1)Perform an experiment for port scanning with nmap

**What is Nmap?**

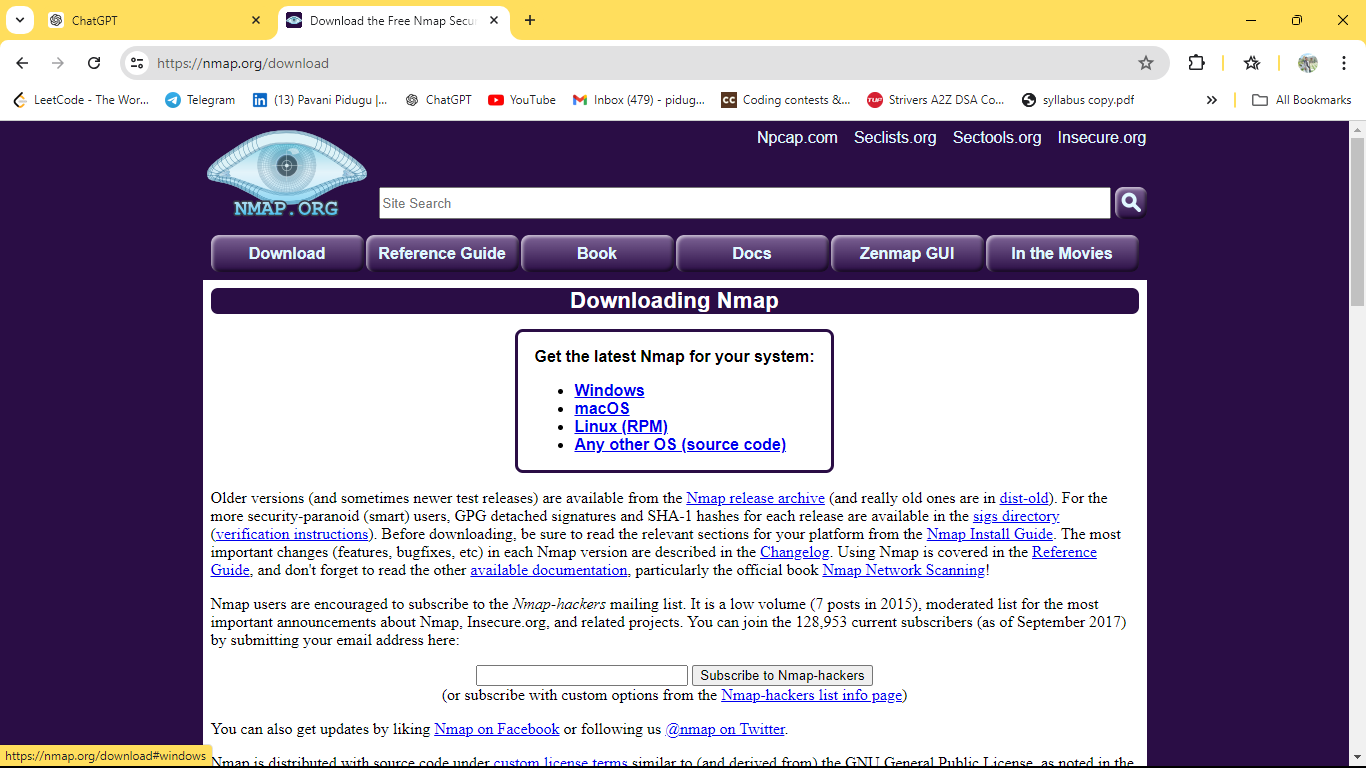
Nmap, short for Network Mapper, is a highly versatile and powerful open-source tool used for network discovery and security auditing. It is widely used by network administrators, security professionals, and enthusiasts for its ability to scan networks and gather detailed information about the devices and services operating within them.

**Key Features of Nmap**

1. **Host Discovery**: Identify devices on a network.
2. **Port Scanning**: Determine which ports on a host are open or closed.
3. **Service Version Detection**: Identify the versions of services running on open ports.
4. **Operating System Detection**: Guess the operating system of a host.
5. **Network Inventory**: Maintain a list of all devices and services on a network.
6. **Security Auditing**: Assess the security of networked devices.

### Port Scanning with Nmap

Port scanning is a technique used to identify open ports and the services running on them on a target machine. Ports are entry points to a machine's services, and knowing which ports are open can provide insights into potential vulnerabilities and the machine's role in the network.



Commands record

Nmap –p scanme.org

Nmap –p 21-100 scanme.org

Namp –p U:53,T:45-78,90 scanme.org

Nmap –F scanme.org

Nmap –p- example.com

Nmap –p http,https scanme.org

Tcp

Nmap –sS scanme.org

Nmap –sA scanme.org

Nmap –sT scanme.org

Nmap –w scanme.org

2)Setup honeypot and monitor the honipot on the network

KFSensor aims to detect, log, and analyze unauthorized attempts to access or compromise computer systems. By simulating vulnerable services, it attracts malicious activity, providing valuable data for network administrators and security professionals to understand and mitigate threats.

#### Key Features of KFSensor:

* **Service Emulation**: Simulates various network services (HTTP, FTP, SMTP, SSH, etc.) to attract attackers.
* **Real-Time Monitoring**: Provides a live view of all activities and alerts as they happen.

### What is a Honeypot?

A **honeypot** is a security mechanism set up to attract and analyze malicious activity. It mimics a target for attackers, creating an environment that appears vulnerable but is actually isolated and monitored. Honeypots serve various purposes, such as detecting new threats, gathering intelligence on attacker behavior, and improving overall network security by identifying vulnerabilities.

#### Benefits of Honeypots:

* **Threat Detection**: Identify and analyze new threats before they reach critical systems.
* **Threat Intelligence**: Gather data on attack methods, tools, and patterns.
* **Security Improvement**: Identify vulnerabilities in the network and improve defensive measures.
* **Research**: Provide valuable data for cybersecurity research and development of new defense mechanisms.

Visitors,ports

Nmap 172.27.48.1

**Example Scenario**

Let's set up a honeypot to monitor HTTP and SSH traffic:

1. **Create an HTTP Sensor**:
   * Right-click "Sensors" -> "New Sensor" -> "HTTP".
   * Configure the HTTP sensor to listen on port 80.
   * Set up a fake web server banner (e.g., "Apache/2.4.1 (Unix)").
2. **Create an SSH Sensor**:
   * Right-click "Sensors" -> "New Sensor" -> "SSH".
   * Configure the SSH sensor to listen on port 22.
   * Set up a fake SSH banner (e.g., "OpenSSH\_7.4p1 Debian-10+deb9u6").
3. **Monitor and Analyze**:
   * In the "Event Log", observe any connection attempts to ports 80 and 22.
   * If an attack is detected, KFSensor will log details such as the IP address of the attacker, the time of the attack, and the type of attack.
   * Analyze the log to understand the attack pattern and potential vulnerabilities in your network.

3) Instal a jcrpt tool(or any other eqvivalent ) and demonstrate Asymmetric ,Symmetric crypto algorithm ,Hash and Digital/PKI signatures studied in theory Network security and management

JCrypTool enables students, teachers, developers, and anyone else interested in cryptography to apply and analyze cryptographic algorithms in a modern, easy-to-use application. JCrypTool includes a wealth of cryptographic mechanisms including classic, symmetric, and asymmetric encryption, hash functions, analysis tools, visualizations and crypto games.

Refer record

Working with sniffers for monitoring network communication (Wireshark).

Wireshark is a network protocol analyzer, sometimes called a *packet analyzer*, designed to provide visibility into network traffic occurring on a network or between machines.

Wireshark is a widely used network protocol analyzer. It's an open-source software tool that allows you to capture and interactively browse the traffic running on a computer network. Originally named Ethereal, Wireshark was created by Gerald Combs in 1998 and has since become the de facto standard for network protocol analysis.

Here are some key features of Wireshark:

1. **Packet Capture:** Wireshark can capture packets from various network interfaces in real-time or read packets from a previously captured file.
2. **Packet Analysis:** It provides a detailed view of captured packets, including packet headers, payloads, and other metadata. It can dissect hundreds of different protocols, allowing you to analyze network traffic at a granular level.
3. **Filtering:** Wireshark offers powerful filtering capabilities, enabling you to focus on specific packets based on criteria like IP addresses, protocols, ports, and more.
4. **Protocol Support:** It supports a wide range of network protocols, including common ones like TCP, UDP, IP, HTTP, DNS, DHCP, SSL/TLS, and many others. It can decode and display protocol-specific information, making it easier to understand the traffic.

**Procedure**

### . Installation:

First, you need to download and install Wireshark on your system. It's available for Windows, macOS, and Linux platforms. You can download it from the official website: [Wireshark.org](https://www.wireshark.org/)

### 2. Capturing Packets:

Once installed, you can launch Wireshark and start capturing packets on your network interface. You can choose the interface you want to capture traffic from and click on the "Start" button to begin capturing packets.

### 3. Filtering:

Wireshark captures a lot of traffic, so it's essential to use filters to focus on the packets you're interested in. Filters can be based on IP addresses, protocols, ports, etc. You can apply filters in the filter toolbar or directly in the filter field.

### 4. Analyzing Packets:

Wireshark displays captured packets in a detailed packet list pane and provides various views and statistics. You can expand each packet to see its details, including headers, payload, and other metadata.

### 5. Following TCP Streams:

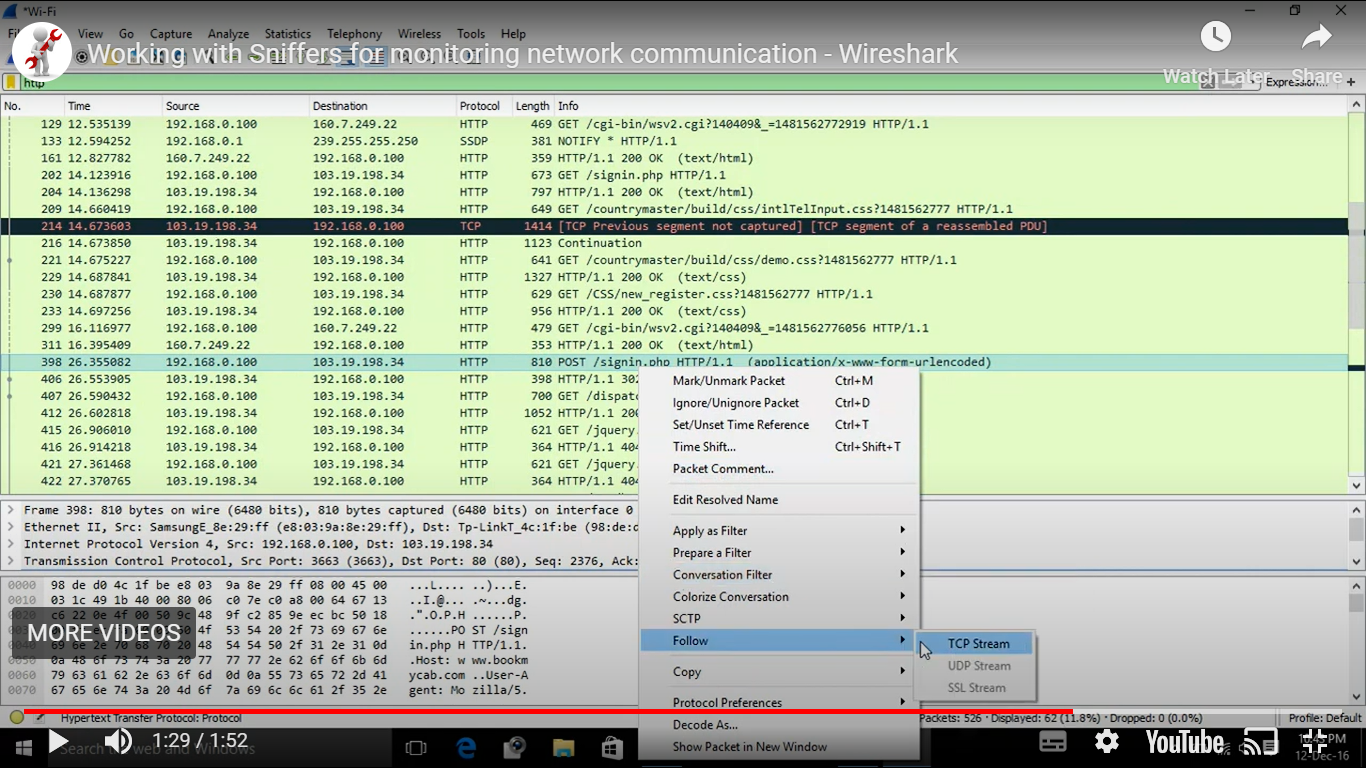
Wireshark allows you to follow TCP streams, which reconstructs the entire conversation between two endpoints. This feature is handy for understanding the flow of data between hosts.

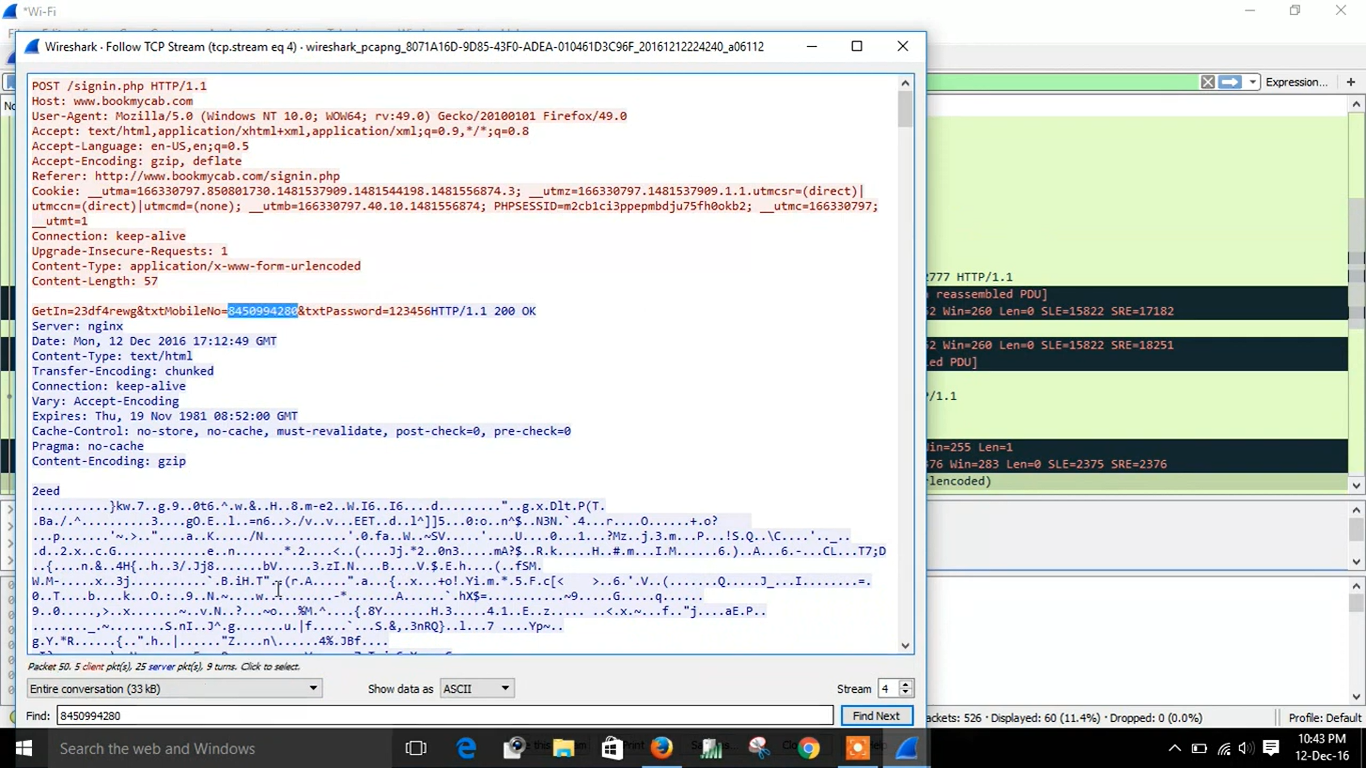
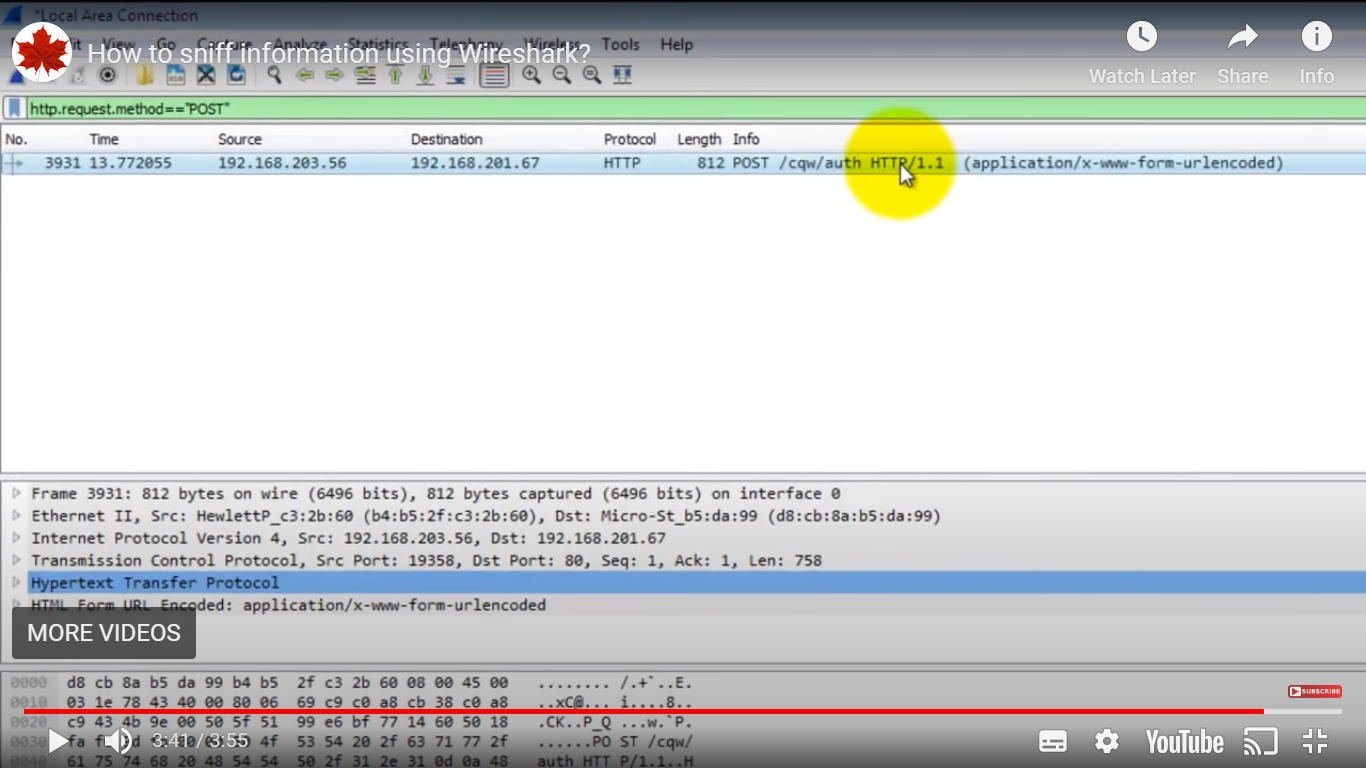
### 6. Protocol Analysis:

Wireshark supports dissecting and analyzing numerous protocols, including common ones like HTTP, DNS, TCP, UDP, and more. It can decode and display protocol-specific information, making it easier to understand the traffic.

### 7. Exporting Data:

You can export captured packets or filtered packets to various formats like plain text, CSV, or PDML for further analysis or reporting purposes.

* **View data traversing various networks**, including wired networks, such as Ethernet; wireless networks; Bluetooth networks; or virtual network interfaces, such as with Docker or a hypervisor.
* **Navigate and view the various layers of the stack**, including application-level protocols, such as HTTP/HTTPS; mail protocols, such as Post Office Protocol 3 and SMTP; and file-sharing protocols, such as Server Message Block and Common Internet File System. Lower down in the stack, we can view TCP/IP and [User Datagram Protocol](https://www.techtarget.com/searchnetworking/definition/UDP-User-Datagram-Protocol). Even lower in the stack, we can view artifacts such as Ethernet frames.
* **Record and capture traffic** for subsequent analysis.
* 



8) CAESER CIPHER

#include <iostream>

#include <string>

using namespace std;

string encrypt(string plaintext, int shift) {

string encrypted\_text = "";

for (char c : plaintext) {

if (islower(c)) {

c = 'A' + (c - 'a' + shift) % 26;

}

encrypted\_text += c;

}

return encrypted\_text;

}

string decrypt(string ciphertext, int shift) {

string decrypted\_text = "";

for (char c : ciphertext) {

if (isupper(c)) {

c = 'a' + (c - 'A' + (26 - shift)) % 26;

}

decrypted\_text += c;

}

return decrypted\_text;

}

int main() {

string plaintext, ciphertext;

int shift;

cout<< "Enter the plaintext(lowercase and digits only): ";

getline(cin, plaintext);

cout<< "Enter the shift value: ";

cin>> shift;

ciphertext = encrypt(plaintext, shift);

cout<< "Encrypted text: " << ciphertext <<endl;

string decrypted\_text = decrypt(ciphertext, shift);

cout<< "Decrypted text: " <<decrypted\_text<<endl;

return 0;

}

Output:

/tmp/lOV2YCNbRj.o

Enter the plaintext(lowercase and digits only): pavani123

Enter the shift value: 3

Encrypted text: SDYDQL123

Decrypted text: pavani123

9)PLAY FAIR

#include<bits/stdc++.h>

#include<string.h>

#define SIZE 5

using namespace std;

void generateKeyTable(string key, char keyTable[SIZE][SIZE]) {

int k = 0;

char vis[26] = {0};

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

if (vis[key[i] - 'A'] == 0 && key[i] != 'J') {

keyTable[k / SIZE][k % SIZE] = key[i];

vis[key[i] - 'A'] = 1;

k++;

}

}

char ch = 'A';

for (int i = k; i< SIZE \* SIZE; i++) {

while (vis[ch - 'A'] == 1 || ch == 'J')

ch++;

keyTable[i / SIZE][i % SIZE] = ch;

vis[ch - 'A'] = 1;

ch++;

}

}

void printKeyTable(char keyTable[SIZE][SIZE]) {

cout<< "Key Table:" <<endl;

for (int i = 0; i< SIZE; i++) {

for (int j = 0; j < SIZE; j++) {

if(keyTable[i][j]=='I')

cout<<"J/";

cout<<keyTable[i][j] << " ";

}

cout<<endl;

}

}

void findPosition(char keyTable[SIZE][SIZE], char ch, int& row, int& col) {

if (ch == 'J') ch = 'I';

for (int i = 0; i< SIZE; i++) {

for (int j = 0; j < SIZE; j++) {

if (keyTable[i][j] == ch) {

row = i;

col = j;

return;

}

}

}

}

string encrypt(char keyTable[SIZE][SIZE], string& plaintext) {

int len = plaintext.length();

string ciphertext;

if (plaintext.length() % 2 != 0) {

plaintext += 'X';

