Savitribai Phule Pune University

S.Y.B.Sc.(Cyber and Digital Science)

SEMESTER- IV

(CDS-246) Lab Course on CDS- 243
(Network Security and Cryptography)

Lab Book

Name:____

College Name:____________________________________

Roll No.:__________________**Division:**__________________

Academic Year : __________________________________

CERTIFICATE

This is to certify that Mr./Ms.____

Seat Number ___________of **S.Y.BSc.(Cyber and Digital Science) Sem-IV** Has successfully completed Laboratory Course on CDS-243 Network Security and Cryptography Workbook Course the year ______________. He/She has scored marks _______out of 10.

Subject Teacher **H.O.D./Coordinator**

Internal Examiner **External Examiner**

Introduction

1. **About the Workbook:**
   This workbook is intended to be used by S.Y. B.Sc. (Cyber and Digital Science) students for Network Security and Cryptography in Sem- IV. This workbook is designed by considering all the practical concepts topics mentioned in the syllabus.

2. **The objectives of this Workbook are:**
   1) Defining the scope of the course.
   2) To bring the uniformity in the practical conduction and implementation in all colleges affiliated to SPPU.
   3) To have continuous assessment of the course and students.
   4) Providing ready reference for the students during practical implementation.
   5) Provide more options to students so that they can have good practice before facing the examination.
   6) Catering to the demand of slow and fast learners and accordingly providing the practice assignments to them.

3. **How to use this Workbook:**
   The workbook contains Eight assignments.
   It is mandatory for students to complete all the assignments in given slot.

4. **Instructions to the students:**
   Please read the following instructions carefully and follow them.

1) Students are expected to carry this workbook every time they come to the lab for practical.
2) Students should prepare for the assignment by reading the relevant material which is mentioned in ready reference.
3) Instructor will specify which problems to solve in the lab during the allotted slot and student should complete them and get verified by the instructor. However, student should spend additional hours in Lab and at home to cover all workbook assignments if needed.

- Students will be assessed for each assignment on a scale from 0 to 5

Not done	0
Incomplete	1
Late Complete	2
Needs improvement	3
Complete	4
Well Done	5

5. **Instruction to the Instructors:**

Make sure that students should follow above instructions.

- Explain the assignment and related concepts using white board if required or by demonstrating the software.
- Give specific input to fill the blanks in queries which can vary from student to student.
- Evaluate each assignment carried out by a student on a scale of 5 as specified above by ticking appropriate box.
- The value should also be entered on assignment completion page of the respective Lab course.

6. **Instructions to the Lab administrator:**
You have to ensure appropriate hardware and software is made available to each student.

Editors:

1) **Mr. Satyavan Kunjir**	Dr. D. Y. Patil ACS College, Pimpri, Pune
2) **Mr. Sammed Bukshete**	Dr. D. Y. Patil ACS College, Pimpri, Pune
3) **Mr. Omkar K Dengi**	Dr. D. Y. Patil ACS College, Pimpri, Pune

Reviewed By:

4) **Dr. Ranjit D. Patil**	Dr. D. Y. Patil ACS College, Pimpri, Pune
5) **Dr. Sujata Patil**	Dr. D. Y. Patil ACS College, Pimpri, Pune

Assignment Index

Sr. No.	Assignment Name	Page Number
1	Network Security Essentials (Sample Case Studies)	7
2	Substitution Techniques	10
3	Implementation of DES & AES algorithm	13
4	Cryptanalysis	16
5	Symmetric Algorithms	19
6	Asymmetric Algorithms	23
7	Review papers on cryptography and network security	31

Signature of Instructor:__________________________

Assignment Completion Sheet

Sr. No.	**Assignment Name**	**Marks (out of 5)**	**Teacher's Sign**
1	Network Security Essentials (Sample Case Studies)		
2	Substitution Techniques		
3	Implementation of DES & AES algorithm		
4	Cryptanalysis		
5	Symmetric Algorithms		
6	Asymmetric Algorithms		
7	Review papers on cryptography and network security		
Total (Out of 35)			
Total (Out of 5)			

Signature of Instructor:___________________________

Assignment 1:
Network Security Essentials (Sample Case Studies)

A] Understanding CIA using ATM System

Objective:

To understand the security requirements associated with an ATM system and their importance.

