**Laboratory Report Cover Sheet**

|  |
| --- |
| SRM Institute of Science and Technology  College of Engineering and Technology  Department of Electronics and Communication Engineering |
| **18ECC303J Computer Communication and Networks**  **Sixth Semester, 2022-23 (Even semester)** |

Experiment based Project

On

**Development and Implementation of RC4 cryptography Algorithm**

|  |  |  |  |
| --- | --- | --- | --- |
| **Topic** | **Mark** | **RA2011004010051** | **RA2011004010052** |
| **Kunal Keshan** | **Vaibhav Mohla** |
| Proposal Submission | 5 |  |  |
| Demonstration | 10 |  |  |
| Presentation | 10 |  |  |
| Document Preparation | 10 |  |  |
| Viva | 5 |  |  |
| **Total** | **40** |  |  |

**REPORT VERIFICATION**

**Date :**

**Staff Name :**

**Signature :**

**Title of the project:**

Implementation of RC4 Encryption and Decryption Algorithms Using Python.

**Name of the members of the Group with Reg. Nos.:**

Siva Ramachandran – RA2011004010049

Kunal Keshan – RA2011004010051

Vaibhav Mohla – RA2011004010052

**Introduction:**

The RC4 algorithm is a symmetric stream cipher used for encryption and

decryption of data. It was developed by Ron Rivest in 1987 and was initially used

for secure communication in the Internet Engineering Task Force (IETF). The

algorithm is known for its simplicity and speed and is widely used in various

applications.

**Software used:** Python 3.8, Pycharm IDE

**RC4 algorithm Theory/Explanation:**

RC4 (Rivest Cipher 4) is a symmetric stream cipher algorithm that is widely

used in various applications, such as secure socket layer (SSL), wireless

network encryption, and Bluetooth. It operates on bytes of data and generates a

stream of pseudo-random bytes, which are XOR-ed with the plaintext to

produce the ciphertext. RC4 encryption and decryption is achieved by using a

secret key of arbitrary length between 40 and 2048 bits.

The RC4 algorithm consists of two main stages: key-scheduling and stream

generation. In the key-scheduling stage, the algorithm creates a permutation of

the 256-byte array, based on the secret key and an initialization vector (IV). The

permutation is achieved by performing a series of swaps between elements of

the array, depending on the key and IV. In the stream generation stage, the

algorithm generates a pseudo-random stream of bytes by repeatedly swapping

elements of the array and generating a byte from the array index, based on the

current state of the algorithm. This stream is XOR-ed with the plaintext to

produce the ciphertext, and vice versa for decryption.

Overall, RC4 is a simple, fast, and widely-used stream cipher that provides a

good level of security for various applications. However, it is vulnerable to

certain attacks, such as key recovery attacks and related key attacks, and

therefore should not be used as the sole security mechanism for critical

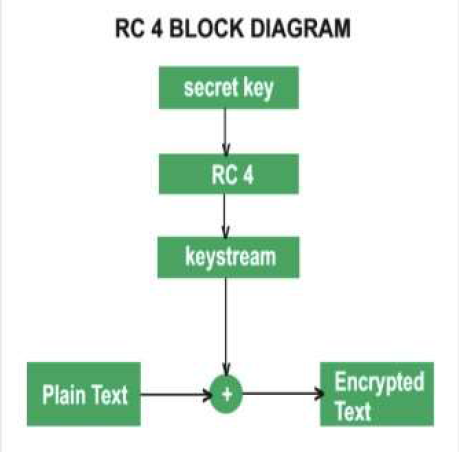
applications.

To encrypt a plaintext message using RC4, the pseudorandom stream is

XORed with the plaintext to produce the ciphertext. To decrypt the ciphertext,

the same pseudorandom stream is XORed with the ciphertext to recover the

original plaintext.



**Implementation Methodology or steps**

RC4 algorithm can be implemented using Python by following these steps:

1. Define a function to perform the key-scheduling stage, which takes the

secret key as input and generates the permutation of the 256-byte array. 2.

Define a function to generate the pseudo-random stream of bytes, which takes

the permutation array as input and generates a stream of bytes to XOR with

the plaintext.

3. Define a function to perform the encryption, which takes the plaintext

and secret key as input, and generates the ciphertext by XOR-ing the plaintext

with the stream of bytes generated in step 2.

4. Define a function to perform the decryption, which takes the ciphertext

and secret key as input, and generates the plaintext by XOR-ing the ciphertext

with the same stream of bytes generated in step 2.

5. Test the encryption and decryption functions with sample plaintext and

secret key, and verify that the output matches the expected ciphertext and

plaintext, respectively.