}

for (int i = 0; i<len; i += 2) {

char ch1 = plaintext[i];

char ch2 = plaintext[i + 1];

if (ch1 == ch2) {

ch2 = 'X';

i--;

}

int row1, col1, row2, col2;

findPosition(keyTable, ch1, row1, col1);

findPosition(keyTable, ch2, row2, col2);

if (row1 == row2) {

ciphertext += keyTable[row1][(col1 + 1) % SIZE];

ciphertext += keyTable[row2][(col2 + 1) % SIZE];

}

else if (col1 == col2) {

ciphertext += keyTable[(row1 + 1) % SIZE][col1];

ciphertext += keyTable[(row2 + 1) % SIZE][col2];

}

else {

ciphertext += keyTable[row1][col2];

ciphertext += keyTable[row2][col1];

}

}

cout<< "Encrypted Text: " << ciphertext <<endl;

return ciphertext;

}

string decrypt(char keyTable[SIZE][SIZE], string& ciphertext) {

int len = ciphertext.length();

string plaintext;

for (int i = 0; i<len; i += 2) {

char ch1 = ciphertext[i];

char ch2 = ciphertext[i + 1];

int row1, col1, row2, col2;

findPosition(keyTable, ch1, row1, col1);

findPosition(keyTable, ch2, row2, col2);

if (row1 == row2) {

plaintext += keyTable[row1][(col1 + SIZE - 1) % SIZE];

plaintext += keyTable[row2][(col2 + SIZE - 1) % SIZE];

}

else if (col1 == col2) {

plaintext += keyTable[(row1 + SIZE - 1) % SIZE][col1];

plaintext += keyTable[(row2 + SIZE - 1) % SIZE][col2];

}

else {

plaintext += keyTable[row1][col2];

plaintext += keyTable[row2][col1];

}

}

plaintext.erase(remove(plaintext.begin(), plaintext.end(), 'X'), plaintext.end());

for (char& c : plaintext) {

c = tolower(c);

}

cout<< "Decrypted Text: " << plaintext <<endl;

return plaintext;

}

int main() {

string key, plaintext;

cout<< "Enter the key: ";

cin>> key;

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

char c = key[i];

if (!isalpha(c)||!isupper(c)) {

cout<< "Invalid key!" <<endl;

return 1;

}

}

char keyTable[SIZE][SIZE];

generateKeyTable(key, keyTable);

printKeyTable(keyTable);

cout<< "Enter the plaintext:";

cin>> plaintext;

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

plaintext[i] = toupper(plaintext[i]);

}

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

char c = plaintext[i];

if (!isalpha(c)) {

cout<< "Invalid plaintext!" <<endl;

return 1;

}

}

string ciphertext=encrypt(keyTable, plaintext);

decrypt(keyTable, ciphertext);

return 0;

}

Output:

/tmp/0lgggiQdnE.o

Enter the key: MONARCHY

Key Table:

M O N A R

C H Y B D

E F G J/I K

L P Q S T

U V W X Z

Enter the plaintext:balloon

Encrypted Text: IBSUPMNA

Decrypted Text: balloon

10)HILL CIPHER

#include <iostream>

using namespace std;

void getKeyMatrix(string key, int keyMatrix[][3]) {

int k = 0;

cout<< "Key Matrix:" <<endl;

for (int i = 0; i< 3; i++) {

for (int j = 0; j < 3; j++) {

keyMatrix[i][j] = (key[k]) % 65;

cout<<keyMatrix[i][j] << " ";

k++;

}

cout<<endl;

}

}

void encrypt(int cipherMatrix[][1], int keyMatrix[][3], int messageVector[][1]) {

int x, i, j;

for (i = 0; i< 3; i++) {

for (j = 0; j < 1; j++) {

cipherMatrix[i][j] = 0;

for (x = 0; x < 3; x++) {

cipherMatrix[i][j] += keyMatrix[i][x] \* messageVector[x][j];

}

cipherMatrix[i][j] = cipherMatrix[i][j] % 26;

}

}

}

void HillCipher(string message, string key) {

int keyMatrix[3][3];

getKeyMatrix(key, keyMatrix);

int messageVector[3][1];

cout<<endl;

for (int i = 0; i< 3; i++) {

messageVector[i][0] = (message[i]) % 65;

cout<< "Message Vector: " <<messageVector[i][0] <<endl;

}

int cipherMatrix[3][1];

encrypt(cipherMatrix, keyMatrix, messageVector);

string CipherText;

for (int i = 0; i< 3; i++)

CipherText += cipherMatrix[i][0] + 65;

cout<< "Ciphertext: " <<CipherText<<endl;