Requirements:

- ☐ Basic understanding of an ATM system
- ☐ Familiarity with security concepts such as confidentiality, integrity, and availability
- ☐ Knowledge of potential threats to an ATM system

Instructions:

Provide an overview of the ATM system and its basic functioning. Explain thE importance of security in an ATM system.

- ➢ **Confidentiality**: Describe the confidentiality requirements of an ATM system. Explain how PINs and card information must be kept confidential to prevent unauthorized access to user accounts. Discuss the importance of confidentiality in an ATM system and the potential consequences of a breach.
- ➢ **Integrity**: Describe the integrity requirements of an ATM system. Explain how the system must ensure that transactions are not altered or tampered with during transmission or processing. Discuss the importance of integrity in an ATM system and the potential consequences of a breach.
- ➢ **Availability**: Describe the availability requirements of an ATM system. Explain how the system must be available to users at all times for transactions. Discuss the importance of availability in an ATM system and the potential consequences of a breach.

1. Describe how an ATM system works, including the process of entering a PIN and using a card for account access.
2. Explain the concept of confidentiality, including examples of sensitive information that should be kept confidential in an ATM system.
3. Discuss the importance of confidentiality in an ATM system and the potential consequences of a breach.
4. Define integrity and provide examples of how it applies to an ATM system.
5. Discuss the importance of integrity in an ATM system and the potential consequences of data tampering.
6. Define availability and provide examples of how it applies to an ATM system.

7. Discuss the importance of availability in an ATM system and the potential consequences of downtime or system failure.
8. Summarize the importance of confidentiality, integrity, and availability in an ATM system and prioritize their importance.

Threats: Discuss the potential threats to an ATM system such as skimming, card trapping, and PIN theft. Explain how these threats can compromise the confidentiality, integrity, and availability of an ATM system.

Conclusion: Summarize the importance of security in an ATM system and the need for confidentiality, integrity, and availability. Emphasize the potential consequences of a breach and the need for constant vigilance to ensure the security of the system.

B] Importance of CIA in Desktop Publishing System

Objective:

To understand the importance of confidentiality, data integrity, and system availability requirements in a desktop publishing system.

\

Requirements:

- ☐ A desktop publishing system
- ☐ Understanding of the types of publications that require confidentiality, data integrity, and system availability.

Instructions:

➢ **Confidentiality Requirement:**

- Example of a type of publication for which confidentiality of the stored data is the most important requirement is a financial report. The financial report contains sensitive information such as financial statements, balance sheets, and income statements.
- Unauthorized access to this information can lead to financial loss and damage to the organization's reputation. Therefore, it is essential to ensure that the stored data is protected and kept confidential to prevent unauthorized access.

➢ **Data Integrity Requirement:**

- Example of a type of publication in which data integrity is the most important requirement is a medical report. The medical report contains critical information such as the patient's medical history, current diagnosis, and treatment plan. Any alteration or manipulation of this information can have serious consequences, such as incorrect diagnosis or incorrect treatment. Therefore, it is essential to ensure that the stored data is accurate and has not been tampered with to maintain data integrity.

➢ **System Availability Requirement:**

- Example of a type of publication in which system availability is the most important requirement is a daily newspaper. The daily newspaper needs to be published and distributed on time, without delay. Any disruption in the publishing system can lead to a delay in the publication of the newspaper, which can result in a loss of readership and revenue. Therefore, it is essential to ensure that the publishing system is available and functioning correctly to maintain system availability

.

Assignment Evaluation

0: Not Done [ ]	**1: Incomplete [ ]**	**2: Late Complete [ ]**
3: Need Improvement [ ]	**4: Complete [ ]**	**5: Well Done [ ]**

Signature of Instructor:______________________________

Assignment 2:
Substitution Techniques

A] Implement a Caesar Cipher Encryption

Objective:

To implement Caesar Cipher encryption in Python.

Requirements:

- ☐ Python programming environment
- ☐ Text editor or IDE

Instructions:

1. Define a function that takes in a plaintext message and a shift value (e.g. 3).
2. Convert the message to all uppercase characters.
3. Loop through each character in the message and shift it by the shift value (using the ord() and chr() functions).
4. If the shifted character goes beyond the range of A-Z, wrap around to the beginning of the alphabet.
5. Return the encrypted message.

Example Code:

```
def caesar_cipher_encrypt(plaintext, shift):
    encrypted_message = ''
    for char in plaintext.upper():
        if char.isalpha():
            shifted_char = chr(((ord(char) - ord('A') + shift) % 26) + ord('A'))
            encrypted_message += shifted_char
        else:
            encrypted_message += char
    return encrypted_message

def main():
    plaintext = input("Enter the plaintext message: ")
    shift = int(input("Enter the shift value: "))

    encrypted_message = caesar_cipher_encrypt(plaintext, shift)
    print("Encrypted message:", encrypted_message)

if __name__ == "__main__":
    main()
```

B] Monoalphabetic Ciphers

Objective:

To implement Monoalphabetic Cipher encryption in Python.

Requirements:

- ☐ Python programming environment.
- ☐ Text editor or IDE

Instructions:

1. Define a dictionary that maps each letter of the alphabet to a corresponding letter (e.g. A -> Q, B->Z,, etc…)
2. Define a function that takes in a plaintext message and the dictionary mapping.
3. Convert the message to all uppercase characters.
4. Loop through each character in the message and substitute it with the corresponding letter from the dictionary.
5. Return the encrypted message.

Example Code:

```
def monoalphabetic_cipher_encrypt(plaintext, mapping):
    plaintext = plaintext.upper()
    encrypted_message = ''
    for char in plaintext:
        if char.isalpha():
            encrypted_message += mapping[char]
        else:
            encrypted_message += char
    return encrypted_message

def main():
    # Define the mapping dictionary
    mapping = {
        'A': 'Q', 'B': 'Z', 'C': 'R', 'D': 'T', 'E': 'Y', 'F': 'U', 'G': 'P',
        'H': 'S', 'I': 'O', 'J': 'K', 'K': 'W', 'L': 'X', 'M': 'V', 'N': 'N',
        'O': 'M', 'P': 'B', 'Q': 'C', 'R': 'D', 'S': 'E', 'T': 'F', 'U': 'G',
        'V': 'H', 'W': 'I', 'X': 'J', 'Y': 'A', 'Z': 'L'
    }

    # Input plaintext message
    plaintext = input("Enter the plaintext message: ")

    # Encrypt the plaintext message
    encrypted_message = monoalphabetic_cipher_encrypt(plaintext, mapping)

    # Output the encrypted message
    print("Encrypted message:", encrypted_message)

if __name__ == "__main__":
    main()
```

Assignment Evaluation

0: Not Done [ ]	**1: Incomplete [ ]**	**2: Late Complete [ ]**
3: Need Improvement [ ]	**4: Complete [ ]**	**5: Well Done [ ]**

Signature of Instructor:_______________________________

Assignment 3:
Implementation of DES & AES algorithm

A] Implementation of DES algorithm in python.

Objectives:

Understand the principles of cryptography

Requirements:

- ☐ Python 3
- ☐ Basic understanding of number theory and modular arithmetic
- ☐ PyCryptodome library (for generating random prime numbers and modular inverse)

Instructions:

1. Import Necessary Modules: Import the necessary modules from Crypto.Cipher and Crypto.Random to use DES encryption and generate random bytes.
2. Define Encryption Function:
   Create a function to encrypt the message using DES encryption.
   Use DES.new(key, DES.MODE_ECB) to create a DES cipher object with the provided key in ECB mode.
   Use the encrypt method of the cipher object to encrypt the message.
3. Define Decryption Function:
   Create a function to decrypt the ciphertext using DES decryption.
   Use DES.new(key, DES.MODE_ECB) to create a DES cipher object with the provided key in ECB mode.
   Use the decrypt method of the cipher object to decrypt the ciphertext.
4. Generate Random Key:
   Generate a random 8-byte key using get_random_bytes(8) from Crypto.Random.
5. Encrypt Message:
   Convert the message to bytes if it's not already in bytes.
   Call the encryption function with the message and the generated key to encrypt the message.
6. Decrypt Message:
   Call the decryption function with the ciphertext and the generated key to decrypt the message.
7. Handle Message:
   Ensure proper handling of byte strings and string conversions when working with encryption and decryption.
8. Testing:
   Test the implementation by encrypting and decrypting messages to verify correctness.

Example code:

```
from Crypto.Cipher import DES
from Crypto.Random import get_random_bytes


def encrypt_message(message, key):
    cipher = DES.new(key, DES.MODE_ECB)
    ciphertext = cipher.encrypt(message)
    return ciphertext

def decrypt_message(ciphertext, key):
    cipher = DES.new(key, DES.MODE_ECB)
    decrypted_message = cipher.decrypt(ciphertext)
    return decrypted_message

def main():
    key = get_random_bytes(8)  # Generate a random 8-byte key
    message = b"Hello, DES!"   # Convert message to bytes

    # Encrypt the message
    ciphertext = encrypt_message(message, key)
    print("Encrypted message:", ciphertext.hex())

    # Decrypt the message
    decrypted_message = decrypt_message(ciphertext, key)
    print("Decrypted message:", decrypted_message.decode())

if __name__ == "__main__":
    main()
```

B: Implementation of AES algorithm in python.

Objectives:

Understand the principles of cryptography

Requirements:

- ☐ Python 3
- ☐ Basic understanding of number theory and modular arithmetic
- ☐ PyCryptodome library (for generating random prime numbers and modular inverse)

Example code:

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

def encrypt_message(message, key):
    cipher = AES.new(key, AES.MODE_ECB)
    ciphertext = cipher.encrypt(message)
    return ciphertext

def decrypt_message(ciphertext, key):
    cipher = AES.new(key, AES.MODE_ECB)
    decrypted_message = cipher.decrypt(ciphertext)
    return decrypted_message

def main():
    key = get_random_bytes(16)  # Generate a random 16-byte key for AES-128
    message = b"Hello, AES!"     # Convert message to bytes

    # Encrypt the message
    ciphertext = encrypt_message(message, key)
    print("Encrypted message:", ciphertext.hex())

    # Decrypt the message
    decrypted_message = decrypt_message(ciphertext, key)
    print("Decrypted message:", decrypted_message.decode())

if __name__ == "__main__":
    main()
```

Assignment Evaluation

0: Not Done [ ]	**1: Incomplete [ ]**	**2: Late Complete [ ]**
3: Need Improvement [ ]	**4: Complete [ ]**	**5: Well Done [ ]**

Signature of Instructor:___________________________________

Assignment 4:
Cryptanalysis

Objectives:

1. Understand the cryptography.
2. To Implement linear cryptanalysis in python.

Requirements:

- ☐ Python 3
- ☐ Basic understanding of number theory and modular arithmetic

Instructions:

1. Define S-Box:: Define the S-box as a lookup table mapping input values to output values.
2. Define Linear Approximation Table:Define a linear approximation table representing the linear relationship between input and output bits of the S-box.
3. Linear Cryptanalysis Function:

   Initialize a dictionary count to store the count of pairs.

   Loop through all possible input and output pairs for the S-box:

   > Calculate input/output pairs for each bit of the S-box.
   >
   > Calculate the inner product of the input/output pairs and the linear approximation table.
   >
   > Increment the count for the corresponding inner product in the count dictionary.

   Find the most probable key(s) by identifying the highest count(s) in the count dictionary.
4. Main Function:

   Call the linear cryptanalysis function.

   Print the most probable key(s) found by linear cryptanalysis.

Example code:

```
# Simple S-box
s_box = [
  0x6, 0x4, 0xc, 0x5,
  0x0, 0x7, 0x2, 0xe,
  0x1, 0xf, 0x3, 0xd,
  0x8, 0xa, 0x9, 0xb
]
```

