**Acropolis Institute of Technology And Research,**

**Indore (M.P.)**



**Network Security- CY503(B)**

**Name – Ashish Prajapat**

**Enrollment No. - 0827CY221016**

**Branch - CS(Cyber Security)**

**Semester- 5th (3rd year)**

**Submitted To - Submitted By-**

**Prof. Nidhi Nigam Ashish Prajapat**

**Index**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr.No.** | **Experiment** | **Date of Exp.** | **Date of sub.** | **Grade** |
| 1. | Write a program to convert plain text to cipher text using Caesar Cipher. | 03/09/24 | 08/10/24 |  |
| 2. | Write CTF report. | 04/09/24 | 08/10/24 |  |
| 3. | Study Of different Cryptographic tools. | 10/09/24 | 08/10/24 |  |
| 4. | Try Hack Me | 11/09/24 | 08/10/24 |  |
| 5. | Configure VPN in Windows 11. | 24/09/24 | 08/10/24 |  |
| 6. | To study of different software vulnerabilities | 08/10/24 | 09/10/24 |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Experiment - 1**

**Hill Cipher**

Hill Cipher is a polygraphic substitution cipher that uses linear algebra to encrypt and decrypt messages. It was invented by Lester Hill in 1929.

Hill Cipher Working:

Encryption:

1. Convert plaintext to numerical values (A=0, B=1, ..., Z=25).

2. Divide plaintext into blocks of size n (e.g., n=3).

3. Represent each block as a vector (column matrix).

4. Multiply each vector by a key matrix (n x n) to get ciphertext vector.

5. Convert ciphertext vector back to numerical values.

Decryption:

1. Convert ciphertext to numerical values.

2. Divide ciphertext into blocks of size n.

3. Represent each block as a vector.

4. Multiply each vector by the inverse of the key matrix.

5. Convert resulting vector back to plaintext.

Key Matrix:

- Must be square (n x n) and invertible.

- Typically, a random matrix with determinant ≠ 0.

**Program:-**

*#include <iostream>*

*using namespace std;*

*// Function to generate the key matrix*

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

*{*

*int k = 0;*

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

*{*

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

*{*

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

*k++;*

*}*

*}*

*}*

*// Function to encrypt the message*

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

*{*

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

*{*

*cipherMatrix[i][0] = 0;*

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

*{*

*cipherMatrix[i][0] += keyMatrix[i][j] \* messageVector[j][0];*

*}*

*cipherMatrix[i][0] = cipherMatrix[i][0] % 26;*

*}*

*}*

*// Function to implement Hill Cipher encryption*

*void HillCipher(string message, string key)*

*{*

*int keyMatrix[3][3];*

*getKeyMatrix(key, keyMatrix);*

*int messageVector[3][1];*

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

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

*int cipherMatrix[3][1];*

*encrypt(cipherMatrix, keyMatrix, messageVector);*

*string CipherText;*

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

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

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

*}*

*int main()*

*{*

*string message, key;*

*// Taking user input for the message and key*

*cout<< "Enter a 3-letter message (uppercase): ";*

*cin>> message;*

*cout<< "Enter a 9-letter key (uppercase): ";*

*cin>> key;*

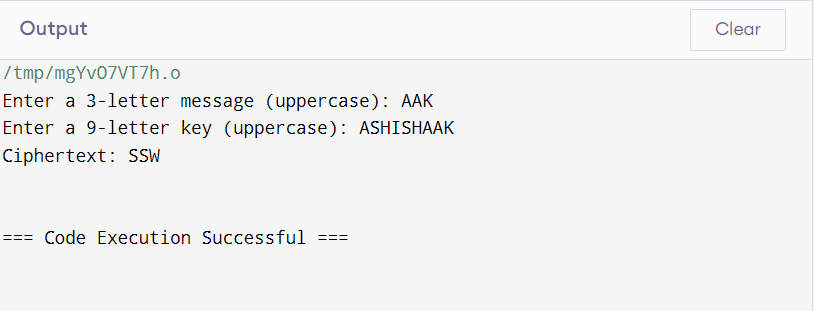
*// Perform encryption*

*HillCipher(message, key)*

*return 0;*

*}*

*OUTPUT:-*

**

***Caesar cipher***

***The Caesar Cipher is a type of substitution cipher where each letter in the plaintext is shifted by a fixed number of positions down the alphabet. It is one of the simplest and most well-known encryption techniques.***

***PROGRAM:-***

*#include<iostream>*

*#include<string.h>*

*using namespace std;*

*int main() {*

*cout<< "Enter the message:\n";*

*char msg[100];*

*cin.ignore(); // To ignore any newline character left in the buffer*

*cin.getline(msg, 100); // Take the message as input*

*int i, length, choice, key;*

*cout<< "Enter key: ";*

*cin>> key; // Take the key as input*

*length = strlen(msg);*

*cout<< "Enter your choice \n1. Encryption \n2. Decryption \n";*

*cin>> choice;*

*if (choice == 1) { // For encryption*

*char ch;*

*for (i = 0; msg[i] != '\0'; ++i) {*

*ch = msg[i];*

*// Encrypt for lowercase letter*

*if (ch>= 'a' &&ch<= 'z') {*

*ch = ch + key;*

*if (ch> 'z') {*

*ch = ch - 'z' + 'a' - 1;*

*}*

*msg[i] = ch;*

*}*

*// Encrypt for uppercase letter*

*else if (ch>= 'A' &&ch<= 'Z') {*

*ch = ch + key;*

*if (ch> 'Z') {*

*ch = ch - 'Z' + 'A' - 1;*

*}*

*msg[i] = ch;*

*}*

*}*

*cout<< "Encrypted message: " << msg <<endl;*

*}*

*else if (choice == 2) { // For decryption*

*char ch;*

*for (i = 0; msg[i] != '\0'; ++i) {*

*ch = msg[i];*

*// Decrypt for lowercase letter*

*if (ch>= 'a' &&ch<= 'z') {*

*ch = ch - key;*

*if (ch< 'a') {*

*ch = ch + 'z' - 'a' + 1;*

*}*

*msg[i] = ch;*

*}*

*// Decrypt for uppercase letter*

*else if (ch>= 'A' &&ch<= 'Z') {*

*ch = ch - key;*

*if (ch< 'A') {*

*ch = ch + 'Z' - 'A' + 1;*

*}*

*msg[i] = ch;*

*}*

*}*

*cout<< "Decrypted message: " << msg <<endl;*

*}*

*else {*

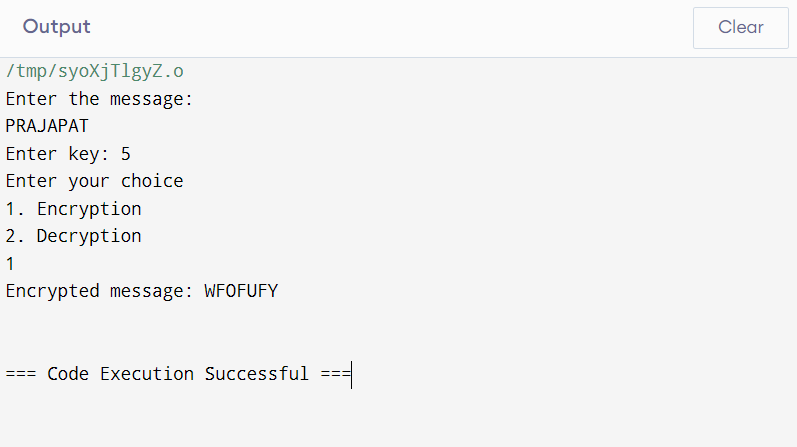
*cout<< "Invalid choice!" <<endl;*

*}*

*return 0;*

*}*

*Output:-*

**

***Playfair***

***Playfair cipher is a polygraphic substitution cipher that uses a 5x5 grid to encrypt and decrypt messages. It was invented by Charles Wheatstone in 1854 and named after Lord Playfair, who promoted it.***

***Program:-***

*#include <iostream>*

*#include <string>*

*using namespace std;*

*class playfair {*

*public:*

*string msg;*

*char n[5][5];*

*void play(string k, string t, bool m, bool e) {*

*createEncoder(k, m);*

*getText(t, m, e);*

*if (e)*

*applyPlayfair(1); // Encryption*

*else*

*applyPlayfair(-1); // Decryption*

*print();*

*}*

*private:*

*void applyPlayfair(int dir) {*

*int j, k, p, q;*

*string nmsg;*

*for (string::const\_iterator it = msg.begin(); it != msg.end(); ++it) {*

*if (getPos(\*it++, j, k)) {*

*if (getPos(\*it, p, q)) {*

*// For same row*

*if (j == p) {*

*nmsg += getChar(j, k + dir);*

*nmsg += getChar(p, q + dir);*

*}*

*// For same column*

*else if (k == q) {*

*nmsg += getChar(j + dir, k);*

*nmsg += getChar(p + dir, q);*

*}*

*// Form rectangle*

*else {*

*nmsg += getChar(j, q);*

*nmsg += getChar(p, k);*

*}*

*}*

*}*

*}*

*msg = nmsg;*

*}*

*void print() { // Print the solution*

*cout<< "\n\nSolution: " <<endl;*

*for (size\_ti = 0; i<msg.length(); i += 2) {*

*cout<< msg[i] <<msg[i + 1] << " ";*

*if ((i / 2 + 1) % 13 == 0) // Break after every 13 pairs*

*cout<<endl;*

*}*

*cout<<endl<<endl;*

*}*

*char getChar(int a, int b) { // Get the characters from the table*

*return n[(a + 5) % 5][(b + 5) % 5];*

*}*

*bool getPos(char l, int& c, int& d) { // Get the position of the character*

*for (int y = 0; y < 5; ++y)*

*for (int x = 0; x < 5; ++x)*

*if (n[y][x] == l) {*

*c = y;*

*d = x;*

*return true;*

*}*

*return false;*

*}*

*void getText(string t, bool m, bool e) { // Prepare the message*

*msg.clear();*

*for (string::iterator it = t.begin(); it != t.end(); ++it) {*

*\*it = toupper(\*it);*

*if (\*it < 'A' || \*it > 'Z')*

*continue;*

*if (\*it == 'J' && m)*

*\*it = 'I';*

*msg += \*it;*

*}*

*if (e) {*

*string nmsg;*

*size\_t len = msg.length();*

*for (size\_t x = 0; x <len; x += 2) {*

*nmsg += msg[x];*

*if (x + 1 <len) {*

*if (msg[x] == msg[x + 1])*

*nmsg += 'X';*

*nmsg += msg[x + 1];*

*}*

*}*

*msg = nmsg;*

*}*

*if (msg.length() % 2 != 0) // Add 'X' if message length is odd*

*msg += 'X';*

*}*

*void createEncoder(string key, bool m) { // Create the 5x5 table*

*key += "ABCDEFGHIJKLMNOPQRSTUVWXYZ";*

*string s = "";*

*for (string::iterator it = key.begin(); it != key.end(); ++it) {*

*\*it = toupper(\*it);*

*if (\*it < 'A' || \*it > 'Z')*

*continue;*

*if ((\*it == 'J' && m) || (s.find(\*it) != string::npos))*

*continue;*

*s += \*it;*

*}*

*copy(s.begin(), s.end(), &n[0][0]); // Copy characters to the 5x5 matrix*

*}*

*};*

*int main() {*

*string k, i, msg;*

*bool m, c;*

*cout<< "Encrypt or Decrypt? ";*

*getline(cin, i);*

*c = (i[0] == 'e' || i[0] == 'E');*

*cout<< "Enter a key: ";*

*getline(cin, k);*

*cout<< "I <-> J (Y/N): ";*

*getline(cin, i);*

*m = (i[0] == 'y' || i[0] == 'Y');*

*cout<< "Enter the message: ";*

*getline(cin, msg);*

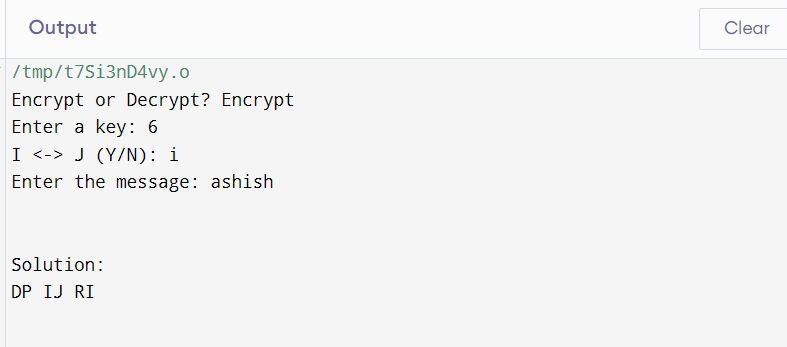
*playfair pf;*

*pf.play(k, msg, m, c);*

*return 0;*

*}*

*Output:-*

**

***Vigenere***

***Vigenère cipher is a polyalphabetic substitution cipher that uses a keyword to encrypt and decrypt messages. It is considered one of the most secure hand ciphers.***

***Program:-***

*#include <iostream>*

*#include <string>*

*using namespace std;*

*class Vig {*

*public:*

*string k;*

*// Constructor to set the key in uppercase*

*Vig(string k) {*

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

*if (k[i] >= 'A' && k[i] <= 'Z')*

*this->k += k[i];*

*else if (k[i] >= 'a' && k[i] <= 'z')*

*this->k += k[i] + 'A' - 'a'; // Convert to uppercase*

*}*

*}*

*// Encryption function*

*string encryption(string t) {*

*string output;*

*for (int i = 0, j = 0; i<t.length(); ++i) {*

*char c = t[i];*

*// Convert lowercase letters to uppercase*

*if (c >= 'a' && c <= 'z')*

*c += 'A' - 'a';*

*else if (c < 'A' || c > 'Z') // Ignore non-alphabet characters*

*continue;*

*// Encrypt character*

*output += (c + k[j] - 2 \* 'A') % 26 + 'A'; // Shift character and adjust ASCII*

*j = (j + 1) % k.length();*

*}*

*return output;*

*}*

*// Decryption function*

*string decryption(string t) {*

*string output;*

*for (int i = 0, j = 0; i<t.length(); ++i) {*

*char c = t[i];*

*// Convert lowercase letters to uppercase*

*if (c >= 'a' && c <= 'z')*

*c += 'A' - 'a';*

*else if (c < 'A' || c > 'Z') // Ignore non-alphabet characters*

*continue;*

*// Decrypt character*

*output += (c - k[j] + 26) % 26 + 'A'; // Reverse shift and adjust ASCII*

*j = (j + 1) % k.length();*

*}*

*return output;*

*}*

*};*

*int main() {*

*// Initialize the cipher with the key*

*Vig v("WELCOME");*

*// Original message*

*string ori = "Meet me after attack";*

*// Encryption and decryption*

*string encrypt = v.encryption(ori);*

*string decrypt = v.decryption(encrypt);*

*// Display the results*

*cout<< "Original Message: " <<ori<<endl;*

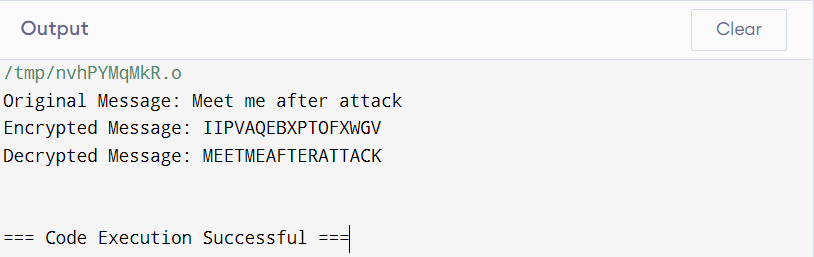
*cout<< "Encrypted Message: " << encrypt <<endl;*

*cout<< "Decrypted Message: " << decrypt <<endl;*

*return 0;*

*}*

*Output:-*

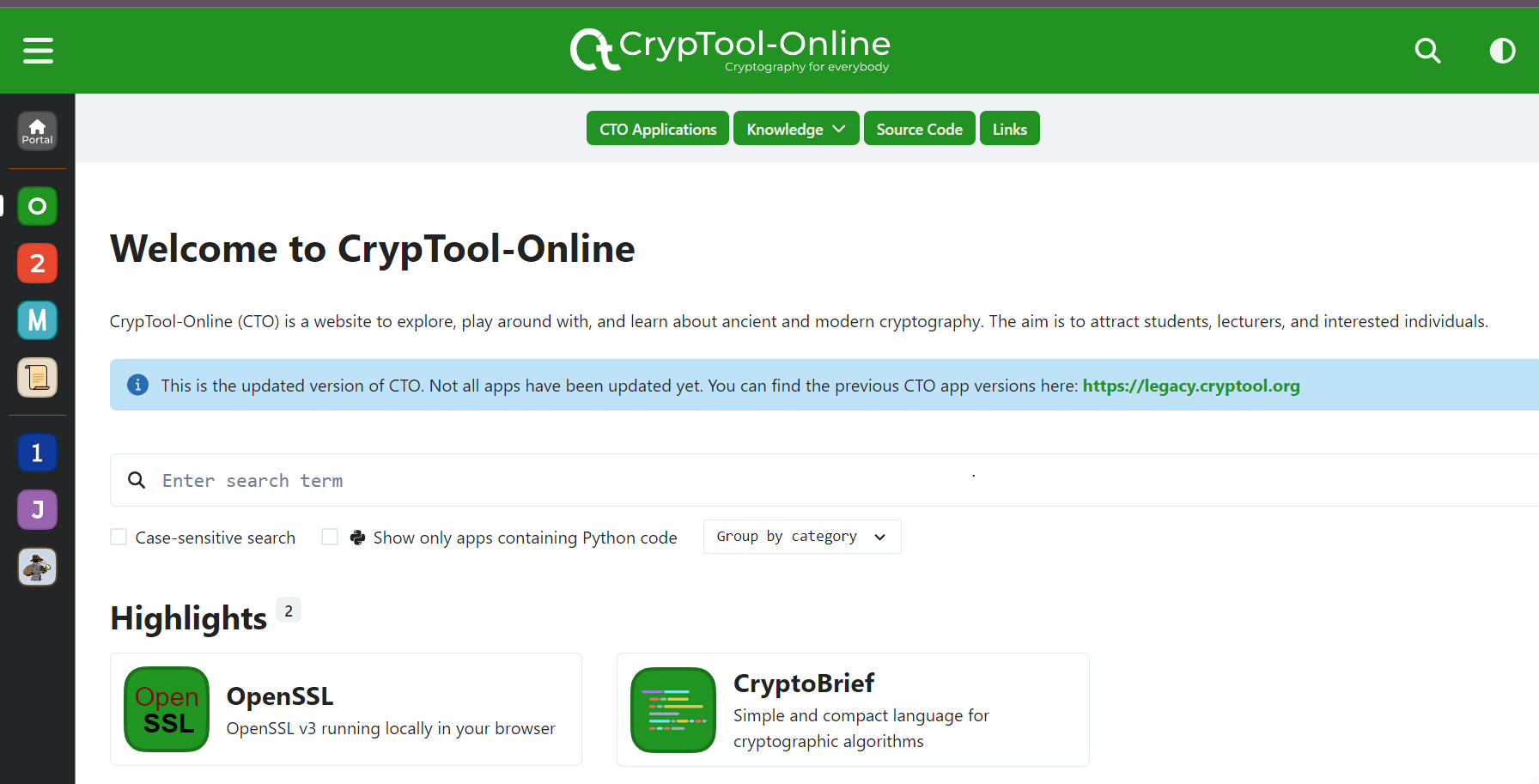


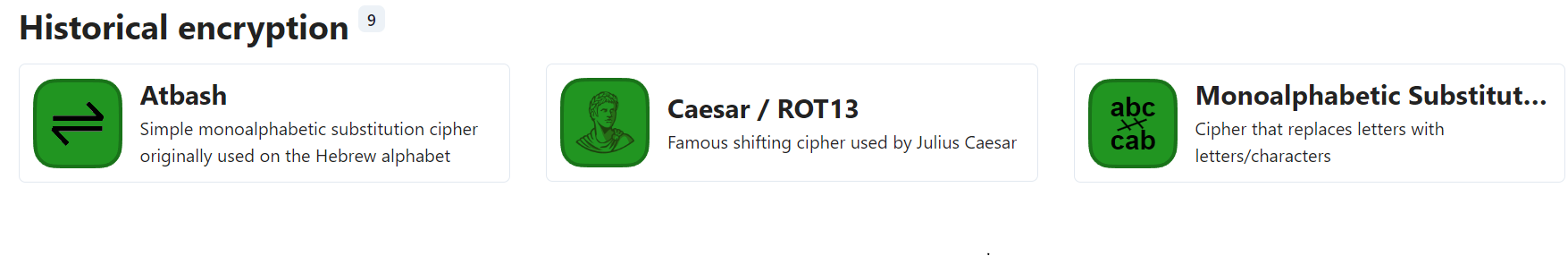
**Experiment -2**

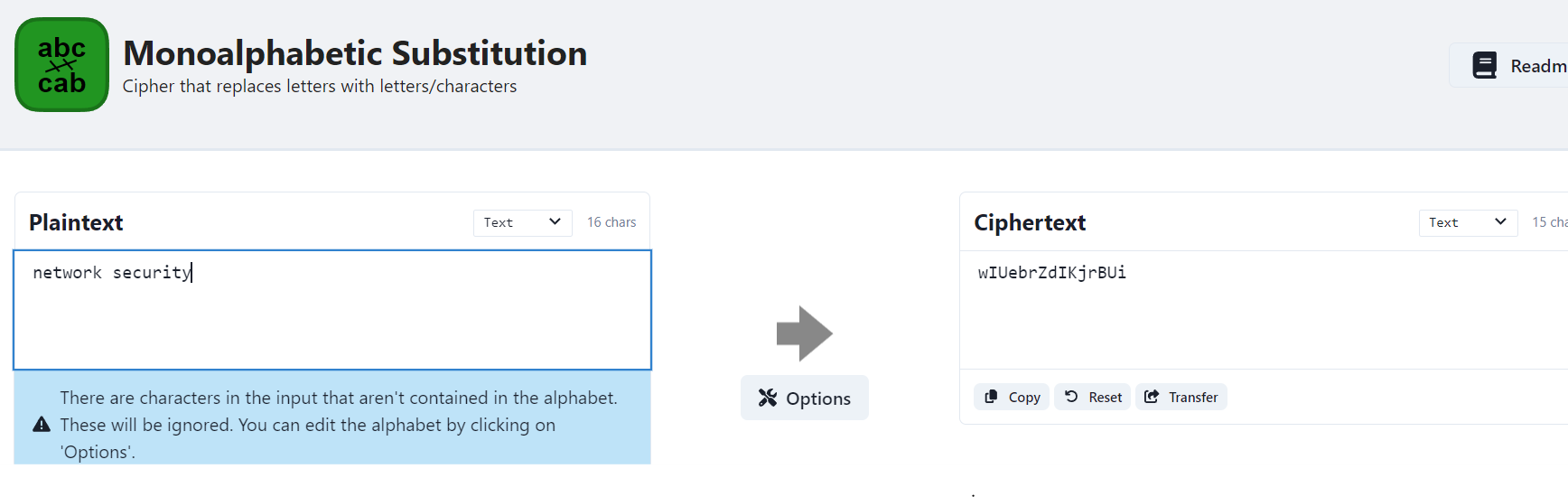
**Study of cryptography tools**

**1. CrypTool**

* **Description**: An interactive e-learning platform that allows users to learn about various cryptographic methods, including classical encryption techniques like Caesar cipher, Vigenère cipher, and more.





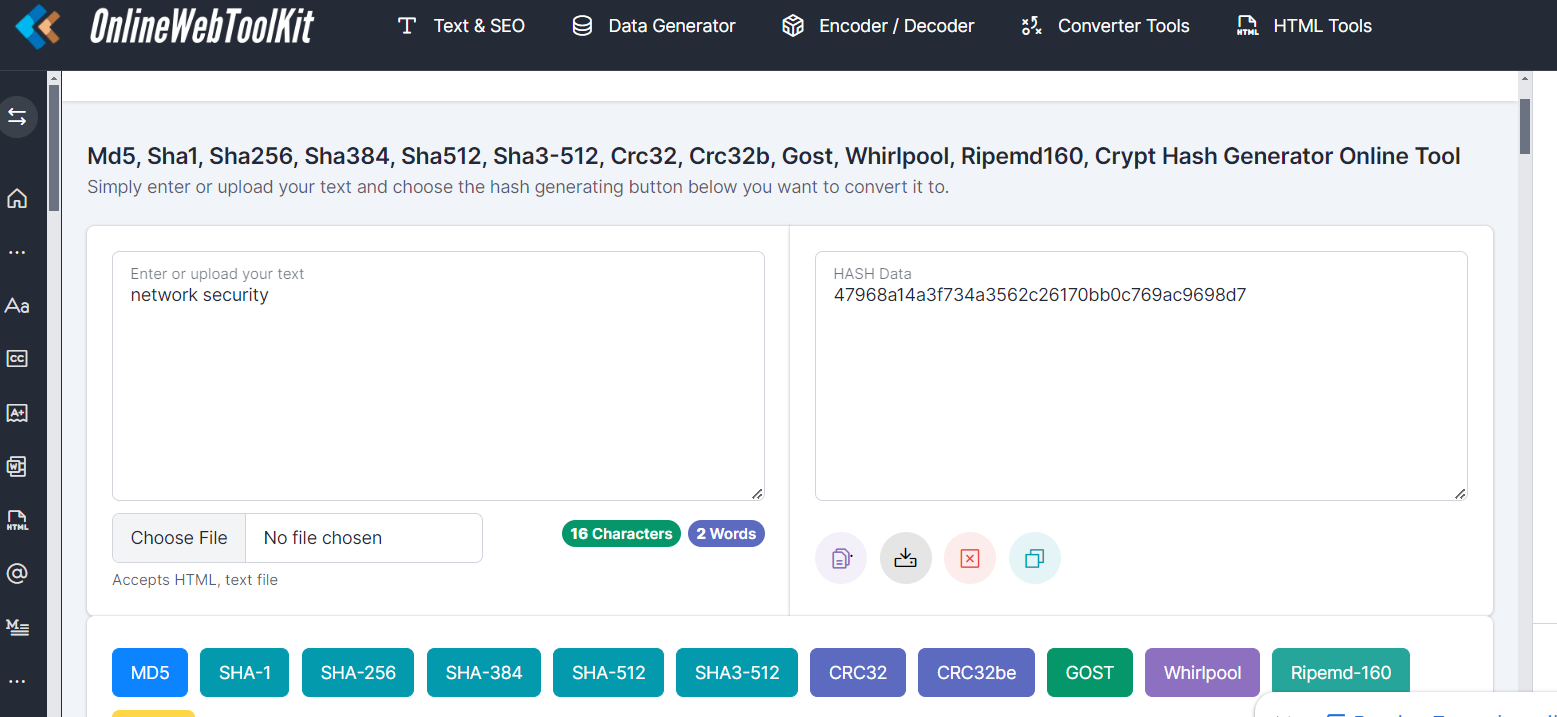


**Hash Generator( OnlineWebToolKit)**

A Hash Generator is an online tool that allows users to create hash values from input text using various hashing algorithms. Hashing is a cryptographic process that transforms input data (like passwords or files) into a fixed-size string of characters, which typically appears random. Hashes are often used for data integrity verification, secure password storage, and digital signatures.

**Key Features of Hash Generators**

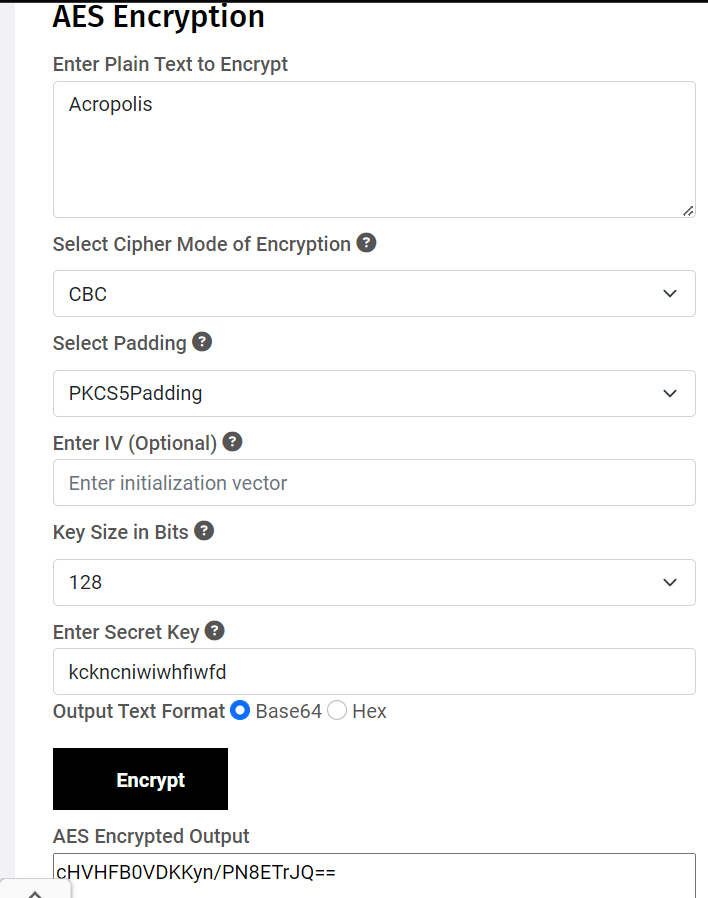
1. **Multiple Hashing Algorithms**:
   * Most hash generators support various algorithms, allowing users to choose the one that best fits their needs. Common algorithms include:
     + **MD5**: Produces a 128-bit hash value, typically represented as a 32-character hexadecimal number. While fast, MD5 is not considered secure for cryptographic purposes due to vulnerabilities.
     + **SHA-1**: Generates a 160-bit hash value and is more secure than MD5, but has also been found to have vulnerabilities over time.
     + **SHA-256**: Part of the SHA-2 family, it creates a 256-bit hash value and is widely used in security applications and protocols, including SSL and TLS.
     + **SHA-512**: Similar to SHA-256 but produces a 512-bit hash value, offering greater security.
     + **bcrypt**: Specifically designed for hashing passwords with a salt to protect against rainbow table attacks.



**Online AES Encryption Tool (only cript)**

**Description**: The Online AES Encryption Tool is a web-based application that enables users to securely encrypt and decrypt messages using the AES (Advanced Encryption Standard) algorithm. This tool supports various key lengths (128, 192, and 256 bits) and different operational modes (like CBC, ECB, CFB, OFB, and CTR), allowing users to customize their encryption process according to their security needs.

**Key Features:**

* **AES Encryption and Decryption**: Easily encrypt plaintext into ciphertext and decrypt ciphertext back to plaintext using AES.
* **Support for Different Key Lengths**: Users can choose between 128, 192, or 256-bit keys, enhancing security based on the required level.
* **Multiple Modes of Operation**: Select from various modes of operation:
  + **ECB (Electronic Codebook)**: Simplest mode, but less secure for identical plaintext blocks.
  + **CBC (Cipher Block Chaining)**: More secure than ECB, using an initialization vector (IV) for added randomness.
  + **CFB (Cipher Feedback)**: Allows encryption of data in smaller increments.
  + **OFB (Output Feedback)**: Similar to CFB, but the ciphertext is fed back for the next encryption block.
  + **CTR (Counter)**: Converts block cipher into a stream cipher, allowing for high-speed encryption.
* **User-Friendly Interface**: Simple and intuitive web interface that allows users to input plaintext, select options, and receive encrypted output quickly.
* **Base64 Encoding/Decoding**: Automatically encodes or decodes the output in Base64 format, making it easier to handle binary data in text format.
* **No Installation Required**: As a web-based tool, it requires no downloads or installations, allowing users to access it from any device with an internet connection.
* 

**CodeChef**

**What**

A simple, intuitive web app for analysing and decoding data without having to deal with complex tools or programming languages. CyberChef encourages both technical and non-technical people to explore data formats, encryption and compression.

**Why**

Digital data comes in all shapes, sizes and formats in the modern world – CyberChef helps to make sense of this data all on one easy-to-use platform.

**How**

The interface is designed with simplicity at its heart. Complex techniques are now as trivial as drag-and-drop. Simple functions can be combined to build up a "recipe", potentially resulting in complex analysis, which can be shared with other users and used with their input.

For those comfortable writing code, CyberChef is a quick and efficient way to prototype solutions to a problem which can then be scripted once proven to work.

**Who**

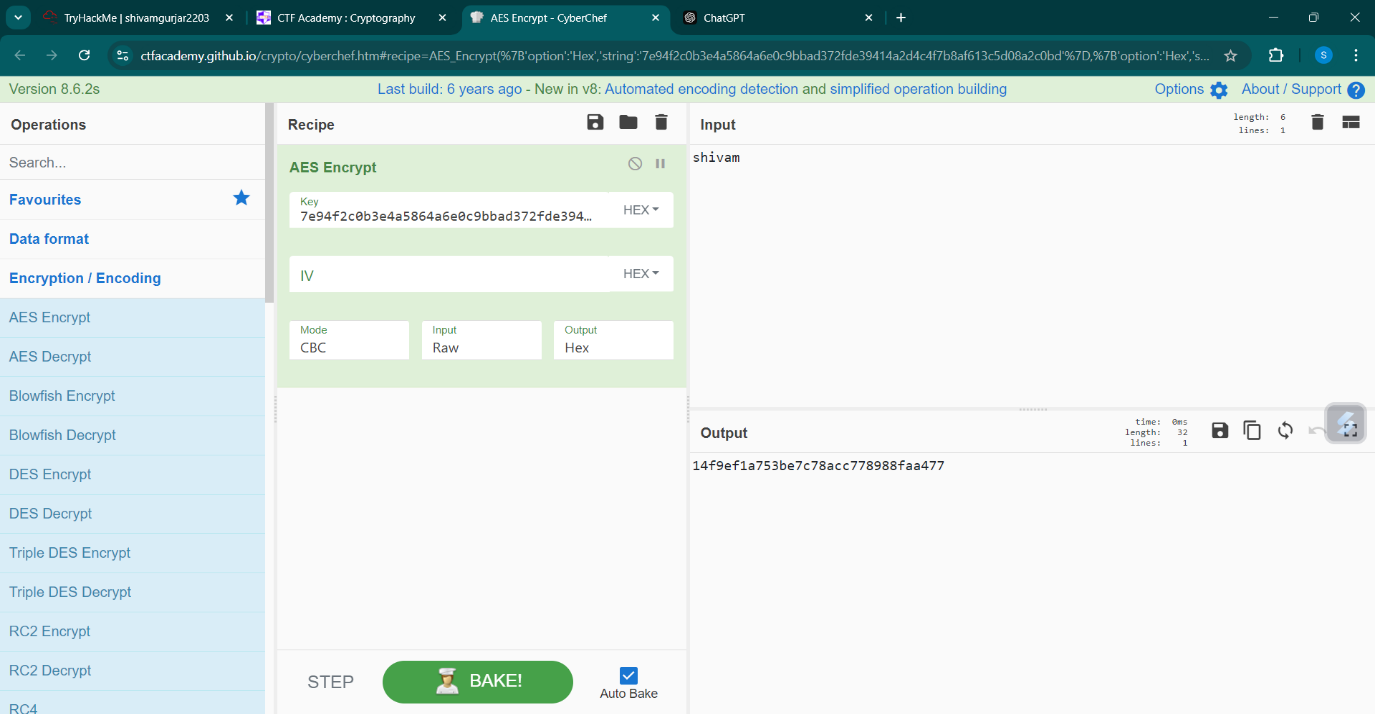
It is expected that CyberChef will be useful for cybersecurity and antivirus companies. It should also appeal to the academic world and any individuals or companies involved in the analysis of digital data, be that software developers, analysts, mathematicians or casual puzzle solvers.

**Aim**

It is hoped that by releasing CyberChef through [GitHub](https://github.com/gchq/CyberChef), contributions can be added which can be rolled out into future versions of the tool.

There are around 200 useful operations in CyberChef for anyone working on anything vaguely Internet-related, whether you just want to convert a timestamp to a different format, decompress gzipped data, create a SHA3 hash, or parse an X.509 certificate to find out who issued it.

It’s the Cyber Swiss Army Knife.



**Experiment-3**

**CTF Challenge Report**

**Challenge Overview:**

The first challenge in this CTF focused on basic cryptography, where we were tasked with deciphering an encoded message to retrieve the hidden flag. The encoded message was scrambled using a cipher, and based on the challenge description and hints, it was highly suggestive that a Caesar cipher or its variant, ROT13, was used to encode the message. The goal was to determine the number of shifts in the cipher and then decode the message to retrieve the flag.

**Step-by-Step Breakdown:**

1. **Challenge Objective:** The problem statement provided the encoded message:

Code:-

Gur frpergcnffjbeqvfuvqqrahaqre gur oevqtr. Gur syntvfpgsn{pvcure\_qrpvcure}

The challenge hinted that the message had been encoded using a cipher. Our task was to:

* + Identify the type of cipher used.
  + Determine how many shifts were applied in the encoded message.
  + Decode the message and retrieve the flag.

1. **Decoding the Cipher:** The hint provided in the challenge description suggested the use of **CyberChef**, an online tool designed for encoding and decoding various ciphers. Based on the structure of the encoded text and the common use of Caesar ciphers in introductory challenges, I suspected that the **ROT13 cipher** had been used.
   * **ROT13 Cipher:** This is a special case of the Caesar cipher, where each letter in the message is shifted by 13 positions in the alphabet. It’s a simple but effective method for obscuring text.
   * Since ROT13 is its own inverse (applying the same transformation twice restores the original message), decoding it is as simple as encoding the text again with a 13-letter shift.
2. **Using CyberChef:** To validate this hypothesis, I followed the steps below:
   * Navigated to **CyberChef** and entered the provided encoded message.
   * Applied the **ROT13 decryption** operation to the message.
   * The decrypted message was:

csharp

Copy code

The secret password is hidden under the bridge. The flag is ctfa{cipher\_decipher}

1. **Analyzing the Decoded Message:** From the decoded message, it became evident that the challenge was using a ROT13 encoding scheme. The message provided both the location of the secret password ("hidden under the bridge") and the flag for submission:

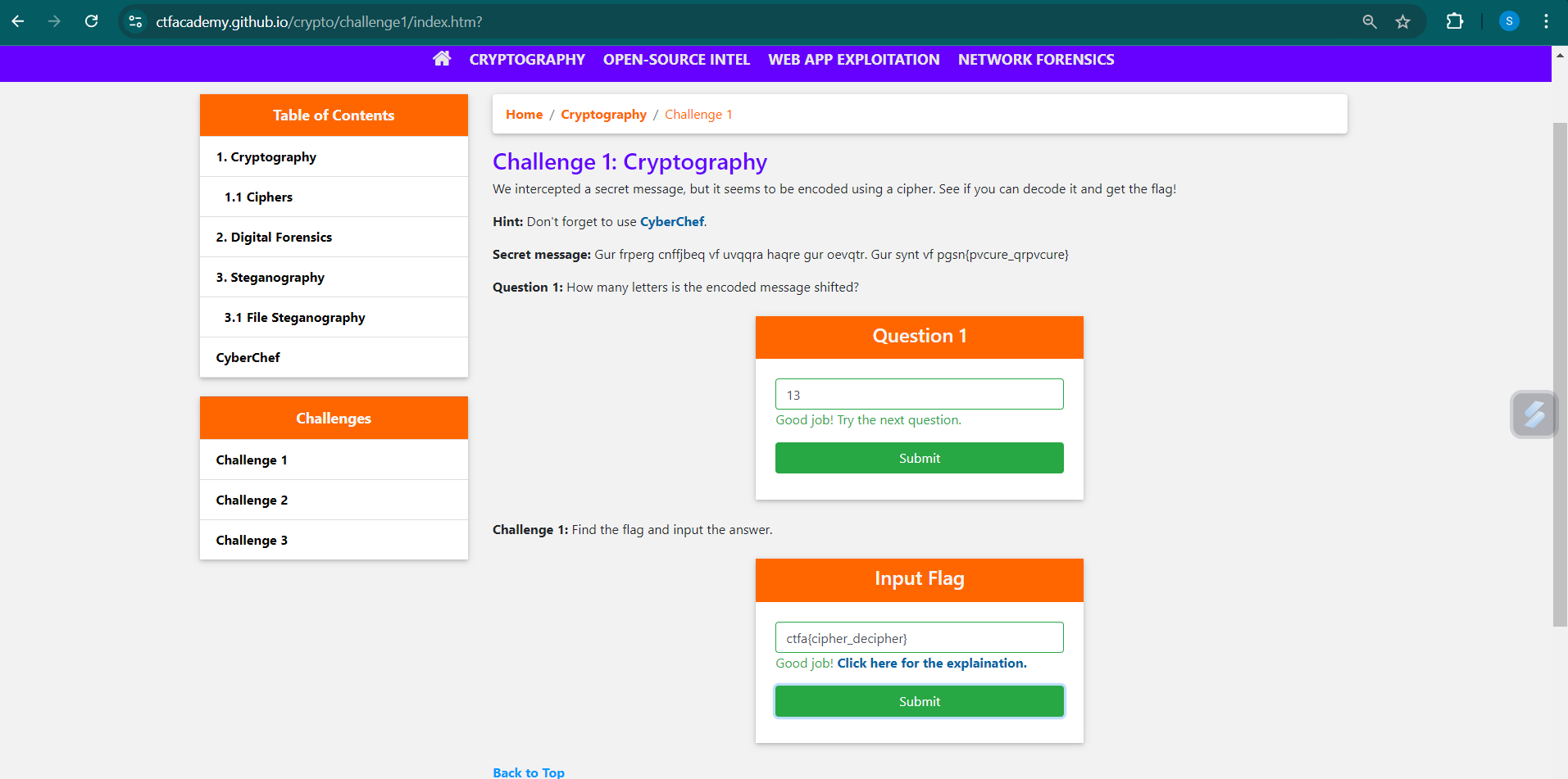
code

ctfa{cipher\_decipher}

1. **Submission of the Flag:**
   * The flag for this challenge, **ctfa{cipher\_decipher}**, was successfully submitted in the input box, leading to a successful completion of the first challenge.

**Tools and Methods Used:**

* **CyberChef:**
  + This is a web-based tool that supports a wide array of operations such as encoding, decoding, encryption, and decryption using various ciphers, including ROT13.
  + The tool allowed me to quickly input the encoded text and apply the necessary decryption operation.
* **ROT13 Cipher:**
  + A simple substitution cipher where each letter is shifted by 13 positions.
  + Commonly used in introductory cryptography challenges due to its simplicity and ease of decryption.



**Reflection:**

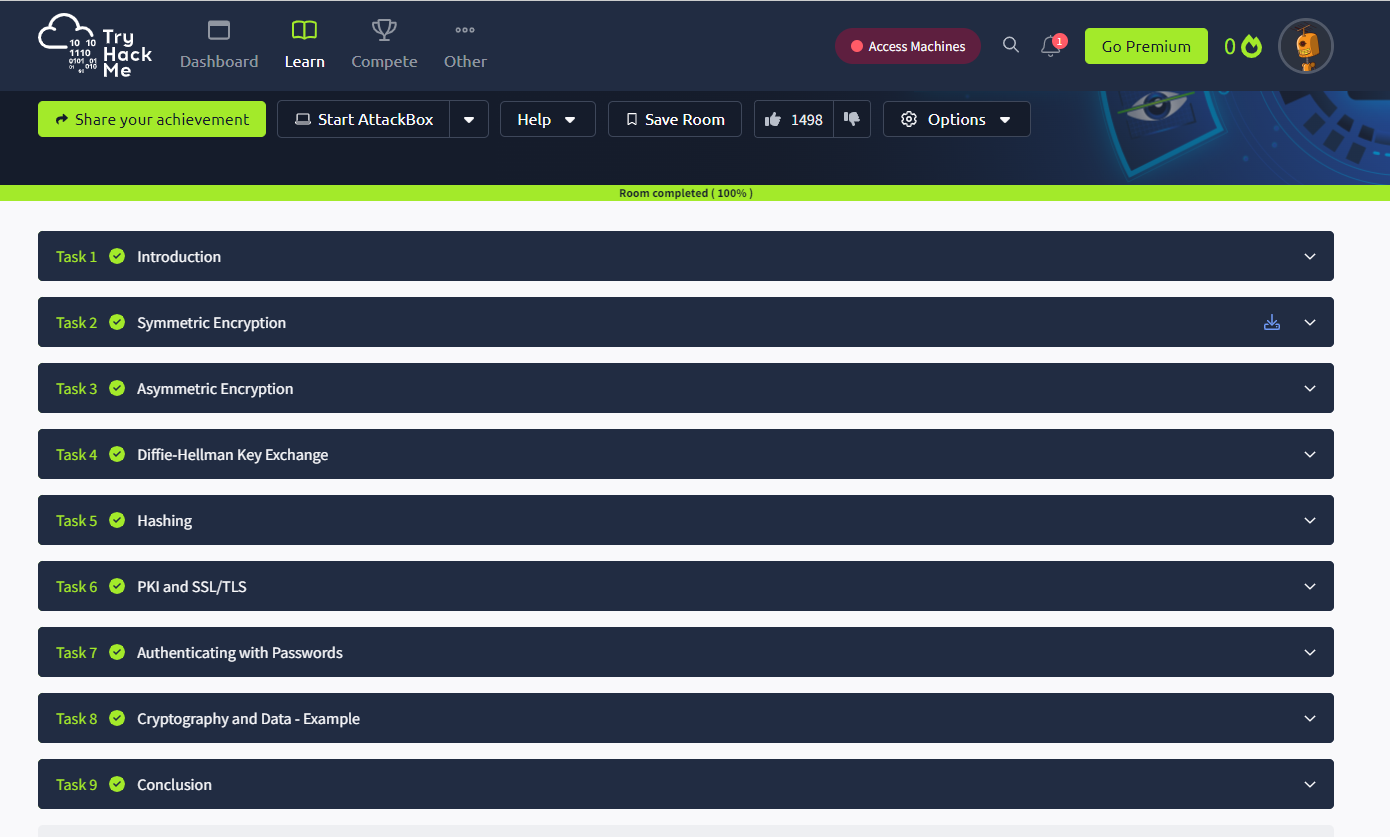
The challenge provided a great introduction to basic cryptographic techniques, specifically the Caesar cipher and its ROT13 variant. While this was a straightforward problem, it underscored the importance of recognizing patterns in encrypted messages and leveraging appropriate tools (like CyberChef) for efficient decoding. Understanding simple ciphers such as Caesar and ROT13 forms the foundation for tackling more complex cryptographic challenges in future levels of the competition.

**Conclusion:**

By identifying the ROT13 cipher and using the appropriate decryption method, I was able to successfully decode the hidden message and retrieve the flag **ctfa{cipher\_decipher}**. This challenge served as a fundamental exercise in understanding the basics of cryptography, providing essential skills that will be useful in future cryptographic tasks within the CTF.

**Experiment – 4**

**Cryptography Study Summary from TryHackMe**

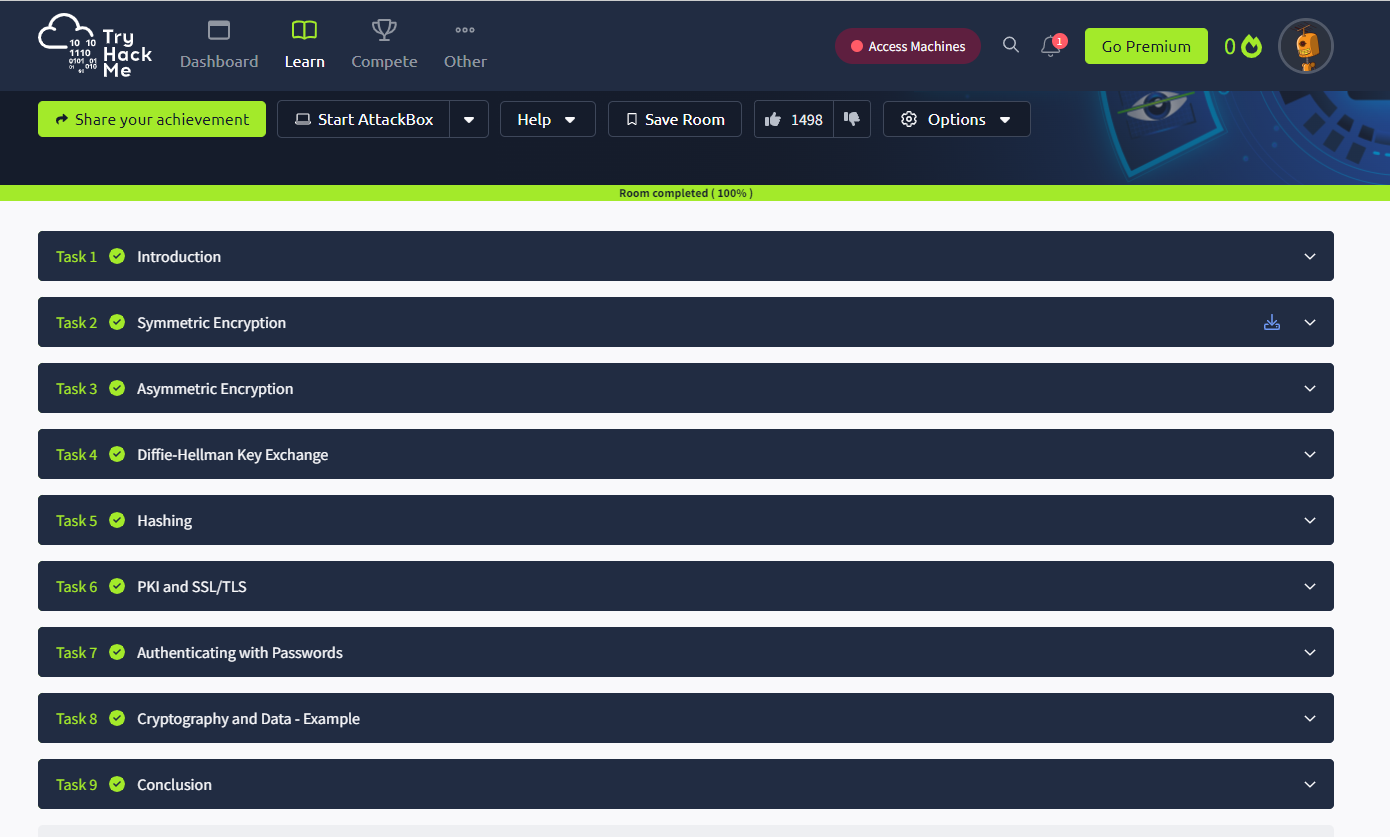


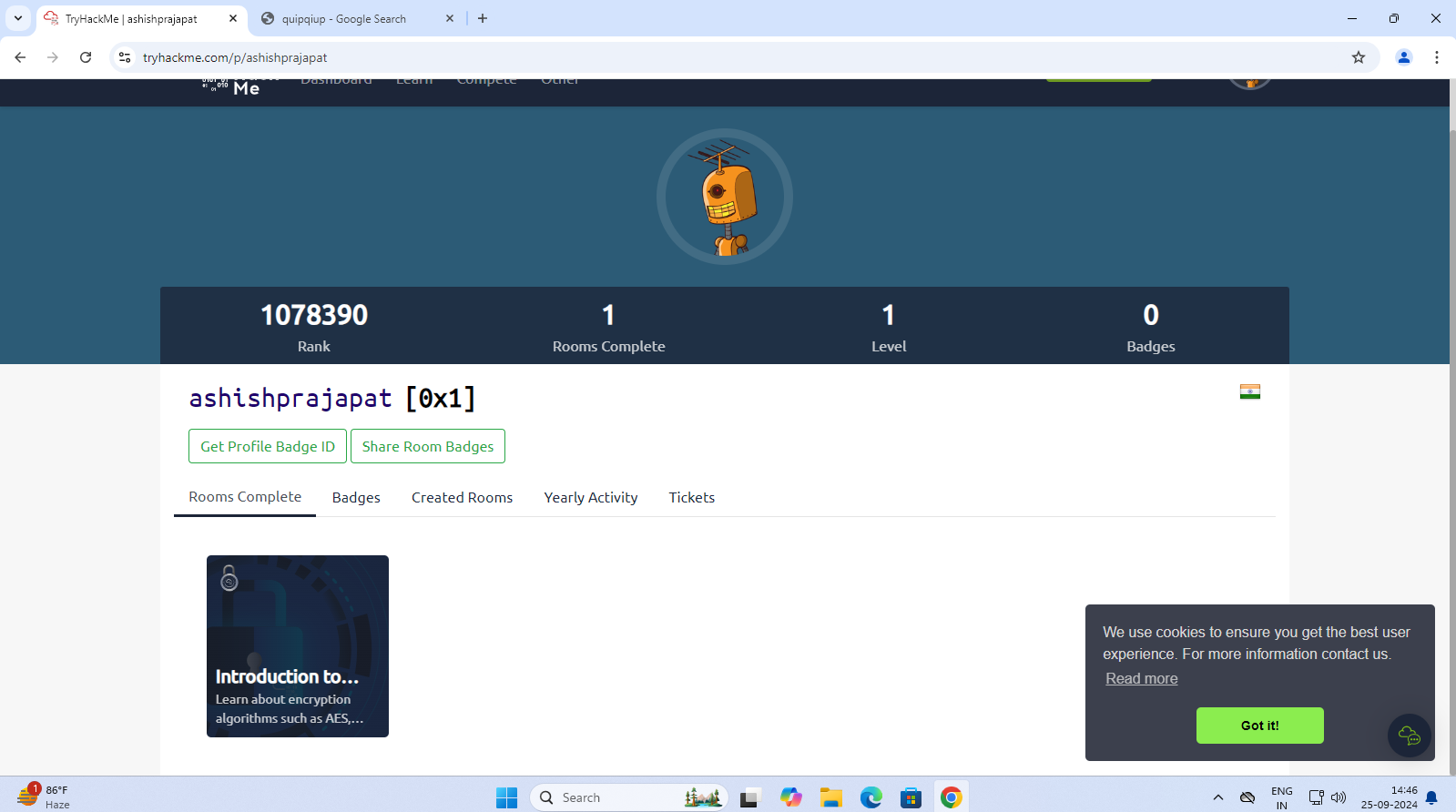
In this experiment, I completed the **"Introduction to Cryptography"** room on TryHackMe, gaining hands-on experience with various cryptographic principles and their applications. The room covered the following topics in detail:

1. **Symmetric Encryption (Task 2):** I explored symmetric encryption, where the same key is used for both encryption and decryption. This technique is efficient but requires both parties to share the key securely. Algorithms like AES (Advanced Encryption Standard) were studied, which are widely used for securing data due to their strength and speed.
2. **Asymmetric Encryption (Task 3):** In contrast to symmetric encryption, asymmetric encryption utilizes a pair of keys—a public key for encryption and a private key for decryption. This method enhances security since only the recipient can decrypt the data. RSA (Rivest–Shamir–Adleman) was the primary algorithm covered, which is commonly used for securing emails, digital signatures, and more.
3. **Diffie-Hellman Key Exchange (Task 4):** This topic focused on the Diffie-Hellman protocol, which allows two parties to securely exchange cryptographic keys over an insecure channel. The protocol ensures that even if an attacker intercepts the communication, they won’t be able to derive the shared secret key. This method is the basis for many modern encryption schemes.
4. **Hashing (Task 5):** Hashing is the process of converting data into a fixed-length string, known as a hash, regardless of the size of the input. This technique is widely used for data integrity verification and password storage. I learned about popular hashing algorithms like MD5 and SHA-256, their use cases, and vulnerabilities like hash collisions in weaker algorithms.
5. **Public Key Infrastructure (PKI) and SSL/TLS (Task 6):** PKI is a system of digital certificates, Certificate Authorities (CAs), and registration authorities used to secure communications on the internet. I also learned about SSL/TLS protocols, which use certificates to encrypt data between a user and a website, providing secure communication channels for sensitive information like banking transactions.
6. **Authenticating with Passwords (Task 7):** This task delved into secure methods for password authentication, discussing the importance of salting and hashing passwords before storage to protect against brute force attacks. Additionally, password policies and multi-factor authentication (MFA) methods were covered as a way to enhance security.
7. **Cryptography and Data - Real-world Example (Task 8):** This section explored a practical example of cryptography in action, showing how encryption and hashing techniques are employed in industries like finance, healthcare, and online services to secure sensitive data and prevent unauthorized access.

**Conclusion**

This study provided me with a strong foundational understanding of cryptographic methods and their applications. I learned not only about the theory behind encryption, hashing, and key exchanges, but also how these methods are applied in the real world to secure communications and data. The hands-on tasks within the TryHackMe platform helped solidify my understanding of these concepts, making it a valuable learning experience.





**Experiment – 5**

**To study of VPN (Virtual Private Network)?**

A VPN (Virtual Private Network) is a service or technology that creates a secure and encrypted connection over a less secure network, such as the public internet. It allows users to access private networks or securely browse the internet by tunneling their internet traffic through a server controlled by the VPN provider. This protects the data from potential hackers or monitoring by ISPs (Internet Service Providers).