}

int main() {

string message, key;

cout<< "Enter the plaintext (must be 3 characters): ";

cin>> message;

cout<< "Enter the key (must be 9 characters): ";

cin>> key;

HillCipher(message, key);

return 0;

}

OUTPUT:

12) RSA

#include <iostream>

#include <cmath>

using namespace std;

int gcd(int a, int b) {

if (a == 0)

return b;

return gcd(b % a, a);

}

int find\_d(int e, int phi)

{

for (int i = 1; i< phi; i++)

{

if ((i \* e) % phi == 1)

{

return i;

}

}

return -1;

}

int modExp(int M, int e, int n) {

int result = 1;

M = M % n;

while (e > 0) {

if (e % 2 == 1)

result = (result \* M) % n;

e = e >> 1;

M = (M \* M) % n;

}

return result;

}

int main() {

int p, q;

cout<< "Enter the value of p: ";

cin>> p;

cout<< "Enter the value of q: ";

cin>> q;

double n = p \* q;

double phi = (p - 1) \* (q - 1);

double e = 2;

while (e < phi) {

if (gcd(e, phi) == 1)

break;

else

e++;

}

double d = find\_d(e,phi);

cout<< "p: " << p <<endl;

cout<< "q: " << q <<endl;

cout<< "n: " << n <<endl;

cout<< "phi(n): " << phi <<endl;

cout<< "e: " << e <<endl;

cout<< "d: " << d <<endl;

int plaintext;

cout<< "\nEnter the plaintext (as an integer): ";

cin>> plaintext;

int ciphertext = modExp(plaintext, e, n);

cout<< "Encrypted ciphertext: " << ciphertext <<endl;

int decrypted\_text = modExp(ciphertext, d, n);

cout<< "Decrypted plaintext: " <<decrypted\_text<<endl;

return 0;

}

OUTPUT:

/tmp/9hHNGXNf8m.o

Enter the value of p: 17

Enter the value of q: 11

p: 17

q: 11

n: 187

phi(n): 160

e: 3

d: 107

Enter the plaintext (as an integer): 88

Encrypted ciphertext: 44

Decrypted plaintext: 88

Hillcipher

#include<bits/stdc++.h>

using namespace std;

float a[2][2], c[2][2], encrypt[2], decrypt[2], mes[2];

string enc = "";

string decr = "";

void encryption();

void decryption();

void inverse();

void getInput() {

string msg;

cout << "\nEnter msg :";

cin >> msg;

cout << "Enter 2x2 matrix for key (It should be invertible):\n";

for (int i = 0; i < 2; i++)

for (int j = 0; j < 2; j++) {

cin >> a[i][j];

c[i][j] = a[i][j];

}

inverse();

// Reset the strings

enc = "";

decr = "";

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

mes[i % 2] = msg[i] - 97;

if(i%2)

{

encryption();

decryption();

}

}

}

void inverse() {

c[0][1] = -1 \* c[0][1];

c[1][0] = -1 \* c[1][0];

float temp;

temp = c[0][0];

c[0][0] = c[1][1];

c[1][1] = temp;

temp = (a[0][0] \* a[1][1]) - (a[0][1] \* a[1][0]);

for (int i = 0; i < 2; i++)

for (int j = 0; j < 2; j++)

c[i][j] = c[i][j] / temp;

}

void encryption() {

for (int i = 0; i < 2; i++)

encrypt[i] = 0;

for (int i = 0; i < 2; i++)

for (int k = 0; k < 2; k++)

encrypt[i] = encrypt[i] + a[i][k] \* mes[k];

for (int i = 0; i < 2; i++) {

char ch = (char)(fmod(encrypt[i], 26) + 97);

enc += ch;

}

}

void decryption() {

for (int i = 0; i < 2; i++)

decrypt[i] = 0;

for (int i = 0; i < 2; i++)

for (int k = 0; k < 2; k++)

decrypt[i] = decrypt[i] + c[i][k] \* encrypt[k % 2];

for (int i = 0; i < 2; i++) {

char ch = (char)(fmod(decrypt[i], 26) + 97);

decr += ch;

}

}

int main()

{

getInput();

cout << "\nEncrypted string :" << enc << endl;

cout << "\nDecrypted string :" << decr << endl;

return 0;

}

Des

import javax.crypto.Cipher;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

import javax.crypto.spec.SecretKeySpec;

import java.util.Base64;

public class DESExample {

public static void main(String[] args) throws Exception {

String plainText = "Hello world.";

System.out.println("Original Text: " + plainText);

SecretKey secretKey = generateKey();

String encryptedText = encrypt(plainText, secretKey);

System.out.println("Encrypted Text: " + encryptedText);

String decryptedText = decrypt(encryptedText, secretKey);

System.out.println("Decrypted Text: " + decryptedText);

