**(1) Implement RC4 in any programming language and demonstrate encryption and decryption of a plaintext stream.**

**RC4 Algorithm Implementation**

**Source 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()

**Output:**  
Plain text : 001010010010

Key : 101001000001

n : 3

S : [0, 1, 2, 3, 4, 5, 6, 7]

Plain text ( in array form ): [1, 2, 2, 2]

Key list : [5, 1, 0, 1, 5, 1, 0, 1]

KSA iterations :

0 [5, 1, 2, 3, 4, 0, 6, 7]

1 [5, 7, 2, 3, 4, 0, 6, 1]

2 [5, 2, 7, 3, 4, 0, 6, 1]

3 [5, 2, 7, 0, 4, 3, 6, 1]

4 [5, 2, 7, 0, 6, 3, 4, 1]

5 [5, 2, 3, 0, 6, 7, 4, 1]

6 [5, 2, 3, 0, 6, 7, 4, 1]

7 [1, 2, 3, 0, 6, 7, 4, 5]

The initial permutation array is : [1, 2, 3, 0, 6, 7, 4, 5]

PGRA iterations :

0 [1, 3, 2, 0, 6, 7, 4, 5]

1 [1, 3, 6, 0, 2, 7, 4, 5]

2 [1, 3, 6, 2, 0, 7, 4, 5]

3 [1, 3, 6, 2, 0, 7, 4, 5]

Key stream : [7, 1, 6, 1]

Cipher text : 110011100011

---------------------------------------------------------

KSA iterations :

0 [5, 1, 2, 3, 4, 0, 6, 7]

1 [5, 7, 2, 3, 4, 0, 6, 1]

2 [5, 2, 7, 3, 4, 0, 6, 1]

3 [5, 2, 7, 0, 4, 3, 6, 1]

4 [5, 2, 7, 0, 6, 3, 4, 1]

5 [5, 2, 3, 0, 6, 7, 4, 1]

6 [5, 2, 3, 0, 6, 7, 4, 1]

7 [1, 2, 3, 0, 6, 7, 4, 5]

The initial permutation array is : [1, 2, 3, 0, 6, 7, 4, 5]

Key stream : [7, 1, 6, 1]

PGRA iterations :

0 [1, 3, 2, 0, 6, 7, 4, 5]

1 [1, 3, 6, 0, 2, 7, 4, 5]

2 [1, 3, 6, 2, 0, 7, 4, 5]

3 [1, 3, 6, 2, 0, 7, 4, 5]

Decrypted text : 001010010010

(2) Plot the key stream to show its randomness.

import matplotlib.pyplot as plt

def generate\_keystream\_plot():

global S, pt

def encryption():

global key, plain\_text, n

# Given text and key

plain\_text = "001011111010"

key = "000101111101"

# 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)

# Plotting the randomness of the keystream

keystream = key\_stream # Assuming key\_stream is the generated keystream

print("Keystream: ", keystream)

plt.figure(figsize=(15, 5))

# Plot the numbers

plt.plot(keystream)

# Customize the plot (optional)

plt.title('Randomness of Keystream')

plt.xlabel('Iteration')

plt.ylabel('Generated Number')

# Display the plot

plt.show()

# Run the encryption function to generate keystream and plot its randomness

generate\_keystream\_plot()