```
# Linear approximation table for the S-box
linear_approximation_table = [
    [0, 0, 0, 0],
    [0, 1, 1, 0],
    [0, 1, 1, 0],
    [0, 0, 1, 1],
    [0, 0, 1, 1],
    [0, 1, 0, 0],
    [0, 1, 0, 0],
    [0, 0, 0, 0]
]

# Linear cryptanalysis function
def linear_cryptanalysis():
    # Initialize a dictionary to store the count of pairs
    count = {}

    # Loop through all possible input and output pairs
    for i in range(16):
        for j in range(16):
            # Calculate input/output pairs for each S-box
            in_pairs = [(i >> k) & 0x1 for k in range(4)]
            out_pairs = [(s_box[in_pair] >> j) & 0x1 for in_pair in in_pairs]

            # Calculate the inner product of the input/output pairs and the linear approximation table
            inner_product = sum(in_pairs[k] * linear_approximation_table[k][j] for k in range(4))

            # Increment the count for the corresponding inner product
            if inner_product in count:
                count[inner_product] += 1
            else:
                count[inner_product] = 1

    # Find the most probable key
    max_count = max(count.values())
    probable_keys = [key for key, value in count.items() if value == max_count]
    return probable_keys

def main():
    probable_keys = linear_cryptanalysis()
    print("Most probable key(s) found by linear cryptanalysis:", probable_keys)

if __name__ == "__main__":
    main()
```

Assignment Evaluation

0: Not Done [ ]	**1: Incomplete [ ]**	**2: Late Complete [ ]**
3: Need Improvement [ ]	**4: Complete [ ]**	**5: Well Done [ ]**

Signature of Instructor:____________________________________

Assignment 5:
Symmetric Algorithms

A] Implement RC4 in python

Objectives:

To implementation of the RC4 algorithm

Requirements:

- ☐ Python 3

Instructions:

Pseudo-Random Generation Algorithm (PRGA):
Initialization:
Initialize two counters i and j to 0.
Pseudo-Random Generation:
Increment i by 1 and update it to (i + 1) mod 256.
Increment j by the value of S[i] and update it to (j + S[i]) mod 256.
Swap the values of S[i] and S[j].
Calculate the pseudo-random byte k as S[(S[i] + S[j]) mod 256].
Output the result of plaintext XOR k.

Example code:

```
def rc4(key, plaintext):
    # Initialization
    S = list(range(256))
    j = 0

    # Key-scheduling algorithm (KSA)
    for i in range(256):
        j = (j + S[i] + key[i % len(key)]) % 256
        S[i], S[j] = S[j], S[i]

    # Pseudo-random generation algorithm (PRGA)
    i = j = 0
    ciphertext = []
    for char in plaintext:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        k = S[(S[i] + S[j]) % 256]
        ciphertext.append(char ^ k)  # XOR operation

    return bytes(ciphertext)

def main():
    # Example usage
    key = bytearray(b"SecretKey")  # Key must be a bytearray or bytes object
    plaintext = bytearray(b"Hello, RC4!")  # Plaintext must be a bytearray or bytes object
    ciphertext = rc4(key, plaintext)
    print("Ciphertext:", ciphertext.hex())

    decrypted_text = rc4(key, ciphertext)
    print("Decrypted text:", decrypted_text.decode())

if __name__ == "__main__":
    main()
```

B] Implement Blowfish in Python

Objectives:

To implementation of the blowfish algorithm

Requirements:

- ☐ Python 3

Instructions:

1. Key Generation:
   Generate a random 16-byte key using get_random_bytes.
2. Plaintext Definition:
   Define the plaintext message as a bytes object.
3. Encryption:
   Encrypt the plaintext message using the generated key and the encrypt_message function.
4. Decryption:
   Decrypt the ciphertext using the same key and the decrypt_message function.
5. Output:
   Print the encrypted ciphertext and the decrypted plaintext.

Example code:

```
from Crypto.Cipher import Blowfish
from Crypto.Random import get_random_bytes

def encrypt_message(key, plaintext):
    cipher = Blowfish.new(key, Blowfish.MODE_ECB)
    ciphertext = cipher.encrypt(plaintext)
    return ciphertext

def decrypt_message(key, ciphertext):
    cipher = Blowfish.new(key, Blowfish.MODE_ECB)
    plaintext = cipher.decrypt(ciphertext)
    return plaintext

def main():
    key = get_random_bytes(16)  # Generate a random 16-byte key
    plaintext = b"Hello, Blowfish!"  # Convert message to bytes

