**Network Security and Cryptography Lab**

Of

**Bachelor of Technology (VIth semester)**

In the department of

**Computer Science and Engineering**

By

**Mood Mahesh Nayak (21131A05C5)**

****

**Faculty**

**Dr. Ch. Sita Kumari**

**(Associate Professor)**

**Department of Computer Science and Engineering**

**Gayatri Vidya Parishad College of Engineering (Autonomous)**

**(Affiliated to JNTU – Kakinada)**

**Visakhapatnam**

**(2023-2024)**

**Index**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sl. No. | Name of the Experiment | Date | Page No. | Remarks |
|  | Ceaser Cipher Algorithm | 13-12-2023 | 3-4 |  |
|  | Hill Cipher Algorithm | 20-12-2023 | 5-6 |  |
|  | Simple DES Algorithm | 27-12-2023 | 7-9 |  |
|  | RSA Algorithm | 10-01-2024 | 10-12 |  |
|  | Diffie Hellman Key Exchange Algorithm | 24-01-2024 | 13-14 |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**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:**