0] + 1) % 5][arr[1]]
      result[i+1] = keyT[(arr[2] + 1) % 5][arr[1]]
    else:
      result[i] = keyT[arr[0]][arr[3]]
      result[i+1] = keyT[arr[2]][arr[1]]

  return ''.join(result)

def encryptByPlayfairCipher(string):
  keyT = []
  string = toLowerCase(removeSpaces(string))
  string = prepare(string)
  generateKeyTable(keyT)
  return encrypt(string, keyT)
```

```
def decrypt(string, keyT):
  n = len(string)
  arr = [0]*4

  result = list(string)
  for i in range(0, n, 2):
    search(keyT, result[i], result[i+1], arr)

    if arr[0] == arr[2]:
      result[i] = keyT[arr[0]][(arr[1] - 1) % 5]
      result[i+1] = keyT[arr[0]][(arr[3] - 1) % 5]
    elif arr[1] == arr[3]:
      result[i] = keyT[(arr[0] - 1) % 5][arr[1]]
      result[i+1] = keyT[(arr[2] - 1) % 5][arr[1]]
    else:
      result[i] = keyT[arr[0]][arr[3]]
      result[i+1] = keyT[arr[2]][arr[1]]

  return ''.join(result)

def decryptByPlayfairCipher(string):
```

```
  keyT = []
  string = toLowerCase(removeSpaces(string))
  generateKeyTable(keyT)
  return decrypt(string, keyT)


def printKeyTable(keyT):
  print("Key Table:")
  for i in range(5):
    for j in range(5):
      print(keyT[i][j], end=" ")
    print()
  print()


keyT = []
generateKeyTable(keyT)
printKeyTable(keyT)


string = input("Enter the string to encrypt: ")
string = toLowerCase(removeSpaces(string))
string = prepare(string)


encrypted_text = encryptByPlayfairCipher(string)
```

```
print("Cipher text:", encrypted_text)



decrypted_text = decryptByPlayfairCipher(encrypted_text)

print("Decrypted text:", decrypted_text)

print()
```

2. Rail Fence

```
def rail_fence_encrypt(message, rails):

    message = message.replace(" ", "").upper()

    fence = ["" for _ in range(rails)]

    rail = 0

    direction = 1



    for char in message:

        fence[rail] += char

        rail += direction

        if rail == 0 or rail == rails - 1:

            direction *= -1



    return "".join(fence)



def rail_fence_decrypt(ciphertext, rails):
```

```
    ciphertext = ciphertext.replace(" ", "").upper()
    n = len(ciphertext)

    fence = [["" for _ in range(n)] for _ in range(rails)]

    rail = 0
    direction = 1
    for i in range(n):
        fence[rail][i] = "*"
        rail += direction
        if rail == 0 or rail == rails - 1:
            direction *= -1

    index = 0
    for r in range(rails):
        for c in range(n):
            if fence[r][c] == "*" and index < n:
                fence[r][c] = ciphertext[index]
                index += 1

    result = []
    rail = 0
```

```
    direction = 1

    for i in range(n):

        result.append(fence[rail][i])

        rail += direction

        if rail == 0 or rail == rails - 1:

            direction *= -1



    return "".join(result)


if __name__ == "__main__":

    message = input("Enter message to encrypt: ")

    rails = int(input("Enter number of rails: "))



    encrypted = rail_fence_encrypt(message, rails)

    print("Encrypted:", encrypted)



    decrypted = rail_fence_decrypt(encrypted, rails)

    print("Decrypted:", decrypted)
```

Output:

```
PS D:\KJSCE\BTech\TY\Sem V\AC> python playfair.py
Key Table:
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

Enter the string to encrypt: HELLO WORLD
Cipher text: kcnvmpymqmcy
Decrypted text: helxloworldx
```

```
PS D:\KJSCE\BTech\TY\Sem V\AC> python railfence.py
Enter message to encrypt: Hello world
Enter number of rails: 3
Encrypted: HOLELWRDLO
Decrypted: HELLOWORLD
```

Post Lab Subjective/Objective type Questions:

1. **Explain the key differences between substitution ciphers**
   Key Differences Between Substitution Ciphers:
   In Caesar Cipher shifts each letter by a fixed amount (e.g., A becomes D (+3)).
   In Monoalphabetic Cipher each letter maps to a unique substitute (e.g., A becomes Q).
   Polyalphabetic Cipher uses multiple cipher alphabets (e.g., Vigenère).
   Playfair Cipher encrypts pairs of letters, using a 5x5 matrix, repeated letters are replaced (e.g., "LL" becomes "LX") to avoid patterns.

2. **In the Playfair cipher, why is it important to replace repeated letters in a digraph.**
   Repeating letters in a digraph (e.g., "LL") would create predictable patterns, weakening the cipher. Replacing repeated letters with a filler (like "X") ensures stronger encryption.

3. **"Weak security is worse than no security." Do you agree? Justify your answer with an example.**
   Yes, because weak security can give a false sense of safety. For example, WEP encryption in Wi-Fi was easily cracked, leaving users vulnerable. Weak security leads to overconfidence, while no security might encourage caution. Essentially, weak security is a greater risk than no security at all.

Conclusion:

I have successfully completed the experiment on implementing classical encryption and decryption techniques using Caesar Cipher, Playfair Cipher, and Transposition Ciphers. Through this experiment, I understood how substitution and transposition ciphers work to secure plaintext by altering or rearranging characters. I also learned the importance of key usage, handling repeated letters in Playfair Cipher, and how classical cryptography laid the foundation for modern encryption techniques. This experiment enhanced my understanding of the basic principles of cryptography and its role in information security.