    # Encrypt the message
    ciphertext = encrypt_message(key, plaintext)
    print("Encrypted message:", ciphertext.hex())

    # Decrypt the message
    decrypted_message = decrypt_message(key, ciphertext)
    print("Decrypted message:", decrypted_message.decode())

if __name__ == "__main__":
    main()
```

C] Implementing the IDEA algorithm in Python

Objectives:

1. Understand the principles of block ciphers
2. Implement key generation, encryption, and decryption functions for IDEA algorithm
3. Verify the correctness of the implementation by encrypting and decrypting messages

Requirements:

- ☐ Python 3
- ☐ Basic understanding of number theory and modular arithmetic
- ☐ PyCryptodome library (for generating random numbers and XOR operation)

Instructions:

1. Import the PyCryptodome library
2. Define a function to generate a 128-bit key
3. Define a function to generate 52 subkeys from the main key
4. Define a function to encrypt a 64-bit block using the subkeys
5. Define a function to decrypt a 64-bit block using the subkeys
6. Test the implementation by encrypting and decrypting messages

Example code:

```
from Crypto.Cipher import IDEA
from Crypto.Random import get_random_bytes

def generate_key():
    return get_random_bytes(16)

def generate_subkeys(main_key):
    subkeys = []
    for i in range(8):
        subkey = main_key[i*2:i*2+2]
        subkeys.append(int.from_bytes(subkey, byteorder='big'))
    return subkeys
```

```
def encrypt_block(block, subkeys):
    cipher = IDEA.new(subkeys)
    return cipher.encrypt(block)

def decrypt_block(block, subkeys):
    cipher = IDEA.new(subkeys)
    return cipher.decrypt(block)

def main():
    # Generate a 128-bit key
    key = generate_key()
    print("Generated Key:", key.hex())

    # Generate subkeys from the main key
    subkeys = generate_subkeys(key)
    print("Generated Subkeys:", subkeys)

    # Test encryption and decryption
    plaintext = b'Hello World!'
    print("Plaintext:", plaintext)

    # Pad the plaintext to ensure it's a multiple of 8 bytes
    plaintext += b'\x00' * (8 - len(plaintext) % 8)

    # Encrypt the plaintext
    encrypted = encrypt_block(plaintext, subkeys)
    print("Encrypted:", encrypted.hex())

    # Decrypt the ciphertext
    decrypted = decrypt_block(encrypted, subkeys)
    print("Decrypted:", decrypted.rstrip(b'\x00'))

if __name__ == "__main__":
    main()
```

Assignment Evaluation

0: Not Done [ ]	**1: Incomplete [ ]**	**2: Late Complete [ ]**
3: Need Improvement [ ]	**4: Complete [ ]**	**5: Well Done [ ]**

Signature of Instructor:____________________________

Assignment 6:
Asymmetric Algorithms

A] Implement Knapsack algorithm in Python

Objectives:

1. Understand the principles of public-key cryptography
2. Implement key generation, encryption, and decryption functions for Knapsack algorithm
3. Verify the correctness of the implementation by encrypting and decrypting messages.

Requirements:

- ☐ Python 3
- ☐ Basic understanding number theory and modular arithmetic.

Instructions:

1. Define a super increasing sequence of integers
2. Define a modulus greater than the sum of the super increasing sequence Generate a public key using the super increasing sequence and modulus
3. Generate a private key using the inverse modulo of the sum of the super increasing sequence and modulus
4. Define a function to encrypt a message using the public key
5. Define a function to decrypt a message using the private key
6. Test the implementation by encrypting and decrypting messages

Example code:

```
from random import randint

def generate_superincreasing_sequence(length):

    return [randint(1, 10)] + [sum(sequence) + randint(sequence[-1] + 1,

sequence[-1] * 2) for _ in range(length - 1)]
```

```
def generate_modulus(sequence):
    return sum(sequence) + randint(1, 10)

def generate_public_key(sequence, modulus):
    return [elem * randint(1, 10) % modulus for elem in sequence]

def generate_private_key(sequence, modulus):
    return [(elem * pow(sum(sequence), -1, modulus)) % modulus for elem in
sequence]

def encrypt_message(message, public_key):
    return sum(int(char) * public_key[i] for i, char in
enumerate(''.join(format(ord(char), '08b') for char in message)))

def decrypt_message(encrypted_message, private_key, modulus):
    return ''.join(chr(int(''.join('1' if encrypted_message >= key else '0' for key in
reversed(private_key)), 2) for _ in range(0, len(private_key) * 8, 8)))

def main():
    length = 8
    sequence = generate_superincreasing_sequence(length)
    modulus = generate_modulus(sequence)
    public_key = generate_public_key(sequence, modulus)
    private_key = generate_private_key(sequence, modulus)
    message = "Hello, Knapsack!"
    print("Original message:", message)


    encrypted_message = encrypt_message(message, public_key)
    print("Encrypted message:", encrypted_message)


    decrypted_message = decrypt_message(encrypted_message, private_key,
modulus)
    print("Decrypted message:", decrypted_message)


if __name__ == "__main__":
    main()
```

B] Implement the Diffie Hellman Key Exchange algorithm in Python

Objectives:

1. Understand the principles of public-key cryptography.
2. To implement the Diffie Hellman Key Exchange algorithm in Python

Requirements:

- ☐ Python programming environment.
- ☐ Basic understanding of modular arithmetic.

Instructions:

1. Define two prime numbers p and g, where p is a large prime number and g is a primitive root modulo p. These two numbers will be public and known to both parties.

2. Alice and Bob agree on a secret random number a and b respectively. These numbers are kept private.
3. Alice computes A = g^a mod p and sends A to Bob.
4. Bob computes B = g^b mod p and sends B to Alice.
5. Alice computes the shared secret key K = B^a mod p.
6. Bob computes the shared secret key K = A^b mod p.
7. Both Alice and Bob now have the same shared secret key K, which can be used for symmetric encryption.

Example code:

```
import random
def is_prime(n):
    if n <= 1:
        return False
    if n <= 3:
        return True
    if 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 primitive_root(p):
    primitive_roots = []
    for i in range(2, p):
        if is_prime(i):
            s = set()
            for j in range(1, p):
                s.add(pow(i, j, p))
            if len(s) == p - 1:
                primitive_roots.append(i)
    return primitive_roots

def diffie_hellman(p, g):
    a = random.randint(2, p - 1)
    b = random.randint(2, p - 1)

    A = pow(g, a, p)
    B = pow(g, b, p)

    K1 = pow(B, a, p)
    K2 = pow(A, b, p)

    return K1, K2

p = 23  # Example prime number
g = primitive_root(p)[0]  # Example primitive root modulo p

K1, K2 = diffie_hellman(p, g)


print("Shared secret key for Alice:", K1)
print("Shared secret key for Bob:", K2)
```

This code defines a function to check if a number is prime (is_prime), a function to find primitive roots modulo p (primitive_root), and the Diffie-Hellman key exchange function (diffie_hellman). Finally, it generates two random secret numbers for Alice and Bob, computes the shared secret keys, and prints them out.

C] Implementing the RSA algorithm in Python

Objectives:

3. Understand the principles of public-key cryptography.
4. Implement key generation, encryption, and decryption functions for RSA algorithm.
5. Verify the correctness of the implementation by encrypting and decrypting messages.

Requirements:

- ☐ Python 3
- ☐ Basic understanding of number theory and modular arithmetic
- ☐ PyCryptodome library (for generating random prime numbers and modular inverse)

Instructions:

1. Import the PyCryptodome library
2. Define a function to generate two large prime numbers p and q
3. Define a function to calculate Euler's totient function of n using p and q
4. Define a function to generate public and private keys using p, q, and Euler's totient
5. Define a function to encrypt a message using the public key
6. Define a function to decrypt a message using the private key
7. Test the implementation by encrypting and decrypting messages

Example code:

```
from Crypto.PublicKey import RSA
from Crypto.Random import get_random_bytes
from Crypto.Cipher import PKCS1_OAEP


def generate_large_prime():
    # You can implement your own function to generate large prime numbers
    # For simplicity, we'll just use the RSA.generate method to generate a key pair
and extract the prime numbers
    key = RSA.generate(2048)
    p = key.p
    q = key.q
    return p, q
```

```
def calculate_totient(p, q):
    return (p - 1) * (q - 1)

def generate_keys(p, q):
    n = p * q
    totient = calculate_totient(p, q)
    e = 65537  # commonly used value for e
    d = pow(e, -1, totient)  # modular inverse of e modulo totient
    public_key = RSA.construct((n, e))
    private_key = RSA.construct((n, e, d))
    return public_key, private_key

def encrypt_message(message, public_key):
    cipher = PKCS1_OAEP.new(public_key)
    ciphertext = cipher.encrypt(message)
    return ciphertext

def decrypt_message(ciphertext, private_key):
    cipher = PKCS1_OAEP.new(private_key)
    plaintext = cipher.decrypt(ciphertext)
    return plaintext

def main():
    # Generate large prime numbers p and q
    p, q = generate_large_prime()
    print("Generated primes p and q:", p, q)

    # Generate public and private keys
    public_key, private_key = generate_keys(p, q)

    # Test encryption and decryption
    message = b"Hello, RSA!"
    print("Original message:", message)

    # Encrypt the message using the public key
    encrypted_message = encrypt_message(message, public_key)
    print("Encrypted message:", encrypted_message.hex())
    # Decrypt the message using the private key
    decrypted_message = decrypt_message(encrypted_message, private_key)
    print("Decrypted message:", decrypted_message.decode())

if __name__ == "__main__":
    main()
```

Assignment Evaluation

0: Not Done [ ]	**1: Incomplete [ ]**	**2: Late Complete [ ]**
3: Need Improvement [ ]	**4: Complete [ ]**	**5: Well Done [ ]**

Signature of Instructor:____

Assignment 7:

Review papers on cryptography and network security

Refer Research Websites:

1. **Indian Institute of Science (IISc) Digital Repository:** IISc hosts a digital repository that contains research articles, theses, conference papers, and other scholarly works produced by researchers affiliated with the institute. You can access it here: IISc Digital Repository
2. **Academy of Sciences (IAS) Journals:** The Indian Academy of Sciences publishes several journals covering different areas of science and technology. You can access articles from these journals on their website: IAS Journals
3. **National Digital Library of India (NDLI):** NDLI is a digital repository that provides access to a vast collection of academic resources, including research papers, articles, books, and theses. It covers various subjects and disciplines. You can explore it here: NDLI
4. **Indian Council of Agricultural Research (ICAR) Journals:** ICAR publishes several journals focusing on agricultural research. You can access articles from these journals on their website: ICAR Journals
5. **Indian Citation Index (ICI):** ICI is an online bibliographic database that indexes research publications from Indian journals. It provides access to abstracts and full-text articles. You can explore it here: Indian Citation Index
6. **Directory of Open Access Journals (DOAJ):** Although not specific to India, DOAJ is a comprehensive directory of open-access journals covering various subjects. You can search for Indian journals and access research articles from different disciplines. You can explore it here: DOAJ
7. **Sci-Hub:** While not an Indian website, Sci-Hub is a popular platform for accessing research papers from various publishers worldwide. It provides access to millions of research articles, including those behind paywalls. You can access it here: Sci-Hub

Links:

1. https://www.researchgate.net/publication/358242788_A_Study_on_Network_Security_and_Cryptography
2. https://www.irjmets.com/uploadedfiles/paper//issue_1_january_2023/33258/final/fin_irjmets1675167802.pdf
3. https://www.ripublication.com/acst17/acstv10n5_10.pdf

1. Write review paper on network security.
2. Write review paper on substitution techniques.
3. Write review paper on cryptographic algorithms.
4. Write review paper on symmetric algorithms.
5. Write review paper on asymmetric algorithms.