

**Vidyavardhini’s**

**College of Engineering & Technology**

Vasai Road (W)

**Department of**

**Information Technology**

**Security Lab Manual**

|  |  |  |  |
| --- | --- | --- | --- |
| **Semester** | **V** | **Class** | **TE** |
| **Course Code** | **ITL502** | **Academic Year** | **2022-23** |
| **Course Name** | **Security Lab.** | | |
| **Name of Faculty** | **Prof. Thaksen J. Parvat** | | |
| **Suppoting Staff** | **Mr. Nitin Shingane** | | |



**Vidyavardhini’s College of Engineering & Technology**

**Vision**

To be a premier institution of technical education, aiming at becoming a valuable resource for industry and society.

**Mission**

* To provide technologically inspiring environment for learning.
* To promote creativity, innovation and professional activities.
* To inculcate ethical and moral values.
* To cater personal, professional and societal needs through quality education.

**Department Vision:**

To foster and maintain excellence by orienting the captivating minds of the aspiring engineers towards IT- driven technological solutions for the benefits of the society.

**Department Mission:**

* To provide quality education, by employing best and diversified teaching practices and tools, and teaching beyond the confines of the university syllabus.
* To keep students abreast with latest technological advancements in the market.
* To prepare students to troubleshoot and solve IT system problems.

**Program Education Objectives (PEOs):**

* To produce skilled IT Professional to cater social/industrial needs.
* To inculcate an ability to implement modern practices with ethical and professional responsibilities.
* To establish graduate as Business Analyst, System Analyst, Data Scientist, Project Leader.

**Program Specific Outcomes (PSOs):**

The graduates will be able to

* Apply and implement IT solutions in allied fields of engineering to solve real word problems.
* Identify social and industrial problems, provide creative solutions and become quality asset for society and industry.
* Deploy secured solution using Information Technology practices and strategies.

**Program Outcomes (POs):**

Engineering Graduates will be able to:

* **PO1. Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
* **PO2. Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
* **PO3. Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
* **PO4. Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
* **PO5. Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
* **PO6. The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
* **PO7. Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
* **PO8. Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
* **PO9. Individual and teamwork:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
* **PO10. Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
* **PO11. Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
* **PO12. Life-long learning:** Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

**Syllabus (Optional)**

**Hardware & Software Requirements:**

**DETAILED SYLLABUS:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr.**  **No.** | **Module** | **Detailed Content** | **Hours** | **LO**  **Mapping** |
| 0 | **Prerequisite** | Programming Language (C, C++, ,Python, Java), Windows and Unix/Linux operating system. editor  commands (eg nano/vi editor etc) | **02** | - |
| **I** | **Fundamentals of Cryptography** | Classical Encryption techniques (mono-alphabetic and poly-alphabetic substitution techniques: Vigenere cipher, playfair cipher). | **04** | LO1 |
| **II** | **Basics Cryptography** | 1)Block cipher modes of operation using a)Data Encryption Standard b)Advanced Encryption Standard (AES). 2)Public key cryptography: RSA algorithm. 3)Hashing Techniques: HMAC using SHA 4)Digital Signature Schemes – RSA, DSS. | **04** | LO2 |
| **III** | **Network Commands, different Protocols** | 1) Study the use of network reconnaissance tools like WHOIS, dig, traceroute, nslookup to gather information about networks and domain registrars. 2) Study of packet sniffer tools Wireshark, :- a. Observer performance in promiscuous as well as non-promiscuous mode. b. Show the packets can be traced based on different filters. | **06** | LO3 |
| IV | **Network Mapper and Commands** | 1) Download and install nmap. 2) Use it with different options to scan open ports, perform OS fingerprinting, ping scan, tcp port scan, udp port scan, etc. | **04** | LO4 |
| V | **Network Keylogers** | a) Keylogger attack using a keylogger tool. b) Simulate DOS attack using Hping or other tools c) Use the NESSUS/ISO Kali Linux tool to scan the network for vulnerabilities. | **04** | LO5 |
| VI | **Network Security Design** | * 1) Set up IPSec under Linux. 2) Set up Snort and study the logs. 3) Explore the GPG tool to implement email security | **04** | LO6 |

**Text Books:**

1 Build your own Security Lab, Michael Gregg, Wiley India.

2 CCNA Security, Study Guide, TIm Boyles, Sybex.

3 Hands-On Information Security Lab Manual, 4th edition, Andrew Green, Michael Whitman,

Herbert Mattord.

4 The Network Security Test Lab: A Step-by-Step Guide Kindle Edition, Michael Gregg.

**References:**

1 Network Security Bible, Eric Cole, Wiley India.

2 Network Defense and Countermeasures, William (Chuck) Easttom.

3 Principles of Information Security + Hands-on Information Security Lab Manual, 4th Ed. , Michael E. Whitman , Herbert J. Mattord.

4 IITB virtual Lab: <http://cse29-iiith.vlabs.ac.in/>

[**https://www.dcode.fr/enTerm**](https://www.dcode.fr/enTerm) **Work:** Term Work shall consist of at least 10 to 12 practical’s based on the above list. Also Term work Journal must include at least 2 assignments.

**Term Work Marks:** 25 Marks (Total marks) = 15 Marks (Experiment) + 5 Marks (Assignments) + 5 Marks (Attendance)

**Practical & Oral Exam: An Oral & Practical exam will be held based on the above syllabus.**

**ITL (502) Security Lab.**

**Objectives:**

**Sr. No. Lab Objectives**

**The Lab experiments aims:**

1 To apply the knowledge of symmetric cryptography to implement classical ciphers.

2 To analyze and implement public key encryption algorithms, hashing and digital signature algorithms..

3 To explore the different network reconnaissance tools to gather information about networks.

4 To explore the tools like sniffers, port scanners and other related tools for analyzing..

5 To Scan the network for vulnerabilities and simulate attacks.

6To set up intrusion detection systems using open-source technologies and to explore email security.

**Lab Outcomes:**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Lab Outcomes** | **Cognitive levels of attainment as per Bloom’s Taxonomy** |
| On successful completion, of course, learner/student will be able to: | | |
| 1 | Illustrate symmetric cryptography by implementing classical ciphers. | L1, L2 |
| 2 | Demonstrate Key management, distribution and user authentication. | L1, L2 |
| 3 | Explore the different network reconnaissance tools to gather information about networks | L1, L2, L3 |
| 4 | Use tools like sniffers, port scanners and other related tools for analyzing packets in a network. | L1, L2, L3 |
| 5 | Use open-source tools to scan the network for vulnerabilities and simulate attacks. | L1, L2, L3 |
| 6 | Demonstrate the network security system using open source tools. | L1, L2 |

**Year/Sem: T.E/Sem V Subject: Security Lab. ( ITL 502)**

**List of experiments**

|  |  |
| --- | --- |
| **Sr. No.** | **Name of the experiment** |
| 1. | Breaking the Mono-alphabetic Substitution Cipher using Frequency analysis method. |
| 2. | Cryptanalysis or decoding Playfair cipher. |
| 3. | Cryptanalysis or decoding Vigenère cipher. |
| 4. | Encrypt long messages using various modes of operation using AES. |
| 5. | Encrypt long messages using various modes of operation using DES. |
| 6. | Cryptographic Hash Functions and Applications (HMAC): to understand the need, design and applications of collision resistant hash functions. |
| 7. | Implementation and analysis of RSA cryptosystem and Digital signature scheme using RSA. |
| 8. | To Study and understand the functions and applications of a keylogger |
| 9. | Study of packet sniffer tools Wireshark: - a. Observer performance in promiscuous as well as non-promiscuous mode. b. Show the packets can be traced based on different filters. |
| 10 | Download, install Nmap and use it with different options to scan open ports, perform OS fingerprinting, ping scan, tcp port scan, udp port scan, etc. |
| 11. | Use tools like sniffers, port scanners and other related tools for analyzing packets in a network. |
| 12 | Study of Network security by a) Set up IPSec under Linux.  b) Set up Snort and study the logs. |

**Prof. Thaksen Parvat Prof. Thaksen Parvat**

**Subject In charge HOD IT**

**Course Outcomes:**

|  |  |  |
| --- | --- | --- |
| At the end of the course student will be able to: | | Bloom’s Level |
| ITL502.1 | Execute and evaluate different Cryptographic methods for short and long messages. | Apply |
| ITL502.2 | Demonstrate the installation and configuration of network simulator and measure different network scenarios and their performance behaviour. | Apply |
| ITL502.3 | Download, install nmap and use it with different options to scan open ports, perform OS fingerprinting, ping scan, tcp port scan, udp port scan, etc. | Analyse |
| ITL502.4 | Demonstrate tools like sniffers, port scanners and other related tools for analyzing packets in a network. | Analyse |

**Mapping of Experiments with Course Outcomes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Experiment | Course Outcomes | | | |
| ITL502.1 | ITL502.2 | ITL502.3 | ITL502.4 |
| To apply the knowledge of symmetric cryptography to implement classical ciphers. | 3 |  |  |  |
| Identify The Basic Cryptographic Techniques Using Playfair Cipher | 3 |  |  |  |
| To study and implement Vigenere Cipher in fundamental programming languages |  | 3 |  |  |
| To use Advanced Encryption Standard (AES) Algorithm for a practical application like User Message Encryption. |  | 3 |  |  |
| To use Data Encryption Standard (DES) Algorithm for a practical application like User Message Encryption. |  | 3 |  |  |
| To use Hashing Techniques: HMAC and SHA for a practical application like User Message Encryption... |  | 3 |  |  |
| To study and implement RSA asymmetric cryptography key algorithms. |  |  | 3 |  |
| To understand the functions and applications of a keylogger. |  |  | 3 |  |
| To install and study Network Protocol Analyzer, analyze packet header using Wireshark. |  |  |  | 3 |
| To install and implement NMAP. |  |  |  | 3 |

**CERTIFICATE**

**This is to certify that, Mr./MS. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Roll. No.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ of TE Information Technology work in the subject Security Laboratory (ITL 502), Satisfactory in the department IT as prescribed by Mumbai University in the academic year 2023-24 Semester V.**

**Staff In-charge Head of the Department Principal**

Date: 28/10/2023

Place: VCET, Vasai Road

**Index**

**Year/Sem: T.E/Sem V Subject: Network Lab. ( ITL 502)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S.No. | **Name of Experiment** | DOP | DOC | Marks | Sign |
| **Basic Experiments** | | | | | |
| 01 | To apply the knowledge of symmetric cryptography to implement classical ciphers. |  |  |  |  |
| 02 | Identify The Basic Cryptographic Techniques Using Playfair Cipher |  |  |  |  |
| 03 | To study and implement Vigenere Cipher in fundamental programming languages |  |  |  |  |
| 04 | To use Advanced Encryption Standard (AES) Algorithm for a practical application like User Message Encryption. |  |  |  |  |
| 05 | To use Data Encryption Standard (DES) Algorithm for a practical application like User Message Encryption. |  |  |  |  |
| 06 | To use Hashing Techniques: HMAC and SHA for a practical application like User Message Encryption.. |  |  |  |  |
| 07 | To study and implement RSA asymmetric cryptography key algorithms. |  |  |  |  |
| 08 | To understand the functions and applications of a keylogger. |  |  |  |  |
| 09 | To install and study Network Protocol Analyzer, analyze packet header using Wireshark**.** |  |  |  |  |
| 10 | To install and implement NMAP. |  |  |  |  |
|  |  |  |  |  |  |
| **Design Experiments /Project /Case Study/ Assignment** | | | | | |
| 11 | Wireshark |  |  |  |  |
| 12 | Nmap |  |  |  |  |
| 13 | Assignment No. 1 |  |  |  |  |
| 14 | Assignment No. 2 |  |  |  |  |
| 15 | Assignment No. 3 |  |  |  |  |
| 16 | Assignment No. 4 |  |  |  |  |
| 17 | Assignment No. 5 |  |  |  |  |
| 18 | Assignment No. 6 |  |  |  |  |

**Prof. Thaksen Parvat Prof. Thaksen Parvat**

**Subject In charge HOD IT**

**Experiment No. 1**

**Aim** :- To apply the knowledge of symmetric cryptography to implement classical ciphers.

**Theory**:- Cryptography is the technique which is used for doing secure communication between two parties in the public environment where unauthorized users and malicious attackers are present. In cryptography there are two processes i.e. encryption and decryption performed at sender and receiver end respectively. Encryption is the processes where a simple multimedia data is combined with some additional data (known as key) and converted into unreadable encoded format known as Cipher. Decryption is the reverse method as that of encryption where the same or different additional data (key) is used to decode the cipher and it is converted in to the real multimedia data.

**Caesar Cipher in Cryptography**

The Caesar Cipher technique is one of the earliest and simplest methods of encryption technique. It’s simply a type of substitution cipher, i.e., each letter of a given text is replaced by a letter with a fixed number of positions down the alphabet. For example with a shift of 1, A would be replaced by B, B would become C, and so on. The method is apparently named after Julius Caesar, who apparently used it to communicate with his officials.

Thus to cipher a given text we need an integer value, known as a shift which indicates the number of positions each letter of the text been moved down.   
The encryption can be represented using modular arithmetic by first

transforming the letters into numbers, according to the scheme, A = 0, B = 1,…, Z = 25. Encryption of a letter by a shift n can be described mathematically as.

**The formula of encryption is:**

En (x) = (x + k) mod 26

**The formula of decryption is:**

Dn (x) = (xi - k) mod 26

**Example :- encryption**

**Key : 3**

**Plaintext : vartak**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Plaintext** | **v** | **a** | **r** | **t** | **a** | **k** |
| **Plaintext value** | **21** | **0** | **17** | **19** | **0** | **10** |
| (x + k) mod 26 | **24** | **3** | **20** | **22** | **3** | **13** |
| **ciphertext** | **Y** | **D** | **U** | **W** | **D** | **N** |

**Plaintext : vartak**

**Ciphertext : YDUWDN**

**Program & Codes :** So we have implemented The Shift Encryption technique in C++ , Java and Python the 3 generally used programming languages ;

***#1 In C++***

#include <iostream>

using namespace std;

string encrypt(string m, int k)

{

string encryp = "";

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

{

if (isupper(m[i]))

{

// For uppercase

encryp += char(int(m[i] + k - 65) % 26 + 65);

}

else if (islower(m[i]))

{

// For lowercase

encryp += char(int(m[i] + k - 97) % 26 + 97);

}

else

{

// For non-alphabetic characters, just append them as they are

encryp += m[i];

}

}

return encryp;

}

string decrypt(string newmessage, int k)

{

string decrypt = "";

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

{

if (isupper(newmessage[i]))

{

// For uppercase

decrypt += char(int(newmessage[i] - k - 65 + 26) % 26 + 65);

}

else if (islower(newmessage[i]))

{

// For lowercase

decrypt += char(int(newmessage[i] - k - 97 + 26) % 26 + 97);

}

else

{

// For non-alphabetic characters, just append them as they are

decrypt += newmessage[i];

}

}

return decrypt;

}

int main()

{

string m;

int k;

cout << "Enter the Key:\n";

cin >> k;

cout << "Enter a message:\n";

cin.ignore(); // Consume the newline character left in the input buffer

getline(cin, m); // Read the entire line as input

string encryptedMessage = encrypt(m, k);

string decryptedMessage = decrypt(encryptedMessage, k);

cout << "Encrypted Message is:\n" << encryptedMessage << "\n";

cout << "Decrypted Message is:\n" << decryptedMessage;

return 0;

}

***OUTPUT :***

**Enter the key:** 25

**Enter a message:** helloHowAreYou

**Encrypted Message is:** gdkknGnvZqdXnt

**Decrypted Message is:** helloHowAreYou

***#2 In Java***

import java.util.Scanner;

public class Encrypt {

public static String temp = "";

public static int k;

// Encrypts text using a shift cipher

public static String encrypt(String m, int k) {

StringBuilder encrypto = new StringBuilder();

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

if (Character.isUpperCase(m.charAt(i))) {

char ch = (char) (((int) m.charAt(i) + k - 65) % 26 + 65);

encrypto.append(ch);

} else {

char ch = (char) (((int) m.charAt(i) + k - 97) % 26 + 97);

encrypto.append(ch);

}

}

temp = encrypto.toString();

return encrypto.toString();

}

// Decryption

// Decryption

public static String decrypt(String temp, int k) {

StringBuilder decrypto = new StringBuilder();

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

if (Character.isUpperCase(temp.charAt(i))) {

char ch = (char) (((int) temp.charAt(i) - k - 65 + 26) % 26 + 65);

decrypto.append(ch);

} else {

char ch = (char) (((int) temp.charAt(i) - k - 97 + 26) % 26 + 97);

decrypto.append(ch);

}

}

return decrypto.toString();

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter a key");

k = sc.nextInt();

sc.nextLine(); // Consume the newline character left in the input buffer

System.out.println("Enter the text");

String m = sc.nextLine();

System.out.println("Text: " + m);

System.out.println("Key: " + k);

String encryptedText = encrypt(m, k);

System.out.println("Cipher: " + encryptedText);

String decryptedText = decrypt(encryptedText, k);

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

}

}

***OUTPUT :***

**Enter a key:**

**20**

**Enter the text:**

**JavaCipher**

**Text: JavaCipher**

**Key: 20**

**Cipher: DupuWcjbyl**

**Decrypted: JavaCipher**

***#3 in Python***

def encrypt\_func(txt, s):

result = ""

# Traverse the plain text

for char in txt:

# Encrypt uppercase characters in plain text

if char.isupper():

result += chr((ord(char) + s - 65) % 26 + 65)

# Encrypt lowercase characters in plain text

else:

result += chr((ord(char) + s - 97) % 26 + 97)

return result

def decrypt\_func(ciphertext, s):

result1 = ""

# Traverse the ciphertext

for char in ciphertext:

# Decrypt uppercase characters in ciphertext

if char.isupper():

result1 += chr((ord(char) - s - 65) % 26 + 65)

# Decrypt lowercase characters in ciphertext

else:

result1 += chr((ord(char) - s - 97) % 26 + 97)

return result1

s = int(input("Enter the shift key: "))

txt = input("Enter a message: ")

encrypted\_text = encrypt\_func(txt, s)

print("Plain text: " + txt)

print("Shift pattern: " + str(s))

print("Cipher: " + encrypted\_text)

decrypted\_text = decrypt\_func(encrypted\_text, s)

print("Decrypted: " + decrypted\_text)

***OUTPUT :***

**Enter the shift key: 18**

**Enter a message: HelloPythonCipher**

**Plain text: HelloPythonCipher**

**Shift pattern: 18**

**Cipher: ZwddgHqlzgfUahzwj**

**Decrypted: HelloPythonCipher**

**SELF TESTING QUESTIONS :**

Q1 List down all the substitution type Cipher methods in Cryptology.

Q2 What is the difference between The Substitution Cipher and Shift Cipher.

Q3 Give an application of Shift Cipher in Cyber security.

Q4 State drawbacks of Shift Cipher.

Q5 Judging on the working and algorithm of all and only substitution Ciphers which one do you feel is a better option for real time deployment?

**Conclusion** : So therefore we learnt cryptology in Computer Networking , studied simple encryption techniques and implemented shift cipher methodology successfully in the big three languages – C++ , Java & Python.

**Experiment No. 2**

**Aim** : Identify The Basic Cryptographic Techniques Using Playfair Cipher

**Theory :**

When it was first put to the British Foreign Office as a cipher, it was rejected due to its

perceived complexity. However, it was later adopted as a military cipher due to it being

reasonably fast to use, and it requires no special equipment, whilst also providing a stronger

cipher than a Monoalphabetic Substitution Cipher. It was used in the Second Boer War, and

both World War I and World War II to different degrees. It is no longer used by military

forces since the advent of powerful computers, but in its day it provided a relatively secure

cipher which was easy to implement quite quickly.

In order to encrypt using the Playfair Cipher, we must first draw up a Polybius Square (but

without the need for the number headings). This is usually done using a keyword, and either

combining "i" and "j" or omitting "q" from the square.

**Steps to implement the Playfair chipper :**

1. We must now split the plaintext up into digraphs (that is pairs of letters). On each

digraph we perform the following encryption steps:

2. If the digraph consists of the same letter twice (or there is only one letter left by itself

at the end of the plaintext) then insert the letter "X" between the same letters (or at the

end), and then continue with the rest of the steps.

3. If the two letters appear on the same row in the square, then replace each letter by the

letter immediately to the right of it in the square (cycling round to the left hand side if

necessary).

4. If the two letters appear in the same column in the square, then replace each letter by

the letter immediately below it in the square (cycling round to the top of the square if

necessary).

5. Otherwise, form the rectangle for which the two plaintext letters are two opposite

corners. Then replace each plaintext letter with the letter that forms the other corner of

the rectangle that lies on the same row as that plaintext letter (being careful to

maintain the order).

**EXAMPLE:**

Plain Text: This is Final Exam Key : President .

TH IS FI NA LE AX MX

(I) (II)

P R E S I/J

D N T A B

C F G H K

L M O Q U

V W X Y Z

P R E S I/J

D N T A B

C F G H K

L M O Q U

V W X Y Z

P R E S I/J

D N T A B

C F G H K

L M O Q U

V W X Y Z

P R E S I/J

D N T A B

C F G H K

L M O Q U

V W X Y Z

(III) (IV)

(V) (VI)

(VII) (VIII)

Plain Text: This is Final Exam

Chipper Text : AGPIPIKRTBOPYTOW

**C++ PROGRAM :**

#include <bits/stdc++.h>

using namespace std;

#define SIZE 30

// Function to convert the string to lowercase

void toLowerCase(char plain[], int ps) {

int i;

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

if (plain[i] > 64 && plain[i] < 91)

plain[i] += 32;

}

}

// Function to remove all spaces in a string

int removeSpaces(char\* plain, int ps) {

int i, count = 0;

for (i = 0; i < ps; i++)

if (plain[i] != ' ')

plain[count++] = plain[i];

plain[count] = '\0';

return count;

}

// Function to generate the 5x5 key square

void generateKeyTable(char key[], int ks, char keyT[5][5]) {

int i, j, k, flag = 0;

// a 26 character hashmap

// to store count of the alphabet

int dicty[26] = {0};

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

if (key[i] != 'j')

dicty[key[i] - 97] = 2;

}

dicty['j' - 97] = 1;

i = 0;

j = 0;

for (k = 0; k < ks; k++) {

if (dicty[key[k] - 97] == 2) {

dicty[key[k] - 97] -= 1;

keyT[i][j] = key[k];

j++;

if (j == 5) {

i++;

j = 0;

}

}

}

for (k = 0; k < 26; k++) {

if (dicty[k] == 0) {

keyT[i][j] = (char)(k + 97);

j++;

if (j == 5) {

i++;

j = 0;

}

}

}

}

// Function to search for the characters of a digraph

// in the key square and return their position

void search(char keyT[5][5], char a, char b, int arr[]) {

int i, j;

if (a == 'j')

a = 'i';

else if (b == 'j')

b = 'i';

for (i = 0; i < 5; i++) {

for (j = 0; j < 5; j++) {

if (keyT[i][j] == a) {

arr[0] = i;

arr[1] = j;

} else if (keyT[i][j] == b) {

arr[2] = i;

arr[3] = j;

}

}

}

}

// Function to find the modulus with 5

int mod5(int a) { return (a % 5); }

// Function to make the plain text length even

int prepare(char str[], int ptrs) {

if (ptrs % 2 != 0) {

str[ptrs++] = 'z';

str[ptrs] = '\0';

}

return ptrs;

}

// Function for performing the encryption

void encrypt(char str[], char keyT[5][5], int ps) {

int i, a[4];

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

search(keyT, str[i], str[i + 1], a);

if (a[0] == a[2]) {

str[i] = keyT[a[0]][mod5(a[1] + 1)];

str[i + 1] = keyT[a[0]][mod5(a[3] + 1)];

} else if (a[1] == a[3]) {

str[i] = keyT[mod5(a[0] + 1)][a[1]];

str[i + 1] = keyT[mod5(a[2] + 1)][a[1]];

} else {

str[i] = keyT[a[0]][a[3]];

str[i + 1] = keyT[a[2]][a[1]];

}

}

}

// Function for performing the decryption

void decrypt(char str[], char keyT[5][5], int ps) {

int i, a[4];

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

search(keyT, str[i], str[i + 1], a);

if (a[0] == a[2]) {

str[i] = keyT[a[0]][mod5(a[1] - 1)];

str[i + 1] = keyT[a[0]][mod5(a[3] - 1)];

} else if (a[1] == a[3]) {

str[i] = keyT[mod5(a[0] - 1)][a[1]];

str[i + 1] = keyT[mod5(a[2] - 1)][a[1]];

} else {

str[i] = keyT[a[0]][a[3]];

str[i + 1] = keyT[a[2]][a[1]];

}

}

}

// Function to encrypt using Playfair Cipher

void encryptByPlayfairCipher(char str[], char key[]) {

char ps, ks, keyT[5][5];

// Key

ks = strlen(key);

ks = removeSpaces(key, ks);

toLowerCase(key, ks);

// Plaintext

ps = strlen(str);

toLowerCase(str, ps);

ps = removeSpaces(str, ps);

ps = prepare(str, ps);

generateKeyTable(key, ks, keyT);

encrypt(str, keyT, ps);

}

// Function to decrypt using Playfair Cipher

void decryptByPlayfairCipher(char str[], char key[]) {

char ps, ks, keyT[5][5];

// Key

ks = strlen(key);

ks = removeSpaces(key, ks);

toLowerCase(key, ks);

// Ciphertext

ps = strlen(str);

toLowerCase(str, ps);

ps = removeSpaces(str, ps);

ps = prepare(str, ps);

generateKeyTable(key, ks, keyT);

decrypt(str, keyT, ps);

}

int main() {

char str[SIZE], key[SIZE];

// Key to be encrypted

string key1, str1;

cout<<"Enter the key:"<<endl;

cin>>key1;

cout<<"Enter the plain text:"<<endl;

cin>>str1;

strcpy(key, key1.c\_str());

cout << "Key text: " << key << "\n";

// Plaintext to be encrypted

strcpy(str, str1.c\_str());

cout << "Plain text: " << str << "\n";

// Encrypt using Playfair Cipher

encryptByPlayfairCipher(str, key);

cout << "Cipher text: " << str << "\n";

// Decrypt using Playfair Cipher

decryptByPlayfairCipher(str, key);

cout << "Decrypted text: " << str << "\n";

return 0;

}

**Output :**

**Enter the key: thisisakey**

**Enter the plain text: okaylets**

**Key text: thisisakey**

**Plain text: okaylets**

**Cipher text: neicfbha**

**Decrypted text: okaylets**

**JAVA PORGRAM :**

import java.awt.Point;

import java.util.Scanner;

public class PlayfairCipher {

// Length of digraph array

private int length = 0;

// Creates a matrix for Playfair cipher

private String[][] table;

// Main method to test Playfair method

public static void main(String[] args) {

new PlayfairCipher();

}

// Main run of the program, Playfair method

// Constructor of the class

private PlayfairCipher() {

// Prompts the user for the keyword to use for encoding & creates tables

System.out.print("Enter the key for playfair cipher: ");

Scanner sc = new Scanner(System.in);

String key = parseString(sc);

while (key.equals(""))

key = parseString(sc);

table = this.cipherTable(key);

// Prompts the user for the message to be encoded

System.out.print("Enter the plaintext to be encipher: ");

String input = parseString(sc);

while (input.equals(""))

input = parseString(sc);

// Encodes and then decodes the encoded message

String output = cipher(input);

String decodedOutput = decode(output);

// Output the results to the user

keyTable(table);

printResults(output, decodedOutput);

}

// Parses an input string to remove numbers, punctuation, replaces J's with I's, and makes the string all caps

private String parseString(Scanner sc) {

String parse = sc.nextLine();

// Converts all the letters to upper case

parse = parse.toUpperCase();

// Replace anything that is not A-Z with an empty string

parse = parse.replaceAll("[^A-Z]", "");

// Replace the letter 'J' with 'I'

parse = parse.replace("J", "I");

return parse;

}

// Creates the cipher table based on the input string (already parsed)

private String[][] cipherTable(String key) {

// Creates a matrix of 5x5

String[][] playfairTable = new String[5][5];

String keyString = key + "ABCDEFGHIKLMNOPQRSTUVWXYZ";

// Fill the string array with empty strings

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 5; j++) {

playfairTable[i][j] = "";

}

}

for (int k = 0; k < keyString.length(); k++) {

boolean repeat = false;

boolean used = false;

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 5; j++) {

if (playfairTable[i][j].equals("" + keyString.charAt(k))) {

repeat = true;

} else if (playfairTable[i][j].equals("") && !repeat && !used) {

playfairTable[i][j] = "" + keyString.charAt(k);

used = true;

}

}

}

}

return playfairTable;

}

// Cipher: takes input (all upper-case), encodes it, and returns the output

private String cipher(String in) {

length = (int) in.length() / 2 + in.length() % 2;

// Insert 'X' between double-letter digraphs and redefine "length"

for (int i = 0; i < (length - 1); i++) {

if (in.charAt(2 \* i) == in.charAt(2 \* i + 1)) {

in = new StringBuffer(in).insert(2 \* i + 1, 'X').toString();

length = (int) in.length() / 2 + in.length() % 2;

}

}

// Makes the plaintext of even length and creates an array of digraphs

String[] digraph = new String[length];

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

if (j == (length - 1) && in.length() / 2 == (length - 1))

// If not even, add 'X' at the end of the plaintext

in = in + "X";

digraph[j] = in.charAt(2 \* j) + "" + in.charAt(2 \* j + 1);

}

// Encodes the digraphs and returns the output

String out = "";

String[] encDigraphs = new String[length];

encDigraphs = encodeDigraph(digraph);

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

out = out + encDigraphs[k];

return out;

}

// Encryption logic: encodes the digraph input with the cipher's specifications

private String[] encodeDigraph(String[] di) {

String[] encipher = new String[length];

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

char a = di[i].charAt(0);

char b = di[i].charAt(1);

int r1 = (int) getPoint(a).getX();

int r2 = (int) getPoint(b).getX();

int c1 = (int) getPoint(a).getY();

int c2 = (int) getPoint(b).getY();

// Executes if the letters of the digraph appear in the same row

if (r1 == r2) {

c1 = (c1 + 1) % 5;

c2 = (c2 + 1) % 5;

}

// Executes if the letters of the digraph appear in the same column

else if (c1 == c2) {

r1 = (r1 + 1) % 5;

r2 = (r2 + 1) % 5;

} else {

int temp = c1;

c1 = c2;

c2 = temp;

}

// Performs the table look-up and puts those values into the encoded array

encipher[i] = table[r1][c1] + "" + table[r2][c2];

}

return encipher;

}

// Decryption logic: decodes the output given from the cipher and decode methods

private String decode(String out) {

String decoded = "";

for (int i = 0; i < out.length() / 2; i++) {

char a = out.charAt(2 \* i);

char b = out.charAt(2 \* i + 1);

int r1 = (int) getPoint(a).getX();

int r2 = (int) getPoint(b).getX();

int c1 = (int) getPoint(a).getY();

int c2 = (int) getPoint(b).getY();

if (r1 == r2) {

c1 = (c1 + 4) % 5;

c2 = (c2 + 4) % 5;

} else if (c1 == c2) {

r1 = (r1 + 4) % 5;

r2 = (r2 + 4) % 5;

} else {

// Swapping logic

int temp = c1;

c1 = c2;

c2 = temp;

}

decoded = decoded + table[r1][c1] + table[r2][c2];

}

// Returns the decoded message

return decoded;

}

// Returns a point containing the row and column of the letter

private Point getPoint(char c) {

Point pt = new Point(0, 0);

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 5; j++) {

if (c == table[i][j].charAt(0)) {

pt = new Point(i, j);

}

}

}

return pt;

}

// Function prints the key-table in matrix form for Playfair cipher

private void keyTable(String[][] printTable) {

System.out.println("Playfair Cipher Key Matrix: ");

System.out.println();

// Loop iterates for rows

for (int i = 0; i < 5; i++) {

// Loop iterates for columns

for (int j = 0; j < 5; j++) {

// Prints the key-table in matrix form

System.out.print(printTable[i][j] + " ");

}

System.out.println();

}

System.out.println();

}

// Method that prints all the results

private void printResults(String encipher, String dec) {

System.out.print("Encrypted Message: ");

// Prints the encrypted message

System.out.println(encipher);

System.out.println();

System.out.print("Decrypted Message: ");

// Prints the decrypted message

System.out.println(dec);

}

}

**Output:**

**Enter the key for playfair cipher: PLAYFAIRKEY**

**Enter the plaintext to be encipher: TEXTPLAINALL**

**Playfair Cipher Key Matrix:**

**P L A Y F**

**I R K E B**

**C D G H M**

**N O Q S T**

**U V W X Z**

**Encrypted Message: SBZSLAPKQPAA**

**Decrypted Message: TEXTPLAINALL**

**PYTHON PROGRAM :**

key=input("Enter key: ")

key=key.replace(" ", "")

key=key.upper()

def matrix(x,y,initial):

return [[initial for i in range(x)] for j in range(y)]

result=list()

for c in key: #storing key

if c not in result:

if c=='J':

result.append('I')

else:

result.append(c)

flag=0

for i in range(65,91): #storing other character

if chr(i) not in result:

if i==73 and chr(74) not in result:

result.append("I")

flag=1

elif flag==0 and i==73 or i==74:

pass

else:

result.append(chr(i))

k=0

my\_matrix=matrix(5,5,0) #initialize matrix

for i in range(0,5): #making matrix

for j in range(0,5):

my\_matrix[i][j]=result[k]

k+=1

def locindex(c): #get location of each character

loc=list()

if c=='J':

c='I'

for i ,j in enumerate(my\_matrix):

for k,l in enumerate(j):

if c==l:

loc.append(i)

loc.append(k)

return loc

def encrypt(): #Encryption

msg=str(input("ENTER MSG: "))

msg=msg.upper()

msg=msg.replace(" ", "")

i=0

for s in range(0,len(msg)+1,2):

if s<len(msg)-1:

if msg[s]==msg[s+1]:

msg=msg[:s+1]+'X'+msg[s+1:]

if len(msg)%2!=0:

msg=msg[:]+'X'

print("CIPHER TEXT:",end=' ')

while i<len(msg):

loc=list()

loc=locindex(msg[i])

loc1=list()

loc1=locindex(msg[i+1])

if loc[1]==loc1[1]:

print("{}{}".format(my\_matrix[(loc[0]+1)%5][loc[1]],my\_matrix[(loc1[0]+1)%5][loc1[1]]),end=' ')

elif loc[0]==loc1[0]:

print("{}{}".format(my\_matrix[loc[0]][(loc[1]+1)%5],my\_matrix[loc1[0]][(loc1[1]+1)%5]),end=' ')

else:

print("{}{}".format(my\_matrix[loc[0]][loc1[1]],my\_matrix[loc1[0]][loc[1]]),end=' ')

i=i+2

def decrypt(): #decryption

msg=str(input("ENTER CIPHER TEXT: "))

msg=msg.upper()

msg=msg.replace(" ", "")

print("PLAIN TEXT:",end=' ')

i=0

while i<len(msg):

loc=list()

loc=locindex(msg[i])

loc1=list()

loc1=locindex(msg[i+1])

if loc[1]==loc1[1]:

print("{}{}".format(my\_matrix[(loc[0]-1)%5][loc[1]],my\_matrix[(loc1[0]-1)%5][loc1[1]]),end=' ')

elif loc[0]==loc1[0]:

print("{}{}".format(my\_matrix[loc[0]][(loc[1]-1)%5],my\_matrix[loc1[0]][(loc1[1]-1)%5]),end=' ')

else:

print("{}{}".format(my\_matrix[loc[0]][loc1[1]],my\_matrix[loc1[0]][loc[1]]),end=' ')

i=i+2

while(1):

choice=int(input("\n 1.Encryption \n 2.Decryption: \n 3.EXIT\nEnter your choice: "))

if choice==1:

encrypt()

elif choice==2:

decrypt()

elif choice==3:

exit()

else:

print("Choose correct choice")

**OUTPUT:**

**Enter key: ciphergod**

**1.Encryption 2.Decryption 3.EXIT**

**Enter your choice: 1**

**ENTER MSG: testmessage**

**CIPHER TEXT: UH TU UA XP UO AI**

**1.Encryption 2.Decryption: 3.EXIT**

**Enter your choice: 2**

**ENTER CIPHER TEXT: UHTUUAXPUOAI**

**PLAIN TEXT: TE ST ME SX SA GE**

**1.Encryption 2.Decryption: 3.EXIT**

**Enter your choice: 3**

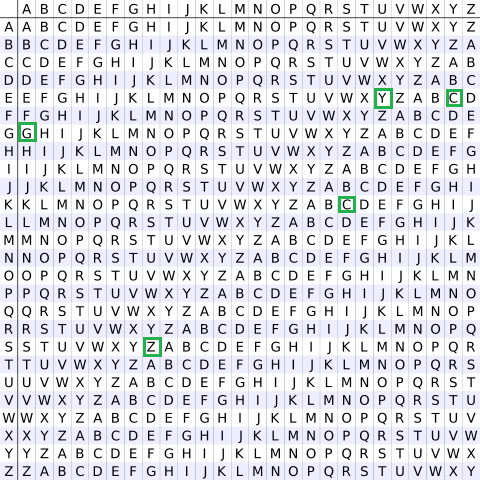
Conclusion : Thus implement playfair chipper using different techniques

**Experiment No. 3**

**Aim :**  To study and implement Vigenere Cipher in fundamental programming languages .

**Theory :**

**Vigenere Cipher** - Vigenere Cipher is a method of encrypting alphabetic text. It uses a simple form of [polyalphabetic substitution](https://en.wikipedia.org/wiki/Polyalphabetic_cipher). A polyalphabetic cipher is any cipher based on substitution, using multiple substitution alphabets. The encryption of the original text is done using the [Vigenère square or Vigenère table](https://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher#/media/File:Vigen%C3%A8re_square_shading.svg).

* The table consists of the alphabets written out 26 times in different rows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible[Caesar Ciphers](https://www.geeksforgeeks.org/caesar-cipher/).
* At different points in the encryption process, the cipher uses a different alphabet from one of the rows.
* The alphabet used at each point depends on a repeating keyword.
* 

**Algorithm For Shift Cipher :**

**Encryption:**

The first letter of the plaintext, G is paired with A, the first letter of the key. So use row G and column A of the Vigenère square, namely G. Similarly, for the second letter of the plaintext, the second letter of the key is used, the letter at row E, and column Y is C. The rest of the plaintext is enciphered in a similar fashion.

**Decryption:**  
Decryption is performed by going to the row in the table corresponding to the key, finding the position of the ciphertext letter in this row, and then using the column’s label as the plaintext.

A more **easy implementation** could be to visualize Vigenère algebraically by converting [A-Z] into numbers [0–25].

**Encryption**

The plaintext(P) and key(K) are added modulo 26.

Ei = (Pi + Ki) mod 26

**Decryption**

Di = (Ei - Ki + 26) mod 26

**Program & Codes :**  So we have implemented The Vigenere Encryption technique in C++ , Java and Python the 3 generally used programming languages ;

***#1 In C++***

#include<bits/stdc++.h>

using namespace std;

string generateKey(string str, string key) {

int x = str.size();

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

if (i == key.size())

i = 0;

key.push\_back(key[i]);

}

return key;

}

string cipherText(string str, string key) {

string cipher\_text;

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

char x = (str[i] + key[i] - 2 \* 'A') % 26 + 'A'; // Changed modulo logic

cipher\_text.push\_back(x);

}

return cipher\_text;

}

string originalText(string cipher\_text, string key) {

string orig\_text;

for (int i = 0; i < cipher\_text.size(); i++) {

char x = (cipher\_text[i] - key[i] + 26) % 26 + 'A'; // Changed modulo logic

orig\_text.push\_back(x);

}

return orig\_text;

}

int main() {

string str;

string keyword;

cout << "Enter a string: ";

cin >> str;

cout << "Enter a key: ";

cin >> keyword;

string key = generateKey(str, keyword);

string cipher\_text = cipherText(str, key);

cout << "Ciphertext: " << cipher\_text << "\n";

cout << "Original/Decrypted Text: " << originalText(cipher\_text, key) << endl;

return 0;

}

***OUTPUT :***

**Enter a string: VIGENERE**

**Enter a key: CIPHER**

**Ciphertext: XQVLRVTM**

**Original/Decrypted Text: VIGENERE**

***#2 In Java***

import java.util.\*;

public class Vigenere {

static String generateKey(String str, String key) {

int x = str.length();

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

if (x == i)

i = 0;

if (key.length() == str.length())

break;

key += (key.charAt(i));

}

return key;

}

static String cipherText(String str, String key) {

String cipher\_text = "";

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

int x = (str.charAt(i) + key.charAt(i)) % 26;

// convert into alphabets(ASCII)

x += 'A';

cipher\_text += (char) (x);

}

return cipher\_text;

}

static String originalText(String cipher\_text, String key) {

String orig\_text = "";

for (int i = 0; i < cipher\_text.length() &&

i < key.length(); i++) {

int x = (cipher\_text.charAt(i) -

key.charAt(i) + 26) % 26;

// convert into alphabets(ASCII)

x += 'A';

orig\_text += (char) (x);

}

return orig\_text;

}

static String LowerToUpper(String s) {

StringBuffer str = new StringBuffer(s);

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

if (Character.isLowerCase(s.charAt(i))) {

str.setCharAt(i, Character.toUpperCase(s.charAt(i)));

}

}

s = str.toString();

return s;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter a string\n");

String Str = sc.nextLine();

System.out.println("Enter a a keyword\n");

String Keyword = sc.nextLine();

String str = LowerToUpper(Str);

String keyword = LowerToUpper(Keyword);

String key = generateKey(str, keyword);

String cipher\_text = cipherText(str, key);

System.out.println("Ciphertext : "

+ cipher\_text + "\n");

System.out.println("Original/Decrypted Text : "

+ originalText(cipher\_text, key));

}

}

***OUTPUT :***

**Enter a string: JAVACIPHER**

**Enter a keyword: VIGENERE**

**Ciphertext : EIBEPMGLZZ**

**Original/Decrypted Text : JAVACIPHER**

***#3 in Python***

def generateKey(string, key):

key = list(key)

if len(string) == len(key):

return key

else:

for i in range(len(string) - len(key)):

key.append(key[i % len(key)])

return "".join(key)

def cipherText(string, key):

cipher\_text = []

for i in range(len(string)):

x = (ord(string[i]) + ord(key[i]) - 2 \* ord('A')) % 26 + ord('A')

cipher\_text.append(chr(x))

return "".join(cipher\_text)

def originalText(cipher\_text, key):

orig\_text = []

for i in range(len(cipher\_text)):

x = (ord(cipher\_text[i]) - ord(key[i]) + 26) % 26 + ord('A')

orig\_text.append(chr(x))

return "".join(orig\_text)

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

string = input("Enter a string: ")

keyword = input("Enter a keyword: ")

key = generateKey(string, keyword)

cipher\_text = cipherText(string, key)

print("Ciphertext:", cipher\_text)

print("Original/Decrypted Text:", originalText(cipher\_text, key))

***OUTPUT :***

**Enter a string: PYTHONCIPHER**

**Enter a keyword: VIGENERE**

**Ciphertext: KGZLBRTMKPKV**

**Original/Decrypted Text: PYTHONCIPHER**

**SELF TESTING QUESTIONS :**

Q1 List down all the substitution type Cipher methods in Cryptology .

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Q2 What is the difference between The Vigenere Cipher and Shift Cipher .

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Q3 Give an application of Vigenere Cipher in Cybersecurity.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Q4 State drawbacks of Vigenere Cipher .

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Q5 Judging on the working and algorithm of all and only substitution Ciphers do you feel vigenere cipher is a better option for real time deployment ?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Conclusion** : So therefore we learnt vigenere cryptology in Computer Networking , studied its cryptanalysis and implemented its algorithm methodology successfully in the big three languages – C++ , Java & Python

**Experiment No. 4**

**Aim:** To use Advanced Encrytion Standard (AES) Algorithm for a practical application like User Message Encryption.

**Theory:** The **AES algorithm** (also known as the **Rijndael algorithm**) is a symmetrical block cipher algorithm that takes plain text in blocks of 128 bits and converts them to ciphertext using keys of 128, 192, and 256 bits. Since the AES algorithm is considered secure, it is in the worldwide standard.

**Algorithm:**

Step – 1: Derive the set of round keys from the cipher key.

Step – 2: Initialize the state array with the block data (plaintext).

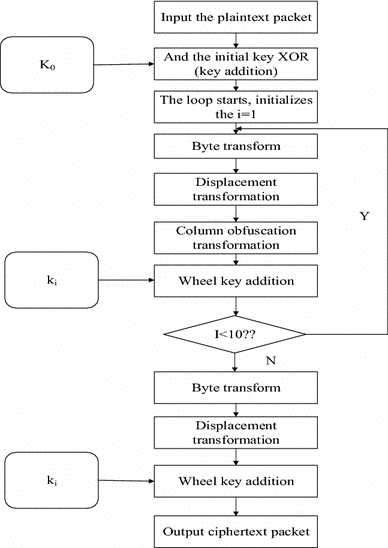
Step – 3: Add the initial round key to the starting state array.

Step – 4: Perform nine rounds of state manipulation.

Step – 5: Perform the tenth and final round of state manipulation.

Step – 6: Copy the final state array out as the encrypted data (ciphertext).

**Flowchart:**



**Program & Codes:** So we have implemented the advanced encryption technique in C++, Java 7 Python the 3 generally used programming languages.

#1 In C++ (Encryption):

#include "pch.h"

#include <iostream>

#include "aes.h"

#include <Windows.h>

#include "osrng.h"

using CryptoPP::AutoSeededRandomPool;

#include <iostream>

using std::cout;

using std::cerr;

using std::endl;

#include <string>

using std::string;

#include <cstdlib>

using std::exit;

#include "cryptlib.h"

using CryptoPP::Exception;

#include "hex.h"

using CryptoPP::HexEncoder;

using CryptoPP::HexDecoder;

#include "filters.h"

using CryptoPP::StringSink;

using CryptoPP::StringSource;

using CryptoPP::StreamTransformationFilter;

#include "aes.h"

using CryptoPP::AES;

#include "ccm.h"

using CryptoPP::CBC\_Mode;

#include "assert.h"

int main(int argc, char\* argv[])

{

AutoSeededRandomPool prng;

byte key[AES::DEFAULT\_KEYLENGTH];

prng.GenerateBlock(key, sizeof(key));

byte iv[AES::BLOCKSIZE];

prng.GenerateBlock(iv, sizeof(iv));

string plain = "CBC Mode Test";

string cipher, encoded, recovered;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Pretty print key

encoded.clear();

StringSource(key, sizeof(key), true,

new HexEncoder(

new StringSink(encoded)

) // HexEncoder

); // StringSource

cout << "key: " << encoded << endl;

// Pretty print iv

encoded.clear();

StringSource(iv, sizeof(iv), true,

new HexEncoder(

new StringSink(encoded)

) // HexEncoder

); // StringSource

cout << "iv: " << encoded << endl;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

try

{

cout << "plain text: " << plain << endl;

CBC\_Mode< AES >::Encryption e;

e.SetKeyWithIV(key, sizeof(key), iv);

// The StreamTransformationFilter removes

// padding as required.

StringSource s(plain, true,

new StreamTransformationFilter(e,

new StringSink(cipher)

) // StreamTransformationFilter

); // StringSource

#if 0

StreamTransformationFilter filter(e);

filter.Put((const byte\*)plain.data(), plain.size());

filter.MessageEnd();

const size\_t ret = filter.MaxRetrievable();

cipher.resize(ret);

filter.Get((byte\*)cipher.data(), cipher.size());

#endif

}

catch (const CryptoPP::Exception& e)

{

cerr << e.what() << endl;

exit(1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Pretty print

encoded.clear();

StringSource(cipher, true,

new HexEncoder(

new StringSink(encoded)

) // HexEncoder

); // StringSource

cout << "cipher text: " << encoded << endl;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

try

{

CBC\_Mode< AES >::Decryption d;

d.SetKeyWithIV(key, sizeof(key), iv);

// The StreamTransformationFilter removes

// padding as required.

StringSource s(cipher, true,

new StreamTransformationFilter(d,

new StringSink(recovered)

) // StreamTransformationFilter

); // StringSource

#if 0

StreamTransformationFilter filter(d);

filter.Put((const byte\*)cipher.data(), cipher.size());

filter.MessageEnd();

const size\_t ret = filter.MaxRetrievable();

recovered.resize(ret);

filter.Get((byte\*)recovered.data(), recovered.size());

#endif

cout << "recovered text: " << recovered << endl;

}

catch (const CryptoPP::Exception& e)

{

cerr << e.what() << endl;

exit(1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

return 0;

}

Output:

**key: B8E971CD9C7DB8F3E6CB8221104889F0**

**iv: FADF5FB52A9210FF0E4AC031AF622E5E**

**plain text: CBC Mode Test**

**cipher text: 22B0345D31F3082445A8808EA8E75121**

**recovered text: CBC Mode Test**

2. **Java:**

**import** javax.crypto.Cipher;

**import** javax.crypto.SecretKey;

**import** javax.crypto.SecretKeyFactory;

**import** javax.crypto.spec.IvParameterSpec;

**import** javax.crypto.spec.PBEKeySpec;

**import** javax.crypto.spec.SecretKeySpec;

**import** java.nio.charset.StandardCharsets;

**import** java.security.InvalidAlgorithmParameterException;

**import** java.security.InvalidKeyException;

**import** java.security.NoSuchAlgorithmException;

**import** java.security.spec.InvalidKeySpecException;

**import** java.security.spec.KeySpec;

**import** java.util.Base64;

**import** javax.crypto.BadPaddingException;

**import** javax.crypto.IllegalBlockSizeException;

**import** javax.crypto.NoSuchPaddingException;

**public** **class** AESExample

{

/\* Private variable declaration \*/

**private** **static** **final** String SECRET\_KEY = "123456789";

**private** **static** **final** String SALTVALUE = "abcdefg";

/\* Encryption Method \*/

**public** **static** String encrypt(String strToEncrypt)

{

**try**

{

/\* Declare a byte array. \*/

**byte**[] iv = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

IvParameterSpec ivspec = **new** IvParameterSpec(iv);

/\* Create factory for secret keys. \*/

SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");

/\* PBEKeySpec class implements KeySpec interface. \*/

KeySpec spec = **new** PBEKeySpec(SECRET\_KEY.toCharArray(), SALTVALUE.getBytes(), 65536, 256);

SecretKey tmp = factory.generateSecret(spec);

SecretKeySpec secretKey = **new** SecretKeySpec(tmp.getEncoded(), "AES");

Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");

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

/\* Retruns encrypted value. \*/

**return** Base64.getEncoder()

.encodeToString(cipher.doFinal(strToEncrypt.getBytes(StandardCharsets.UTF\_8)));

}

**catch** (InvalidAlgorithmParameterException | InvalidKeyException | NoSuchAlgorithmException | InvalidKeySpecException | BadPaddingException | IllegalBlockSizeException | NoSuchPaddingException e)

{

System.out.println("Error occured during encryption: " + e.toString());

}

**return** **null**;

}

/\* Decryption Method \*/

**public** **static** String decrypt(String strToDecrypt)

{

**try**

{

/\* Declare a byte array. \*/

**byte**[] iv = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

IvParameterSpec ivspec = **new** IvParameterSpec(iv);

/\* Create factory for secret keys. \*/

SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");

/\* PBEKeySpec class implements KeySpec interface. \*/

KeySpec spec = **new** PBEKeySpec(SECRET\_KEY.toCharArray(), SALTVALUE.getBytes(), 65536, 256);

SecretKey tmp = factory.generateSecret(spec);

SecretKeySpec secretKey = **new** SecretKeySpec(tmp.getEncoded(), "AES");

Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");

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

/\* Retruns decrypted value. \*/

**return** **new** String(cipher.doFinal(Base64.getDecoder().decode(strToDecrypt)));

}

**catch** (InvalidAlgorithmParameterException | InvalidKeyException | NoSuchAlgorithmException | InvalidKeySpecException | BadPaddingException | IllegalBlockSizeException | NoSuchPaddingException e)

{

System.out.println("Error occured during decryption: " + e.toString());

}

**return** **null**;

}

/\* Driver Code \*/

**public** **static** **void** main(String[] args)

{

/\* Message to be encrypted. \*/

String originalval = "AES Encryption";

/\* Call the encrypt() method and store result of encryption. \*/

String encryptedval = encrypt(originalval);

/\* Call the decrypt() method and store result of decryption. \*/

String decryptedval = decrypt(encryptedval);

/\* Display the original message, encrypted message and decrypted message on the console. \*/

System.out.println("Original value: " + originalval);

System.out.println("Encrypted value: " + encryptedval);

System.out.println("Decrypted value: " + decryptedval);

}

}

**Output:**

**Original value: AES Encryption**

**Encrypted value: V5E9I52IxhMaW4+hJh156g==**

**Decrypted value: AES Encryption**

**3. Python:**

import base64  
import hashlib  
from Crypto.Cipher import AES  
from Crypto import Random  
BLOCK\_SIZE = 16  
pad = lambda s: s + (BLOCK\_SIZE - len(s) % BLOCK\_SIZE) \* chr(BLOCK\_SIZE - len(s) % BLOCK\_SIZE)  
unpad = lambda s: s[:-ord(s[len(s) - 1:])]  
def encrypt(plain\_text, key):  
 private\_key = hashlib.sha256(key.encode("utf-8")).digest()  
 plain\_text = pad(plain\_text)  
 print("After padding:", plain\_text)  
 iv = Random.new().read(AES.block\_size)  
 cipher = AES.new(private\_key, AES.MODE\_CBC, iv)  
 return base64.b64encode(iv + cipher.encrypt(plain\_text))  
def decrypt(cipher\_text, key):  
 private\_key = hashlib.sha256(key.encode("utf-8")).digest()  
 cipher\_text = base64.b64decode(cipher\_text)  
 iv = cipher\_text[:16]  
 cipher = AES.new(private\_key, AES.MODE\_CBC, iv)  
 return unpad(cipher.decrypt(cipher\_text[16:]))  
message=input("Enter message to encrypt: ");  
key = input("Enter encryption key: ")  
encrypted\_msg = encrypt(message, key)  
print("Encrypted Message:", encrypted\_msg)  
decrypted\_msg = decrypt(encrypted\_msg, key)  
print("Decrypted Message:", bytes.decode(decrypted\_msg))

**Output:**

**Enter message to encrypt: hello**

**Enter encryption key: 12345**

**After padding: hello**

**Encrypted Message: b’r3V0A0Ssjw/4ZOKL42/hWSQ0LKy7lt9b)Vt7D&5RA3E=’**

**Decrypted Message: hello**

**Conclusion:** We have successfully implemented Advanced Encryption Standard (AES) Algorithm using various modes of operations.

**Experiment No. 5**

**Aim :**  To use Data Encryption Standard (DES) Algorithm for a practical application like User Message Encryption.

**Theory :**

**Data encryption standard (DES)** has been found vulnerable to very powerful attacks and therefore, the popularity of DES has been found slightly on the decline. DES is a block cipher and encrypts data in blocks of size of 64 bits each, which means 64 bits of plain text go as the input to DES, which produces 64 bits of ciphertext. The same algorithm and key are used for encryption and decryption, with minor differences. The key length is 56 bit

**ALGORITHM:**

**STEP-1:** Read the 64-bit plain text.

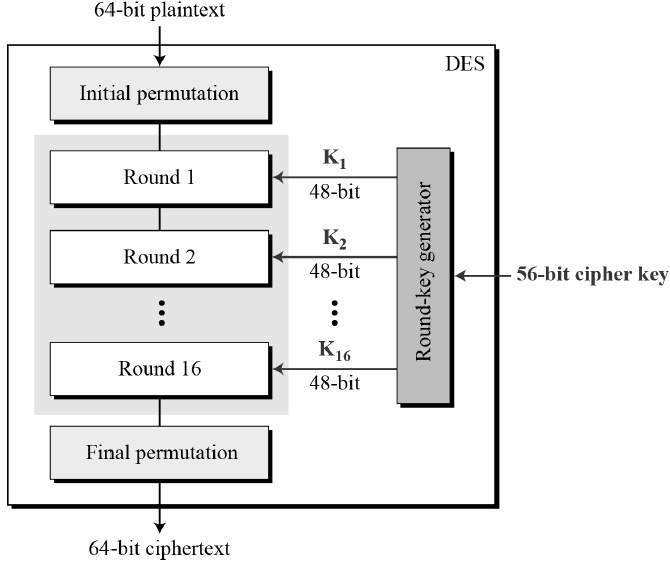
**STEP-2:** Split it into two 32-bit blocks and store it in two different arrays.

**STEP-3:** Perform XOR operation between these two arrays.

**STEP-4:** The output obtained is stored as the second 32-bit sequence and the original second 32-bit sequence forms the first part.

**STEP-5:** Thus the encrypted 64-bit cipher text is obtained in this way. Repeat the same process for the remaining plain text characters.

**Flowchart :**



**Program & Codes :**  So we have implemented The Shift Encryption technique in C++ , Java and Python the 3 generally used programming languages ;

**#1 In C++**

// Including dependancies

#include <iostream>

#include <string>

#include <cmath>

using namespace std;

// Array to hold 16 keys

string round\_keys[16];

// String to hold the plain text

string pt;

// Function to convert a number in decimal to binary

string convertDecimalToBinary(int decimal)

{

string binary;

while(decimal != 0) {

binary = (decimal % 2 == 0 ? "0" : "1") + binary;

decimal = decimal/2;

}

while(binary.length() < 4){

binary = "0" + binary;

}

return binary;

}

// Function to convert a number in binary to decimal

int convertBinaryToDecimal(string binary)

{

int decimal = 0;

int counter = 0;

int size = binary.length();

for(int i = size-1; i >= 0; i--)

{

if(binary[i] == '1'){

decimal += pow(2, counter);

}

counter++;

}

return decimal;

}

// Function to do a circular left shift by 1

string shift\_left\_once(string key\_chunk){

string shifted="";

for(int i = 1; i < 28; i++){

shifted += key\_chunk[i];

}

shifted += key\_chunk[0];

return shifted;

}

// Function to do a circular left shift by 2

string shift\_left\_twice(string key\_chunk){

string shifted="";

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

for(int j = 1; j < 28; j++){

shifted += key\_chunk[j];

}

shifted += key\_chunk[0];

key\_chunk= shifted;

shifted ="";

}

return key\_chunk;

}

// Function to compute xor between two strings

string Xor(string a, string b){

string result = "";

int size = b.size();

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

if(a[i] != b[i]){

result += "1";

}

else{

result += "0";

}

}

return result;

}

// Function to generate the 16 keys.

void generate\_keys(string key){

// The PC1 table

int pc1[56] = {

57,49,41,33,25,17,9,

1,58,50,42,34,26,18,

10,2,59,51,43,35,27,

19,11,3,60,52,44,36,

63,55,47,39,31,23,15,

7,62,54,46,38,30,22,

14,6,61,53,45,37,29,

21,13,5,28,20,12,4

};

// The PC2 table

int pc2[48] = {

14,17,11,24,1,5,

3,28,15,6,21,10,

23,19,12,4,26,8,

16,7,27,20,13,2,

41,52,31,37,47,55,

30,40,51,45,33,48,

44,49,39,56,34,53,

46,42,50,36,29,32

};

// 1. Compressing the key using the PC1 table

string perm\_key ="";

for(int i = 0; i < 56; i++){

perm\_key+= key[pc1[i]-1];

}

// 2. Dividing the key into two equal halves

string left= perm\_key.substr(0, 28);

string right= perm\_key.substr(28, 28);

for(int i=0; i<16; i++){

// 3.1. For rounds 1, 2, 9, 16 the key\_chunks

// are shifted by one.

if(i == 0 || i == 1 || i==8 || i==15 ){

left= shift\_left\_once(left);

right= shift\_left\_once(right);

}

// 3.2. For other rounds, the key\_chunks

// are shifted by two

else{

left= shift\_left\_twice(left);

right= shift\_left\_twice(right);

}

// Combining the two chunks

string combined\_key = left + right;

string round\_key = "";

// Finally, using the PC2 table to transpose the key bits

for(int i = 0; i < 48; i++){

round\_key += combined\_key[pc2[i]-1];

}

round\_keys[i] = round\_key;

}

}

// Implementing the algorithm

string DES(){

// The initial permutation table

int initial\_permutation[64] = {

58,50,42,34,26,18,10,2,

60,52,44,36,28,20,12,4,

62,54,46,38,30,22,14,6,

64,56,48,40,32,24,16,8,

57,49,41,33,25,17,9,1,

59,51,43,35,27,19,11,3,

61,53,45,37,29,21,13,5,

63,55,47,39,31,23,15,7

};

// The expansion table

int expansion\_table[48] = {

32,1,2,3,4,5,4,5,

6,7,8,9,8,9,10,11,

12,13,12,13,14,15,16,17,

16,17,18,19,20,21,20,21,

22,23,24,25,24,25,26,27,

28,29,28,29,30,31,32,1

};

// The substitution boxes. The should contain values

// from 0 to 15 in any order.

int substition\_boxes[8][4][16]=

{{

14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7,

0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8,

4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0,

15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13

},

{

15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10,

3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5,

0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15,

13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9

},

{

10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8,

13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,1,

13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7,

1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12

},

{

7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15,

13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9,

10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4,

3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14

},

{

2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9,

14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6,

4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14,

11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3

},

{

12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11,

10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8,

9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6,

4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13

},

{

4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1,

13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6,

1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2,

6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12

},

{

13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7,

1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2,

7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8,

2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11

}};