**Programme:**

import random

def Key\_Scheduling(key):

key\_length = len(key)

if key\_length > 256:

raise ValueError("Key too long (max length = 256)")

S = list(range(256))

j = 0

for i in range(256):

j = (j + S[i] + key[i % key\_length]) % 256

S[i], S[j] = S[j], S[i]

return S

def pad\_key(key):

padded\_key = bytearray(256)

key\_len = len(key)

if key\_len > 256:

raise ValueError("Key too long (max length = 256)")

padded\_key[:key\_len] = bytearray(key.encode())

padded\_key[key\_len:] = bytearray(256 - key\_len)

return padded\_key

def stream\_generation(S):

i = 0

j = 0

while True:

i = (i + 1) % 256

j = (j + S[i]) % 256

S[i], S[j] = S[j], S[i]

K = S[(S[i] + S[j]) % 256]

yield K

def encrypt(plaintext, key):

key = pad\_key(key)

S = Key\_Scheduling(key)

keystream = stream\_generation(S)

encrypted\_text = ""

for c in plaintext:

if c.isalpha():

if c.islower():

c = chr((ord(c) + next(keystream) - 97) % 26 + 97)

elif c.isupper():

c = chr((ord(c) + next(keystream) - 65) % 26 + 65)

elif c.isnumeric():

c = str((int(c) + next(keystream)) % 10)

encrypted\_text += c

return encrypted\_text

def decrypt(ciphertext, key):

key = pad\_key(key)

S = Key\_Scheduling(key)

keystream = stream\_generation(S)

decrypted\_text = ""

for c in ciphertext:

if c.isalpha():

if c.islower():

c = chr((ord(c) - next(keystream) - 97) % 26 + 97)

elif c.isupper():

c = chr((ord(c) - next(keystream) - 65) % 26 + 65)

elif c.isnumeric():

c = str((int(c) - next(keystream)) % 10)

decrypted\_text += c

return decrypted\_text

ed = input('Enter 1 for Encrypt, or 2 for Decrypt: ').upper()

if ed == '1':

text= input("Enter a plaintext: ")

key=input("Enter the key: ")

encrypted\_text = encrypt(text, key)

print("Cipher text is:", encrypted\_text)

elif ed == '2':

TEXT = input("Enter cipher text: ")

KEY = input("Enter key: ")

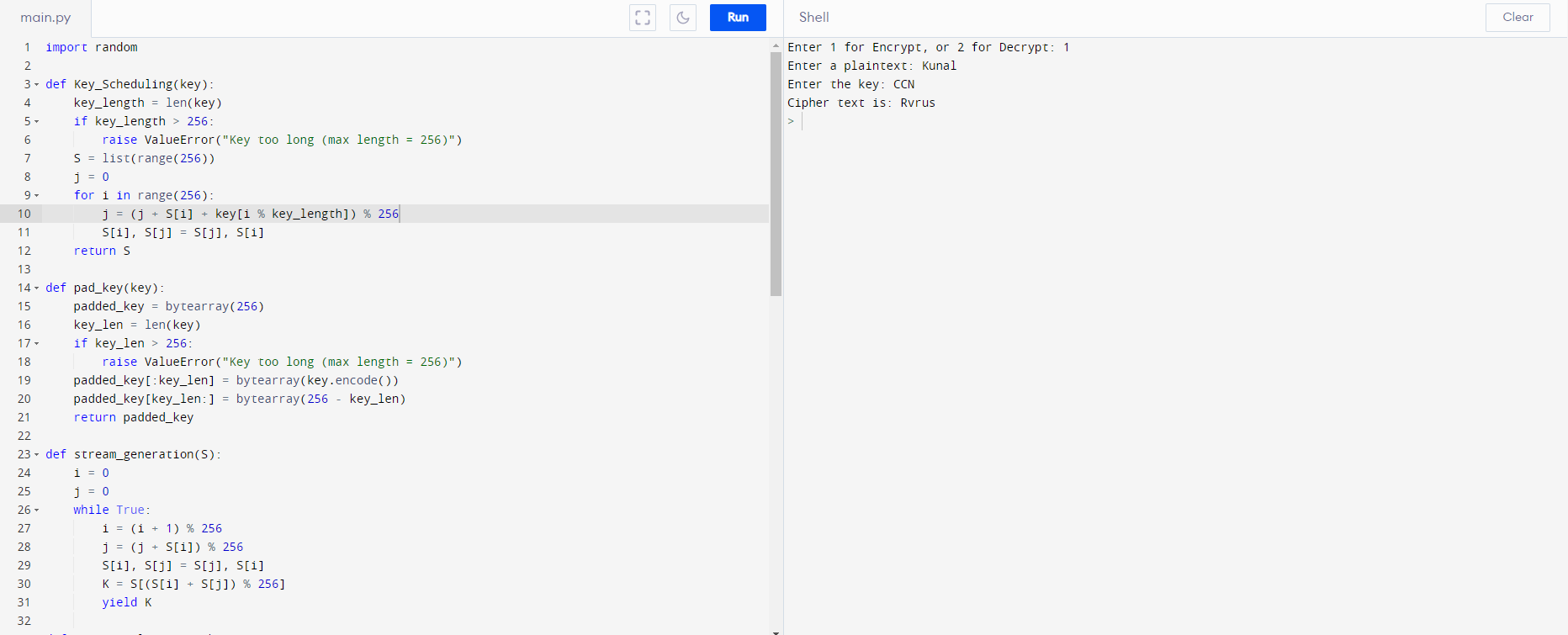
decrypted\_text = decrypt(TEXT, KEY)

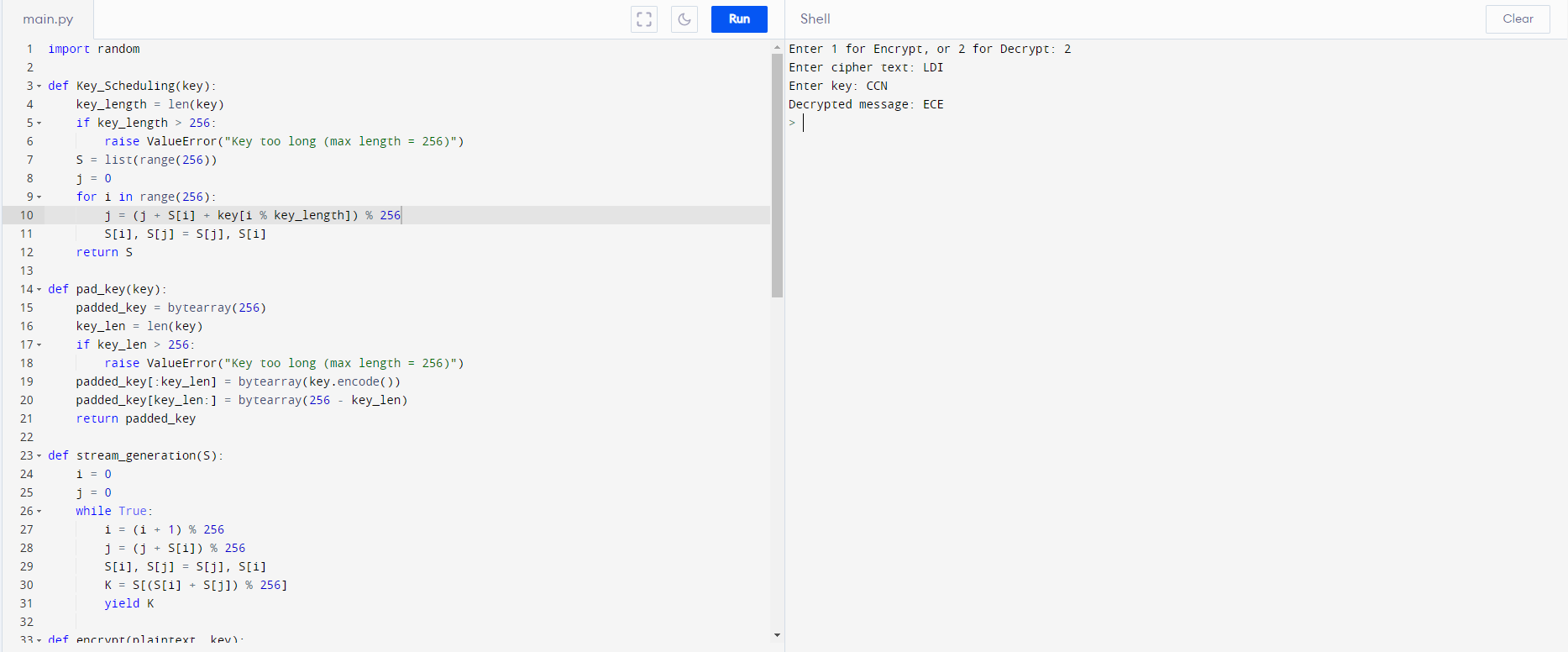
print("Decrypted message:", decrypted\_text)

else:

print('Error in input - try again.')

**Demonstration Result and Screenshots:**

****

****

**Tabulation  
Key: CCN**

|  |  |  |
| --- | --- | --- |
| S.No | Plain text | Cipher Text |
| 1 | Kunal | Rvrus |
| 2 | Sivaram | Zjzuybx |
| 3 | Vaibhav | Cbmvobg |
| 4 | SRM | ZSQ |
| 5 | ECE | LDI |

**Conclusion:**

The RC4 algorithm is a widely used stream cipher for encryption and decryption

of data. It is known for its simplicity and speed. In this report, we have explained

the theory behind the RC4 algorithm, its implementation using Python and the

software used. And the functions that provides a convenient and easy-to-use

interface for implementing the RC4 algorithm in Python.

**Reference List**

1. Data communications and networking I Behrouz A Forouzan

2. <https://www.youtube.com/watch?v=Pl-ySf5abv8>