Network Security and Cryptography Lab

Of

Bachelor of Technology (VIth semester)

In the department of

Computer Science and Engineering

By

Mood Mahesh Nayak (21131A05C5)

<u>Faculty</u>

Dr. Ch. Sita Kumari

(Associate Professor)

<u>Department of Computer Science and Engineering</u>

Gayatri Vidya Parishad College of Engineering (Autonomous)

(Affiliated to JNTU – Kakinada)

Visakhapatnam

(2023-2024)

Index

Week - 1

Aim: Implement the Ceaser Cipher Algorithm

Program:

```
def encryption(text, key):
    cipher = ""
    for i in text:
        if i == " ":
            cipher += " "
        else:
            if i.isdigit():
                cipher += chr(((ord(i)-ord('0')+key) % 10)+ord('0'))
            elif ord(i) in range(97, 123):
                cipher += chr(((ord(i)-ord('a')+key) % 26)+ord('a'))
            elif ord(i) in range(65, 91):
                cipher += chr(((ord(i)-ord('A')+key) % 26)+ord('A'))
            else:
                cipher += chr((ord(i)+key) % 128)
    return cipher

def decryption(cipher, key):
    text = ""
    for i in cipher:
        if i == " ":
            text += " "
        else:
            if i.isdigit():
                text += chr(((ord(i)-ord('0')-key) % 10)+ord('0'))
            elif ord(i) in range(97, 123):
                text += chr(((ord(i)-ord('a')-key) % 26)+ord('a'))
```

```
            elif ord(i) in range(65, 91):
                text += chr(((ord(i)-ord('A')-key) % 26)+ord('A'))
            else:
                text += chr((ord(i)-key) % 128)
    return text
    # text = encryption(cipher, -key)
    # return text

key = int(input("Enter Key: "))
cipher = encryption(input("Enter Plain text: "), key)
text = decryption(cipher, key)
print("Cipher Text: ", cipher.upper())
print("Plain Text: ",  text)
```

Output:

Enter key : 3

Enter Plain text : meet me after 3pm @Gvp

Cipher text: PHHW PH DIWHU 6SP CJYS

Plain text after decryption : meet me after 3pm @Gvp

Week – 2

Aim: Implement the Hill Cipher Algorithm

Program:

```
import numpy as np
from math import sqrt
from sympy import Matrix

plainText = input("Enter the plain text: ")
key = input("Enter the key: ")

key_length = len(key)
text_length = len(plainText)
key_matrix_dim = int(sqrt(key_length))

def construct_matrix(text, key):
    key_matrix = np.array([ord(i)-ord('A') for i in key])
    key_matrix = key_matrix.reshape(key_matrix_dim, key_matrix_dim)
    text_matrix = np.array([ord(i)-ord('A') for i in text])
    text_matrix = text_matrix.reshape(
        text_length//key_matrix_dim, key_matrix_dim)
    return key_matrix, text_matrix

def Encryption():
    key_matrix, plainText_matrix = construct_matrix(plainText, key)
    cipher = np.array([])
    for i in range(text_length // key_matrix_dim):
        row = np.matmul(key_matrix, plainText_matrix[i])
        cipher = np.append(cipher, list(map(chr, row % 26 + ord('A'))))
    return cipher
```

```
cipher_matrix = Encryption()
print("Cipher text: ", "".join(cipher_matrix.flatten()))

def Decryption():
    key_matrix, Cipher_matrix = construct_matrix(cipher_matrix, key)
    A = Matrix(key_matrix)
    key_matrix_inv = A.inv_mod(26)
    text = np.array([])
    for i in range(text_length // key_matrix_dim):
        row = np.matmul(key_matrix_inv, Cipher_matrix[i])
        text = np.append(text, list(map(chr, row % 26 + ord('A'))))
    return text

print("Plaintext: ", "".join(Decryption()))
```

Output:

Enter the plain text: PAYMOREMONEY

Enter the key: RRFVSVCCT

Cipher text: LNSHDLEWMTRW

Plaintext: PAYMOREMONEY

Week – 3

Aim: Implement the Simple – DES Algorithm

Program:

```
IP = [2, 6, 3, 1, 4, 8, 5, 7]
EP = [4, 1, 2, 3, 2, 3, 4, 1]
IP_INVERSE = [4, 1, 3, 5, 7, 2, 8, 6]
P10 = [3, 5, 2, 7, 4, 10, 1, 9, 8, 6]
P8 = [6, 3, 7, 4, 8, 5, 10, 9]
P4 = [2, 4, 3, 1]

S0 = [[1, 0, 3, 2],
      [3, 2, 1, 0],
      [0, 2, 1, 3],
      [3, 1, 3, 2]]

S1 = [[0, 1, 2, 3],
      [2, 0, 1, 3],
      [3, 0, 1, 0],
      [2, 1, 0, 3]]

def permutate(original, key):
    newkey = ''
    for i in key:
        newkey += original[i - 1]
    return newkey

def left_half(bits):
    return bits[:len(bits)//2]
```

```
def right_half(bits):
    return bits[len(bits)//2:]

def shift(bits):
    rotated_left_half = left_half(bits)[1:] + left_half(bits)[0]
    rotated_right_half = right_half(bits)[1:] + right_half(bits)[0]
    return rotated_left_half + rotated_right_half

def key1():
    return permutate(shift(permutate(KEY, P10)), P8)

def key2():
    return permutate(shift(shift(shift(permutate(KEY, P10)))), P8)

def xor(bits, key):
    new = ''
    for bit, key_bit in zip(bits, key):
        new += str(((int(bit) + int(key_bit)) % 2))
    return new

def lookup_in_sbox(bits, sbox):
    row = int(bits[0] + bits[3], 2)
    col = int(bits[1] + bits[2], 2)
    return '{0:02b}'.format(sbox[row][col])

def f_k(bits, key):
    L = left_half(bits)
    R = right_half(bits)
    bits = permutate(R, EP)
    bits = xor(bits, key)
    bits = lookup_in_sbox(left_half(bits), S0) + lookup_in_sbox(right_half(bits), S1)
    bits = permutate(bits, P4)
```

```
    return xor(bits, L)

def encrypt(plain_text):
    bits = permutate(plain_text, IP)
    temp = f_k(bits, key1())
    # swap
    bits = right_half(bits) + temp
    bits = f_k(bits, key2())
    print("Cipher Text: ", permutate(bits + temp, IP_INVERSE))
    return permutate(bits + temp, IP_INVERSE)

def decrypt(cipher_text):
    bits = permutate(cipher_text, IP)
    temp = f_k(bits, key2())
    bits = right_half(bits) + temp
    bits = f_k(bits, key1())
    print("Original Message: ", permutate(bits + temp, IP_INVERSE))

KEY = input("Enter key: ")
cipher = encrypt(input("Enter Plain text: "))
decrypt(cipher)
```

Output:

Enter key: 1010000010

Enter Plain text: 10010111

Cipher Text: 00111000

Original Message: 10010111

Week – 4

Aim: Implement RSA Algorithm

Program:

```
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

def get_prime_input():
    while True:
        try:
            num = int(input("Enter a prime number: "))
            if is_prime(num):
                return num
            else:
                print("Please enter a prime number.")
        except ValueError:
            print("Invalid input. Please enter a valid integer.")

def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

def mod_inverse(a, m):
    m0, x0, x1 = m, 0, 1
```

```
    while a > 1:
        q = a // m
        m, a = a % m, m
        x0, x1 = x1 - q * x0, x0
    return x1 + m0 if x1 < 0 else x1

def generate_keypair(p, q):
    n = p * q
    phi = (p - 1) * (q - 1)
    print(phi)
    e = 2
    while gcd(e, phi) != 1:
        e += 1
    d = mod_inverse(e, phi)
    return ((e, n), (d, n))

def encrypt(message, public_key):
    e, n = public_key
    cipher_text = [pow(ord(char), e, n) for char in message]
    return cipher_text

def decrypt(cipher_text, private_key):
    d, n = private_key
    plain_text = ''.join([chr(pow(char, d, n)) for char in cipher_text])
    return plain_text

def main():
    print("RSA Encryption and Decryption without random module")
    p = get_prime_input()
    q = get_prime_input()

    public_key, private_key = generate_keypair(p, q)
```

```
    print("Public Key:", public_key)
    print("Private Key:", private_key)

    message = input("Enter a message to encrypt: ")


    cipher_text = encrypt(message, public_key)
    print("Encrypted Message:", cipher_text)
    cipher = [i % 127 for i in cipher_text]
    cipher_ = [chr(i) for i in cipher_text]
    print("Encrypted Message:", ''.join(cipher_))
    decrypted_message = decrypt(cipher_text, private_key)
    print("Decrypted Message:", decrypted_message)

main()
```

Output:

```
Enter a prime number: 59
Enter a prime number: 29
Public Key (e, n): (3, 1711)
Private Key (d, n): (1083, 1711)
Enter a message to encrypt: mohit
Encrypted Message: [1513, 542, 737, 989, 464]
Encrypted Message: שȞˡϝǐ
Decrypted Message: mohit
```

Week – 5

Aim: Implement Diffie-Hellman Key exchange algorithm

Program:

```
def mod_exp(base, exponent, modulus):
    result = 1
    base = base % modulus
    while exponent > 0:
        if exponent % 2 == 1:
            result = (result * base) % modulus
        exponent = exponent // 2
        base = (base * base) % modulus
    return result



def diffie_hellman():
    p = int(input("Enter p: "))
    g = int(input("Enter primitive root : "))
    a = int(input("Enter A's secret key: "))
    b = int(input("Enter B's secret key: "))
    A = mod_exp(g, a, p)
    B = mod_exp(g, b, p)

    print("A Sent to B : ", A)
    print("B Sent to A : ", B)

    secret_key_alice = mod_exp(B, a, p)
    secret_key_bob = mod_exp(A, b, p)

    print("Shared secret key for A:", secret_key_alice)
```

```
print("Shared secret key for B:", secret_key_bob)
diffie_hellman()
```

Output:

Enter p: 17

Enter primitive root : 5

Enter A's secret key: 4

Enter B's secret key: 6

A Sent to B : 13

B Sent to A : 2

Shared secret key for A: 16

Shared secret key for B: 16

Week – 6

Aim: Implement SHA-1 Algorithm

Program: