POORNIMA
COLLEGE OF ENGINEERING
An autonomous institution approved by RTU, AICTE & UGC • NAAC A+ Accredited

Department Of Information Technology

Subject: Security Lab **Subject Code: 7IT4-22**

Exp 1:

Implement the following Substitution & Transposition Techniques concepts:

a) Caesar Cipher b) Rail fence row & Column Transformation

a) Caesar Cipher:

```
def encrypt_fun(text,shift):
   result = ""

   for i in range(len(text)):
      character = text[i]

      # Encrypt uppercase characters
      if (character.isupper()):
         result += chr((ord(character) + shift-65) % 26 + 65)

      # Encrypt lowercase characters
      else:
         result += chr((ord(character) + shift - 97) % 26 + 97)

   return result

text = "CaesarCipher"
shift = 3
print ("Text  : " + text)
print ("Shift : " + str(shift))
print ("Cipher: " + encrypt_fun(text,shift))
```

OUTPUT:

```
Text  : CaesarCipher
Shift : 3
Cipher: FdhvduFlskhu
```

b) Rail fence row & Column Transformation :

```
def encryptRailFence(text, key):


    rail = [['\n' for i in range(len(text))]
                for j in range(key)]


    dir_down = False
    row, col = 0, 0

    for i in range(len(text)):


        if (row == 0) or (row == key - 1):
            dir_down = not dir_down


        rail[row][col] = text[i]
        col += 1


        if dir_down:
            row += 1
        else:
            row -= 1

    result = []
    for i in range(key):
        for j in range(len(text)):
            if rail[i][j] != '\n':
                result.append(rail[i][j])
    return("" . join(result))


def decryptRailFence(cipher, key):


    rail = [['\n' for i in range(len(cipher))]
                for j in range(key)]


    dir_down = None
    row, col = 0, 0
```

```
for i in range(len(cipher)):
    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)):
        if ((rail[i][j] == '*') and
        (index < len(cipher))):
            rail[i][j] = cipher[index]
            index += 1


result = []
row, col = 0, 0
for i in range(len(cipher)):


    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))


if __name__ == "__main__":
    print(encryptRailFence("encryptRailFence", 2))
    print(encryptRailFence("encryptRailFence", 3))



    print(decryptRailFence("ecytalecnrpRiFne", 2))
    print(decryptRailFence("eyaenrpRiFnectlc", 3))
```

OUTPUT:

```
ecytalecnrpRiFne
eyaenrpRiFnectlc
encryptRailFence
encryptRailFence
```

Exp 2:

Implement the Diffie-Hellman Key Exchange mechanism using HTML and JavaScript. Consider the end user as one of the parties (Alice) and the JavaScript application as other party (bob).

```
<html>

<body>
<h1> Diffie-Hellman Key Exchange mechanism </h1>
</body>
<script>
function power(a, b, p)
{
   if (b == 1)
      return a;
   else
      return((Math.pow(a, b)) % p);
}

var P, G, x, a, y, b, ka, kb;

P = 23;
document.write("The value of P:" + P + "<br>");

G = 9;
document.write("The value of G:" + G + "<br>");

a = 4;
document.write("The private key a for Alice:" +
            a + "<br>");

x = power(G, a, P);

b = 3;
document.write("The private key b for Bob:" +
            b + "<br>");

y = power(G, b, P);

ka = power(y, a, P); // Secret key for Alice
kb = power(x, b, P); // Secret key for Bob

document.write("Secret key for the Alice is:" +
            ka + "<br>");
document.write("Secret key for the Bob is:" +
            kb + "<br>");



</script>
  </html>
```

Output:

```
Diffie-Hellman Key Exchange mechanism
The value of P:23
The value of G:9
The private key a for Alice:4
The private key b for Bob:3
Secret key for the Alice is:9
Secret key for the Bob is:9
```

Exp 3:

Implement the following Attack: a) Dictionary Attack b) Brute Force Attack

a) Dictionary Attack

```
import hashlib

# List of commonly used passwords and their variations

common_passwords = ["password", "password123", "letmein", "qwerty", "123456",
"abc123", "admin", "welcome", "monkey", "sunshine"]

password_variations = ["", "123", "1234", "12345", "123456", "!", "@", "#", "$", "%", "^",
"&", "*", "(", ")", "-", "_", "+", "=", "/", "\\", "|", "[", "]", "{", "}", "<", ">"]

# Hash of the password to be attacked

hashed_password = hashlib.sha256(b"mypass12#@").hexdigest()

print("Hashed Password is:")

print(hashed_password)

# Try out all possible combinations of common passwords and their variations

for password in common_passwords:

    for variation in password_variations:

        possible_password = password + variation

        hashed_possible_password = hashlib.sha256(possible_password.encode()).hexdigest()

       #print(hashed_possible_password)

        if hashed_possible_password == hashed_password:

            print(f"Password found: {possible_password}")

            break

    else:

        continue

    break

else:

    print("Password not found")
```

OUTPUT:

```
Hashed Password is:

7914d20d1a95f83e938ebc0d0a731b86f1352e49d5aa1d27f1cf1011d32a4528

Password not found
```

b) Brute Force Attack

```
import itertools
import string

def bruteforce_attack(password):
    chars = string.printable.strip()
    attempts = 0
    for length in range(1, len(password) + 1):
        for guess in itertools.product(chars, repeat=length):
            attempts += 1
            guess = ''.join(guess)
            if guess == password:
                return (attempts, guess)
    return (attempts, None)

password = input("Input the password to crack: ")
attempts, guess = bruteforce_attack(password)
if guess:
    print(f"Password cracked in {attempts} attempts. The password is {guess}.")
else:
    print(f"Password not cracked after {attempts} attempts.")
```

OUTPUT:

```
 Input the password to crack: abcd
Password cracked in 9243692 attempts. The password is abcd.
```

Exp 4:

Installation of Wire shark, tcpdump, etc and observe data transferred in client server communication using UDP/TCP and identify the UDP/TCP datagram.

1. **<u>Step Wireshark :</u>**

a. **Download and Install Wireshark:**

- Go to the Wireshark website to download the version that is compatible with your operating system.

- Adhere to the website's installation instructions.

b. **Begin Packet Capturing:**

- Open Wireshark and choose the network interface from which to begin collecting packets.
- The display filter in Wireshark. The display filter in Wireshark's default configuration is a bar that sits right above the column display. Here is where we enter expressions to narrow down what we can see in a pcap file, be it Ethernet frames, IP packets, or TCP segments.
- There are several local interfaces available; please choose one.
- Press the Start button.
- In essence, you are recording and intercepting data packets as they pass through a network interface when you capture packets.

c. **Analyze Packets:**
- Wireshark will show packets as they come through the chosen interface in real time.To limit the packets that are shown based on parameters such as source, destination, protocol, etc., you can apply filters.

2. **<u>Step tcpdump :</u>**

- **Launch a Terminal or Command Prompt:**
On Unix-based systems, open a terminal window. As an administrator, run the Command Prompt on Windows.

- **Begin Packet Capturing:**
In the first case, run dumpcap -i <interface>-w<output_file>, where <interface> is the network interface that you choose to start capturing from.

- **View Captured Packets:**
tcpdump will present captured packets in a readable format on the terminal window.