Name:-Amrit Ranjan

Reg No:-21BCE1847

Cryptography And Network Security

Assignment 7

Faculty Name:-Dr Tapabarta Roy

RC4 Encryption and Decryption



Ques:-Write down a program to implement RC4 encryption and decryption choosing your own parameters.

Code:-

# Function for encryption

def encryption():

    global key, plain\_text, n

    # Given text and key

    plain\_text = "001010010010"

    key = "101001000001"

    # n is the no: of bits to

    # be considered at a time

    n = 3

    print("Plain text : ", plain\_text)

    print("Key : ", key)

    print("n : ", n)

    print(" ")

    # The initial state vector array

    S = [i for i in range(0, 2\*\*n)]

    print("S : ", S)

    key\_list = [key[i:i + n] for i in range(0, len(key), n)]

    # Convert to key\_stream to decimal

    for i in range(len(key\_list)):

        key\_list[i] = int(key\_list[i], 2)

    # Convert to plain\_text to decimal

    global pt

    pt = [plain\_text[i:i + n] for i in range(0, len(plain\_text), n)]

    for i in range(len(pt)):

        pt[i] = int(pt[i], 2)

    print("Plain text ( in array form ): ", pt)

    # Making key\_stream equal

    # to length of state vector

    diff = int(len(S)-len(key\_list))

    if diff != 0:

        for i in range(0, diff):

            key\_list.append(key\_list[i])

    print("Key list : ", key\_list)

    print(" ")

    # Perform the KSA algorithm

    def KSA():

        j = 0

        N = len(S)

        # Iterate over the range [0, N]

        for i in range(0, N):

            # Find the key

            j = (j + S[i]+key\_list[i]) % N

            # Update S[i] and S[j]

            S[i], S[j] = S[j], S[i]

            print(i, " ", end ="")

            # Print S

            print(S)

        initial\_permutation\_array = S

        print(" ")

        print("The initial permutation array is : ",

            initial\_permutation\_array)

    print("KSA iterations : ")

    print(" ")

    KSA()

    print(" ")

    # Perform PGRA algorithm

    def PGRA():

        N = len(S)

        i = j = 0

        global key\_stream

        key\_stream = []

        # Iterate over [0, length of pt]

        for k in range(0, len(pt)):

            i = (i + 1) % N

            j = (j + S[i]) % N

            # Update S[i] and S[j]

            S[i], S[j] = S[j], S[i]

            print(k, " ", end ="")

            print(S)

            t = (S[i]+S[j]) % N

            key\_stream.append(S[t])

        # Print the key stream

        print("Key stream : ", key\_stream)

        print(" ")

    print("PGRA iterations : ")

    print(" ")

    PGRA()

    # Performing XOR between generated

    # key stream and plain text

    def XOR():

        global cipher\_text

        cipher\_text = []

        for i in range(len(pt)):

            c = key\_stream[i] ^ pt[i]

            cipher\_text.append(c)

    XOR()

    # Convert the encrypted text to

    # bits form

    encrypted\_to\_bits = ""

    for i in cipher\_text:

        encrypted\_to\_bits += '0'\*(n-len(bin(i)[2:]))+bin(i)[2:]

    print(" ")

    print("Cipher text : ", encrypted\_to\_bits)

encryption()

print("---------------------------------------------------------")

# Function for decryption of data

def decryption():

    # The initial state vector array

    S = [i for i in range(0, 2\*\*n)]

    key\_list = [key[i:i + n] for i in range(0, len(key), n)]

    # Convert to key\_stream to decimal

    for i in range(len(key\_list)):

        key\_list[i] = int(key\_list[i], 2)

    # Convert to plain\_text to decimal

    global pt

    pt = [plain\_text[i:i + n] for i in range(0, len(plain\_text), n)]

    for i in range(len(pt)):

        pt[i] = int(pt[i], 2)

    # making key\_stream equal

    # to length of state vector

    diff = int(len(S)-len(key\_list))

    if diff != 0:

        for i in range(0, diff):

            key\_list.append(key\_list[i])

    print(" ")