}

public static SecretKey generateKey() throws Exception {

KeyGenerator keyGenerator = KeyGenerator.getInstance("DES");

return keyGenerator.generateKey();

}

public static String encrypt(String plainText, SecretKey secretKey) throws Exception {

Cipher cipher = Cipher.getInstance("DES");

cipher.init(Cipher.ENCRYPT\_MODE, secretKey);

byte[] encryptedBytes = cipher.doFinal(plainText.getBytes());

return Base64.getEncoder().encodeToString(encryptedBytes);

}

public static String decrypt(String encryptedText, SecretKey secretKey) throws Exception {

Cipher cipher = Cipher.getInstance("DES");

cipher.init(Cipher.DECRYPT\_MODE, secretKey);

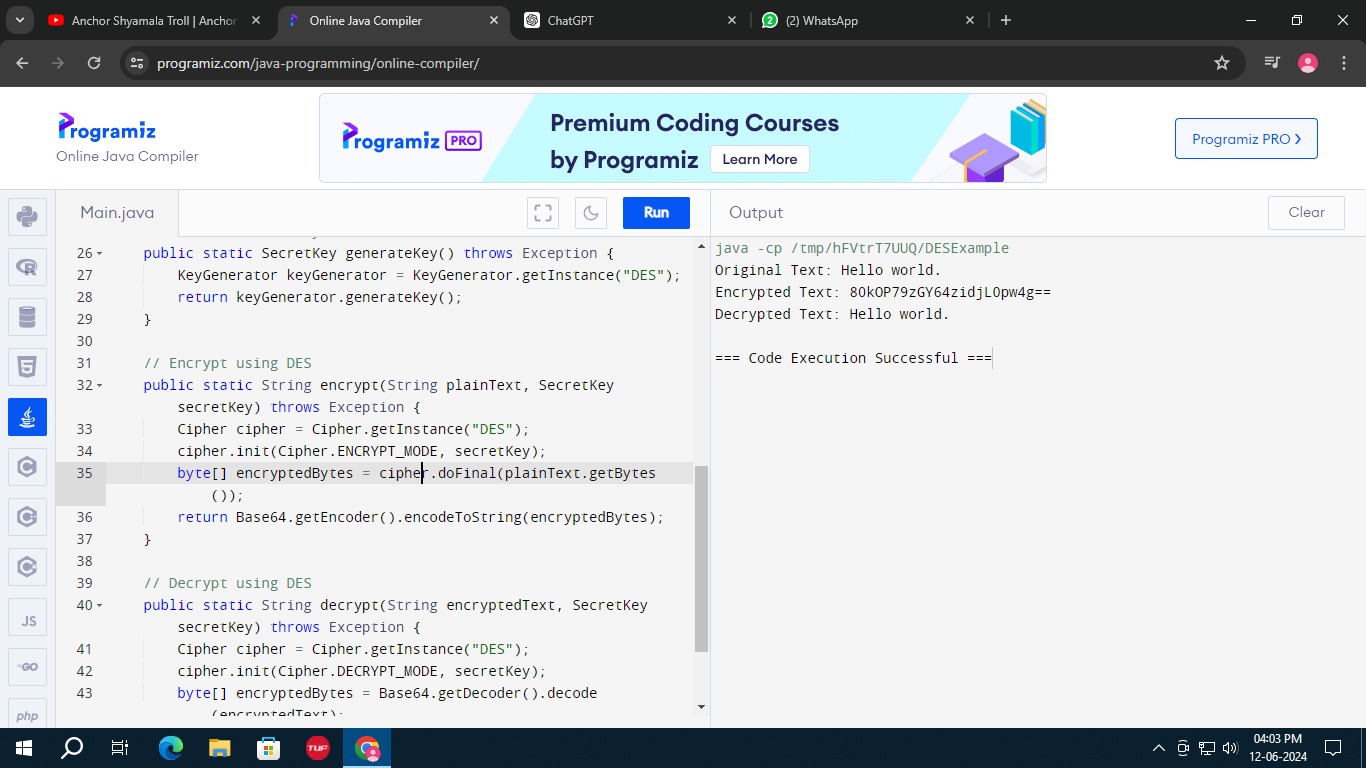
byte[] encryptedBytes = Base64.getDecoder().decode(encryptedText);

byte[] decryptedBytes = cipher.doFinal(encryptedBytes);

return new String(decryptedBytes);

}

}



SHA

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

public class SHA1Example {

public static void main(String[] args) throws NoSuchAlgorithmException {

String input = "Hello, World!";

MessageDigest md = MessageDigest.getInstance("SHA-1");

byte[] hash = md.digest(input.getBytes());

StringBuilder hexString = new StringBuilder();

for (byte b : hash) hexString.append(String.format("%02x", b));

System.out.println("SHA-1 Hash: " + hexString.toString());

}

}