**Practical-1**

**Aim: Perform encryption and decryption using Caesar substitution cipher. Perform Brute Force attack on the ciphertext to retrieve plaintext.**

**Code:**

def caesar\_encrypt(plaintext, shift):  
 ciphertext = ''  
 for char in plaintext:  
 if char.isalpha():  
 # For each character, it checks if  
 # it is an alphabet letter (char.isalpha()). If it is, it calculates the shifted  
 # character based on whether it's lowercase or uppercase and appends it to the ciphertext.  
 shift\_amount = shift % 26  
 if char.islower():  
 shifted = chr(((ord(char) - ord('a') + shift\_amount) % 26) + ord('a'))  
 else:  
 shifted = chr(((ord(char) - ord('A') + shift\_amount) % 26) + ord('A'))  
 ciphertext += shifted  
 elif char.isdigit():  
 shifted = str((int(char) + shift) % 10)  
 ciphertext += shifted  
 else:  
 ciphertext += char  
 return ciphertext  
  
def caesar\_decrypt(ciphertext, shift):  
 plaintext = ''  
 for char in ciphertext:  
 if char.isalpha():  
 shift\_amount = shift % 26  
 if char.islower():  
 shifted = chr(((ord(char) - ord('a') - shift\_amount) % 26) + ord('a'))  
 else:  
 shifted = chr(((ord(char) - ord('A') - shift\_amount) % 26) + ord('A'))  
 plaintext += shifted  
 elif char.isdigit():  
 shifted = str((int(char) - shift) % 10)  
 plaintext += shifted  
 else:  
 plaintext += char  
 return plaintext  
  
def caesar\_brute\_force(ciphertext):  
 decrypted\_texts = []  
 for shift in range(26):  
 decrypted\_text = ''  
 for char in ciphertext:  
 if char.isalpha():  
 if char.islower():  
 decrypted\_char = chr(((ord(char) - ord('a') - shift) % 26) + ord('a'))  
 else:  
 decrypted\_char = chr(((ord(char) - ord('A') - shift) % 26) + ord('A'))  
 decrypted\_text += decrypted\_char  
 else:  
 decrypted\_text += char  
 decrypted\_texts.append(decrypted\_text)  
 return decrypted\_texts  
  
def get\_input():  
 plaintext = input("Enter the text you want to encrypt and decrypt: ")  
 shift\_str = input("Enter the shift (a positive number for encryption, a negative number for decryption): ")  
 if shift\_str.isdigit():  
 shift = int(shift\_str)  
 else:  
 print("Invalid input for shift. Please enter a valid number.")  
 return None, None  
 return plaintext, shift  
  
  
plaintext, shift = get\_input()  
  
if plaintext is not None and shift is not None:  
 encrypted\_text = caesar\_encrypt(plaintext, shift)  
 print("Encrypted:", encrypted\_text)  
  
 decrypted\_text = caesar\_decrypt(encrypted\_text, shift)  
 print("Decrypted:", decrypted\_text)  
  
 decrypted\_texts = caesar\_brute\_force(encrypted\_text)  
 for i, text in enumerate(decrypted\_texts):  
 print(f"Shift {i}: {text}")

**OutPut:**

**A screen shot of a computer

Description automatically generated**

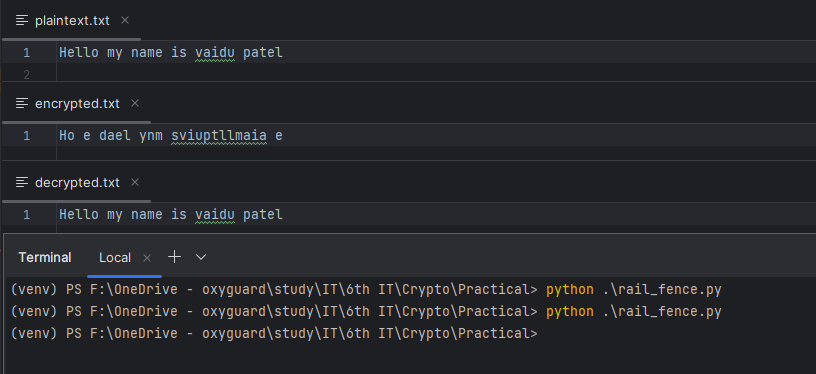
**Practical-2**

**Aim: Perform encryption and decryption using Rail Fence transposition cipher. Perform encryption by fetching data from .txt file and decryption through file operations.**

**Code:**

def encryptRailFence(plain\_text, key):  
 rail = [['\n' for i in range(len(plain\_text))] for j in range(key)]  
 dir\_down = False  
 row, col = 0, 0  
 for i in range(len(plain\_text)):  
 if (row == 0) or (row == key - 1):  
 dir\_down = not dir\_down  
 rail[row][col] = plain\_text[i]  
 col += 1  
 if dir\_down:  
 row += 1  
 else:  
 row -= 1  
 result = []  
 for i in range(key):  
 for j in range(len(plain\_text)):  
 if rail[i][j] != '\n':  
 result.append(rail[i][j])  
 return "".join(result)  
  
def decryptRailFence(cipher\_text, key):  
 rail = [['\n' for i in range(len(cipher\_text))] for j in range(key)]  
 dir\_down = None  
 row, col = 0, 0  
 for i in range(len(cipher\_text)):  
 if row == 0:  
 dir\_down = True  
 if row == key - 1:  
 dir\_down = False  
 rail[row][col] = '\*'  
 col += 1  
 if dir\_down:  
 row += 1  
 else:  
 row -= 1  
 index = 0  
 for i in range(key):  
 for j in range(len(cipher\_text)):  
 if (rail[i][j] == '\*') and (index < len(cipher\_text)):  
 rail[i][j] = cipher\_text[index]  
 index += 1  
 result = []  
 row, col = 0, 0  
 for i in range(len(cipher\_text)):  
 if row == 0:  
 dir\_down = True  
 if row == key - 1:  
 dir\_down = False  
 if rail[row][col] != '\*':  
 result.append(rail[row][col])  
 col += 1  
 if dir\_down:  
 row += 1  
 else:  
 row -= 1  
 return "".join(result)  
  
key = 3  
with open('plaintext.txt', 'r') as f:  
 plain\_text = f.read().replace('\n', '')  
cipher\_text = encryptRailFence(plain\_text, key)  
with open('encrypted.txt', 'w') as f:  
 f.write(cipher\_text)  
  
with open('encrypted.txt', 'r') as f:  
 cipher\_text = f.read().replace('\n', '')  
plain\_text = decryptRailFence(cipher\_text, key)  
with open('decrypted.txt', 'w') as f:  
 f.write(plain\_text)

**OutPut:**

****

**Practical-3**

**Aim: Perform encryption and decryption using Playfair substitution cipher. Perform encryption and decryption for both alphabetic and alphanumeric data types.**

**Code:**

import string  
def genKeyMat(key):  
 atoz = string.ascii\_lowercase.replace('j', '.')  
 key\_matrix = ['' for i in range(5)]  
 i = 0  
 j = 0  
 for c in key:  
 if c in atoz:  
 key\_matrix[i] += c  
 atoz = atoz.replace(c, '.')  
  
 j += 1  
 if j > 4:  
 i += 1  
 j = 0  
 for c in atoz:  
 if c != '.':  
 key\_matrix[i] += c  
  
 j += 1  
 if j>4:  
 i += 1  
 j = 0  
 return key\_matrix  
  
  
  