    # KSA algorithm

    def KSA():

        j = 0

        N = len(S)

        # Iterate over the range [0, N]

        for i in range(0, N):

            j = (j + S[i]+key\_list[i]) % N

            # Update S[i] and S[j]

            S[i], S[j] = S[j], S[i]

            print(i, " ", end ="")

            print(S)

        initial\_permutation\_array = S

        print(" ")

        print("The initial permutation array is : ",

            initial\_permutation\_array)

    print("KSA iterations : ")

    print(" ")

    KSA()

    print(" ")

    # Perform PRGA algorithm

    def do\_PGRA():

        N = len(S)

        i = j = 0

        global key\_stream

        key\_stream = []

        # Iterate over the range

        for k in range(0, len(pt)):

            i = (i + 1) % N

            j = (j + S[i]) % N

            # Update S[i] and S[j]

            S[i], S[j] = S[j], S[i]

            print(k, " ", end ="")

            print(S)

            t = (S[i]+S[j]) % N

            key\_stream.append(S[t])

    print("Key stream : ", key\_stream)

    print(" ")

    print("PGRA iterations : ")

    print(" ")

    do\_PGRA()

    # Perform XOR between generated

    # key stream and cipher text

    def do\_XOR():

        global original\_text

        original\_text = []

        for i in range(len(cipher\_text)):

            p = key\_stream[i] ^ cipher\_text[i]

            original\_text.append(p)

    do\_XOR()

    # convert the decrypted text to

    # the bits form

    decrypted\_to\_bits = ""

    for i in original\_text:

        decrypted\_to\_bits += '0'\*(n-len(bin(i)[2:]))+bin(i)[2:]

    print(" ")