// The permutation table

int permutation\_tab[32] = {

16,7,20,21,29,12,28,17,

1,15,23,26,5,18,31,10,

2,8,24,14,32,27,3,9,

19,13,30,6,22,11,4,25

};

// The inverse permutation table

int inverse\_permutation[64]= {

40,8,48,16,56,24,64,32,

39,7,47,15,55,23,63,31,

38,6,46,14,54,22,62,30,

37,5,45,13,53,21,61,29,

36,4,44,12,52,20,60,28,

35,3,43,11,51,19,59,27,

34,2,42,10,50,18,58,26,

33,1,41,9,49,17,57,25

};

//1. Applying the initial permutation

string perm = "";

for(int i = 0; i < 64; i++){

perm += pt[initial\_permutation[i]-1];

}

// 2. Dividing the result into two equal halves

string left = perm.substr(0, 32);

string right = perm.substr(32, 32);

// The plain text is encrypted 16 times

for(int i=0; i<16; i++) {

string right\_expanded = "";

// 3.1. The right half of the plain text is expanded

for(int i = 0; i < 48; i++) {

right\_expanded += right[expansion\_table[i]-1];

}; // 3.3. The result is xored with a key

string xored = Xor(round\_keys[i], right\_expanded);

string res = "";

// 3.4. The result is divided into 8 equal parts and passed

// through 8 substitution boxes. After passing through a

// substituion box, each box is reduces from 6 to 4 bits.

for(int i=0;i<8; i++){

// Finding row and column indices to lookup the

// substituition box

string row1= xored.substr(i\*6,1) + xored.substr(i\*6 + 5,1);

int row = convertBinaryToDecimal(row1);

string col1 = xored.substr(i\*6 + 1,1) + xored.substr(i\*6 + 2,1) + xored.substr(i\*6 + 3,1) + xored.substr(i\*6 + 4,1);;

int col = convertBinaryToDecimal(col1);

int val = substition\_boxes[i][row][col];

res += convertDecimalToBinary(val);

}

// 3.5. Another permutation is applied

string perm2 ="";

for(int i = 0; i < 32; i++){

perm2 += res[permutation\_tab[i]-1];

}

// 3.6. The result is xored with the left half

xored = Xor(perm2, left);

// 3.7. The left and the right parts of the plain text are swapped

left = xored;

if(i < 15){

string temp = right;

right = xored;

left = temp;

}

}

// 4. The halves of the plain text are applied

string combined\_text = left + right;

string ciphertext ="";

// The inverse of the initial permuttaion is applied

for(int i = 0; i < 64; i++){

ciphertext+= combined\_text[inverse\_permutation[i]-1];

}

//And we finally get the cipher text

return ciphertext;

}

int main(){

// A 64 bit key

string key= "1010101010111011000010010001100000100111001101101100110011011101";

// A block of plain text of 64 bits

pt= "1010101111001101111001101010101111001101000100110010010100110110";

string apt = pt;

// Calling the function to generate 16 keys

generate\_keys(key);

cout<<"Plain text: "<<pt<<endl;

// Applying the algo

string ct= DES();

cout<<"Ciphertext: "<<ct<<endl;

// Reversing the round\_keys array for decryption

int i = 15;

int j = 0;

while(i > j)

{

string temp = round\_keys[i];

round\_keys[i] = round\_keys[j];

round\_keys[j] = temp;

i--;

j++;

}

pt = ct;

string decrypted = DES();

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

// Comapring the initial plain text with the decrypted text

if (decrypted == apt){

cout<<"Plain text encrypted and decrypted successfully."<<endl;

}

}

**Output:**

**Plain text: 1010101111001101111001101010101111001101000100110010010100110110**

**Ciphertext: 1001111000100110100111110101101011111010010011011011101101110000**

**Decrypted text:1010101111001101111001101010101111001101000100110010010100110110**

**Plain text encrypted and decrypted successfully.**

***#2 Java Code***

import java.util.\*;

class Main {

private static class DES {

// CONSTANTS

// Initial Permutation Table

int[] IP = { 58, 50, 42, 34, 26, 18,

10, 2, 60, 52, 44, 36, 28, 20,

12, 4, 62, 54, 46, 38,

30, 22, 14, 6, 64, 56,

48, 40, 32, 24, 16, 8,

57, 49, 41, 33, 25, 17,

9, 1, 59, 51, 43, 35, 27,

19, 11, 3, 61, 53, 45,

37, 29, 21, 13, 5, 63, 55,

47, 39, 31, 23, 15, 7 };

// Inverse Initial Permutation Table

int[] IP1 = { 40, 8, 48, 16, 56, 24, 64,

32, 39, 7, 47, 15, 55,

23, 63, 31, 38, 6, 46,

14, 54, 22, 62, 30, 37,

5, 45, 13, 53, 21, 61,

29, 36, 4, 44, 12, 52,

20, 60, 28, 35, 3, 43,

11, 51, 19, 59, 27, 34,

2, 42, 10, 50, 18, 58,

26, 33, 1, 41, 9, 49,

17, 57, 25 };

// first key-hePermutation Table

int[] PC1 = { 57, 49, 41, 33, 25,

17, 9, 1, 58, 50, 42, 34, 26,

18, 10, 2, 59, 51, 43, 35, 27,

19, 11, 3, 60, 52, 44, 36, 63,

55, 47, 39, 31, 23, 15, 7, 62,

54, 46, 38, 30, 22, 14, 6, 61,

53, 45, 37, 29, 21, 13, 5, 28,

20, 12, 4 };

// second key-Permutation Table

int[] PC2 = { 14, 17, 11, 24, 1, 5, 3,

28, 15, 6, 21, 10, 23, 19, 12,

4, 26, 8, 16, 7, 27, 20, 13, 2,

41, 52, 31, 37, 47, 55, 30, 40,

51, 45, 33, 48, 44, 49, 39, 56,

34, 53, 46, 42, 50, 36, 29, 32 };

// Expansion D-box Table

int[] EP = { 32, 1, 2, 3, 4, 5, 4,

5, 6, 7, 8, 9, 8, 9, 10,

11, 12, 13, 12, 13, 14, 15,

16, 17, 16, 17, 18, 19, 20,

21, 20, 21, 22, 23, 24, 25,

24, 25, 26, 27, 28, 29, 28,

29, 30, 31, 32, 1 };

// Straight Permutation Table

int[] P = { 16, 7, 20, 21, 29, 12, 28,

17, 1, 15, 23, 26, 5, 18,

31, 10, 2, 8, 24, 14, 32,

27, 3, 9, 19, 13, 30, 6,

22, 11, 4, 25 };

// S-box Table

int[][][] sbox = {

{ { 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7 },

{ 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8 },

{ 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0 },

{ 15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13 } },

{ { 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10 },

{ 3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5 },

{ 0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15 },

{ 13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9 } },

{ { 10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8 },

{ 13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1 },

{ 13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7 },

{ 1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12 } },

{ { 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15 },

{ 13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9 },

{ 10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4 },

{ 3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14 } },

{ { 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9 },

{ 14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6 },

{ 4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14 },

{ 11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3 } },

{ { 12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11 },

{ 10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8 },

{ 9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6 },

{ 4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13 } },

{ { 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1 },

{ 13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6 },

{ 1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2 },

{ 6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12 } },

{ { 13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7 },

{ 1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2 },

{ 7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8 },

{ 2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11 } }

};

int[] shiftBits = { 1, 1, 2, 2, 2, 2, 2, 2,

1, 2, 2, 2, 2, 2, 2, 1 };

// hexadecimal to binary conversion

String hextoBin(String input) {

int n = input.length() \* 4;

input = Long.toBinaryString(

Long.parseUnsignedLong(input, 16));

while (input.length() < n)

input = "0" + input;

return input;

}

// binary to hexadecimal conversion

String binToHex(String input) {

int n = (int) input.length() / 4;

input = Long.toHexString(

Long.parseUnsignedLong(input, 2));

while (input.length() < n)

input = "0" + input;

return input;

}

// per-mutate input hexadecimal

// according to specified sequence

String permutation(int[] sequence, String input) {

String output = "";

input = hextoBin(input);

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

output += input.charAt(sequence[i] - 1);

output = binToHex(output);

return output;

}

// xor 2 hexadecimal strings

String xor(String a, String b) {

// hexadecimal to decimal(base 10)

long t\_a = Long.parseUnsignedLong(a, 16);

// hexadecimal to decimal(base 10)

long t\_b = Long.parseUnsignedLong(b, 16);

// xor

t\_a = t\_a ^ t\_b;

// decimal to hexadecimal

a = Long.toHexString(t\_a);

// prepend 0's to maintain length

while (a.length() < b.length())

a = "0" + a;

return a;

}

// left Circular Shifting bits

String leftCircularShift(String input, int numBits) {

int n = input.length() \* 4;

int perm[] = new int[n];

for (int i = 0; i < n - 1; i++)

perm[i] = (i + 2);

perm[n - 1] = 1;

while (numBits-- > 0)

input = permutation(perm, input);

return input;

}

// preparing 16 keys for 16 rounds

String[] getKeys(String key) {

String keys[] = new String[16];

// first key permutation

key = permutation(PC1, key);

for (int i = 0; i < 16; i++) {

key = leftCircularShift(

key.substring(0, 7), shiftBits[i])

+ leftCircularShift(key.substring(7, 14),

shiftBits[i]);

// second key permutation

keys[i] = permutation(PC2, key);

}

return keys;

}

// s-box lookup

String sBox(String input) {

String output = "";

input = hextoBin(input);

for (int i = 0; i < 48; i += 6) {

String temp = input.substring(i, i + 6);

int num = i / 6;

int row = Integer.parseInt(

temp.charAt(0) + "" + temp.charAt(5), 2);

int col = Integer.parseInt(

temp.substring(1, 5), 2);

output += Integer.toHexString(

sbox[num][row][col]);

}

return output;

}

String round(String input, String key, int num) {

// fk

String left = input.substring(0, 8);

String temp = input.substring(8, 16);

String right = temp;

// Expansion permutation

temp = permutation(EP, temp);

// xor temp and round key

temp = xor(temp, key);

// lookup in s-box table

temp = sBox(temp);

// Straight D-box

temp = permutation(P, temp);

// xor

left = xor(left, temp);

System.out.println("Round "

+ (num + 1) + " "

+ right.toUpperCase()

+ " " + left.toUpperCase() + " "

+ key.toUpperCase());

// swapper

return right + left;

}

String encrypt(String plainText, String key) {

int i;

// get round keys

String keys[] = getKeys(key);

// initial permutation

plainText = permutation(IP, plainText);

System.out.println(

"After initial permutation: "

+ plainText.toUpperCase());

System.out.println(

"After splitting: L0="

+ plainText.substring(0, 8).toUpperCase()

+ " R0="

+ plainText.substring(8, 16).toUpperCase() + "\n");

// 16 rounds

for (i = 0; i < 16; i++) {

plainText = round(plainText, keys[i], i);

}

// 32-bit swap

plainText = plainText.substring(8, 16)

+ plainText.substring(0, 8);

// final permutation

plainText = permutation(IP1, plainText);

return plainText;

}

String decrypt(String plainText, String key) {

int i;

// get round keys

String keys[] = getKeys(key);

// initial permutation

plainText = permutation(IP, plainText);

System.out.println(

"After initial permutation: "

+ plainText.toUpperCase());

System.out.println(

"After splitting: L0="

+ plainText.substring(0, 8).toUpperCase()

+ " R0=" + plainText.substring(8, 16).toUpperCase()

+ "\n");

// 16-rounds

for (i = 15; i > -1; i--) {

plainText = round(plainText, keys[i], 15 - i);

}

// 32-bit swap

plainText = plainText.substring(8, 16)

+ plainText.substring(0, 8);

plainText = permutation(IP1, plainText);

return plainText;

}

}

public static void main(String args[]) {

String text = "123456ABCD132536";

String key = "AABB09182736CCDD";

DES cipher = new DES();

System.out.println("Encryption:\n");

text = cipher.encrypt(text, key);

System.out.println(

"\nCipher Text: " + text.toUpperCase() + "\n");

System.out.println("Decryption\n");

text = cipher.decrypt(text, key);

System.out.println(

"\nPlain Text: "

+ text.toUpperCase());

}

}

**Output:**

***#3 Python***

# Hexadecimal to binary conversion

def hex2bin(s):

mp = {'0': "0000",

'1': "0001",

'2': "0010",

'3': "0011",

'4': "0100",

'5': "0101",

'6': "0110",

'7': "0111",

'8': "1000",

'9': "1001",

'A': "1010",

'B': "1011",

'C': "1100",

'D': "1101",

'E': "1110",

'F': "1111"}

bin = ""

for i in range(len(s)):

bin = bin + mp[s[i]]

return bin

# Binary to hexadecimal conversion

def bin2hex(s):

mp = {"0000": '0',

"0001": '1',

"0010": '2',

"0011": '3',

"0100": '4',

"0101": '5',

"0110": '6',

"0111": '7',

"1000": '8',

"1001": '9',

"1010": 'A',

"1011": 'B',

"1100": 'C',

"1101": 'D',

"1110": 'E',

"1111": 'F'}

hex = ""

for i in range(0, len(s), 4):

ch = ""

ch = ch + s[i]

ch = ch + s[i + 1]

ch = ch + s[i + 2]

ch = ch + s[i + 3]

hex = hex + mp[ch]

return hex

# Binary to decimal conversion

def bin2dec(binary):

binary1 = binary

decimal, i, n = 0, 0, 0

while(binary != 0):

dec = binary % 10

decimal = decimal + dec \* pow(2, i)

binary = binary//10

i += 1

return decimal

# Decimal to binary conversion

def dec2bin(num):

res = bin(num).replace("0b", "")

if(len(res) % 4 != 0):

div = len(res) / 4

div = int(div)

counter = (4 \* (div + 1)) - len(res)

for i in range(0, counter):

res = '0' + res

return res

# Permute function to rearrange the bits

def permute(k, arr, n):

permutation = ""

for i in range(0, n):

permutation = permutation + k[arr[i] - 1]

return permutation

# shifting the bits towards left by nth shifts

def shift\_left(k, nth\_shifts):

s = ""

for i in range(nth\_shifts):

for j in range(1, len(k)):

s = s + k[j]

s = s + k[0]

k = s

s = ""

return k

# calculating xow of two strings of binary number a and b

def xor(a, b):

ans = ""

for i in range(len(a)):

if a[i] == b[i]:

ans = ans + "0"

else:

ans = ans + "1"

return ans

# Table of Position of 64 bits at initial level: Initial Permutation Table

initial\_perm = [58, 50, 42, 34, 26, 18, 10, 2,

60, 52, 44, 36, 28, 20, 12, 4,

62, 54, 46, 38, 30, 22, 14, 6,

64, 56, 48, 40, 32, 24, 16, 8,

57, 49, 41, 33, 25, 17, 9, 1,

59, 51, 43, 35, 27, 19, 11, 3,

61, 53, 45, 37, 29, 21, 13, 5,

63, 55, 47, 39, 31, 23, 15, 7]

# Expansion D-box Table

exp\_d = [32, 1, 2, 3, 4, 5, 4, 5,

6, 7, 8, 9, 8, 9, 10, 11,

12, 13, 12, 13, 14, 15, 16, 17,

16, 17, 18, 19, 20, 21, 20, 21,

22, 23, 24, 25, 24, 25, 26, 27,

28, 29, 28, 29, 30, 31, 32, 1]

# Straight Permutation Table

per = [16, 7, 20, 21,

29, 12, 28, 17,

1, 15, 23, 26,

5, 18, 31, 10,

2, 8, 24, 14,

32, 27, 3, 9,

19, 13, 30, 6,

22, 11, 4, 25]

# S-box Table

sbox = [[[14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7],

[0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8],

[4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0],

[15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13]],

[[15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10],

[3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5],

[0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15],

[13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9]],

[[10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8],

[13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1],

[13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7],

[1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12]],

[[7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15],

[13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9],

[10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4],

[3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14]],

[[2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9],

[14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6],

[4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14],

[11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3]],

[[12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11],

[10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8],

[9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6],

[4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13]],

[[4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1],

[13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6],

[1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2],

[6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12]],

[[13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7],

[1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2],

[7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8],

[2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11]]]

# Final Permutation Table

final\_perm = [40, 8, 48, 16, 56, 24, 64, 32,

39, 7, 47, 15, 55, 23, 63, 31,

38, 6, 46, 14, 54, 22, 62, 30,

37, 5, 45, 13, 53, 21, 61, 29,

36, 4, 44, 12, 52, 20, 60, 28,

35, 3, 43, 11, 51, 19, 59, 27,

34, 2, 42, 10, 50, 18, 58, 26,

33, 1, 41, 9, 49, 17, 57, 25]

def encrypt(pt, rkb, rk):

pt = hex2bin(pt)

# Initial Permutation

pt = permute(pt, initial\_perm, 64)

print("After initial permutation", bin2hex(pt))

# Splitting

left = pt[0:32]

right = pt[32:64]

for i in range(0, 16):

# Expansion D-box: Expanding the 32 bits data into 48 bits

right\_expanded = permute(right, exp\_d, 48)

# XOR RoundKey[i] and right\_expanded

xor\_x = xor(right\_expanded, rkb[i])

# S-boxex: substituting the value from s-box table by calculating row and column

sbox\_str = ""

for j in range(0, 8):

row = bin2dec(int(xor\_x[j \* 6] + xor\_x[j \* 6 + 5]))

col = bin2dec(

int(xor\_x[j \* 6 + 1] + xor\_x[j \* 6 + 2] + xor\_x[j \* 6 + 3] + xor\_x[j \* 6 + 4]))

val = sbox[j][row][col]

sbox\_str = sbox\_str + dec2bin(val)

# Straight D-box: After substituting rearranging the bits

sbox\_str = permute(sbox\_str, per, 32)

# XOR left and sbox\_str

result = xor(left, sbox\_str)

left = result

# Swapper

if(i != 15):

left, right = right, left

print("Round ", i + 1, " ", bin2hex(left),

" ", bin2hex(right), " ", rk[i])

# Combination

combine = left + right

# Final permutation: final rearranging of bits to get cipher text

cipher\_text = permute(combine, final\_perm, 64)

return cipher\_text

pt = "123456ABCD132536"

key = "AABB09182736CCDD"

# Key generation

# --hex to binary

key = hex2bin(key)

# --parity bit drop table

keyp = [57, 49, 41, 33, 25, 17, 9,

1, 58, 50, 42, 34, 26, 18,

10, 2, 59, 51, 43, 35, 27,

19, 11, 3, 60, 52, 44, 36,

63, 55, 47, 39, 31, 23, 15,

7, 62, 54, 46, 38, 30, 22,

14, 6, 61, 53, 45, 37, 29,

21, 13, 5, 28, 20, 12, 4]

# getting 56 bit key from 64 bit using the parity bits

key = permute(key, keyp, 56)

# Number of bit shifts

shift\_table = [1, 1, 2, 2,

2, 2, 2, 2,

1, 2, 2, 2,

2, 2, 2, 1]

# Key- Compression Table : Compression of key from 56 bits to 48 bits

key\_comp = [14, 17, 11, 24, 1, 5,

3, 28, 15, 6, 21, 10,

23, 19, 12, 4, 26, 8,

16, 7, 27, 20, 13, 2,

41, 52, 31, 37, 47, 55,

30, 40, 51, 45, 33, 48,

44, 49, 39, 56, 34, 53,

46, 42, 50, 36, 29, 32]

# Splitting

left = key[0:28] # rkb for RoundKeys in binary

right = key[28:56] # rk for RoundKeys in hexadecimal

rkb = []

rk = []

for i in range(0, 16):

# Shifting the bits by nth shifts by checking from shift table

left = shift\_left(left, shift\_table[i])

right = shift\_left(right, shift\_table[i])

# Combination of left and right string

combine\_str = left + right

# Compression of key from 56 to 48 bits

round\_key = permute(combine\_str, key\_comp, 48)

rkb.append(round\_key)

rk.append(bin2hex(round\_key))

print("Encryption")

cipher\_text = bin2hex(encrypt(pt, rkb, rk))

print("Cipher Text : ", cipher\_text)

print("Decryption")

rkb\_rev = rkb[::-1]

rk\_rev = rk[::-1]

text = bin2hex(encrypt(cipher\_text, rkb\_rev, rk\_rev))

print("Plain Text : ", text)

**Output:**

**Encryption**

**After initial permutation 14A7D67818CA18AD**

**Round 1 18CA18AD 5A78E394 194CD072DE8C**

**Round 2 5A78E394 4A1210F6 4568581ABCCE**

**Round 3 4A1210F6 B8089591 06EDA4ACF5B5**

**Round 4 B8089591 236779C2 DA2D032B6EE3**

**Round 5 236779C2 A15A4B87 69A629FEC913**

**Round 6 A15A4B87 2E8F9C65 C1948E87475E**

**Round 7 2E8F9C65 A9FC20A3 708AD2DDB3C0**

**Round 8 A9FC20A3 308BEE97 34F822F0C66D**

**Round 9 308BEE97 10AF9D37 84BB4473DCCC**

**Round 10 10AF9D37 6CA6CB20 02765708B5BF**

**Round 11 6CA6CB20 FF3C485F 6D5560AF7CA5**

**Round 12 FF3C485F 22A5963B C2C1E96A4BF3**

**Round 13 22A5963B 387CCDAA 99C31397C91F**

**Round 14 387CCDAA BD2DD2AB 251B8BC717D0**

**Round 15 BD2DD2AB CF26B472 3330C5D9A36D**

**Round 16 19BA9212 CF26B472 181C5D75C66D**

**Cipher Text : C0B7A8D05F3A829C**

**Decryption**

**After initial permutation 19BA9212CF26B472**

**Round 1 CF26B472 BD2DD2AB 181C5D75C66D**

**Round 2 BD2DD2AB 387CCDAA 3330C5D9A36D**

**Round 3 387CCDAA 22A5963B 251B8BC717D0**

**Round 4 22A5963B FF3C485F 99C31397C91F**

**Round 5 FF3C485F 6CA6CB20 C2C1E96A4BF3**

**Round 6 6CA6CB20 10AF9D37 6D5560AF7CA5**

**Round 7 10AF9D37 308BEE97 02765708B5BF**

**Round 8 308BEE97 A9FC20A3 84BB4473DCCC**

**Round 9 A9FC20A3 2E8F9C65 34F822F0C66D**

**Round 10 2E8F9C65 A15A4B87 708AD2DDB3C0**

**Round 11 A15A4B87 236779C2 C1948E87475E**

**Round 12 236779C2 B8089591 69A629FEC913**

**Round 13 B8089591 4A1210F6 DA2D032B6EE3**

**Round 14 4A1210F6 5A78E394 06EDA4ACF5B5**

**Round 15 5A78E394 18CA18AD 4568581ABCCE**

**Round 16 14A7D678 18CA18AD 194CD072DE8C**

**Plain Text : 123456ABCD132536**

**SELF TESTING QUESTIONS :**

Q1 What is the main vulnerability of DES?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Q2 Which mode of operation is used in DES?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Q3 What is strength of DES?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Q4 What is the avalanche effect in DES?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Q5 What is expansion table in DES?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Conclusion** : DES uses the same key to encrypt and decrypt a message, so both the sender and the receiver must know and use the same private key. DES was once the go-to, symmetric key algorithm for the encryption of electronic data, but it has been superseded by the more secure Advanced Encryption Standard (AES) algorithm.

**Experiment No. 6**

**Aim:** To use Hashing Techniques: HMAC and SHA for a practical application like User Message Encryption.

# Theory:

A hashing algorithm is a mathematical function that garbles data and makes it unreadable.

Hashing algorithms are one-way programs, so the text can’t be unscrambled and decoded by anyone else. And that’s the point. Hashing protects data at rest, so even if someone gains access to your server, the items stored there remain unreadable.

Hashing can also help you prove that data isn’t adjusted or altered after the author is finished with it. And some people use hashing to help them make sense of reams of data.

Hash Function is a function that has a huge role in making a System Secure as it converts normal data given to it as an irregular value of fixed length. We can imagine it to be a Shaker in our homes.

When we put data into this function it outputs an irregular value. The Irregular value it outputs is known as the **“Hash Value”**. Hash Values are simply numbers but are often written in Hexadecimal. Computers manage values as Binary. The hash value is also data and is often managed in Binary.

**HMAC:**

**HMAC algorithm** stands for Hashed or Hash-based [Message Authentication Code](https://www.geeksforgeeks.org/computer-network-message-authentication-code-works/). It is a result of work done on developing a MAC derived from cryptographic hash functions.

HMAC is a great resistant to cryptanalysis attacks as it uses the Hashing concept twice. HMAC consists of twin benefits of Hashing and MAC and thus is more secure than any other authentication code. RFC 2104 has issued HMAC, and HMAC has been made compulsory to implement in IP security. The FIPS 198 NIST standard has also issued HMAC.



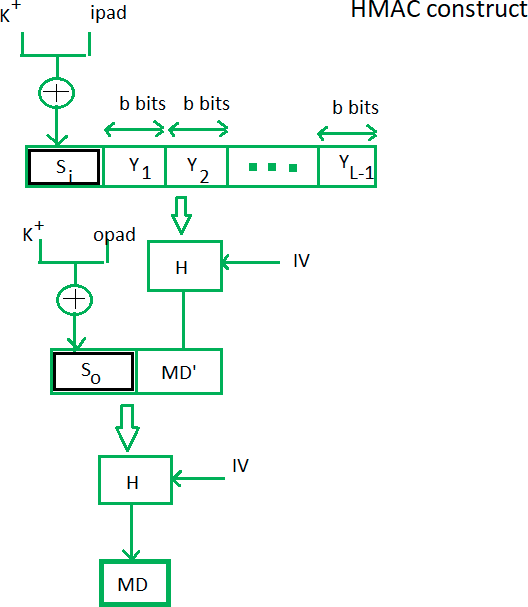
# Objectives –

* As the Hash Function, HMAC is also aimed to be one way, i.e, easy to generate output from input but complex the other way round.
* It aims at being less affected by collisions than the hash functions.
* HMAC reuses the algorithms like MD5 and SHA-1 and checks to replace the embedded hash functions with more secure hash functions, in case found.
* HMAC tries to handle the Keys in a more simple manner.

# HMAC algorithm –

The working of HMAC starts with taking a message M containing blocks of length *b* bits. An input signature is padded to the left of the message and the whole is given as input to a hash function which gives us a temporary message-digest MD’. MD’ again is appended to an output signature and the whole is applied a hash function again, the result is our final message digest MD.

* Here is a simple structure of HMAC:



Here, H stands for Hashing function, M is the original message

Si and So are input and output signatures respectively,

Yi is the ith block in original message M, where I ranges from [1, L) L = the count of blocks in M

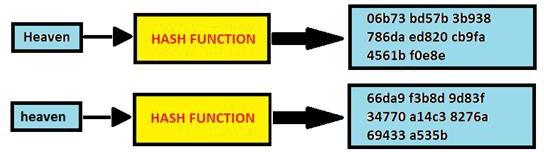
K is the secret key used for hashing IV is an initial vector (some constant)

The generation of input signature and output signature *Si* and *So* respectively.

# SHA

SHA stands for secure hashing algorithm. SHA is a modified version of MD5 and used for hashing data and [certificates.](https://learn.encryptionconsulting.com/digital-certificates/) A hashing algorithm shortens the input data into a smaller form that cannot be understood by using bitwise operations, modular additions, and compression functions. You may be wondering, can hashing be cracked or decrypted? Hashing is similar to [encryption,](https://learn.encryptionconsulting.com/what-is-encryption/) the only difference between hashing and encryption is that hashing is one-way, meaning once the data is hashed, the resulting hash digest cannot be cracked, unless a brute force attack is used.

SHA works in such a way even if a single character of the message changed, then it will generate a different hash. For example, hashing of two similar, but different messages i.e., Heaven and heaven is different. However, there is only a difference of a capital and small letter.



# Different SHA Forms

When learning about SHA forms, several different types of SHA are referenced. Examples of SHA names used are SHA-1, SHA-2, SHA-256, SHA-512, SHA-224, and SHA-384, but in

actuality, there are only two types: SHA-1 and SHA-2. The other larger numbers, like SHA- 256, are just versions of SHA-2 that note the bit lengths of the SHA-2. SHA-1 was the original secure hashing algorithm, returning a 160-bit hash digest after hashing.

**Program and code:**

**SHA-256**

# Java:

import java.math.BigInteger;

import java.nio.charset.StandardCharsets;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

// Java program to calculate SHA-256 hash value

class GFG2 {

public static byte[] getSHA(String input) throws NoSuchAlgorithmException {

// Static getInstance method is called with hashing SHA

MessageDigest md = MessageDigest.getInstance("SHA-256");

// digest() method called to calculate the message digest of an input

// and return an array of bytes

return md.digest(input.getBytes(StandardCharsets.UTF\_8));

}

public static String toHexString(byte[] hash) {

// Convert the byte array into a signum representation

BigInteger number = new BigInteger(1, hash);

// Convert the message digest into a hex value

String hexString = number.toString(16);

// Pad with leading zeros if needed

while (hexString.length() < 64) {

hexString = "0" + hexString;

}

return hexString;

}

// Driver code

public static void main(String[] args) {

try {

System.out.println("HashCode Generated by SHA-256 for:\n");

String s1 = "Batchfour";

System.out.println(s1 + " : " + toHexString(getSHA(s1)));

String s2 = "\n hello world";

System.out.println(s2 + " : " + toHexString(getSHA(s2)));

String s3 = "K1t4fo0V";

System.out.println(s3 + " : " + toHexString(getSHA(s3)));

} catch (NoSuchAlgorithmException e) {

System.out.println("Exception thrown for incorrect algorithm: " + e);

}

}

}

# Output:

**HashCode Generated by SHA-256 for:**

**Batchfour : 8f53ca117a48ba3a0e78fd9889fb103eca5bc9554b6d197e0e86523889735105**

**hello world : 7b80c4913687d5bda92d876fb8252ddd74592c7815e4f1c7d976f40538608835**

**K1t4fo0V : 0a979e43f4874eb24b740c0157994e34636eed0425688161cc58e8b26b1dcf4e**

* + **Python:**

import hashlib

str = "TEITBATCHFOUR"

result = hashlib.sha256(str.encode())

print("The hexadecimal equivalent of SHA256 is:")

print(result.hexdigest())

str = "TEITBATCHFOUR"

result = hashlib.sha384(str.encode())

print("\nThe hexadecimal equivalent of SHA384 is:")

print(result.hexdigest())

str = "TEITBATCHFOUR"

result = hashlib.sha224(str.encode())

print("\nThe hexadecimal equivalent of SHA224 is:")

print(result.hexdigest())

str = "TEITBATCHFOUR"

result = hashlib.sha512(str.encode())

print("\nThe hexadecimal equivalent of SHA512 is:")

print(result.hexdigest())

str = "TEITBATCHFOUR"

result = hashlib.sha1(str.encode())

print("\nThe hexadecimal equivalent of SHA1 is:")

print(result.hexdigest())

# Output:

**The hexadecimal equivalent of SHA256 is:**

**d0d8ee0438a4b9e1383872749bdfebb4d8443ad857c996145d5d8a088cb4a194**

**The hexadecimal equivalent of SHA384 is:**

**a8bbfaef281c11943fad0cab7f33d0baae7835fd4dc308e390a664d7dc10d48a60b1fb6802db9e993fc3992216fffb26**

**The hexadecimal equivalent of SHA224 is:**

**b7f847b413bda5074e1a6734db55840b9f9aa0fff02e7b6618890530**

**The hexadecimal equivalent of SHA512 is:**

**9eac3f63ad01a4249e128017d36182d1e23ac644af8410debd7a925f83956b6424fca96e041e9d6c7dd7bed084803bdd75e0c24df9237084670a8c084907ec1c**

**The hexadecimal equivalent of SHA1 is:**

**ba8835a7f2d04a88f684e0b3fc7e7cc2eca826c8**

* + **C++:**

#include <cstring> #include <fstream> #include "sha256.h"

const unsigned int SHA256::sha256\_k[64] = //UL = uint32

{0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5, 0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,

0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,

0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da, 0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,

0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,

0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,

0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5,

0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3, 0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,

0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2};

void SHA256::transform(const unsigned char \*message, unsigned int block\_nb)

{

uint32 w[64]; uint32 wv[8]; uint32 t1, t2;

const unsigned char \*sub\_block; int i;

int j;

for (i = 0; i < (int) block\_nb; i++) { sub\_block = message + (i << 6); for (j = 0; j < 16; j++) {

SHA2\_PACK32(&sub\_block[j << 2], &w[j]);

}

for (j = 16; j < 64; j++) {

w[j] = SHA256\_F4(w[j - 2]) + w[j - 7] + SHA256\_F3(w[j - 15]) + w[j - 16];

}

for (j = 0; j < 8; j++) { wv[j] = m\_h[j];

}

for (j = 0; j < 64; j++) {

t1 = wv[7] + SHA256\_F2(wv[4]) + SHA2\_CH(wv[4], wv[5], wv[6])

+ sha256\_k[j] + w[j];

t2 = SHA256\_F1(wv[0]) + SHA2\_MAJ(wv[0], wv[1], wv[2]);

wv[7] = wv[6];

wv[6] = wv[5];

wv[5] = wv[4];

wv[4] = wv[3] + t1;

wv[3] = wv[2];

wv[2] = wv[1];

wv[1] = wv[0]; wv[0] = t1 + t2;

}

for (j = 0; j < 8; j++) { m\_h[j] += wv[j];

}

}

}

void SHA256::init()

{

m\_h[0] = 0x6a09e667; m\_h[1] = 0xbb67ae85; m\_h[2] = 0x3c6ef372; m\_h[3] = 0xa54ff53a; m\_h[4] = 0x510e527f; m\_h[5] = 0x9b05688c; m\_h[6] = 0x1f83d9ab; m\_h[7] = 0x5be0cd19; m\_len = 0;

m\_tot\_len = 0;

}

void SHA256::update(const unsigned char \*message, unsigned int len)

{

unsigned int block\_nb;

unsigned int new\_len, rem\_len, tmp\_len; const unsigned char \*shifted\_message;

tmp\_len = SHA224\_256\_BLOCK\_SIZE - m\_len; rem\_len = len < tmp\_len ? len : tmp\_len; memcpy(&m\_block[m\_len], message, rem\_len); if (m\_len + len < SHA224\_256\_BLOCK\_SIZE) {

m\_len += len; return;

}

new\_len = len - rem\_len;

block\_nb = new\_len / SHA224\_256\_BLOCK\_SIZE; shifted\_message = message + rem\_len; transform(m\_block, 1);

transform(shifted\_message, block\_nb);

rem\_len = new\_len % SHA224\_256\_BLOCK\_SIZE; memcpy(m\_block, &shifted\_message[block\_nb << 6], rem\_len); m\_len = rem\_len;

m\_tot\_len += (block\_nb + 1) << 6;

}

void SHA256::final(unsigned char \*digest)

{

unsigned int block\_nb; unsigned int pm\_len; unsigned int len\_b;

int i;

block\_nb = (1 + ((SHA224\_256\_BLOCK\_SIZE - 9)

< (m\_len % SHA224\_256\_BLOCK\_SIZE)));

len\_b = (m\_tot\_len + m\_len) << 3; pm\_len = block\_nb << 6;

memset(m\_block + m\_len, 0, pm\_len - m\_len); m\_block[m\_len] = 0x80; SHA2\_UNPACK32(len\_b, m\_block + pm\_len - 4); transform(m\_block, block\_nb);

for (i = 0 ; i < 8; i++) { SHA2\_UNPACK32(m\_h[i], &digest[i << 2]);

}

}

std::string sha256(std::string input)

{

unsigned char digest[SHA256::DIGEST\_SIZE]; memset(digest,0,SHA256::DIGEST\_SIZE);

SHA256 ctx = SHA256();

ctx.init();

ctx.update( (unsigned char\*)input.c\_str(), input.length()); ctx.final(digest);

char buf[2\*SHA256::DIGEST\_SIZE+1]; buf[2\*SHA256::DIGEST\_SIZE] = 0;

for (int i = 0; i < SHA256::DIGEST\_SIZE; i++)

sprintf(buf+i\*2, "%02x", digest[i]); return std::string(buf);

}

# Conclusion:

We have successfully implemented Hashing Techniques: HMAC and SHA for practical applications like User Message Encryption.

**Experiment No. 7**

**AIM**: To study and implement RSA asymmetric cryptography key algorithms.

**THEORY**: RSA algorithm is asymmetric cryptography algorithm. Asymmetric actually means that

it works on two different keys i.e. Public Key and Private Key. As the name describes that the

Public Key is given to everyone and Private key is kept private.

An example of asymmetric cryptography :

1. A client (for example browser) sends its public key to the server and requests for some data.

2.The server encrypts the data using client’s public key and sends the encrypted data.Client

receives this data and decrypts it.

3.Since this is asymmetric, nobody else except browser can decrypt the data even if a third

party has public key of browser.

The idea! The idea of RSA is based on the fact that it is difficult to factorize a large integer. The

public key consists of two numbers where one number is multiplication of two large prime

numbers. And private key is also derived from the same two prime numbers. So if somebody

can factorize the large number, the private key is compromised. Therefore encryption strength

totally lies on the key size and if we double or triple the key size, the strength of encryption

increases exponentially. RSA keys can be typically 1024 or 2048 bits long, but experts believe

that 1024 bit keys could be broken in the near future. But till now it seems to be an infeasible

task.

The RSA Algorithm

1.Choose two different large random prime numbers p and q

2.Calculate n = p q n is the modulus for the public key and the private keys

3.Calculate ϕ ( n ) = ( p − 1 ) ( q − 1 )

4.Choose an integer k such that 1 < k < ϕ ( n ) and k is co-prime to ϕ ( n ) : k and ϕ ( n )

share no factors other than 1; gcd (k, ϕ ( n ))= 1.

5.k is released as the public key exponent

6.Compute d to satisfy the d k ≡ 1 ( mod ϕ ( n ) ) i.e.: d k = 1 + x ϕ ( n ) for some integer x

7.d is kept as the private key exponent.

**Program and code:**

**//1.C++**

#include <iostream>

#include <inttypes.h>

int64\_t modPow(int64\_t base, int64\_t exponent, int64\_t modulus);

int64\_t modInverse(int64\_t e, int64\_t phi);

void extendedGCD(int64\_t a, int64\_t b, int64\_t\* d, int64\_t\* x, int64\_t\* y);

int main() {

int64\_t p = 61;

int64\_t q = 53;

int64\_t n = p \* q;

int64\_t e = 17;

int64\_t phi = (p - 1) \* (q - 1);

int64\_t k = modInverse(e, phi);

int64\_t d = k;

if (k == -1) {

std::cout << "Modular multiplicative inverse of 'e' does not exist!" << std::endl;

return 1;

}

int64\_t msg = 20;

std::cout << "Message data = " << msg << std::endl;

int64\_t c = modPow(msg, e, n);

std::cout << "Encrypted data = " << c << std::endl;

int64\_t m = modPow(c, d, n);

std::cout << "Original Message Sent = " << m << std::endl;

return 0;

}

int64\_t modPow(int64\_t base, int64\_t exponent, int64\_t modulus) {

int64\_t result = 1;

base = base % modulus;

while (exponent > 0) {

if (exponent % 2 == 1) {

result = (result \* base) % modulus;

}

base = (base \* base) % modulus;

exponent /= 2;

}

return result;

}

int64\_t modInverse(int64\_t e, int64\_t phi) {

int64\_t d, x, y;

extendedGCD(e, phi, &d, &x, &y);

if (d != 1) {

// Modular inverse does not exist

return -1;

}

// Ensure 'd' is positive

while (x < 0) {

x += phi;

}

return x;

}

void extendedGCD(int64\_t a, int64\_t b, int64\_t\* d, int64\_t\* x, int64\_t\* y) {

if (b == 0) {

\*d = a;

\*x = 1;

\*y = 0;

} else {

int64\_t x1, y1;

extendedGCD(b, a % b, d, &x1, &y1);

\*x = y1;

\*y = x1 - (a / b) \* y1;

}

}

**Output**:

**Message data = 20**

**Encrypted data = 3023**

**Original Message Sent = 20**

**//2 java**

// Java Program to Implement the RSA Algorithm

import java.math.BigDecimal;

import java.math.BigInteger;

class RSA {

public static void main(String args[]) {

int p, q, n, z, d = 0, e, i;

// The number to be encrypted and decrypted

int msg = 12;

double c;

BigInteger msgback;

// 1st prime number p

p = 3;

// 2nd prime number q

q = 11;

n = p \* q;

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

System.out.println("the value of z = " + z);

for (e = 2; e < z; e++) {

// e is for the public key exponent

if (gcd(e, z) == 1) {

break;

}

}

System.out.println("the value of e = " + e);

for (i = 0; i <= 9; i++) {

int x = 1 + (i \* z);

// d is for the private key exponent

if (x % e == 0) {

d = x / e;

break;

}

}

System.out.println("the value of d = " + d);

c = (Math.pow(msg, e)) % n;

System.out.println("Encrypted message is : " + c);

// Converting int value of n to BigInteger

BigInteger N = BigInteger.valueOf(n);

// Converting double value of c to BigInteger

BigInteger C = BigDecimal.valueOf(c).toBigInteger();

msgback = (C.pow(d)).mod(N);

System.out.println("Decrypted message is : " + msgback);

}

static int gcd(int e, int z) {

if (e == 0)

return z;

else

return gcd(z % e, e);

}

}

**Output:**

**Message data = 12.0**

**Encrypted data = 3.0**

**Original Message Sent = 12.0**

**//3 RSA.py**

# Python for RSA asymmetric cryptographic algorithm.

# For demonstration, values are

# relatively small compared to practical application

import math

def gcd(a, h):

temp = 0

while(1):

temp = a % h

if (temp == 0):

return h

a = h

h = temp

p = 3

q = 7

n = p\*q

e = 2

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

while (e < phi):

# e must be co-prime to phi and

# smaller than phi.

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

break

else:

e = e+1

# Private key (d stands for decrypt)

# choosing d such that it satisfies

# d\*e = 1 + k \* totient

k = 2

d = (1 + (k\*phi))/e

# Message to be encrypted

msg = 12.0

print("Message data = ", msg)

# Encryption c = (msg ^ e) % n

c = pow(msg, e)

c = math.fmod(c, n)

print("Encrypted data = ", c)

# Decryption m = (c ^ d) % n

m = pow(c, d)

m = math.fmod(m, n)

print("Original Message Sent = ", m)

**Output:**

**Message data = 12.0**

**Encrypted data = 3.0**

**Original Message Sent = 12.0**

**Conclusion:** So therefore we learnt RSA asymmetric cryptography algorithm successfully in

the big three languages c++, java and python.

1. Hashed message is signed by a sender using which type of key. ?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Using the RSA public key cryptosystem, if p = 13, q = 31 and d = 7, then the value of ‘e’ is ?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Where RSA algorithm is used in real life?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. What type of algorithm is RSA?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Why RSA is secure?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Experiment No. 8**

**AIM**: to understand the functions and applications of a keylogger.

**THEORY:**

Keylogger:

A keylogger called a keystroke logger or keyboard capture is a type of surveillance technology used to monitor and record each keystroke on a specific computer. Keylogger software is also available for use on smartphones, such as the Apple iPhone and Android devices. Cybercriminals often use Keyloggers as a spyware tool to steal [personally identifiable information](https://www.computerweekly.com/opinion/Security-Think-Tank-Safeguarding-PII-in-the-current-threat-landscape) (PII), login credentials, and sensitive enterprise data.

The basic functionality of a keylogger is that it records what you type and, in one way or another, reports that information back to whoever installed it on your computer. (We'll go into the details in a moment.) Since much of your interactions with your computer—and with the people you communicate with via your computer—are mediated through your keyboard, the range of potential information the snooper can acquire by this method is truly vast, from passwords and banking information to private correspondence.

Some keyloggers go beyond just logging keystrokes and recording text and snoop in several other ways as well. It's possible for advanced keyloggers to:

* Log clipboard text, recording information that you cut and paste from other documents
* Track activities like opening folders, documents, and applications
* Take and record randomly timed screenshots
* Request the text value of certain on-screen controls, which can be useful for grabbing passwords

Some uses of keyloggers could be considered ethical or appropriate to varying degrees. Keylogger recorders may also be used by:

* employers to observe employees' computer activities;
* parents to supervise their children's Internet usage;
* device owners to track the possible unauthorized activity on their devices; or
* law enforcement agencies to analyze incidents involving computer use.

There are basically two types of Keyloggers:

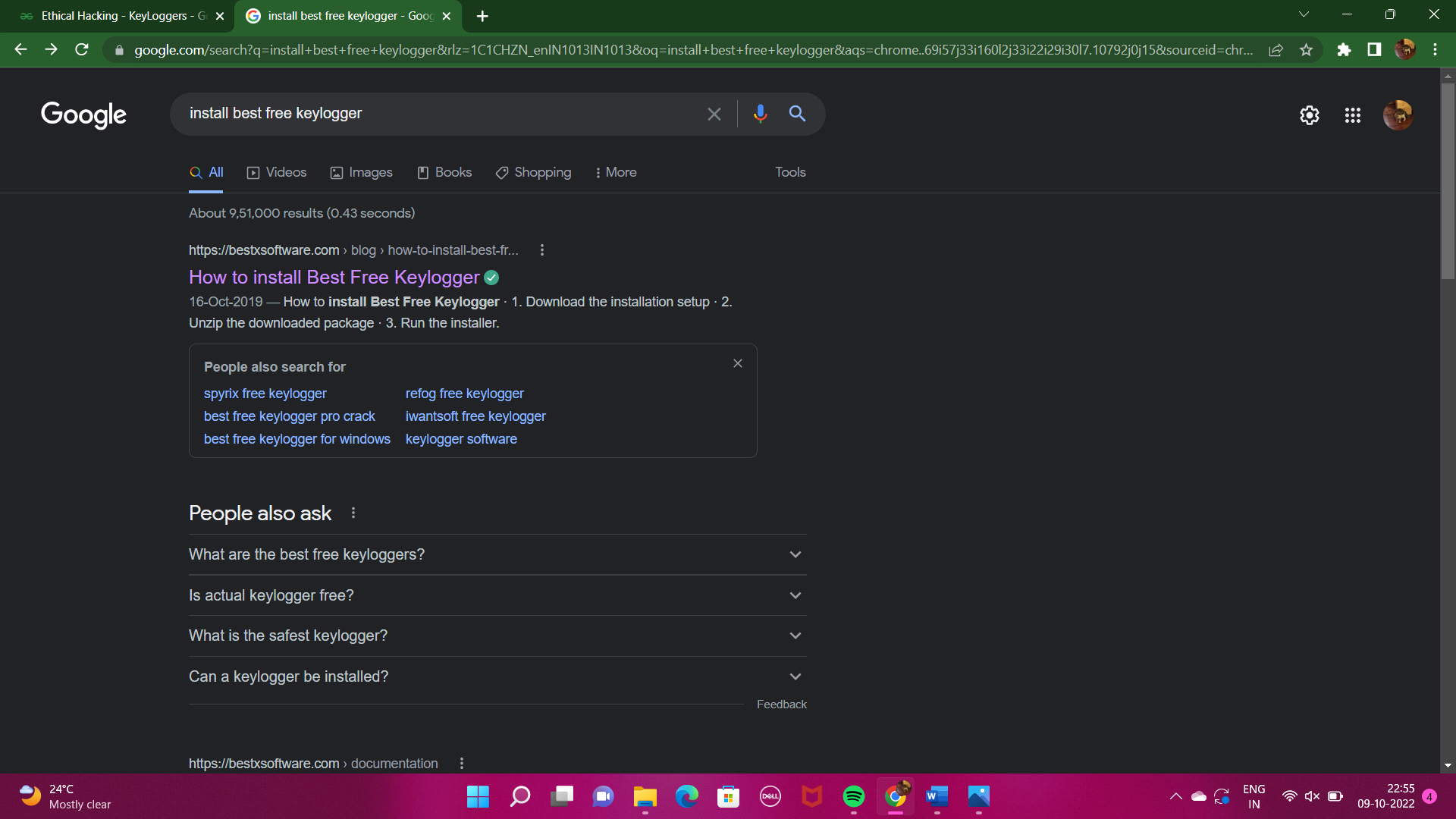
Hardware Keylogger: This is a thumb-size device. It records all the keystrokes you enter from the keyboard and then saves them in its memory. Later this data will be analyzed. The drawback of this device is, It can’t record mouse clicks, can’t take screenshots, and even can’t email, more importantly, It requires physical access to the machine. Hardware Keylogger is advantageous because it’s not hooked into any software nor can it be detected by any software.

Software Keylogger: Software Keylogger can be installed in the victim’s system even if they use an updated Antivirus. There are lots of software available in the market which make a Keylogger undetectable by the latest antivirus, we are going to study them too in upcoming chapters. There are many keyloggers available in the market with various features. Some examples of Software Keyloggers are:

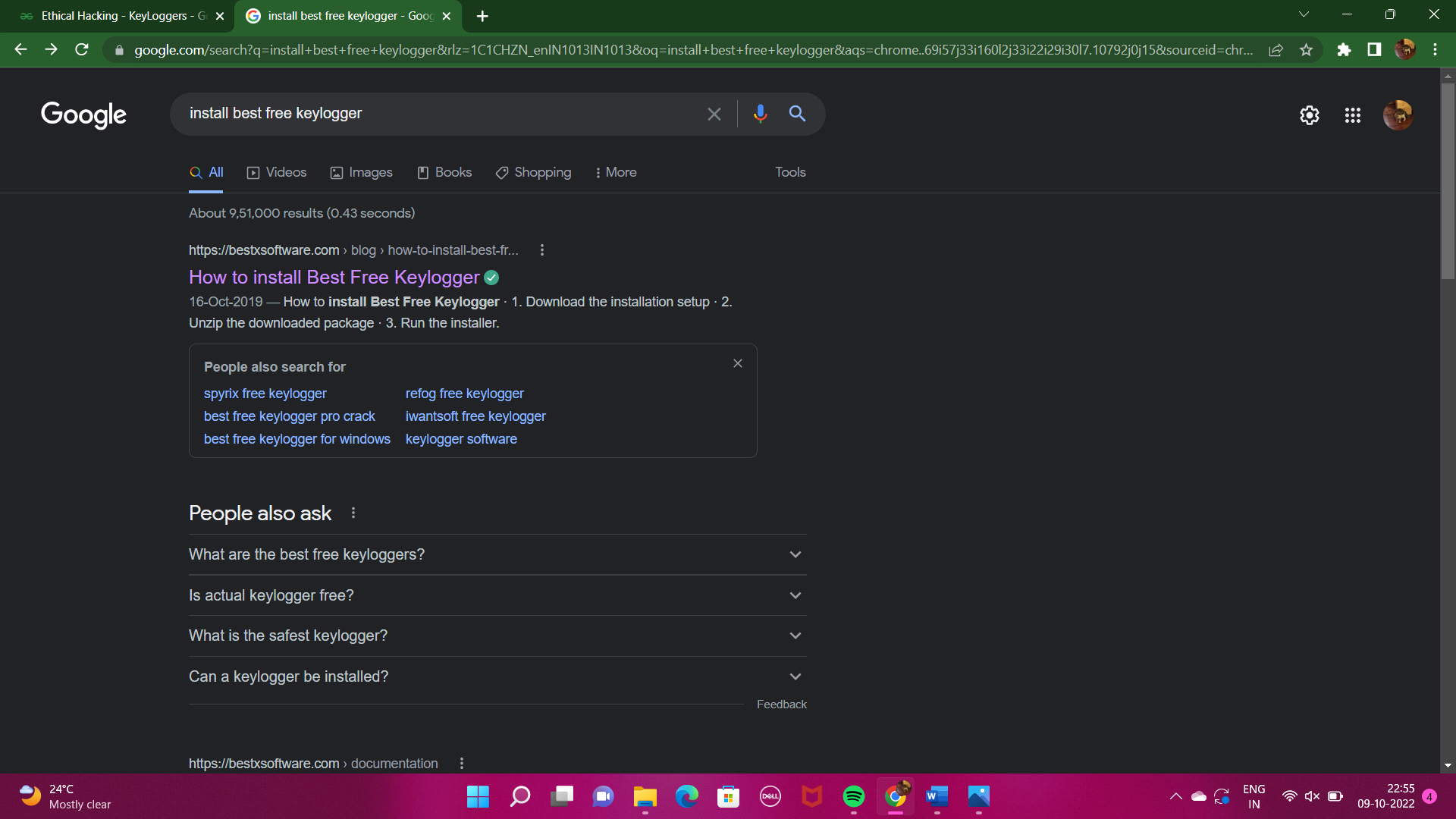
* [Revealer HYPERLINK "https://www.logixoft.com/" HYPERLINK "https://www.logixoft.com/" HYPERLINK "https://www.logixoft.com/" HYPERLINK "https://www.logixoft.com/" HYPERLINK "https://www.logixoft.com/" HYPERLINK "https://www.logixoft.com/" HYPERLINK "https://www.logixoft.com/" Keylogger](https://www.logixoft.com/)
* [Ardamax HYPERLINK "https://www.geeksforgeeks.org/ethical-hacking-keyloggers/www.ardamax.com/keylogger" HYPERLINK "https://www.geeksforgeeks.org/ethical-hacking-keyloggers/www.ardamax.com/keylogger" HYPERLINK "https://www.geeksforgeeks.org/ethical-hacking-keyloggers/www.ardamax.com/keylogger" HYPERLINK "https://www.geeksforgeeks.org/ethical-hacking-keyloggers/www.ardamax.com/keylogger" HYPERLINK "https://www.geeksforgeeks.org/ethical-hacking-keyloggers/www.ardamax.com/keylogger" HYPERLINK "https://www.geeksforgeeks.org/ethical-hacking-keyloggers/www.ardamax.com/keylogger" HYPERLINK "https://www.geeksforgeeks.org/ethical-hacking-keyloggers/www.ardamax.com/keylogger" Keylogger](https://www.geeksforgeeks.org/ethical-hacking-keyloggers/www.ardamax.com/keylogger)
* [WinSpy](https://www.geeksforgeeks.org/ethical-hacking-keyloggers/www.win.spy.com)
* [Invisible Keylogger](https://www.geeksforgeeks.org/ethical-hacking-keyloggers/www.mykeylogger.com)
* [Refog HYPERLINK "https://www.geeksforgeeks.org/ethical-hacking-keyloggers/www.refog.com" HYPERLINK "https://www.geeksforgeeks.org/ethical-hacking-keyloggers/www.refog.com" HYPERLINK "https://www.geeksforgeeks.org/ethical-hacking-keyloggers/www.refog.com" HYPERLINK "https://www.geeksforgeeks.org/ethical-hacking-keyloggers/www.refog.com" HYPERLINK "https://www.geeksforgeeks.org/ethical-hacking-keyloggers/www.refog.com" HYPERLINK "https://www.geeksforgeeks.org/ethical-hacking-keyloggers/www.refog.com" HYPERLINK "https://www.geeksforgeeks.org/ethical-hacking-keyloggers/www.refog.com" Keylogger](https://www.geeksforgeeks.org/ethical-hacking-keyloggers/www.refog.com)
* Best free key logger

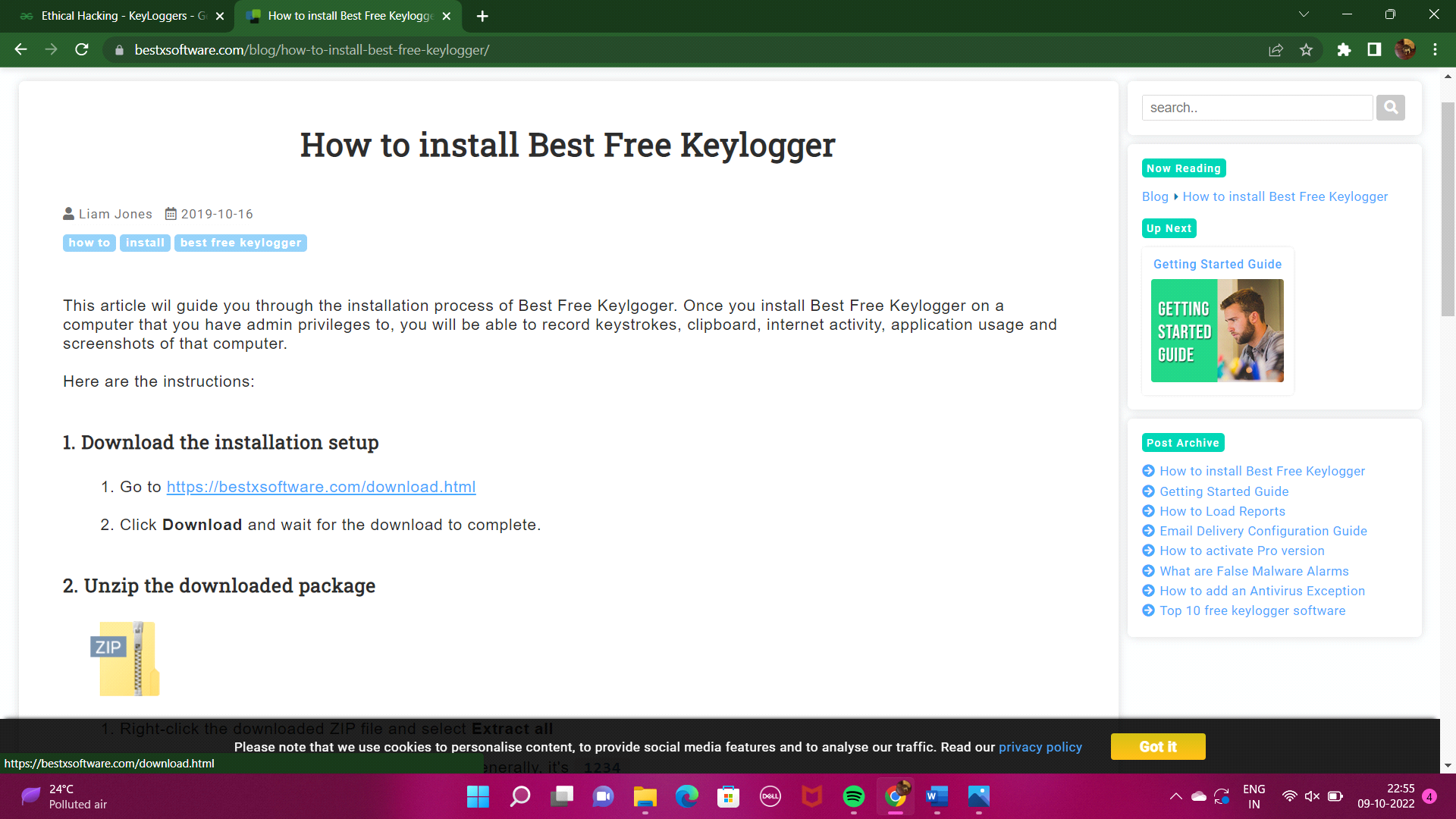
**PROCEDURE:**

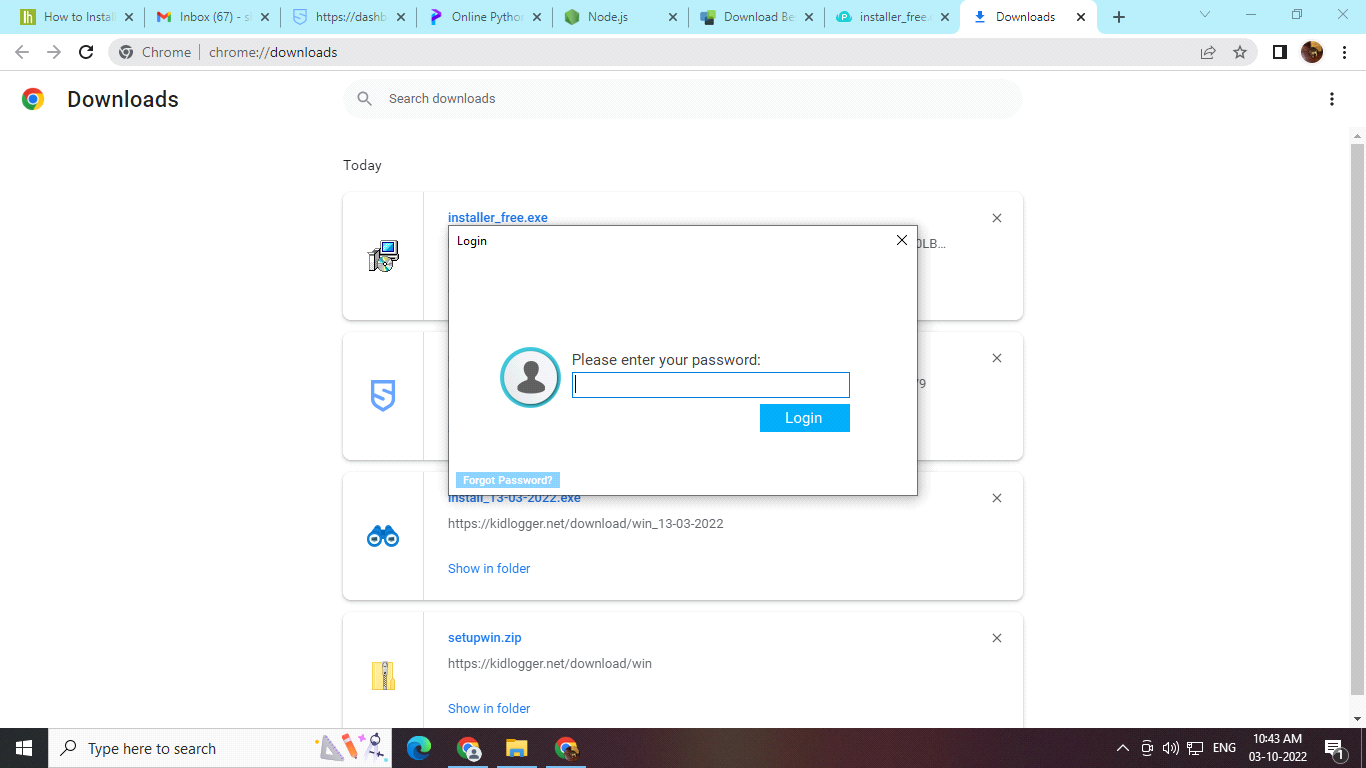
* Go to <https://bestxsoftware.com/download.html>



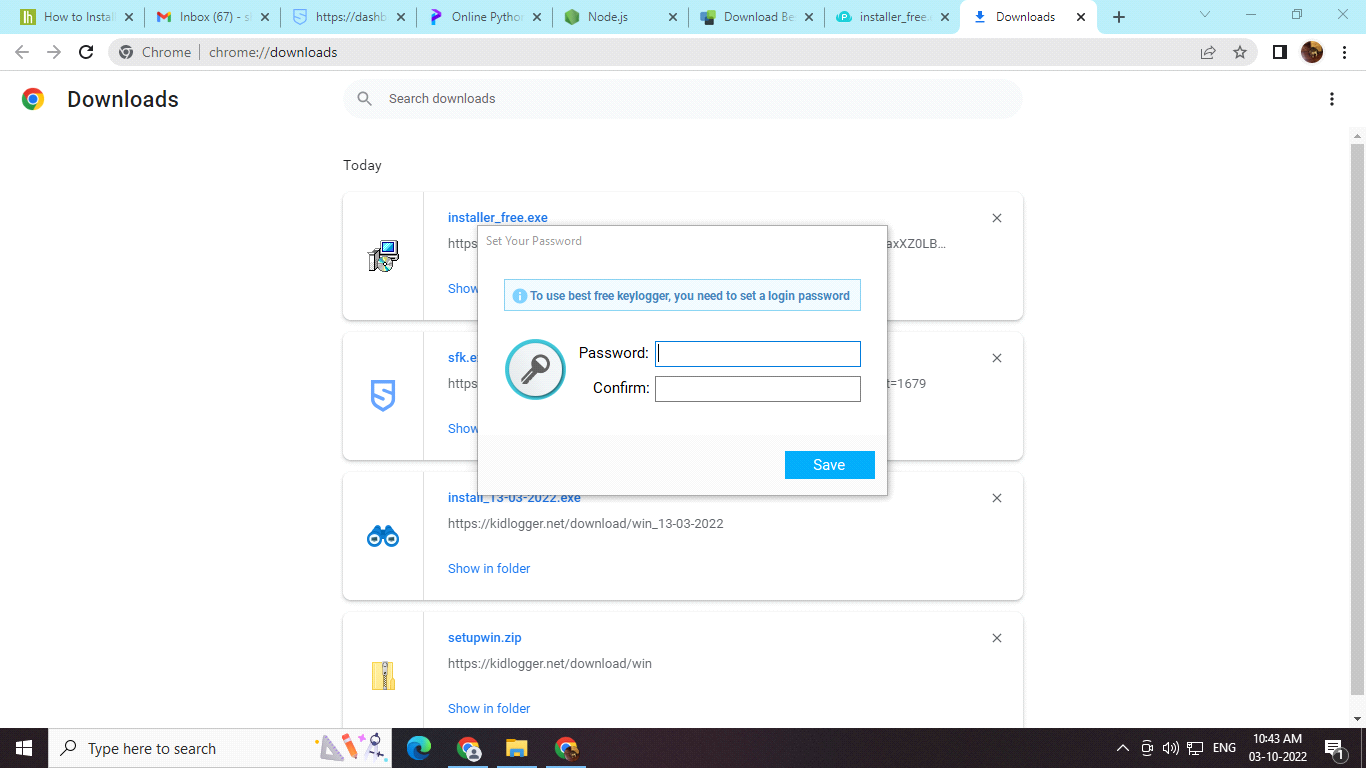
* Click **Download** and wait for the download to complete.



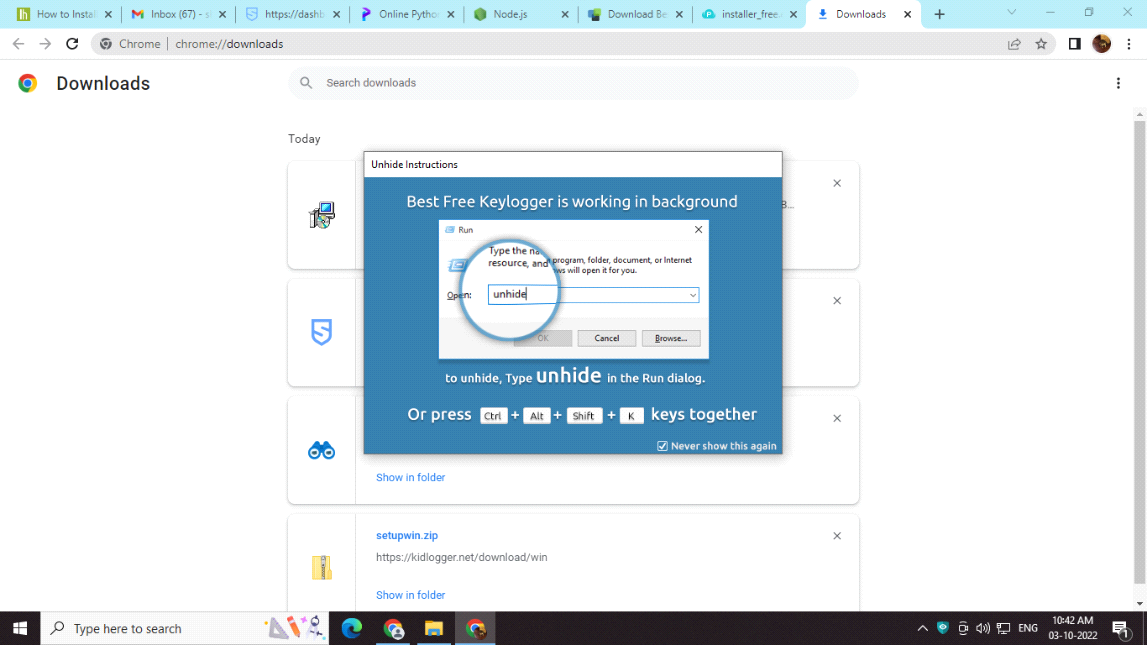
* Right-click the downloaded ZIP file and select **Extract all.**
* Create a new password for the keylogger.



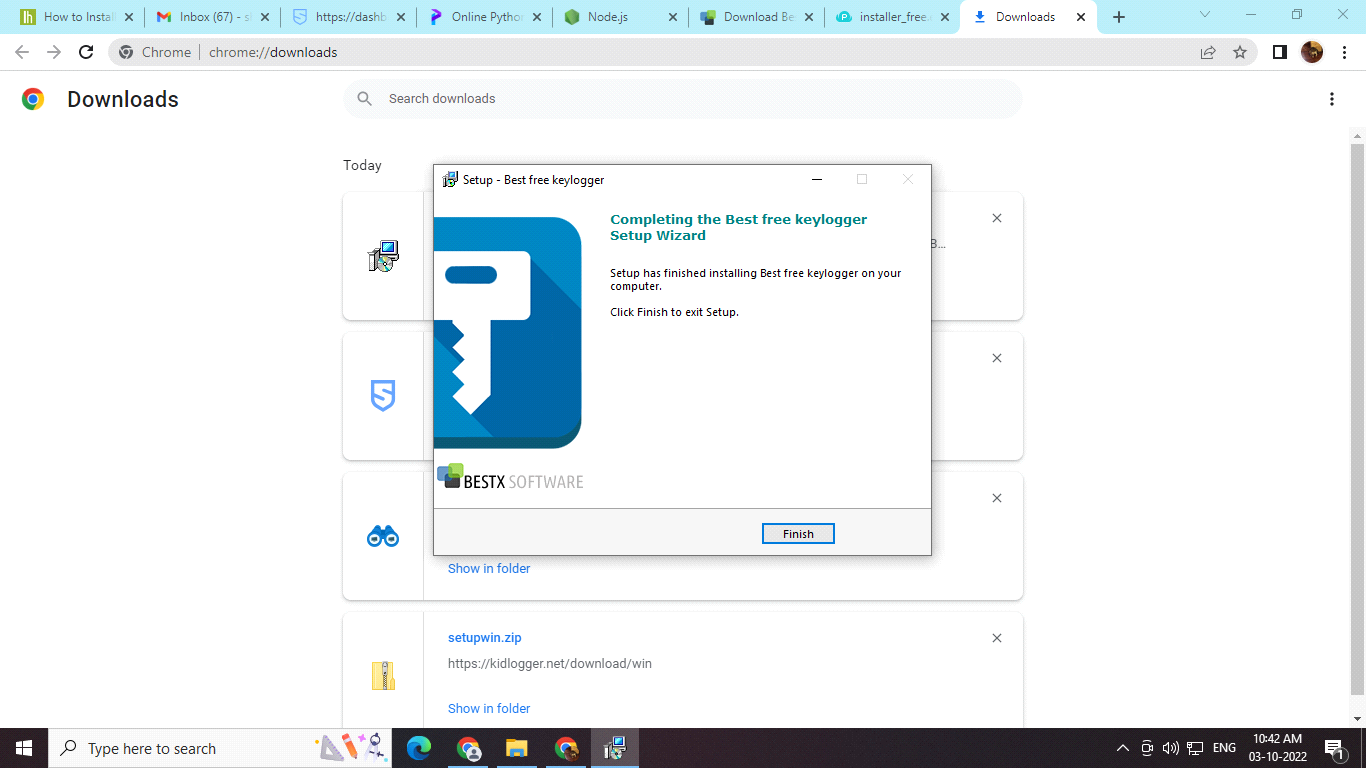
* Once the password is created, confirm and save it



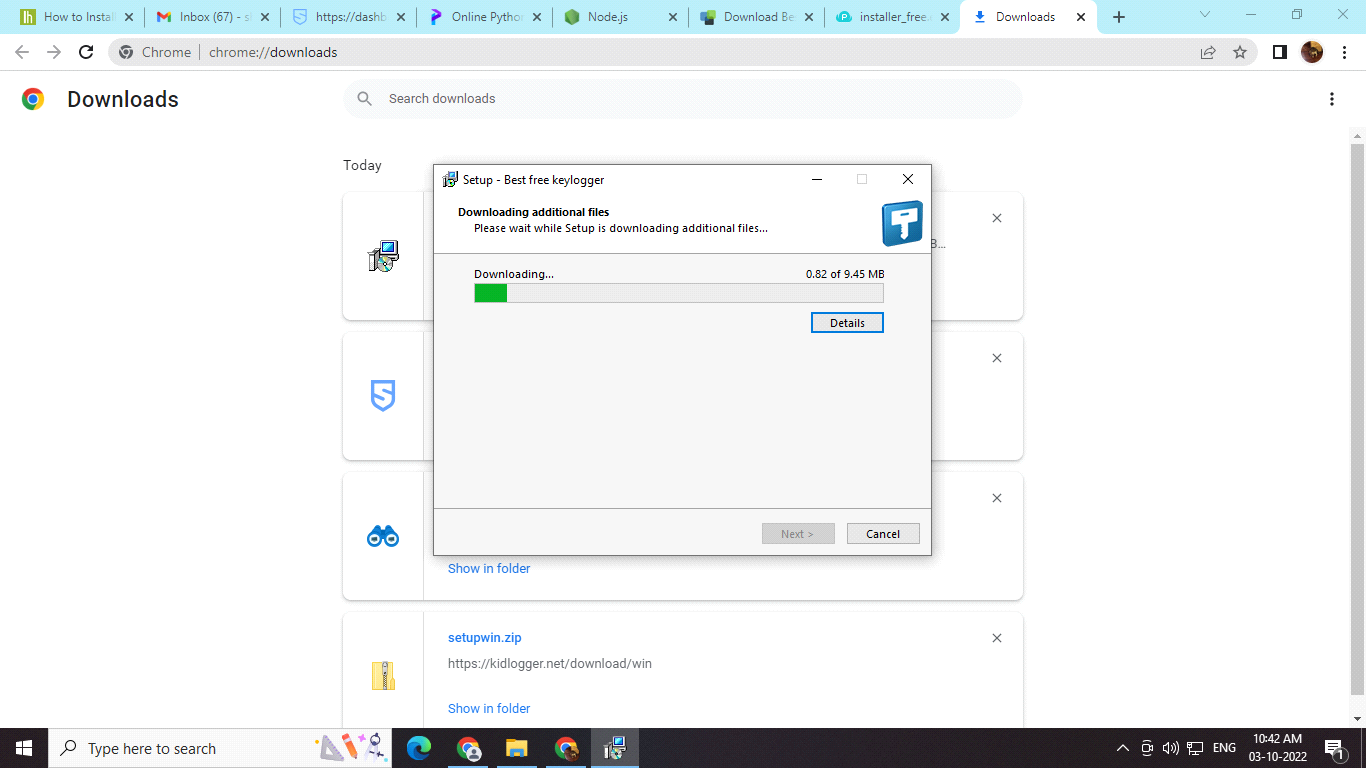
* The keylogger works in the background, so to unhide it press the ctrl + shift + alt + R keys together.



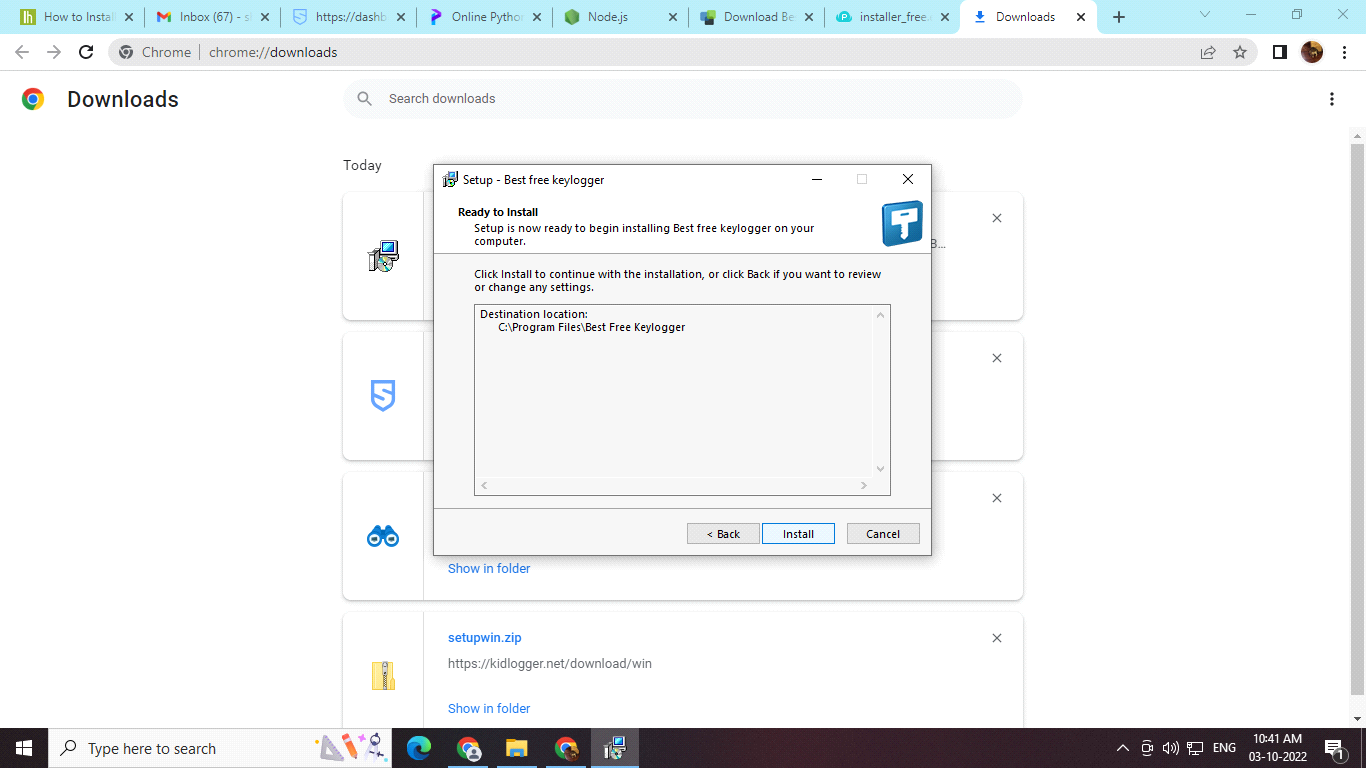
* A setup dialog box will open with a finish button, click finish.



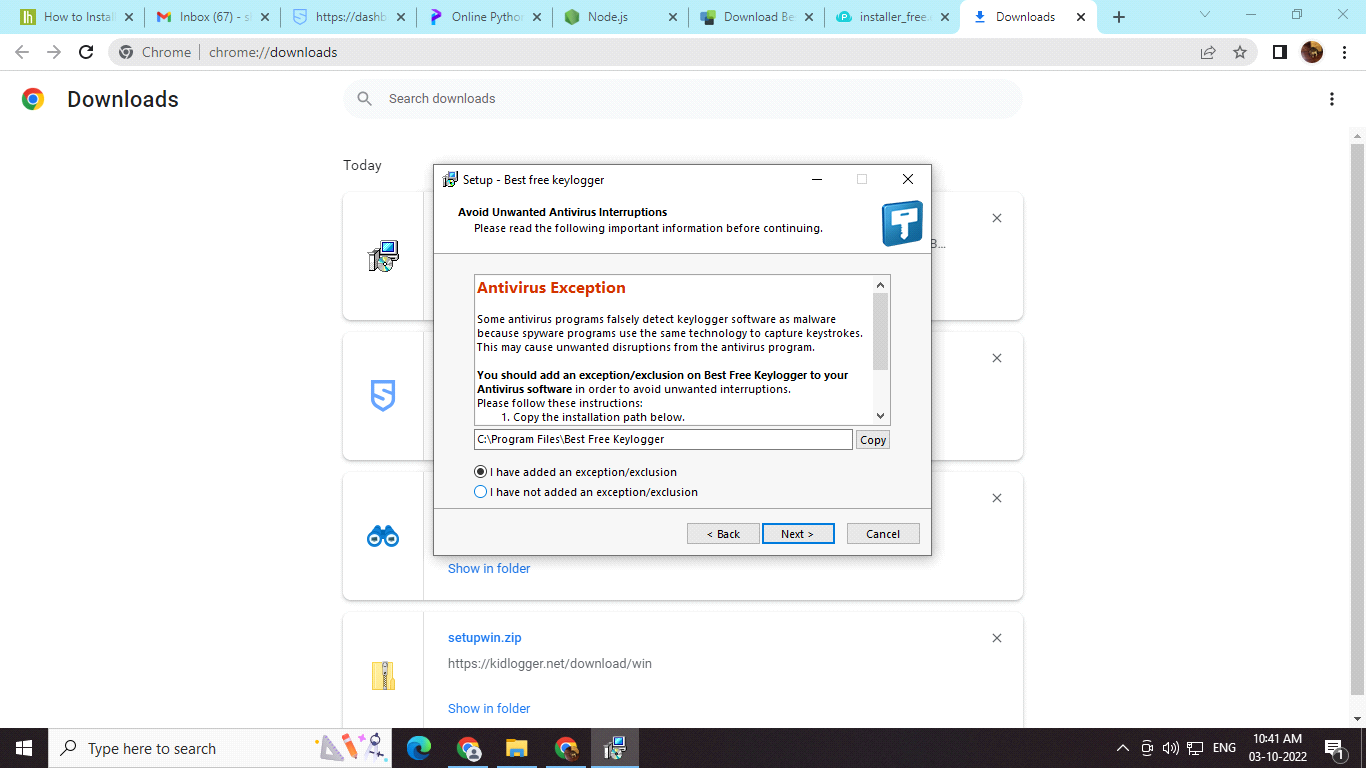
* Next, additional files will be installed for keylogger.



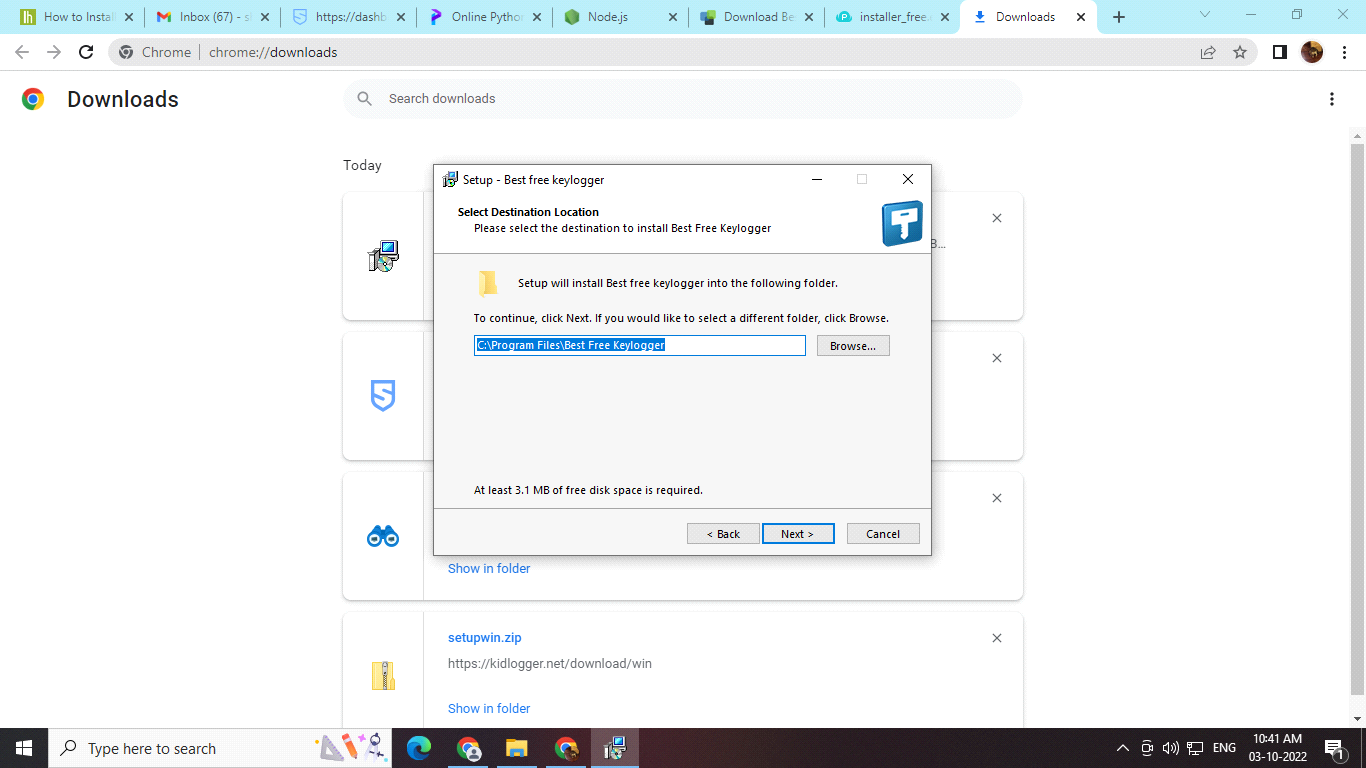
* Ready to install dialogue box will provide further details and address of installed files. Click install to continue.



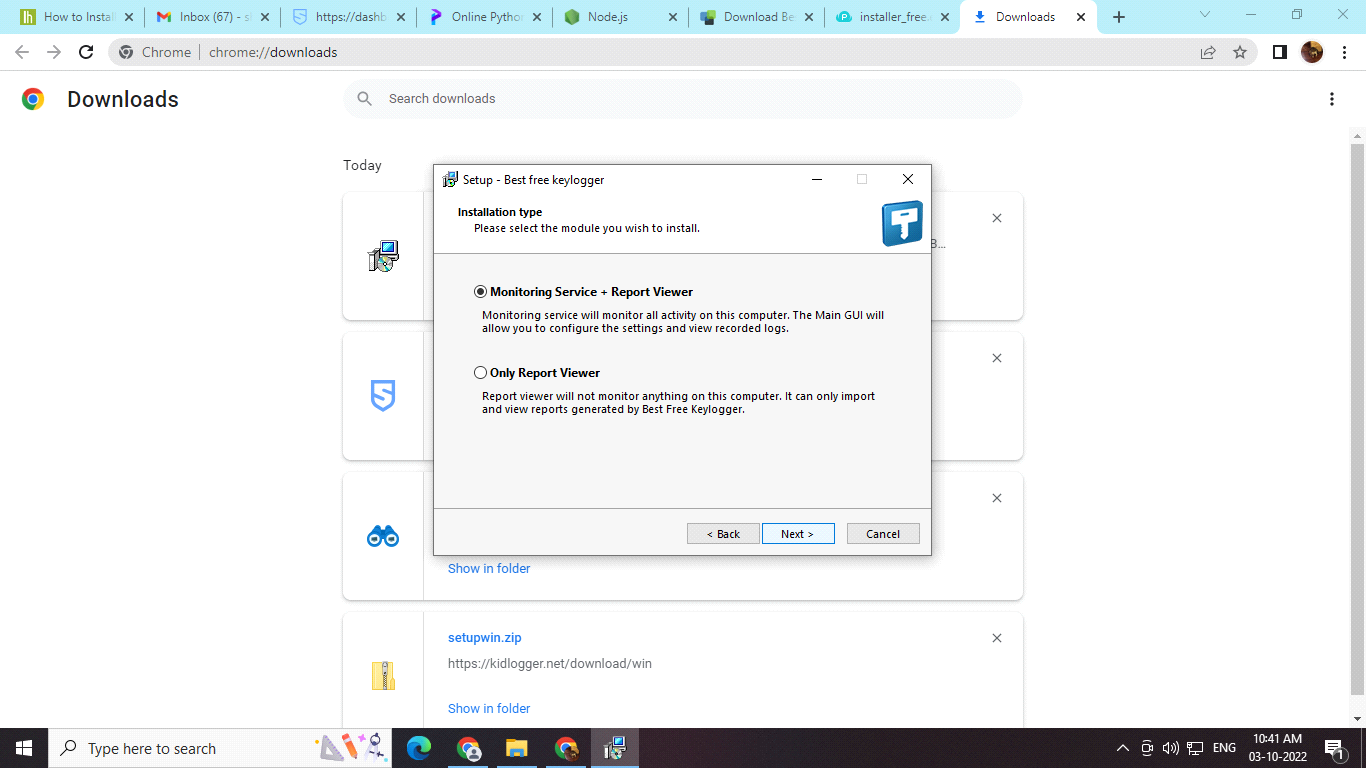
* An antivirus exception will appear, the user has to accept the antivirus exception to install the keylogger. Click next to continue.



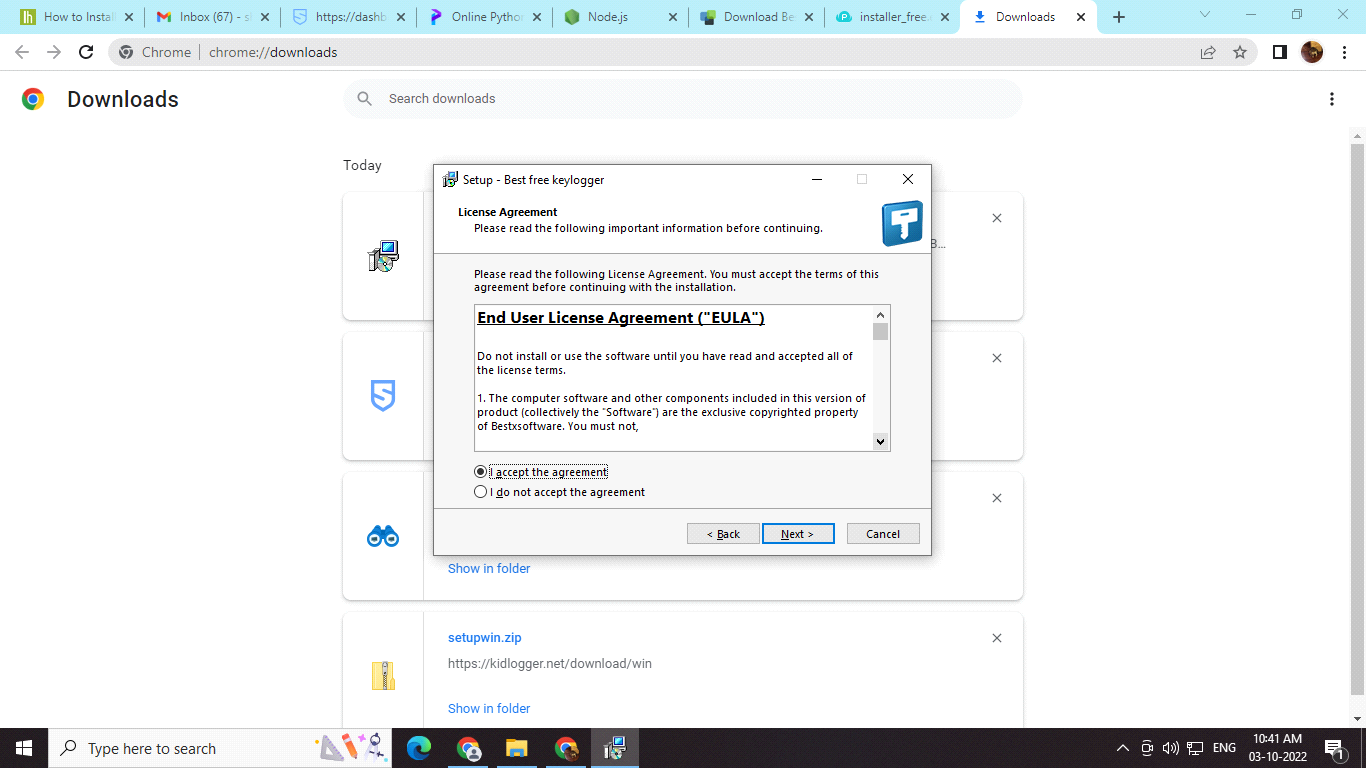
* The final location of the keylogger files will be displayed which is preferred to be unchanged. Click next to continue.



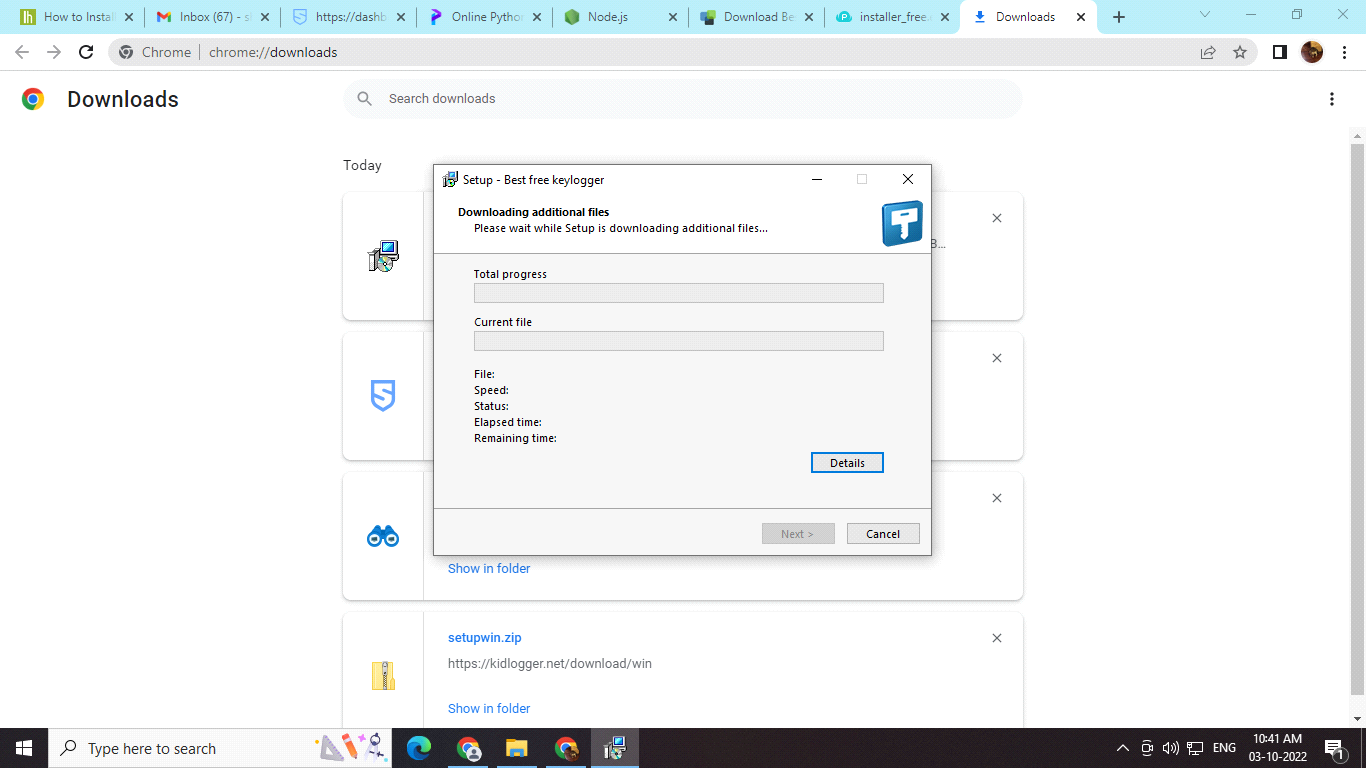
* The user will be asked about the installation type, in order to have a detailed analysis of keylogging it is preferred to go for the monitoring service + report Viewer option. Click next to continue.



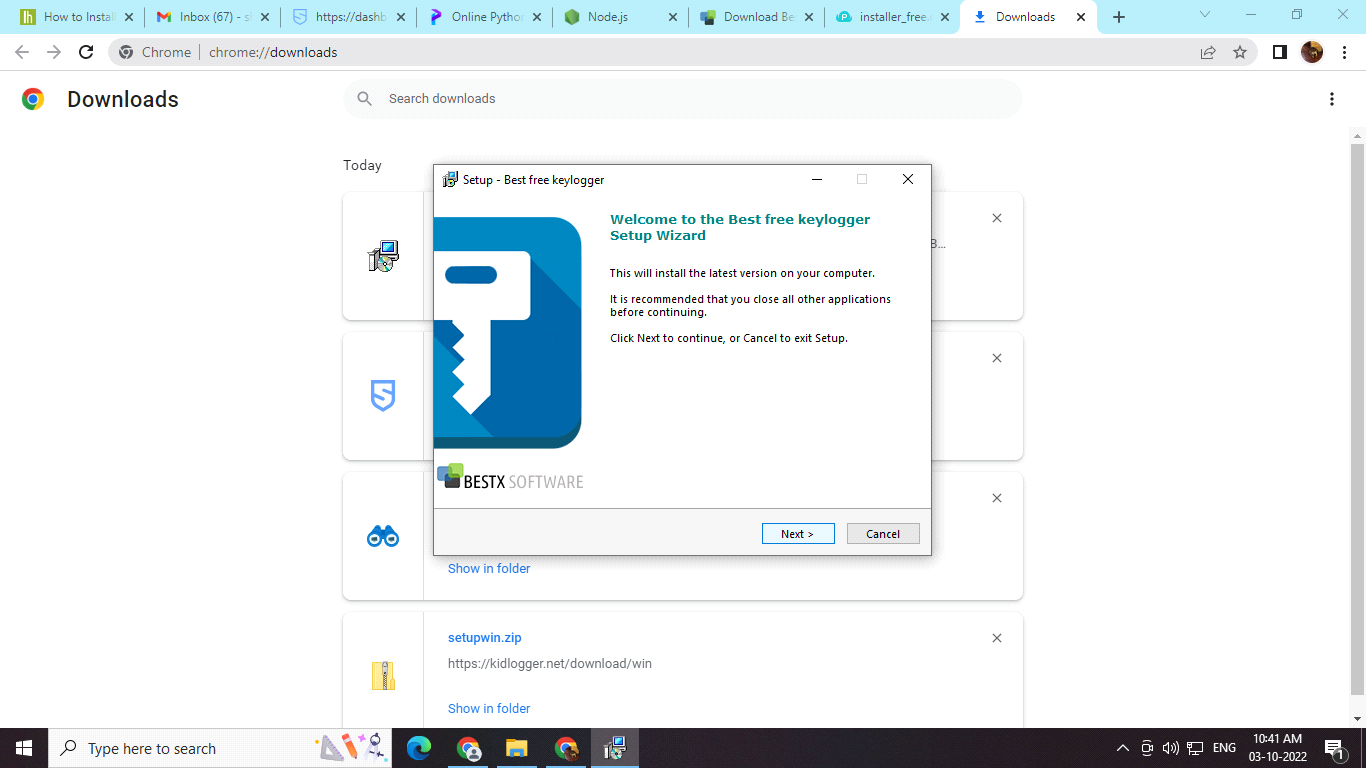
* License agreement window will ask to accept all the terms and conditions regarding the keylogger. Click next to continue.



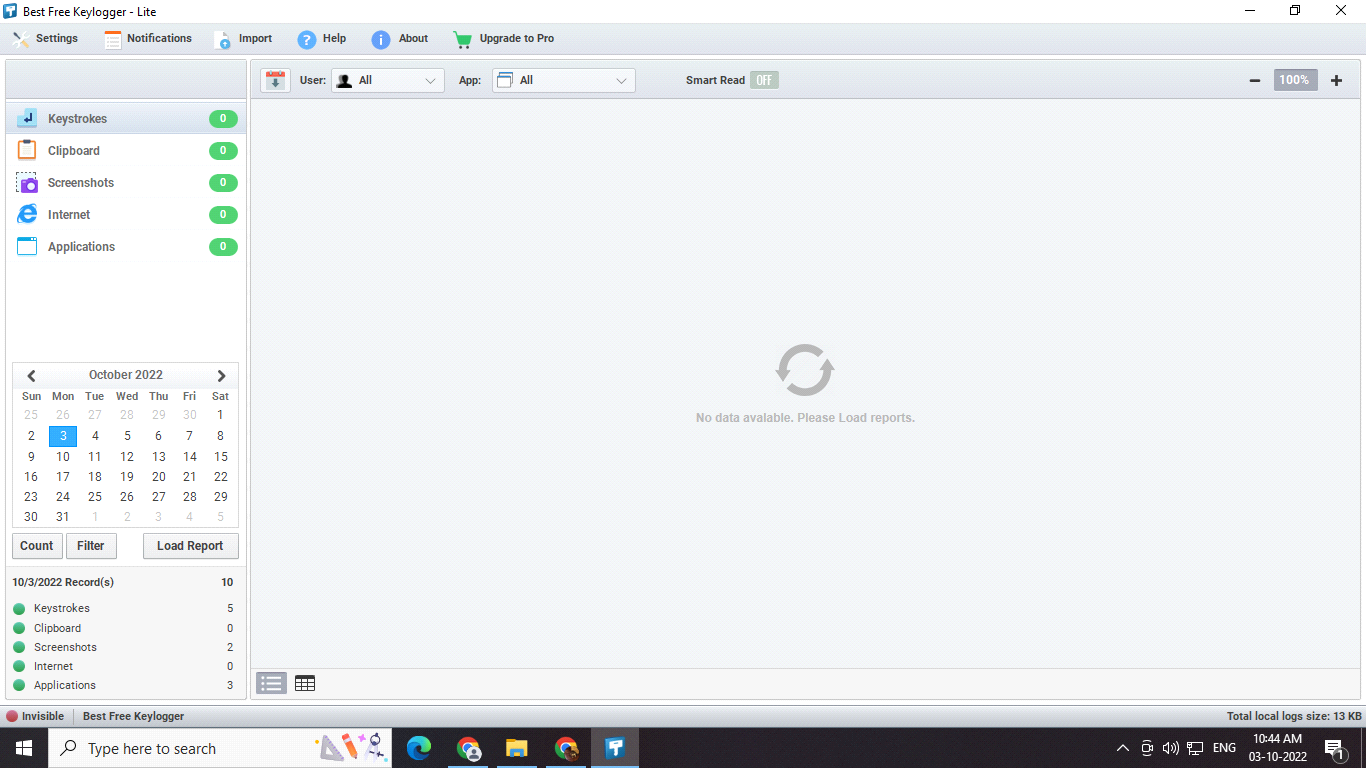
* Again a detailed installation of all the files necessary will occur. Click next again.



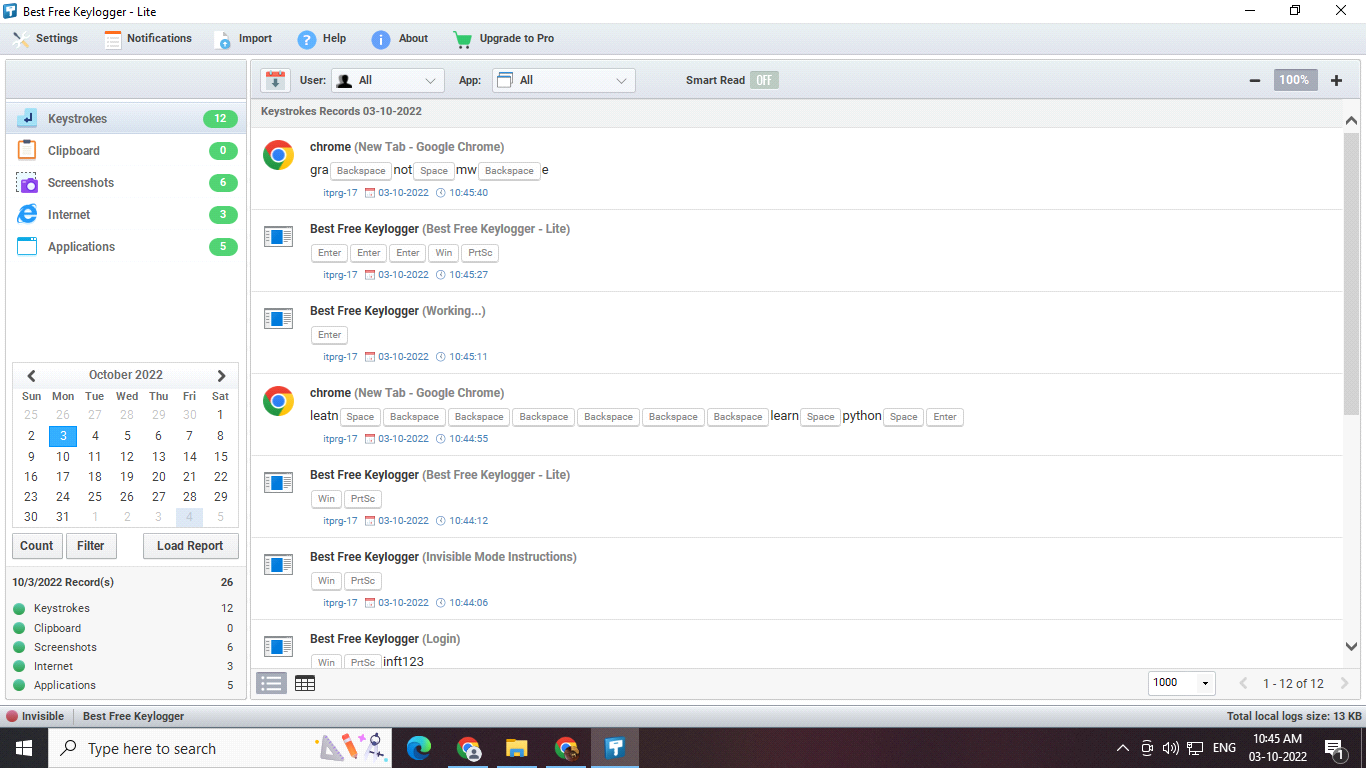
* Once all the installation procedures are done, user will find a desktop icon to access the records .

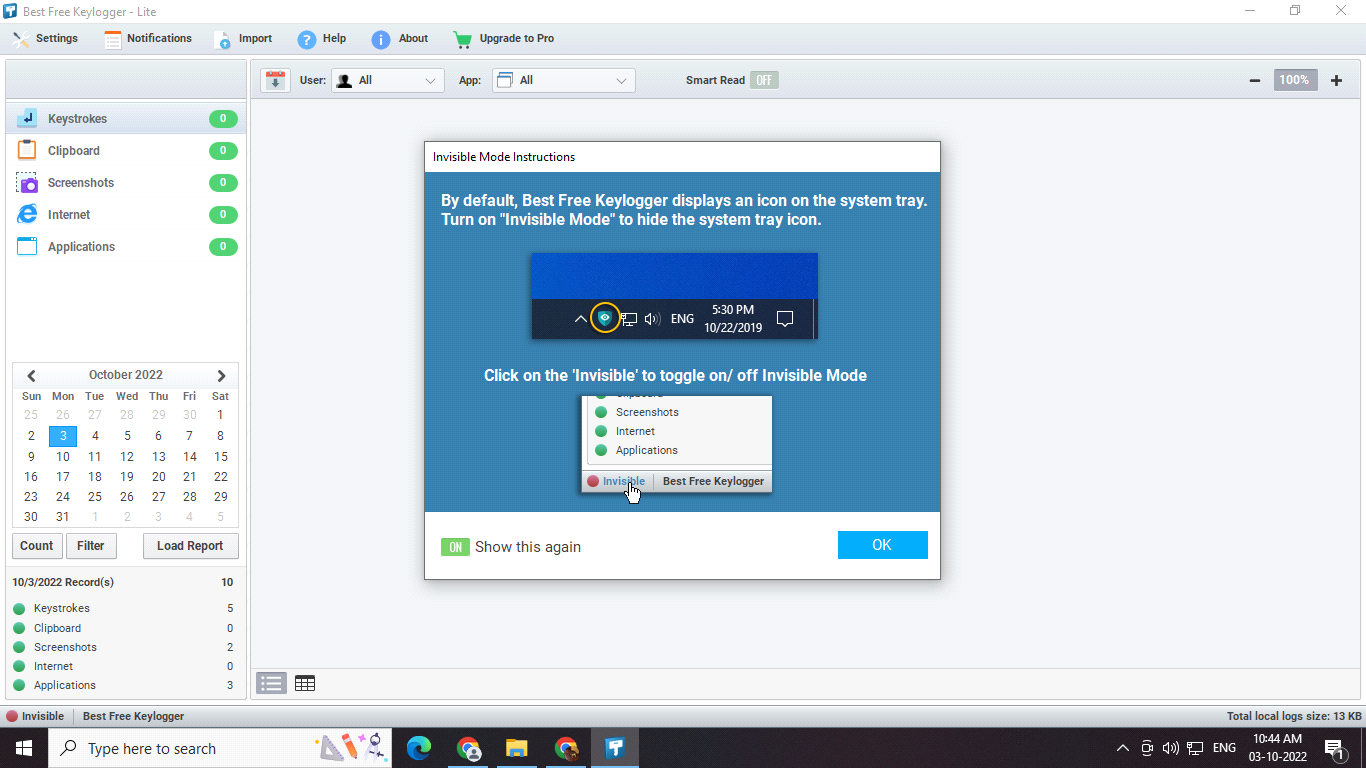
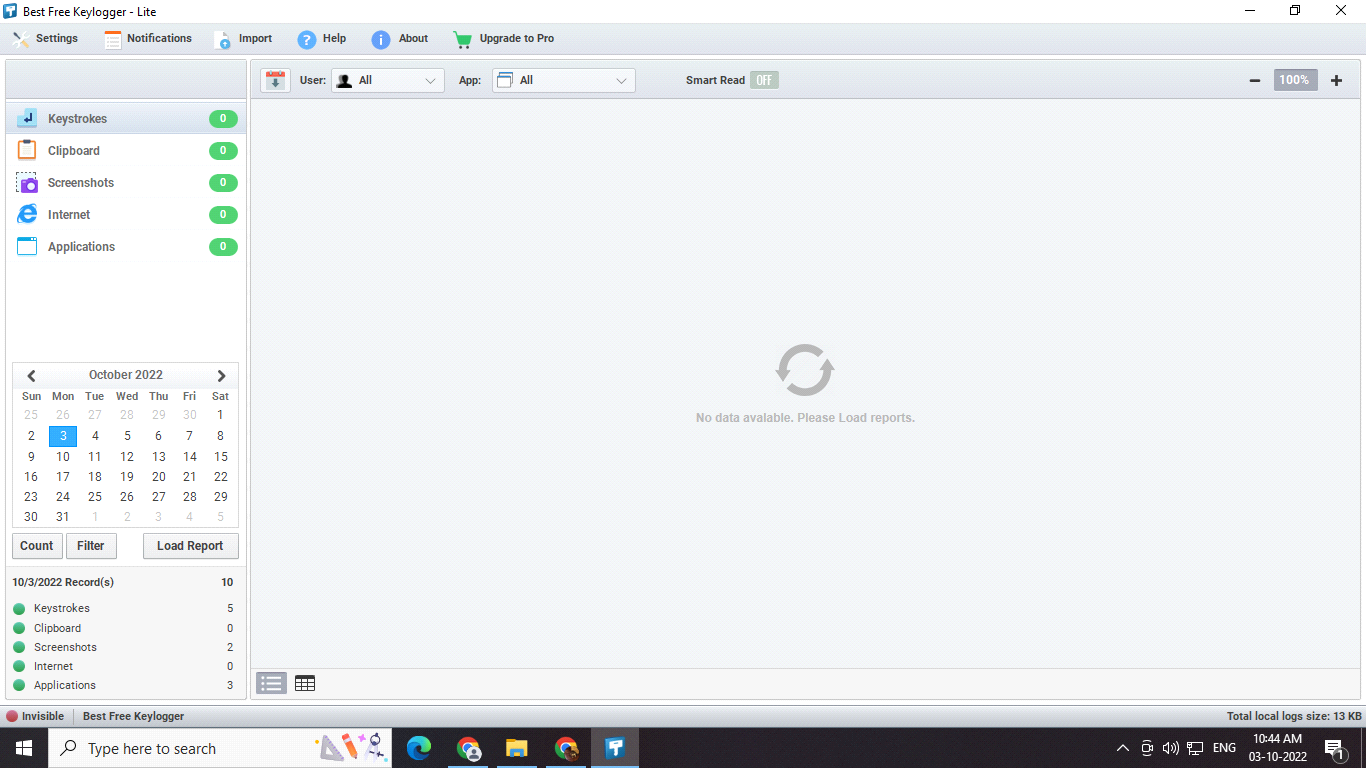


* The recods would be inintially empty.



* So as the user uses different applications and searches on the browser, all this information is accurately noted in the keylogger.





**QUESTIONS:**

1. What is a keylogger? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. State any 3 funtions of a keylogger. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Give examples of hardware keylogger. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. State few applications of a keylogger. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Enlist software keyloggers? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**CONCLUSION:** The functions and applications of a keylogger are studied and examined.

**REFERENCES:**

<https://www.techtarget.com/searchsecurity/definition/keylogger>

<https://www.techtarget.com/searchsecurity/definition/keylogger>

<https://www.geeksforgeeks.org/introduction-to-keyloggers/>

**Experiment No. 9**



**Aim:** To install and study Network Protocol Analyzer, analyse packet header using **Wireshark.**

**API:** Packet Analyzer, Ethereal, Wireshark, Ntop etc.

**Theory :-**

Version 3.6.2 (v3.6.2-0-g626020d9b3c3)

Copyright 1998-2022 Gerald Combs <gerald@wireshark.org> and contributors. License GPLv2+: GNU GPL version 2 or later <<https://www.gnu.org/licenses/gpl-2.0.html>> This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Compiled (64-bit) using Microsoft Visual Studio 2019 (VC++ 14.29, build 30139), with Qt 5.15.2, with libpcap, with GLib 2.66.4, with zlib 1.2.11, with Lua 5.2.4, with GnuTLS 3.6.3 and PKCS #11 support, with Gcrypt 1.8.3, with MIT Kerberos, with MaxMind DB resolver, with nghttp2 1.44.0, with brotli, with LZ4, with Zstandard, with Snappy, with libxml2 2.9.10, with libsmi 0.4.8, with QtMultimedia, with automatic updates using WinSparkle 0.5.7, with AirPcap, with SpeexDSP (using bundled resampler), with Minizip.

Running on 64-bit Windows 10 (21H2), build 19044, with Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz (with SSE4.2), with 7973 MB of physical memory, with GLib 2.66.4, with Qt 5.15.2, with Npcap version 1.55, based on libpcap version 1.10.2-PRE-GIT, with c-ares 1.17.0, with GnuTLS 3.6.3, with Gcrypt 1.8.3, with nghttp2 1.44.0, with brotli 1.0.9, with LZ4 1.9.3, with Zstandard 1.4.0, without AirPcap, with light display mode, without HiDPI, with LC\_TYPE=English\_India.utf8, binary plugins supported (21 loaded).

Wireshark is Open Source Software released under the GNU General Public License.

Check the man page and [https://www.wireshark.org](https://www.wireshark.org/) for more information.

Wireshark is the world’s foremost and widely-used network protocol analyzer. It lets you see what’s happening on your network at a microscopic level and is the de facto (and often de jure) standard across many commercial and non-profit enterprises, government agencies, and educational institutions. Wireshark development thrives thanks to the volunteer contributions of networking experts around the globe and is the continuation of a project started by Gerald Combs in 1998.

**Wireshark has a rich feature set which includes the following:**

* Deep inspection of hundreds of protocols, with more being added all the time
* Live capture and offline analysis
* Standard three-pane packet browser
* Multi-platform: Runs on Windows, Linux, macOS, Solaris, FreeBSD, NetBSD, and many others
* Captured network data can be browsed via a GUI, or via the TTY-mode TShark utility
* The most powerful display filters in the industry
* Rich VoIP analysis

Read/write many different capture file formats: tcpdump (libpcap), Pcap NG, Catapult DCT2000, Cisco Secure IDS iplog, Microsoft Network Monitor, Network General Sniffer® (compressed and uncompressed), Sniffer® Pro, and NetXray®, Network Instruments Observer, NetScreen snoop, Novell LANalyzer, RADCOM WAN/LAN Analyzer, Shomiti/Finisar Surveyor, Tektronix K12xx, Visual Networks Visual UpTime, WildPackets EtherPeek/TokenPeek/AiroPeek, and many others

Capture files compressed with gzip can be decompressed on the fly

Live data can be read from Ethernet, IEEE 802.11, PPP/HDLC, ATM, Bluetooth, USB, Token Ring, Frame Relay, FDDI, and others (depending on your platform)

Decryption support for many protocols, including IPsec, ISAKMP, Kerberos, SNMPv3, SSL/TLS, WEP, and WPA/WPA2

Coloring rules can be applied to the packet list for quick, intuitive analysis

Output can be exported to XML, PostScript®, CSV, or plain text

**Procedure: To download Wireshark**:

1.Open a web browser.

2.Navigate to [http://www.wireshark.org](http://www.wireshark.org/).

3.Select Download Wireshark.

4.Select the Wireshark Windows Installer matching your system type, either 32-bit or 64-bit as determined in Activity 1. Save the program in the Downloads folder.

5.Close the web browser.

**To install Wireshark:**

1.Open Windows Explorer.

2.Select the Downloads folder.

3.Locate the version of Wireshark you downloaded in Activity 2. Double-click on the file to open it.

4.If you see a User Account Control dialog box, select Yes to allow the program to make changes to this computer.

5.Select Next > to start the Setup Wizard.

6.Review the license agreement. If you agree, select I Agree to continue.

7.Select Next > to accept the default components.

8.Select the shortcuts you would like to have created. Leave the file extensions selected. Select Next > to continue.

9.Select Next > to accept the default install location.

10.Select Install to begin installation.

11.Select Next > to install WinPcap.

12.Select Next > to start the Setup Wizard.

13.Review the license agreement. If you agree, select I Agree to continue.

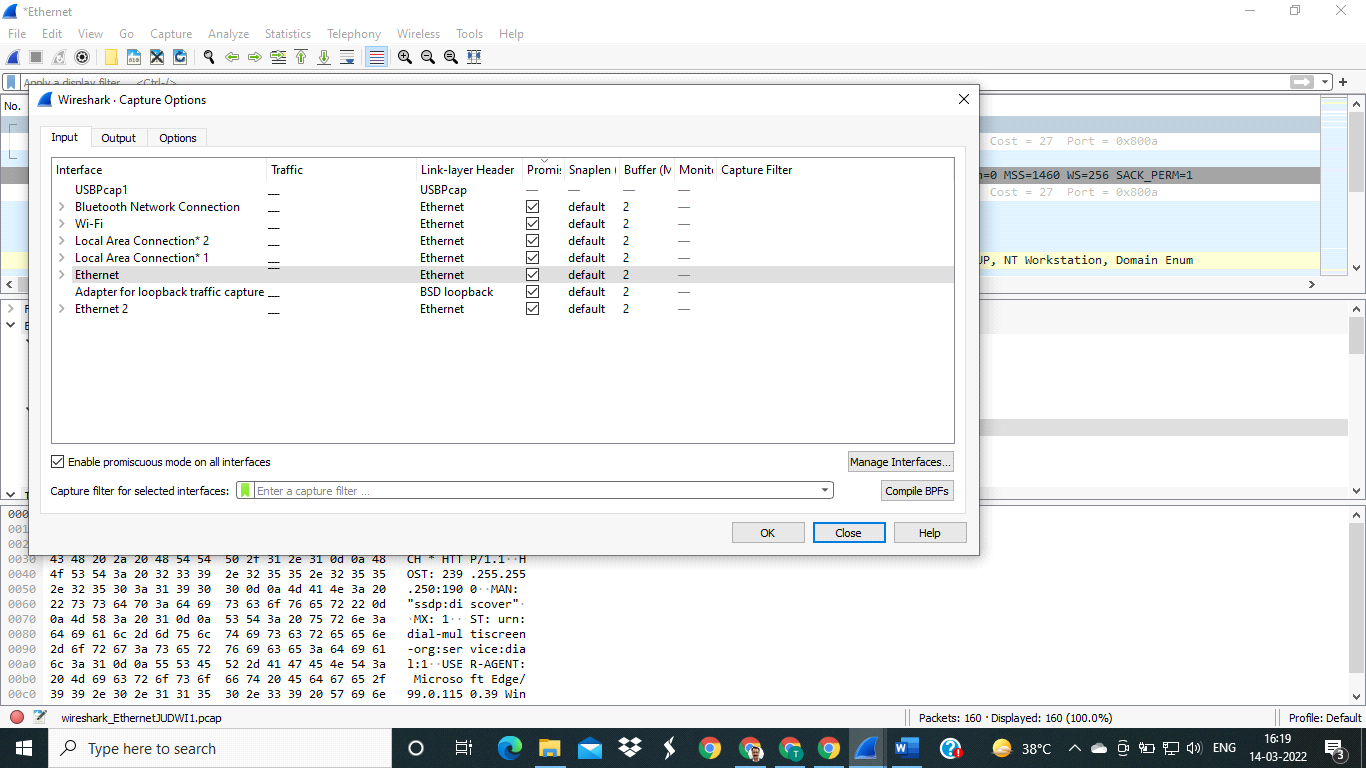
14.Select Install to begin installation.

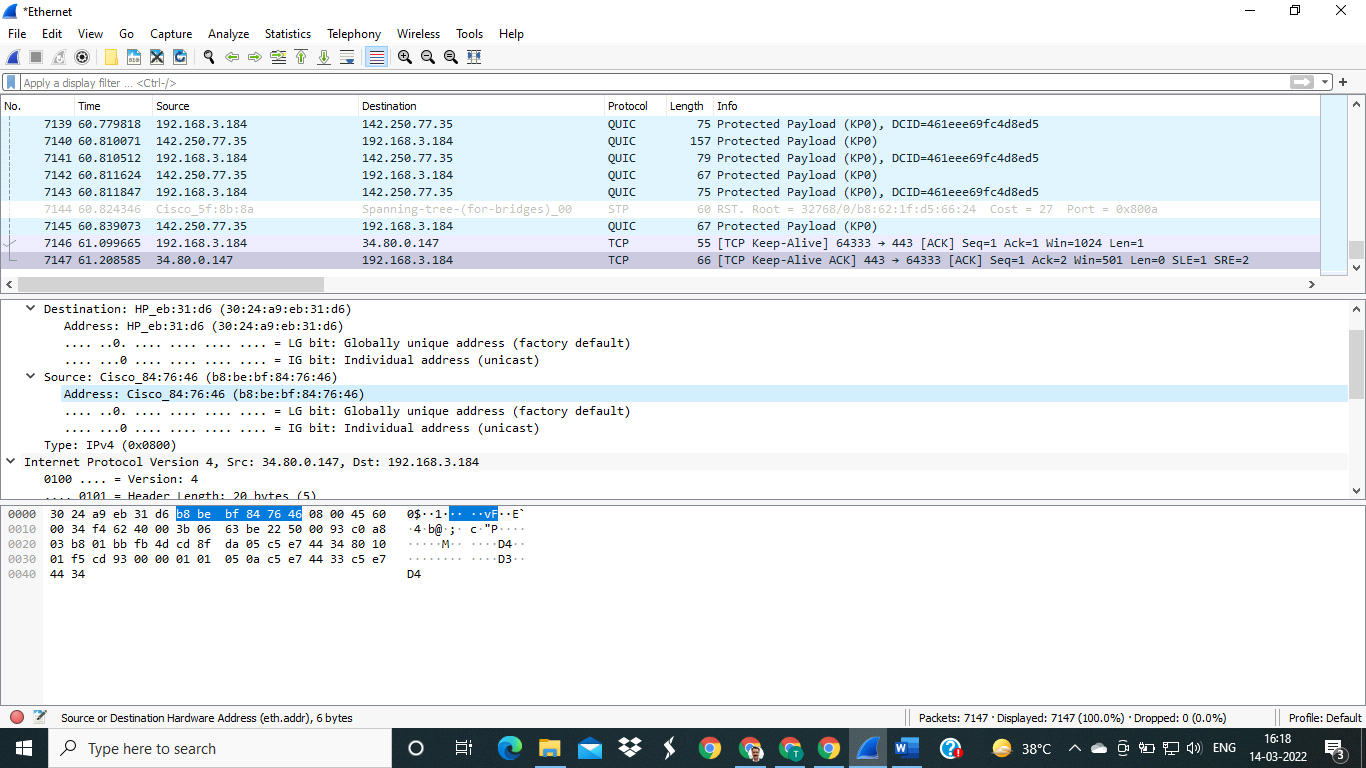
15.Select Finish to complete the installation of WinPcap.

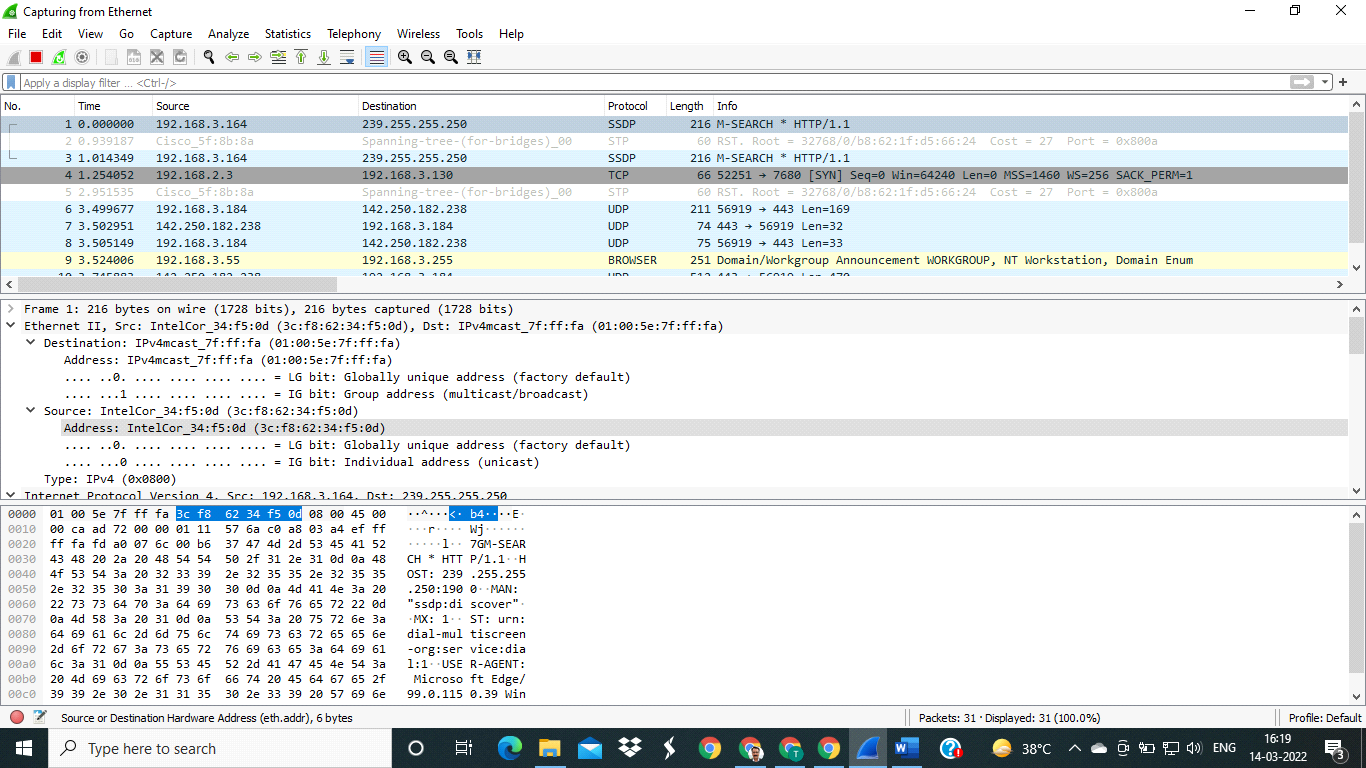
16.Select Next > to continue with the installation of Wireshark.

17.Select Finish to complete the installation of Wireshark.

**Conclusion:** Learnt how to install and configure Wireshark.







---------------------------------------------X-0-X-----------------------------------------------

**APIs:** Packet Analyzer

**Procedure:** ***To capture packer: Set up the Packet Capture:***

1. Click View > Wireless Toolbar. The Wireless Toolbar will appear just below the Main toolbar.

2. Use the Wireless Toolbar to configure the desired channel and channel width.

3. Under Capture, click on AirPcap USB wireless capture adapter to select the capture interface.

Note: If the AirPcap isn't listed, press F5 to refresh the list of available packet capture interfaces. The AirPcap has been discontinued by RiverBed and is 802.11n only.

4. Click the Start Capture button to begin the capture.

5. When you are finished capturing, click the Stop button.

***•Saving the Capture:***

1. To save the capture, click File > Save.

2. Name the file, and click Save.

Note: .Pcap and .Pcap-ng are good filetypes to use for the capture if you plan to use Eye P.A. to open the capture.

3. Eye P.A. can now open the capture file.

**Theory:**

**Packet sniffer \ Packet analyzer:**

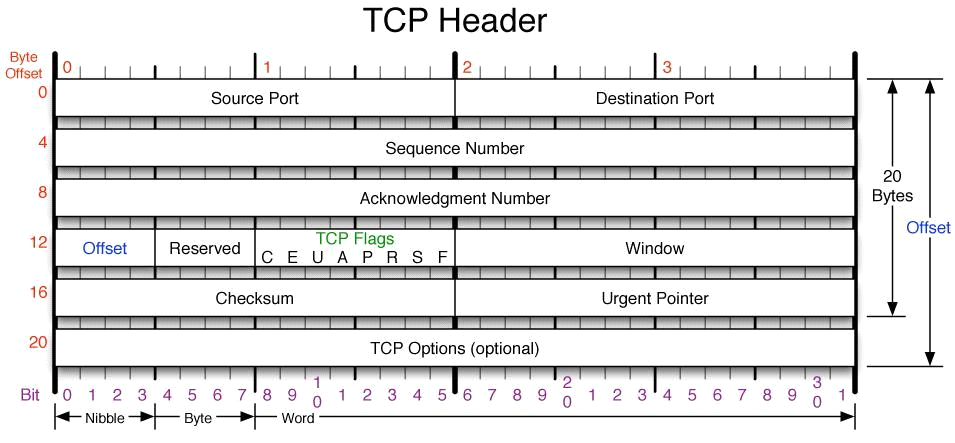
A packet analyzer (also known as a network analyzer, protocol analyzer or packet sniffer or for particular types of networks, an Ethernet sniffer or wireless sniffer) is a computer program or a piece of computer hardware that can intercept and log traffic passing over a digital network or part of a network. As data streams own across the network, the sniffer captures each packet and, if needed, decodes the packet's raw data, showing the values of various fields in the packet, and analyzes its content according to the appropriate RFC or other specifications.

**Different types of packet:**

**1. TCP:**

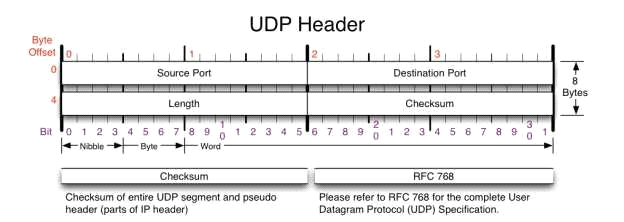
The Transmission Control Protocol (TCP) is one of the core protocols of the Internet protocol suite (IP), and is so common that the entire suite is often called TCP/IP. TCP provides reliable, ordered and error-checked delivery (or notification of failure to deliver) of a stream of octets between programs running on computers connected to a local area network, intranet or the public Internet. It resides at the transport layer. Web browsers use TCP when they connect to servers on the World Wide Web, and it is used to deliver email and transfer files from one location to another. The protocol corresponds to the transport layer of TCP/IP suite. TCP provides a communication service at an intermediate level between an application program and the Internet Protocol (IP). That is, when an application program desires to send a large chunk of data across the Internet using IP, instead of breaking the data into IP-sized pieces and issuing a series of IP requests, the software can issue a single request to TCP and let TCP handle the

IP works by exchanging pieces of information called packets. A packet is a sequence of octets (bytes) and consists of a header followed by a body. The header describes the packet's source, destination and control information. The body contains the data IP is transmitting. Due to network congestion, traffic load balancing, or other unpredictable network behavior, IP packets can be lost, duplicated, or delivered out of order. TCP detects these problems, requests retransmission of lost data, rearranges out-of-order data, and even helps minimize network congestion to reduce the occurrence of the other problems. If the data still remains undelivered, its source is notified of this failure. Once the TCP receiver has reassembled the sequence of octets originally transmitted, it passes them to the receiving application. Thus, TCP abstracts the application's communication from the underlying networking details. TCP is a reliable stream delivery service that guarantees that all bytes received will be identical with bytes sent and in the correct order. Since packet transfer over many networks is not reliable, a technique known as positive acknowledgment with retransmission is used to guarantee reliability of packet transfers. This fundamental technique requires the receiver to respond with an acknowledgment message as it receives the data. The sender keeps a record of each packet it sends. The sender also maintains a timer from when the packet was sent, and retransmits a packet if the timer expires before the message has been acknowledged. The timer is needed in case a packet gets lost or corrupted.



**2. UDP:**

The User Datagram Protocol (UDP) is one of the core members of the Internet protocol Suite. UDP uses a simple connectionless transmission model with a minimum of protocol mechanism. It has no handshaking dialogues, and thus exposes any unreliability of the underlying network protocol to the user's program. There is no guarantee of delivery, ordering, or duplicate protection. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram. With UDP, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network without prior communications to set up special transmission channels or data paths. UDP is suitable for purposes where error checking and correction is either not necessary or is performed in the application, avoiding the overhead of such processing at the network interface level. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system.



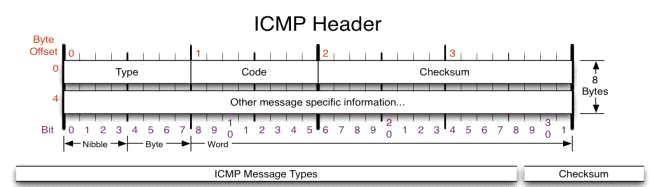
**3.ICMP:**

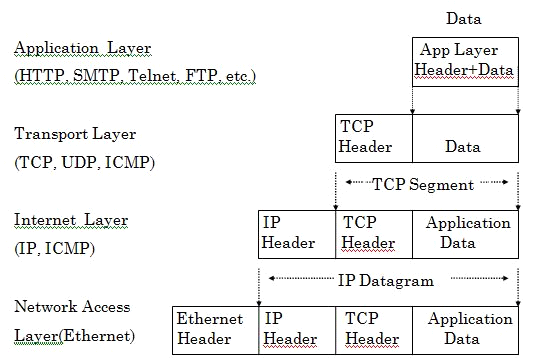
The Internet Control Message Protocol (ICMP) is one of the main protocols of the Internet Protocol Suite. It is used by network devices, like routers, to send error messages indicating, for example, that a requested service is not available or that a host or router could not be reached.

ICMP can also be used to relay query messages. It is assigned protocol number 1.ICMP differs from transport protocols such as TCP and UDP in that it is not typically used to exchange data between systems, nor is it regularly employed by end-user network applications (with the

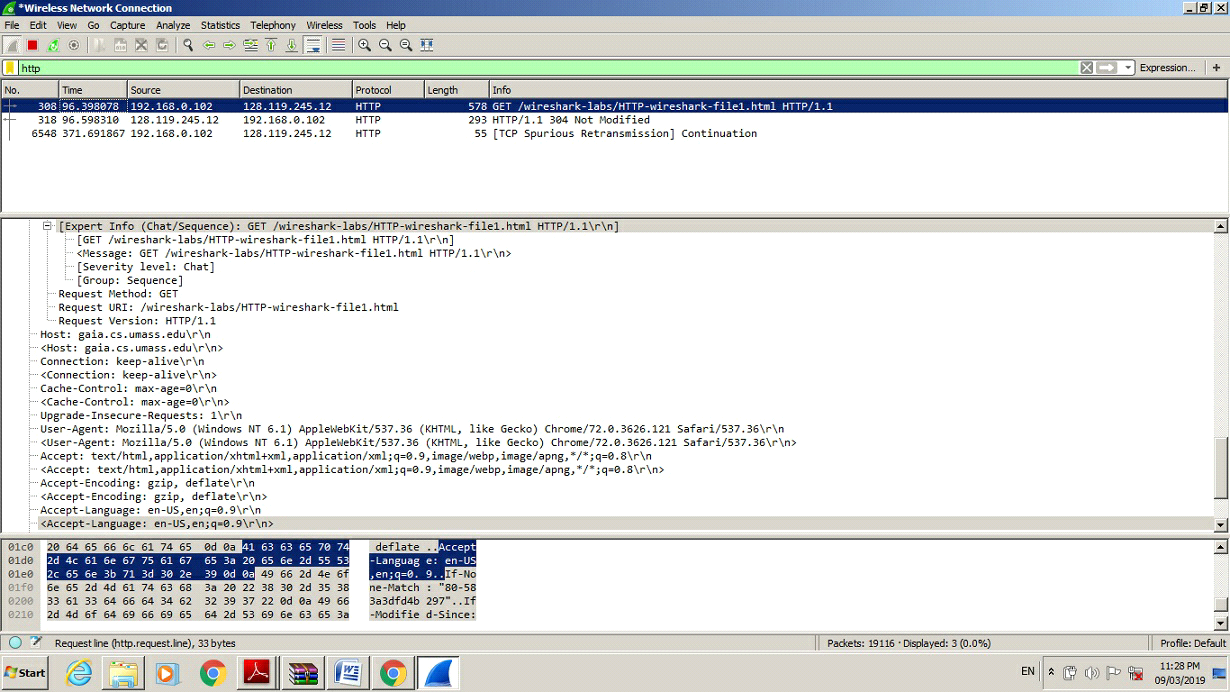
exception of some diagnostic tools like ping and trace route). ICMP for Internet Protocol version 4 (IPv4) is also known as ICMPv4. IPv6 has a similar protocol, ICMPv6. The Internet Control Message Protocol is part of the Internet Protocol Suite, as defined in RFC 792. ICMP messages

**4.IGMP:**

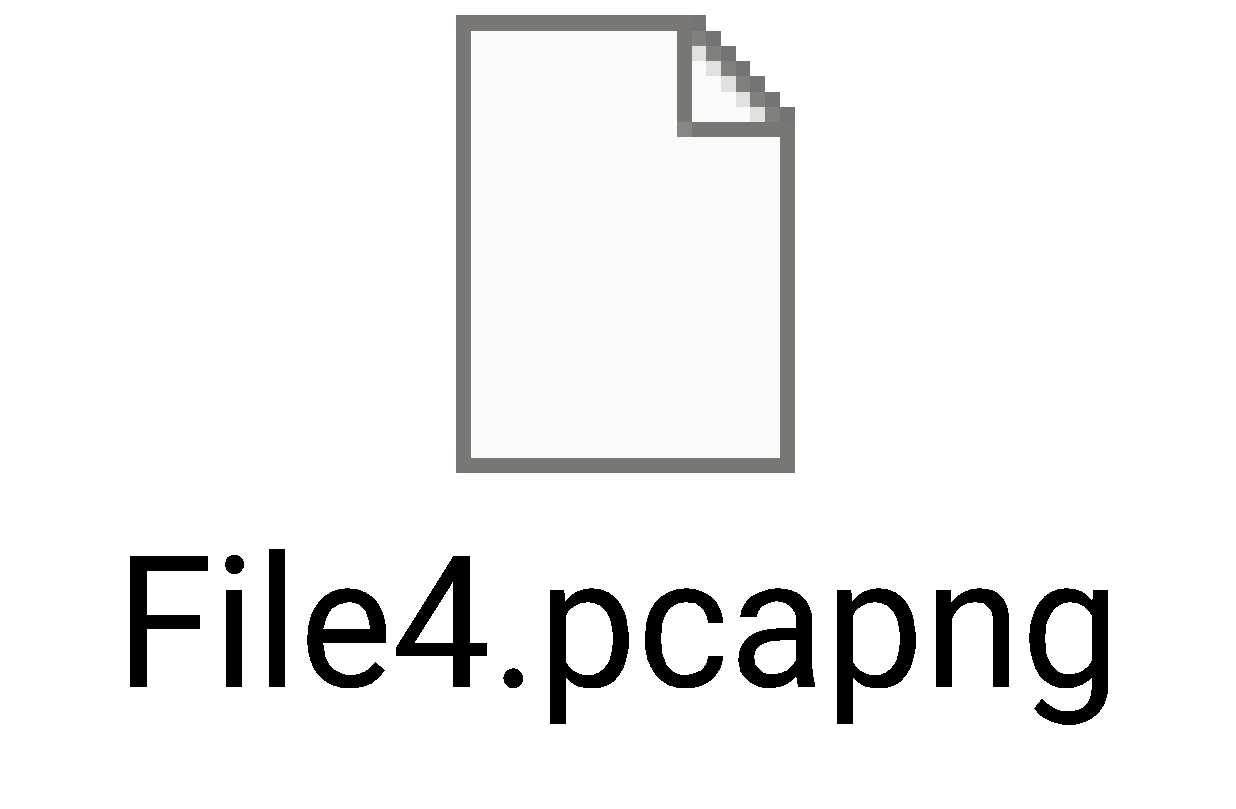
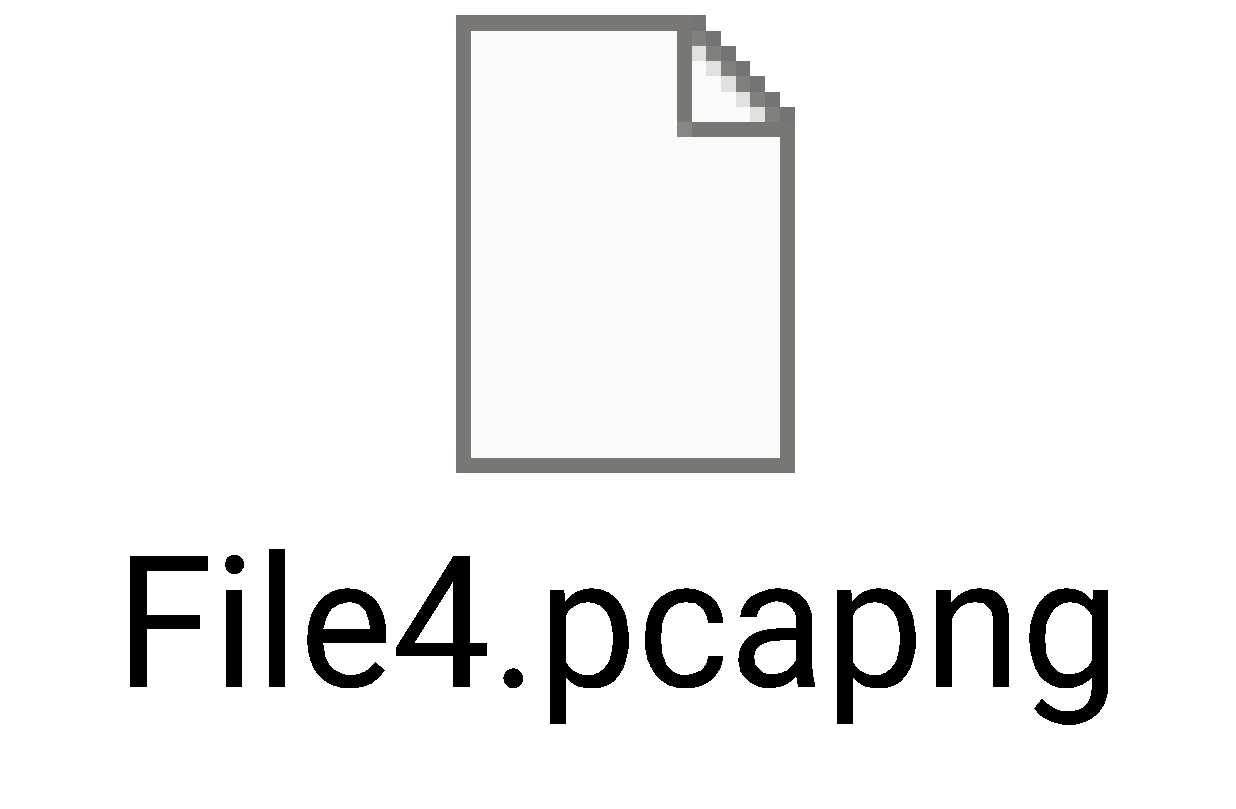
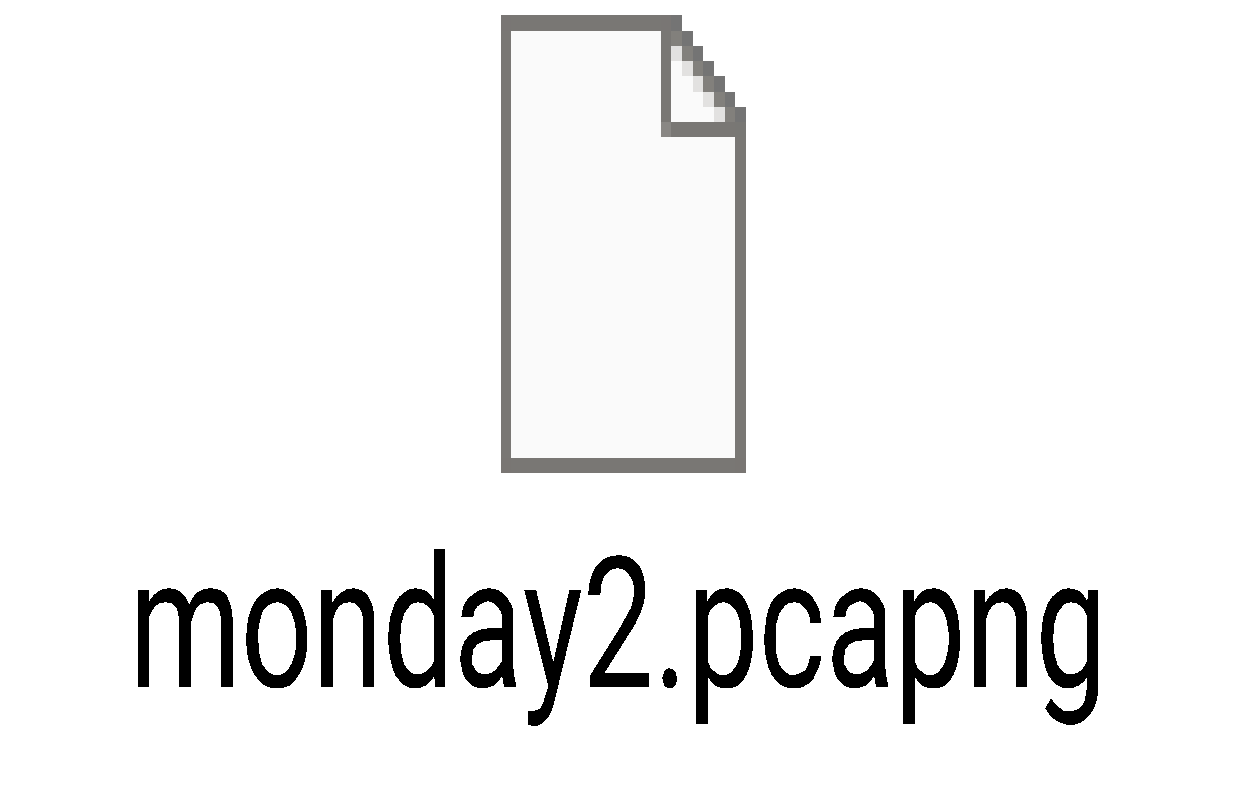
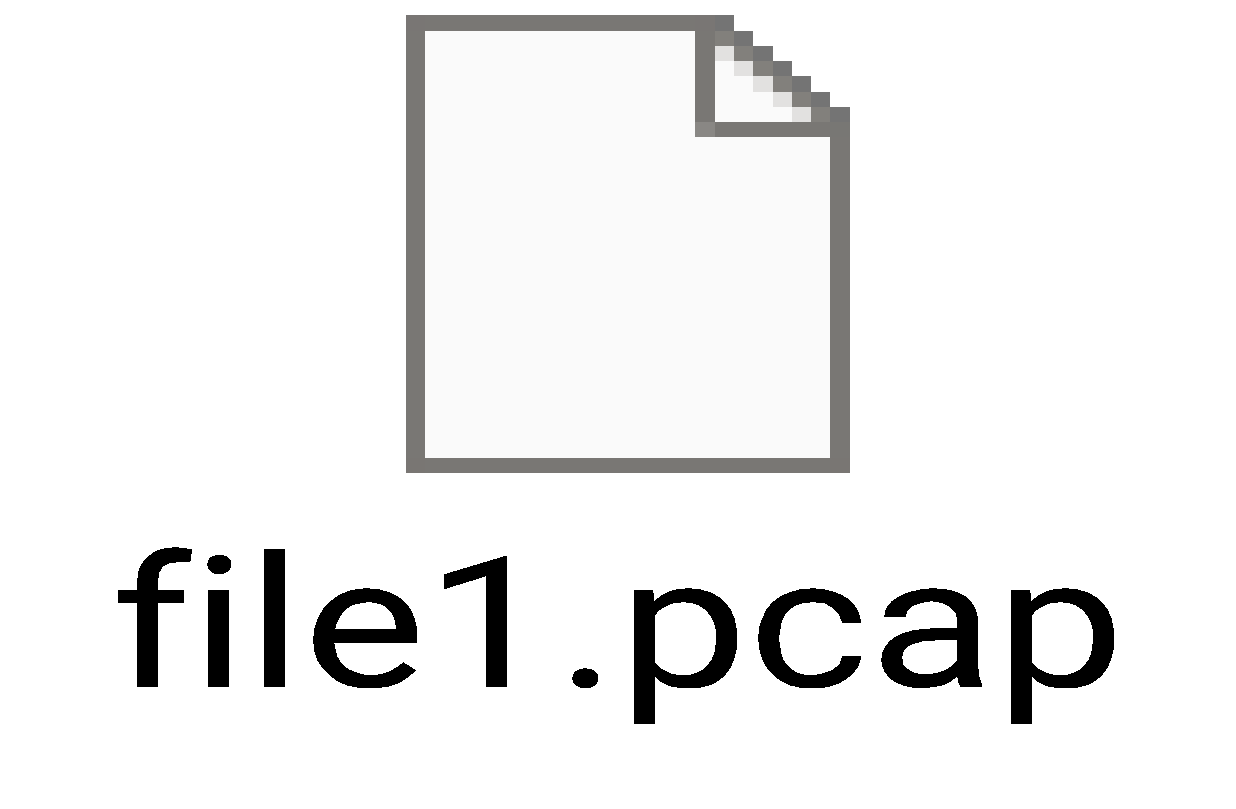
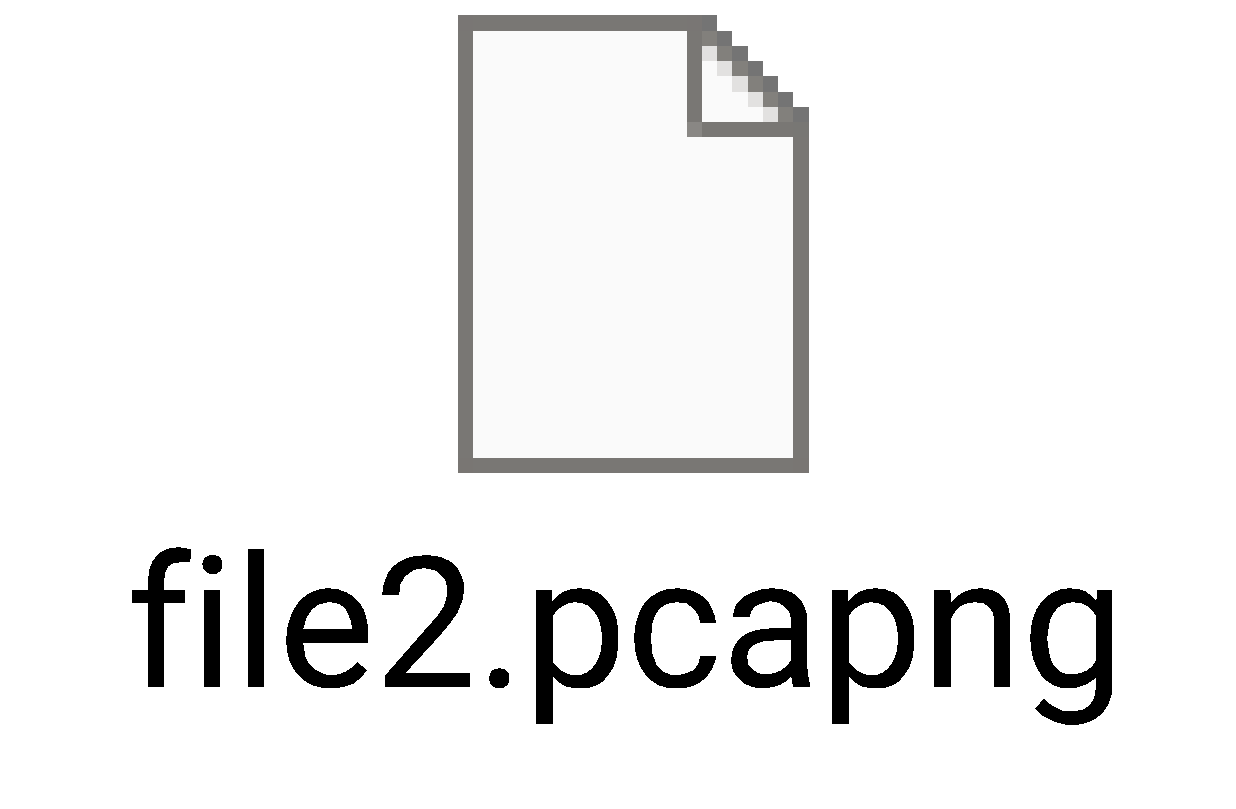
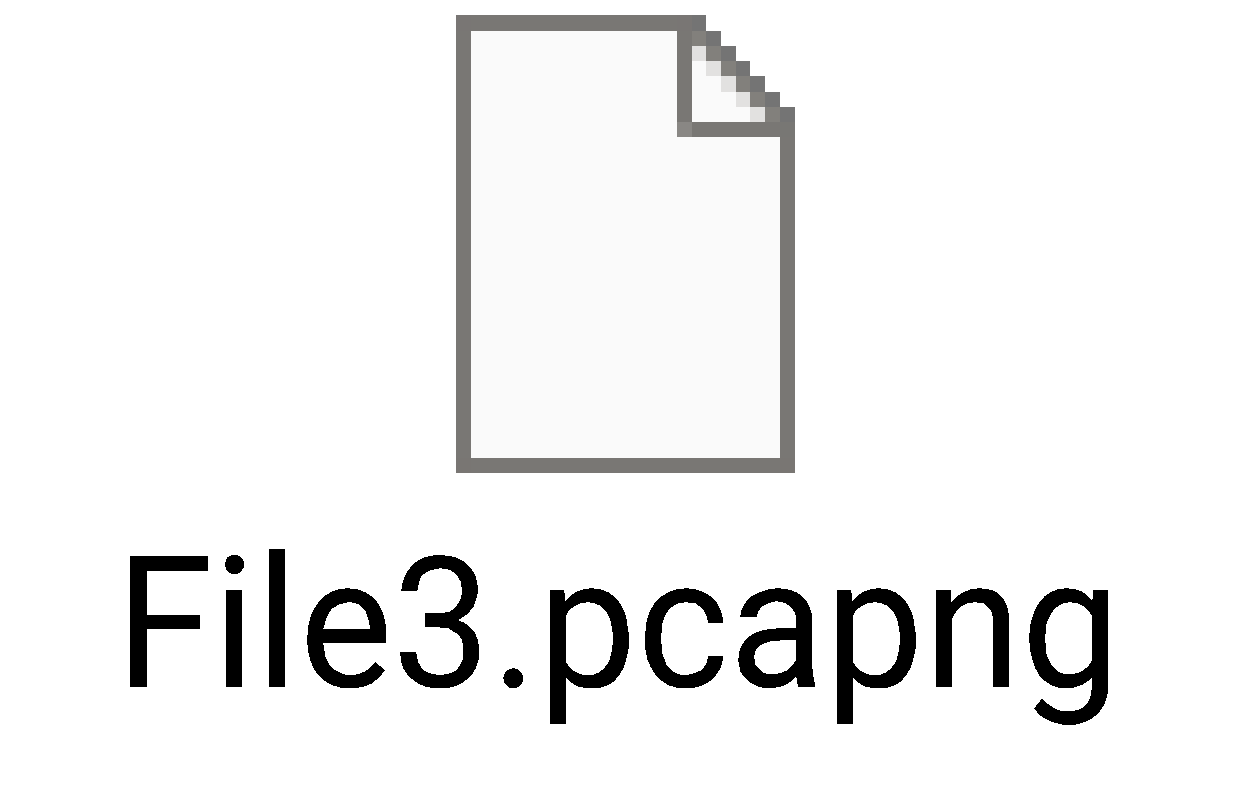
The Internet Group Management Protocol (IGMP) is a communications protocol used by hosts and adjacent routers on IP networks to establish multicast group memberships. IGMP is an integral part of IP multicast.IGMP can be used for one-to-many networking applications such as online streaming video and gaming, and allows more efficient use of resources when supporting these types of applications. IGMP messages are carried in bare IP packets with IP protocol. There is no transport layer used with IGMP messaging, similar to the Internet Control Message Protocol. Membership Queries are sent by multicast routers to determine which multicast addresses are of interest to systems attached to its network. Routers periodically send General Queries to refresh the group membership state for all systems on its network. Group-Specific Queries are used for determining the reception state for a particular multicast address.



**Output:**



**Conclusion:** We have successfully analysed the captured packets from Wireshark.

**Experiment No. 10**

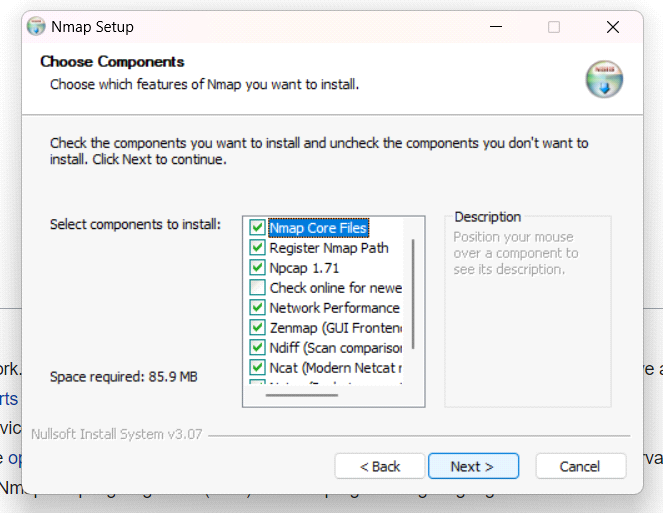
**Aim** : To install and implement NMAP.

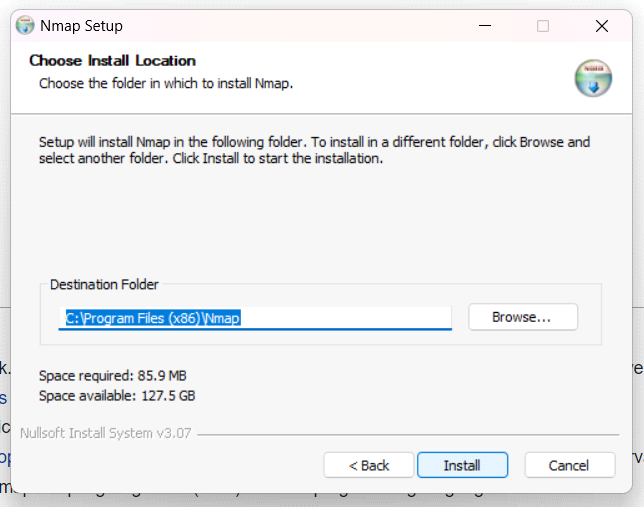
**NMAP** : Nmap (**Network Mapper**) is a network scanner created by Gordon Lyon. Nmap is used to discover hosts and services on a computer network by sending packets and analyzing the responses.

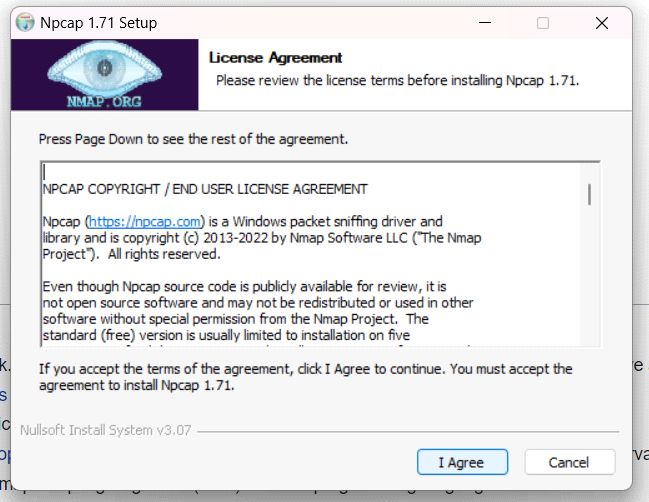
Nmap provides a number of features for probing computer networks, including host discovery and service and operating system detection. These features are extensible by scripts that provide more advanced service detection, vulnerability detection, and other features. Nmap can adapt to network conditions including latency and congestion during a scan.

**INSTALLATION OF NMAP :**

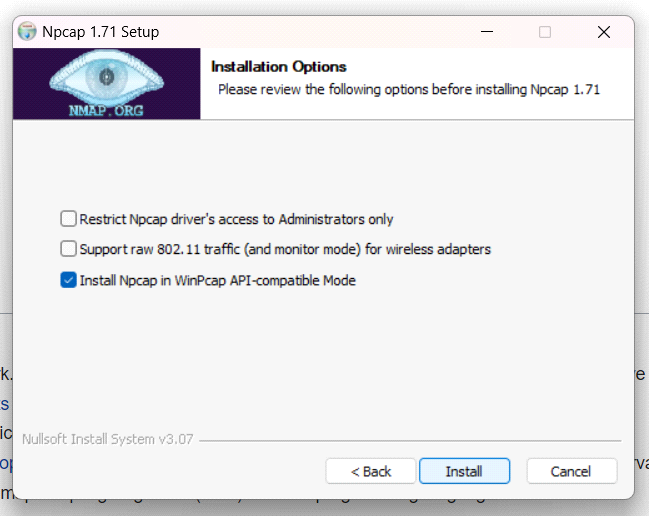
* Run the installer and click next :



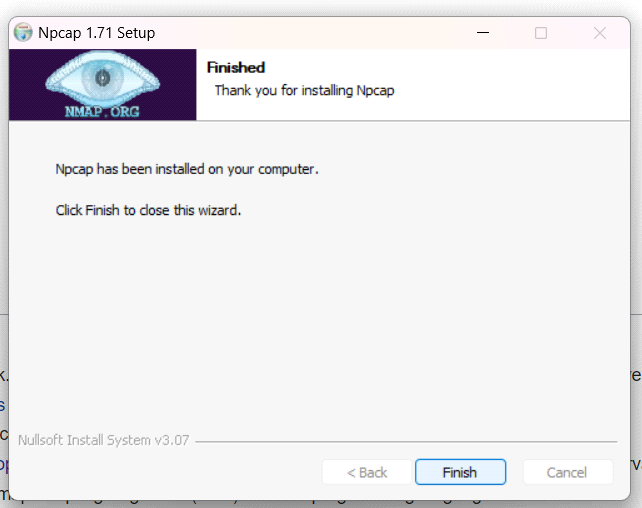
* Click on install :
* 
* **Click on I Agree** :



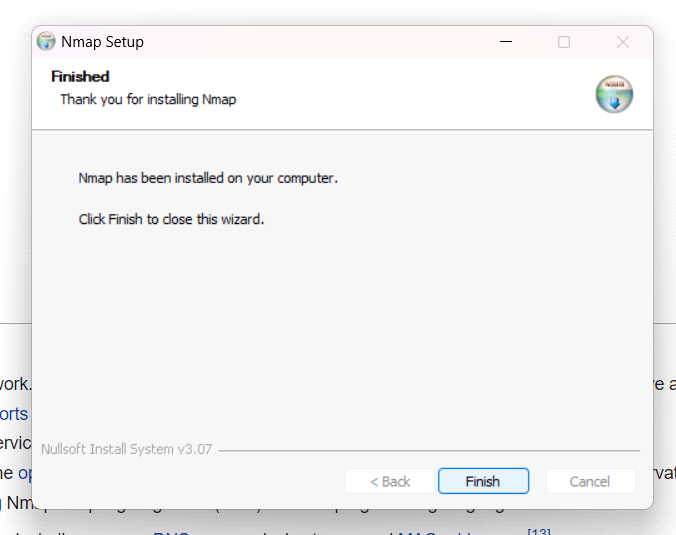
* **Click on Install :**



* Click on Finish :

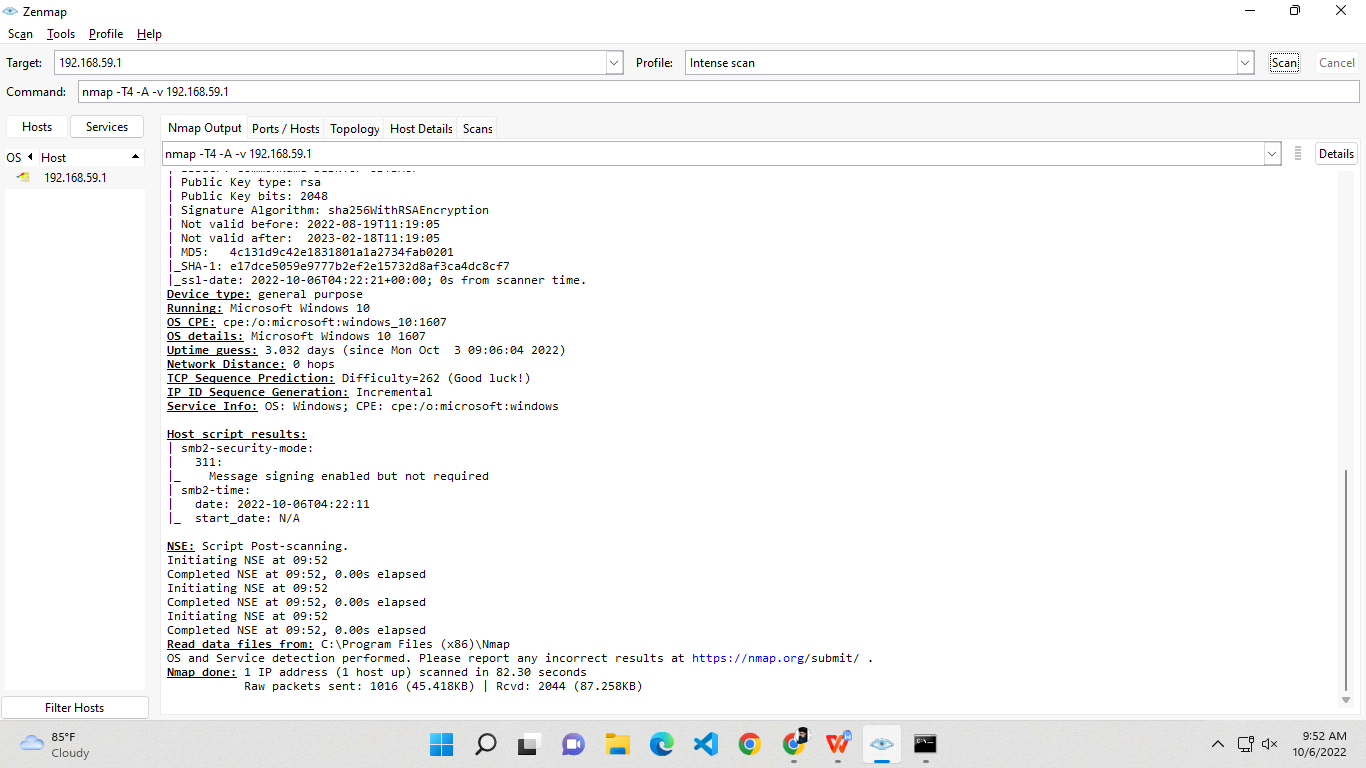


* **Click on Finish :**

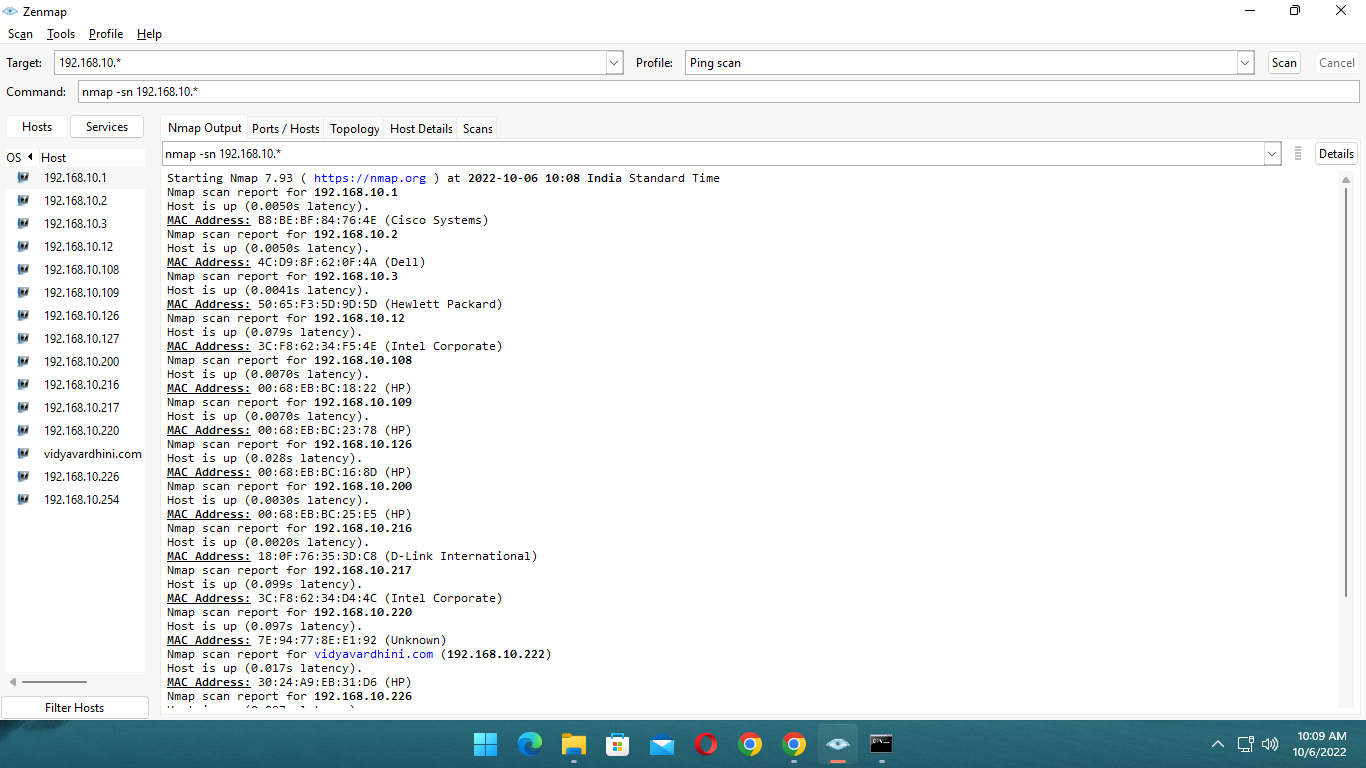


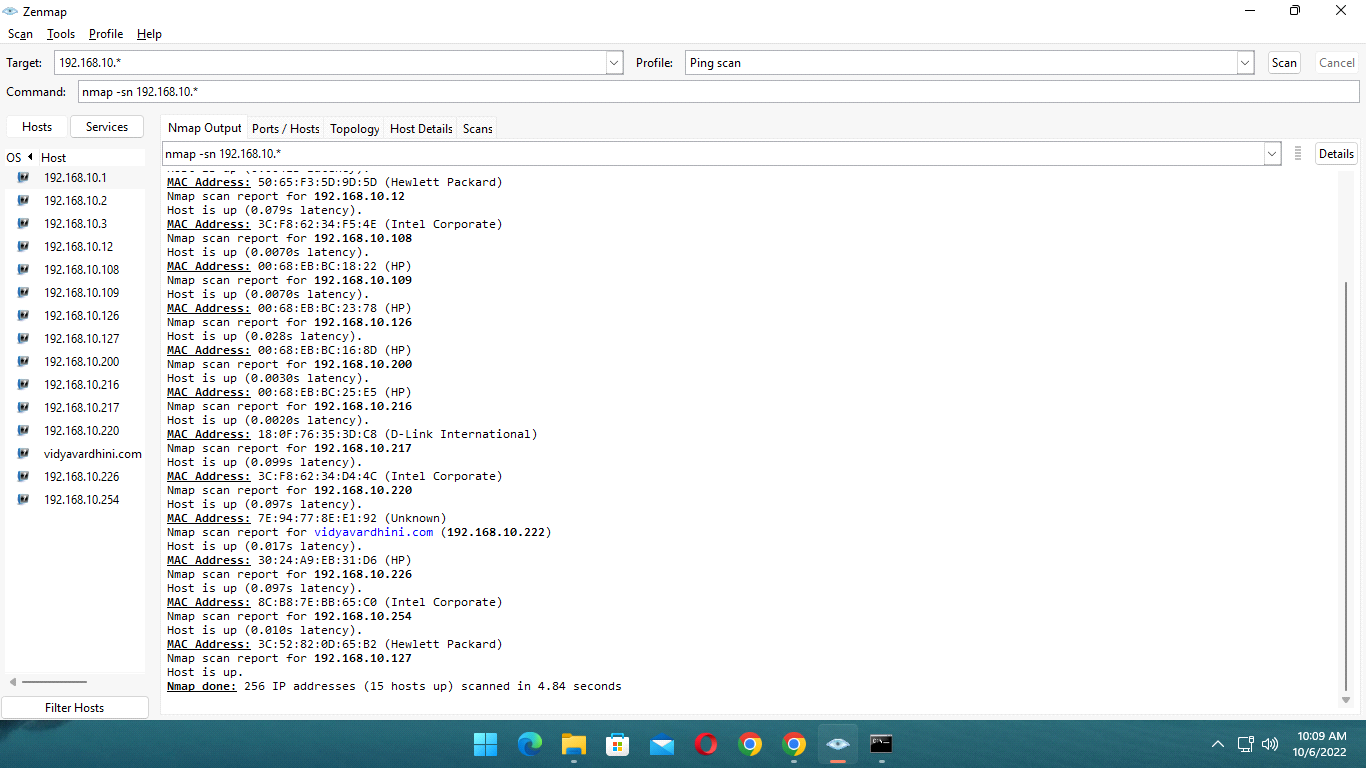
* **IMPLEMENTATION OF NMAP** :

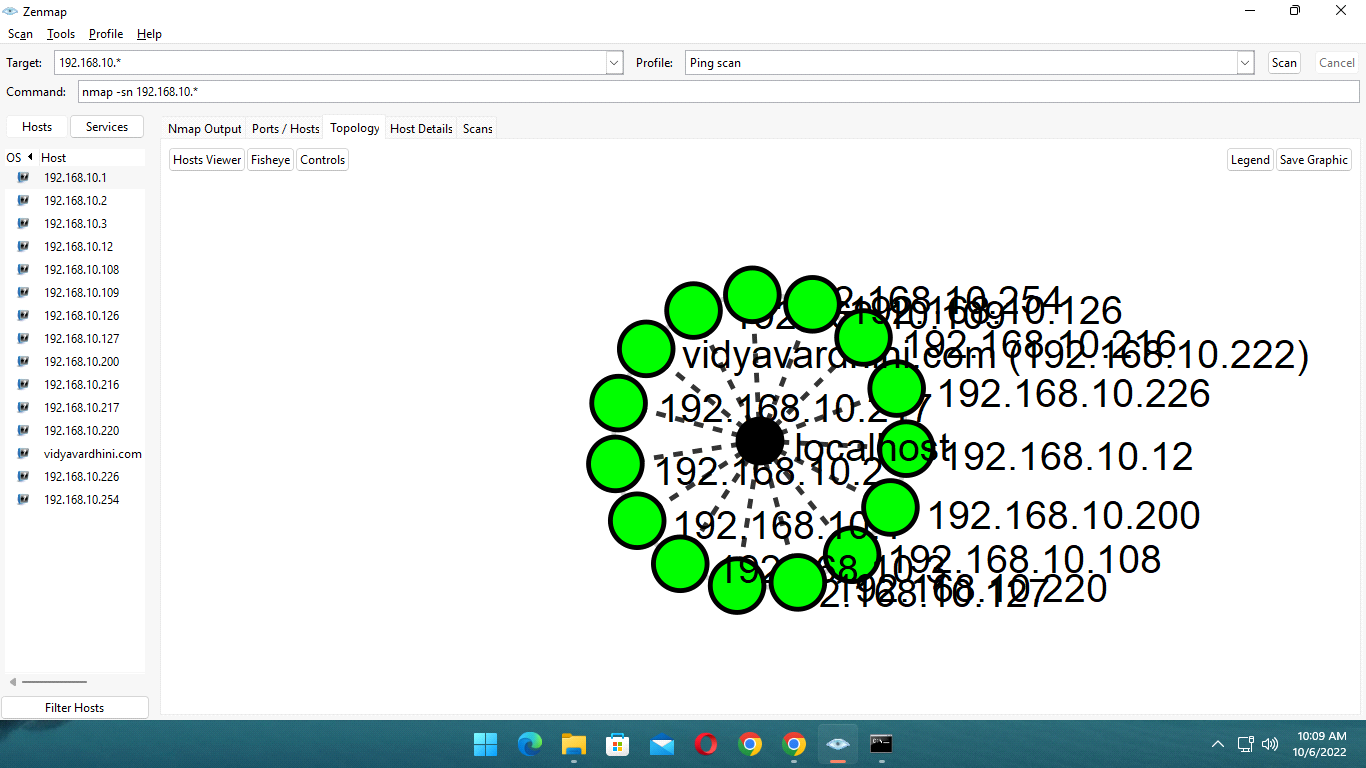
1. nmap-T4-A-v (ip address):



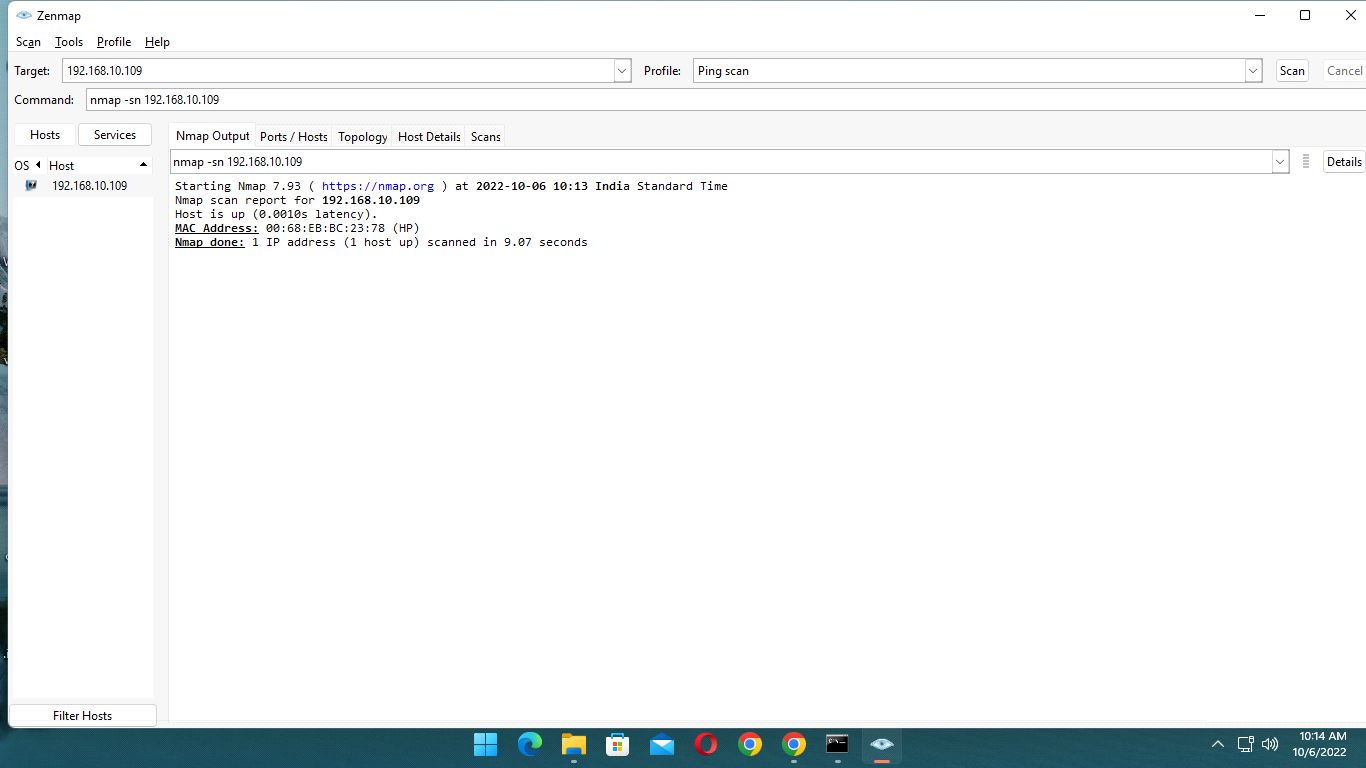
1. nmap -sn (ip address):

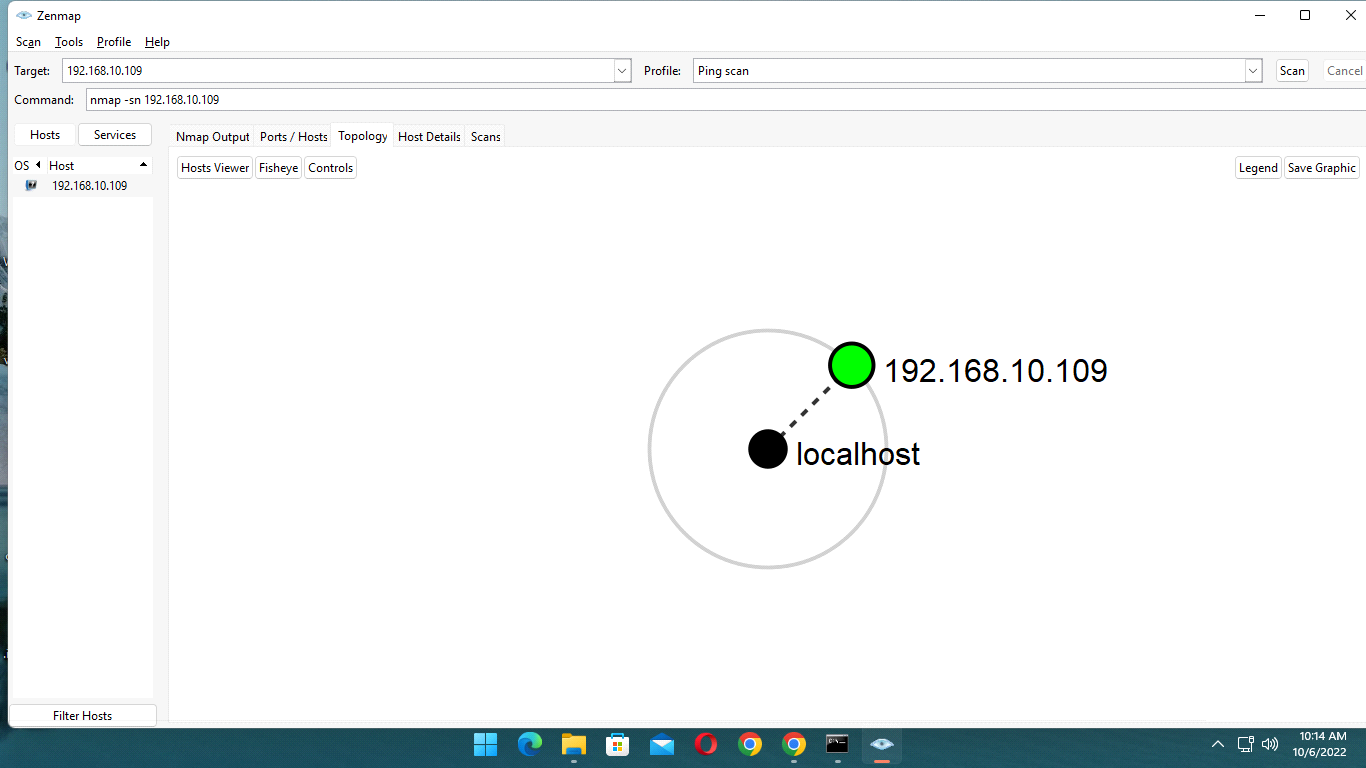






1. nmap -sn (reciever’s ip address) :





* **Conclusion :**

Nmap is clearly the “Swiss Army Knife” of networking, thanks to its inventory of versatile commands. It lets you quickly scan and discover essential information about your network, hosts, ports, firewalls, and operating systems. Nmap has numerous settings, flags, and preferences that help system administrators analyze a network in detail.

Q1.

Q2.