def encrypt(plainText):  
 plaintextpairs = []  
 ciphetextpairs = []  
 # Rule 1: if both latter are same or only one left add "X" after first letter  
 i = 0  
 while i < len(plainText):  
 a = plainText[i]  
 b = ""  
 if (i+1) == len(plainText):  
 b = 'x'  
 else:  
 b = plainText[i+1]  
 if a != b:  
 plaintextpairs.append(a + b)  
 i += 2  
 else :  
 plaintextpairs.append(a + 'x')  
 i += 1  
 # Rule 2: if letters are in same row, replace with letters to their immediate right letter  
 for pair in plaintextpairs:  
 applied\_rule = False  
 for row in key\_matrix:  
 if pair[0] in row and pair[1] in row:  
 j0 = row.find(pair[0])  
 j1 = row.find(pair[1])  
 ciphetextpair = row[(j0 + 1) % 5] + row[(j1 + 1) % 5]  
 ciphetextpairs.append(ciphetextpair)  
 applied\_rule = True  
 if applied\_rule:  
 continue  
 # Rule 3 :If letter are in same column, replace them with immediate below letter  
 for j in range(5):  
 col = "".join([key\_matrix[i][j] for i in range(5)])  
 if pair[0] in col and pair[1] in col:  
 i0 = col.find(pair[0])  
 i1 = col.find(pair[1])  
 ciphetextpair = col[(i0 + 1) % 5] + col[(i1 + 1) % 5]  
 ciphetextpairs.append(ciphetextpair)  
 applied\_rule = True  
 if applied\_rule:  
 continue  
 # Rule 4: not in same column or row,replace them with the letters on same row respectively but at  
 # the other pair of corners of the rectangle define by the original pairs  
 i0 = 0  
 i1 = 0  
 j0 = 0  
 j1 = 0  
 for i in range(5):  
 row = key\_matrix[i]  
 if pair[0] in row:  
 i0 = i  
 j0 = row.find(pair[0])  
 if pair[1] in row:  
 i1 = i  
 j1 = row.find(pair[1])  
 ciphetextpair = key\_matrix[i0][j1] + key\_matrix[i1][j0]  
 ciphetextpairs.append(ciphetextpair)  
 return "".join(ciphetextpairs)  
  
def decrypt(ciphetext):  
 encryptedtextpairs =[]  
 ciphetextpairs = []  
 # Rule 1: if both latter are same or only one left add "X" after first letter  
 i = 0  
 while i<len(ciphetext):  
 a = ciphetext[i]  
 b = ciphetext[i+1]  
 ciphetextpairs.append(a + b)  
 i+=2  
 # print(ciphetextpairs)  
  
 for pair in ciphetextpairs:  
 applied\_rule = False  
 for row in key\_matrix:  
 if pair[0] in row and pair[1] in row:  
 j0 = row.find(pair[0])  
 j1 = row.find(pair[1])  
 encryptedtextpair = row[(j0 + 4) % 5] + row[(j1 + 4) % 5]  
 encryptedtextpairs.append(encryptedtextpair)  
 applied\_rule = True  
 if applied\_rule:  
 continue  
 # Rule 3 :If letter are in same column, replace them with immediate below letter  
 for j in range(5):  
 col = "".join([key\_matrix[i][j] for i in range(5)])  
 if pair[0] in col and pair[1] in col:  
 i0 = col.find(pair[0])  
 i1 = col.find(pair[1])  
 encryptedtextpair = col[(i0 + 4) % 5] + col[(i1 + 4) % 5]  
 encryptedtextpairs.append(encryptedtextpair)  
 applied\_rule = True  
 if applied\_rule:  
 continue  
 # Rule 4: not in same column or row,replace them with the letters on same row respectively but at  
 # the other pair of corners of the rectangle define by the original pairs  
 i0 = 0  
 i1 = 0  
 j0 = 0  
 j1 = 0  
 for i in range(5):  
 row = key\_matrix[i]  
 if pair[0] in row:  
 i0 = i  
 j0 = row.find(pair[0])  
 if pair[1] in row:  
 i1 = i  
 j1 = row.find(pair[1])  
 encryptedtextpair = key\_matrix[i0][j1] + key\_matrix[i1][j0]  
 encryptedtextpairs.append(encryptedtextpair)  
 return "".join(encryptedtextpairs)  
  
  
key = 'playfair example'  
key\_matrix = genKeyMat(key)  
plainText = "hidethegoldinthetreestump"  
ciphetext = encrypt(plainText)  
print("Plain text: ",plainText)  
print("Key: ",key)  
print("Cipher text: ",ciphetext)  
print("Decrypted text: ",decrypt(ciphetext))