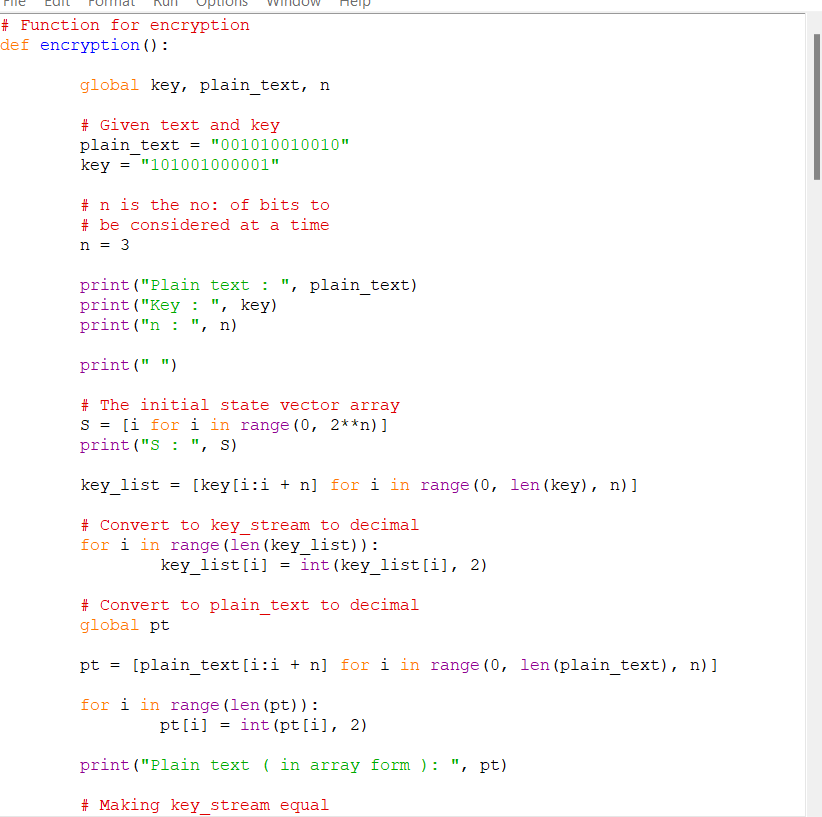
    print("Decrypted text : ",

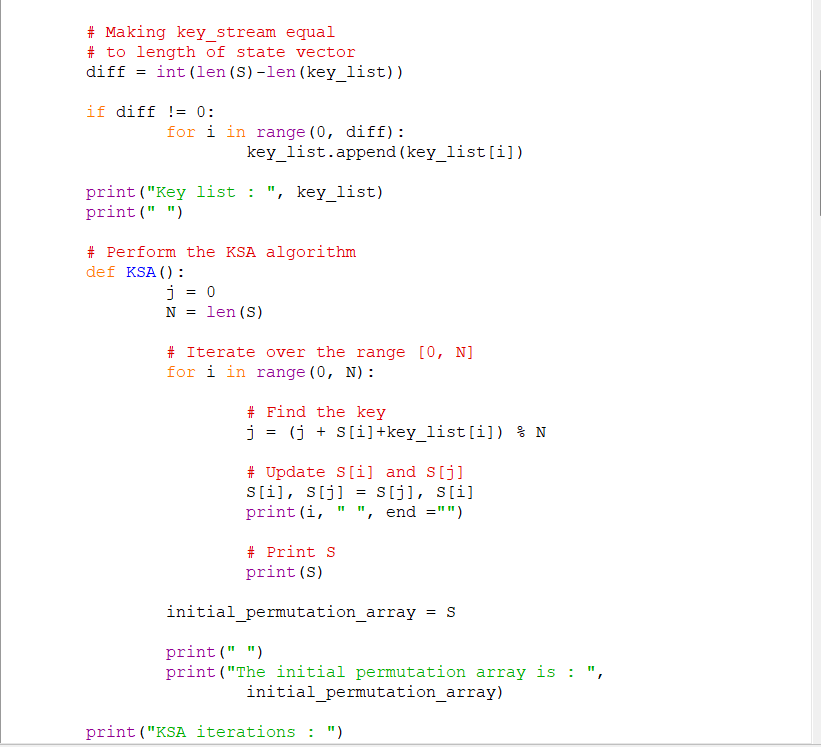
        decrypted\_to\_bits)

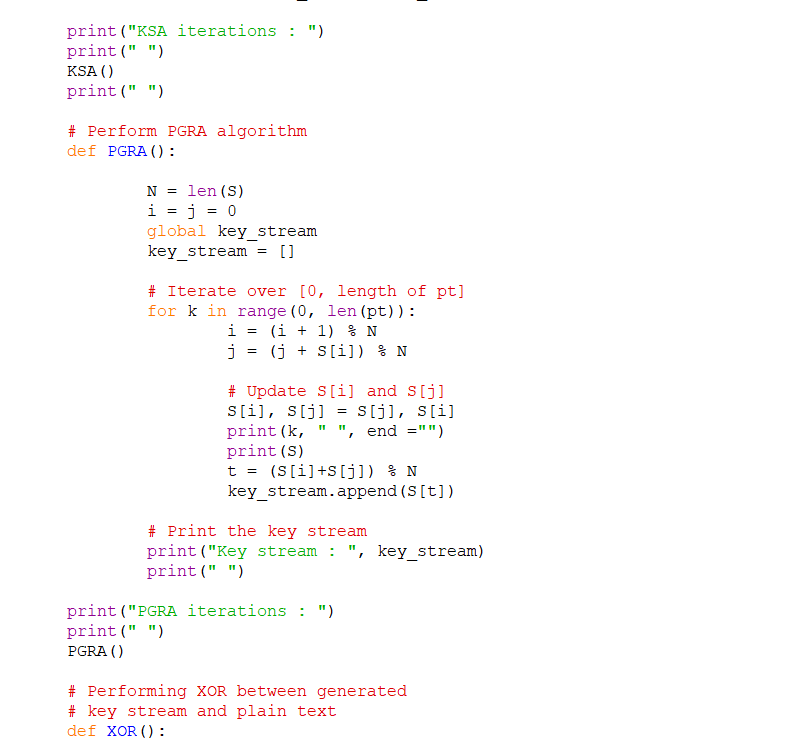
# Driver Code

decryption()

Code ScreenShots:-







Code Output:-