**OutPut:**

A screen shot of a computer

Description automatically generated

**Practical-4**

**Aim: Perform key exchange between two communicating parties using the Diffie-Hellman approach.**

**Code:-**

import random

def generate\_large\_prime():

    while True:

        prime\_candidate = random.randint(100, 1000)

        if is\_prime(prime\_candidate):

            return prime\_candidate

def is\_prime(n):

    if n <= 1:

        return False

    elif n <= 3:

        return True

    elif n % 2 == 0 or n % 3 == 0:

        return False

    i = 5

    while i \* i <= n:

        if n % i == 0 or n % (i + 2) == 0:

            return False

        i += 6

    return True

def generate\_private\_key(p):

    return random.randint(1, p - 1)

def calculate\_public\_key(g, private\_key, p):

    return pow(g, private\_key, p)

def calculate\_shared\_secret(public\_key, private\_key, p):

    return pow(public\_key, private\_key, p)

def diffie\_hellman():

    p = generate\_large\_prime()

    print("Prime number:",p)

    g = 5

    print("Primitive Root(g):",g)

    alice\_private\_key = generate\_private\_key(p)

    bob\_private\_key = generate\_private\_key(p)

    alice\_public\_key = calculate\_public\_key(g, alice\_private\_key, p)

    bob\_public\_key = calculate\_public\_key(g, bob\_private\_key, p)

    alice\_shared\_key = calculate\_shared\_secret(bob\_public\_key, alice\_private\_key, p)

    bob\_shared\_key = calculate\_shared\_secret(alice\_public\_key, bob\_private\_key, p)

    return alice\_shared\_key, bob\_shared\_key

alice\_secret, bob\_secret = diffie\_hellman()

print("Alice's shared secret:", alice\_secret)

print("Bob's shared secret:", bob\_secret)

**output:**

**A screen shot of a computer

Description automatically generated**

**Practical-5**

**Aim:** **Design sign-in page and encrypt passwords stored using AES.**

* **Form is created in React App**

**Code:**

**App.jsx**

import { useState } from "react";

import CryptoJS from "crypto-js";

function generateRandomKeyAndIV() {

  const key = CryptoJS.lib.WordArray.random(32);

  const iv = CryptoJS.lib.WordArray.random(16);

  return { key, iv };

}

function encryptWithAES(text, key, iv) {

  const encrypted = CryptoJS.AES.encrypt(text, key, { iv }).toString();

  return encrypted;

}

function decryptWithAES(encryptedData, key, iv) {

  const decrypted = CryptoJS.AES.decrypt(encryptedData, key, { iv }).toString(

    CryptoJS.enc.Utf8

  );

  return decrypted;

}

function App() {

  const [email, setEmail] = useState("");

  const [password, setPassword] = useState("");

  const [encryptedData, setEncryptedData] = useState("");

  const [decryptedData, setDecryptedData] = useState("");

  const { key, iv } = generateRandomKeyAndIV();

  const handleSubmit = (e) => {

    e.preventDefault();

    console.log(email);

    console.log(password);

    const encrypted = encryptWithAES(password, key, iv);

    setEncryptedData(encrypted);

    const decrypted = decryptWithAES(encrypted, key, iv);

    setDecryptedData(decrypted);

    setEmail("");

    setPassword("");

  };

  return (

    <>

      <div className="container">

        <form className="my-3" onSubmit={handleSubmit}>

          <div className="mb-3">

            <label htmlFor="exampleInputEmail1" className="form-label">

              Email address

            </label>

            <input

              type="email"

              value={email}

              onChange={(e) => {

                setEmail(e.target.value);

              }}

              className="form-control"

              id="exampleInputEmail1"

              aria-describedby="emailHelp"

            />

          </div>

          <div className="mb-3">

            <label htmlFor="exampleInputPassword1" className="form-label">

              Password

            </label>

            <input

              type="password"

              value={password}

              onChange={(e) => {

                setPassword(e.target.value);

              }}

              autoComplete="false"

              className="form-control"

              id="exampleInputPassword1"

            />

          </div>

          <button type="submit" className="btn btn-primary">

            Submit

          </button>

          <div className="my-3">

            <label htmlFor="encrypted" className="form-label">

              Encrypted data for Password

            </label>

            <div className="form-control p-3">{encryptedData}</div>

          </div>

          <div className="my-3">

            <label htmlFor="encrypted" className="form-label">

              Decrypted data for Password

            </label>

            <div className="form-control p-3">{decryptedData}</div>

          </div>

        </form>

      </div>

    </>

  );

}

export default App;

**Output: before**

**A screenshot of a computer

Description automatically generated**

**Output: after**

A screenshot of a computer

Description automatically generated

**Practical-6**

**Aim: Perform encryption and decryption on the following type of data using the RSA algorithm.**

**1) Integer data**

**2) Image file**

**Input Image: vaidik.jpg**

****

**Code:**

import random

import base64

def image\_to\_base64(image\_path):

    with open(image\_path, "rb") as img\_file:

        return base64.b64encode(img\_file.read()).decode('utf-8')

def base64\_to\_image(base64\_data, image\_path):

    with open(image\_path, "wb") as img\_file:

        img\_file.write(base64.b64decode(base64\_data))

def gcd(a, b):

    while b != 0:

        a, b = b, a % b

    return a

def multiplicative\_inverse(e, phi):

    d = 0

    x1, x2 = 0, 1

    y1, y2 = 1, 0

    temp\_phi = phi

    while e > 0:

        temp1 = temp\_phi // e

        temp2 = temp\_phi - temp1 \* e

        temp\_phi = e

        e = temp2

        x = x2 - temp1 \* x1

        y = y2 - temp1 \* y1

        x2 = x1

        x1 = x

        y2 = y1

        y1 = y

    if temp\_phi == 1:

        d = y2 + phi

    return d

def is\_prime(num):

    if num <= 1:

        return False

    elif num <= 3:

        return True

    elif num % 2 == 0 or num % 3 == 0:

        return False

    i = 5

    while i \* i <= num:

        if num % i == 0 or num % (i + 2) == 0:

            return False

        i += 6

    return True

def generate\_keypair(p, q):

    if not (is\_prime(p) and is\_prime(q)):

        raise ValueError("Both numbers must be prime.")

    elif p == q:

        raise ValueError("p and q cannot be equal")

    n = p \* q

    phi = (p - 1) \* (q - 1)

    e = random.randrange(1, phi)

    gcd\_value = gcd(e, phi)

    while gcd\_value != 1:

        e = random.randrange(1, phi)

        gcd\_value = gcd(e, phi)

    d = multiplicative\_inverse(e, phi)

    return ((e, n), (d, n))

def encrypt(pk, plaintext):

    key, n = pk

    cipher = [pow(ord(char), key, n) for char in plaintext]

    return cipher

def decrypt(pk, ciphertext):

    key, n = pk

    plain = [chr(pow(char, key, n)) for char in ciphertext]

    return ''.join(plain)

p = 61

q = 53

public\_key, private\_key = generate\_keypair(p, q)

print("Public Key:", public\_key)

print("Private Key:", private\_key)

message = "Ye Mera Text He Jiska Aaj Postmortem Bole To Encryption Decryption Hoga!!!"

print("Original message:", message)

encrypted\_msg = encrypt(public\_key, message)

print("Encrypted message:", encrypted\_msg)

decrypted\_msg = decrypt(private\_key, encrypted\_msg)

print("Decrypted message:", decrypted\_msg)

image\_path = "vaidik.jpg"

base64\_data = image\_to\_base64(image\_path)

print("Image converted to base64.")

encrypted\_data = encrypt(public\_key, base64\_data)

print("Base64 data encrypted.")

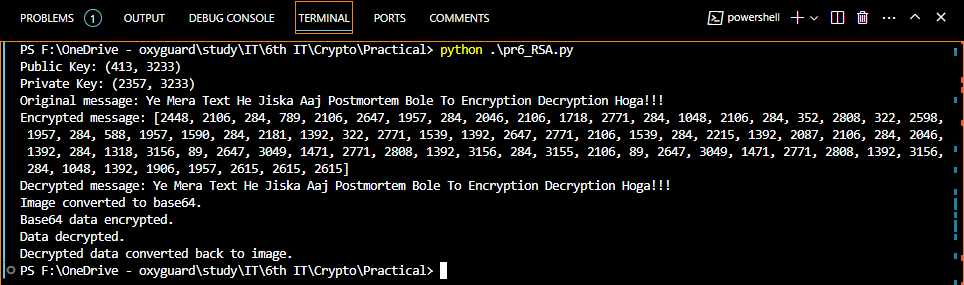
decrypted\_data = decrypt(private\_key, encrypted\_data)

print("Data decrypted.")

base64\_to\_image(decrypted\_data, "decrypted\_image.jpg")

print("Decrypted data converted back to image.")

**Output:**

****

**Output Image: decrypted\_image.jpg**



**Practical-7**

**Aim:** **Perform a practical to configure TELNET and SSH using real devices and analyze packet flows and their payload using Wireshark.**

**Practical-8**

**Aim:** **Write a program to implement a blockchain network. Perform bitcoin mining and demonstrate bitcoin transactions using the web app.**

**Code:**

import datetime

import hashlib

import json

from flask import Flask, jsonify

class Blockchain:

    def \_\_init\_\_(self):

        self.chain = []

        self.create\_block(proof=1, previous\_hash='0')

    def create\_block(self, proof, previous\_hash):

        block = {

            'index': len(self.chain) + 1,

            'timestamp': str(datetime.datetime.now()),

            'proof': proof,

            'previous\_hash': previous\_hash

        }

        self.chain.append(block)

        return block

    def print\_previous\_block(self):

        return self.chain[-1]

    def proof\_of\_work(self, previous\_proof):

        new\_proof = 1

        check\_proof = False

        while check\_proof is False:

            hash\_operation = hashlib.sha256(

                str(new\_proof\*\*2 - previous\_proof\*\*2).encode()).hexdigest()

            if hash\_operation[:5] == '00000':

                check\_proof = True

            else:

                new\_proof += 1

        return new\_proof

    def hash(self, block):

        encoded\_block = json.dumps(block, sort\_keys=True).encode()

        return hashlib.sha256(encoded\_block).hexdigest()

    def chain\_valid(self, chain):

        previous\_block = chain[0]

        block\_index = 1

        while block\_index < len(chain):

            block = chain[block\_index]

            if block['previous\_hash'] != self.hash(previous\_block):

                return False

            previous\_proof = previous\_block['proof']

            proof = block['proof']

            hash\_operation = hashlib.sha256(

                str(proof\*\*2 - previous\_proof\*\*2).encode()).hexdigest()

            if hash\_operation[:5] != '00000':

                return False

            previous\_block = block

            block\_index += 1

        return True

app = Flask(\_\_name\_\_)

blockchain = Blockchain()

@app.route('/mine\_block', methods=['GET'])

def mine\_block():

    previous\_block = blockchain.print\_previous\_block()

    previous\_proof = previous\_block['proof']

    proof = blockchain.proof\_of\_work(previous\_proof)

    previous\_hash = blockchain.hash(previous\_block)

    block = blockchain.create\_block(proof, previous\_hash)

    response = {

        'message': 'A block is MINED',

        'index': block['index'],

        'timestamp': block['timestamp'],

        'proof': block['proof'],

        'previous\_hash': block['previous\_hash']

    }

    return jsonify(response), 200

@app.route('/get\_chain', methods=['GET'])

def display\_chain():

    response = {

        'chain': blockchain.chain,

        'length': len(blockchain.chain)

    }

    return jsonify(response), 200

@app.route('/valid', methods=['GET'])

def valid():

    valid = blockchain.chain\_valid(blockchain.chain)

    if valid:

        response = {'message': 'The Blockchain is valid.'}

    else:

        response = {'message': 'The Blockchain is not valid.'}

    return jsonify(response), 200

@app.route('/')

def home():

    return 'Welcome to the Blockchain!'

@app.route('/favicon.ico')

def favicon():

    return '', 204

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(host='127.0.0.1', port=5000)

**Output:**

**A screenshot of a computer

Description automatically generated**

A screen shot of a computer

Description automatically generated

A blue and white rectangle

Description automatically generated

A screenshot of a computer

Description automatically generated