Key Features of VPN:

* Encryption: VPNs use encryption to secure data sent and received, preventing third parties from eavesdropping on the communication.
* Anonymity: VPNs hide a user's IP address and geographical location, making it appear as though the traffic originates from the VPN server.
* Bypass Geo-restrictions: VPNs can help users access content that might be restricted in certain regions.
* Secure Access to Private Networks: Remote users can securely access a company’s private network through a VPN, as if they were directly connected to it.

What is a Virtual Network?

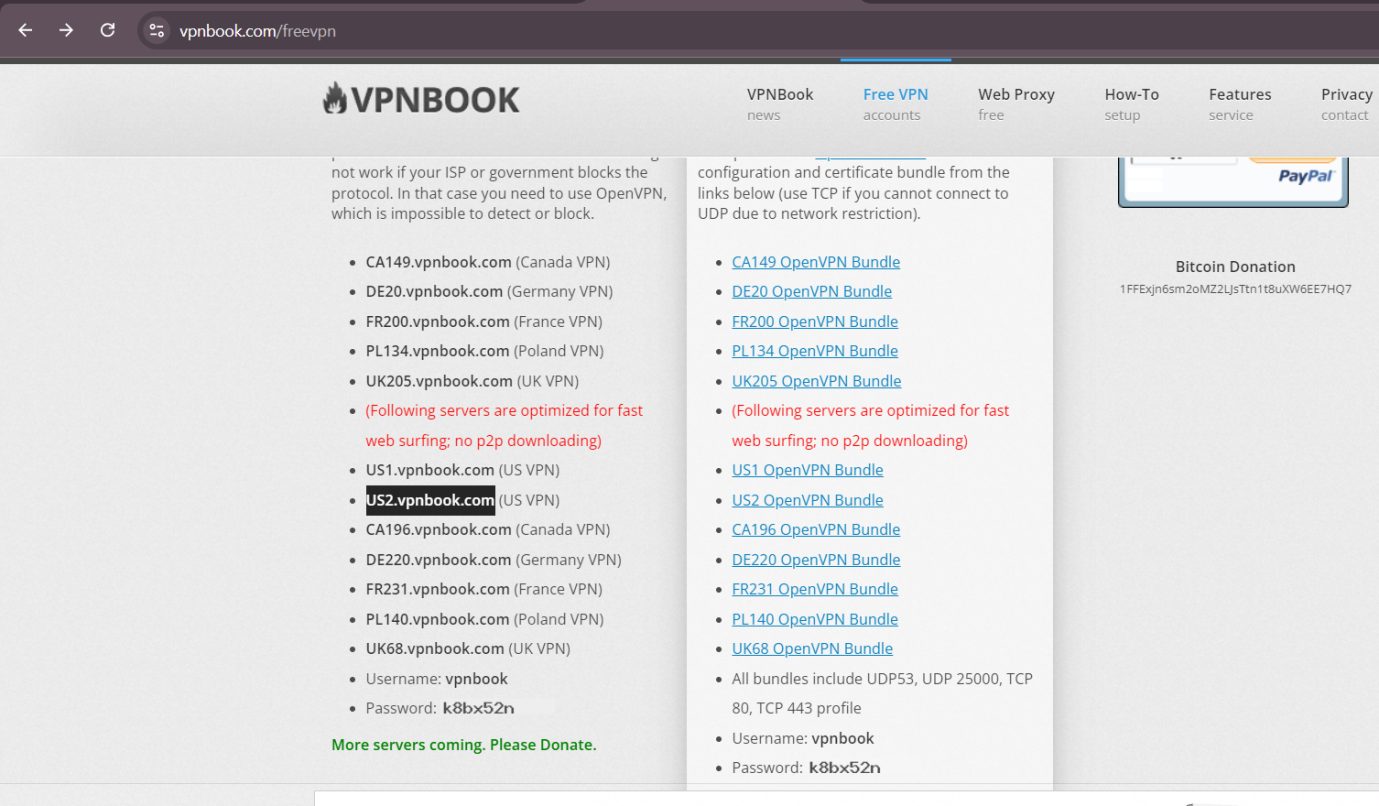
A Virtual Network refers to the creation of network services or devices within a shared infrastructure, typically in cloud environments. It allows resources (such as servers, databases, and virtual machines) to communicate with each other over a simulated or virtualized network instead of physical connections.

Key Features of Virtual Networks:

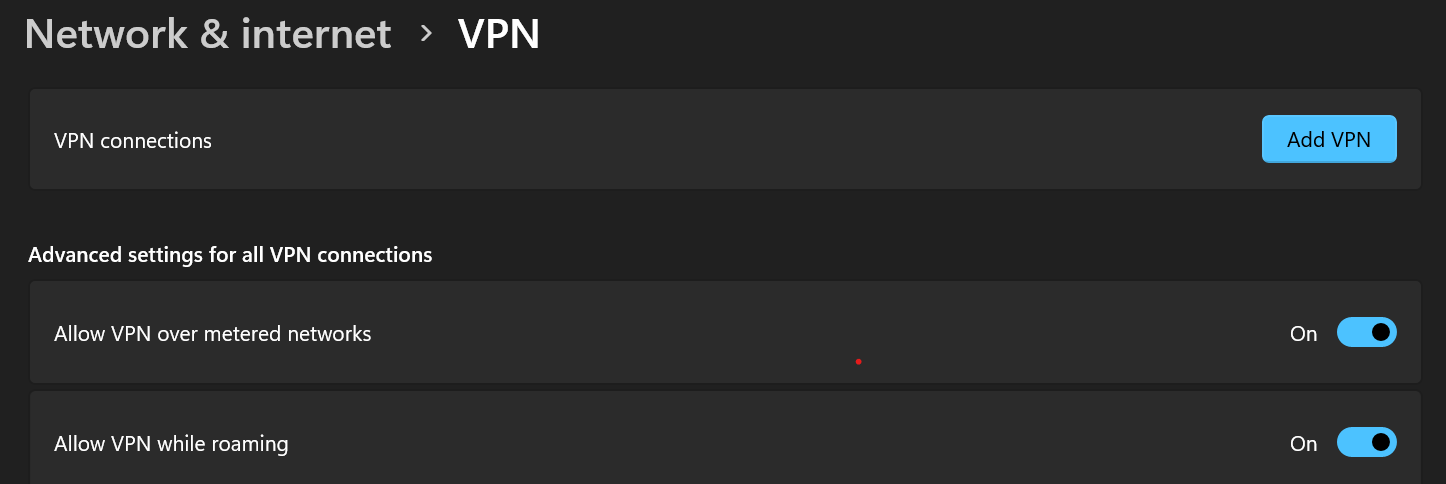
* Resource Isolation: Virtual networks isolate specific traffic, applications, or services in a multi-tenant environment to ensure that they don't interfere with other network traffic.
* Scalability and Flexibility: Virtual networks can be easily created, expanded, or deleted, offering flexibility in cloud environments like AWS, Azure, or Google Cloud.
* Internal Communication: In a cloud environment, virtual networks allow virtual machines or instances to communicate with each other without needing public IP addresses.

**Steps to Configure VPN on Windows 11 Using VPNBook:**

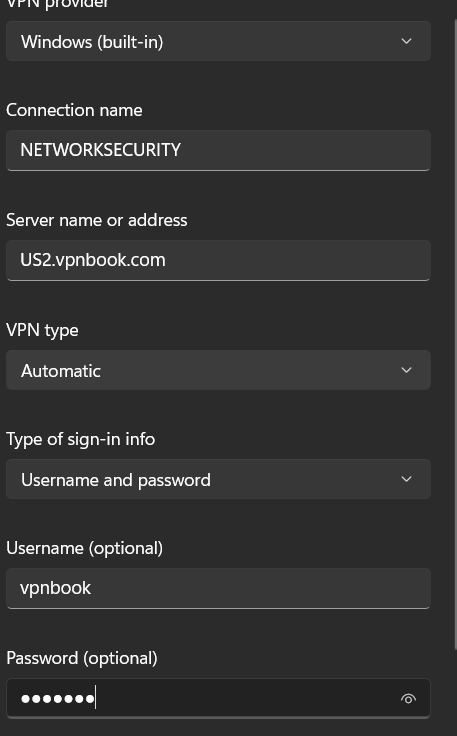
1. **Visit VPNBook Website:**
   * Open a browser and go to the [VPNBook website](https://www.vpnbook.com).
   * Choose a VPN server from the options provided (e.g., **US2.vpnbook.com**) or anything you want .
   * Note the **username** and **password** displayed on the website, as they will be needed during the setup.



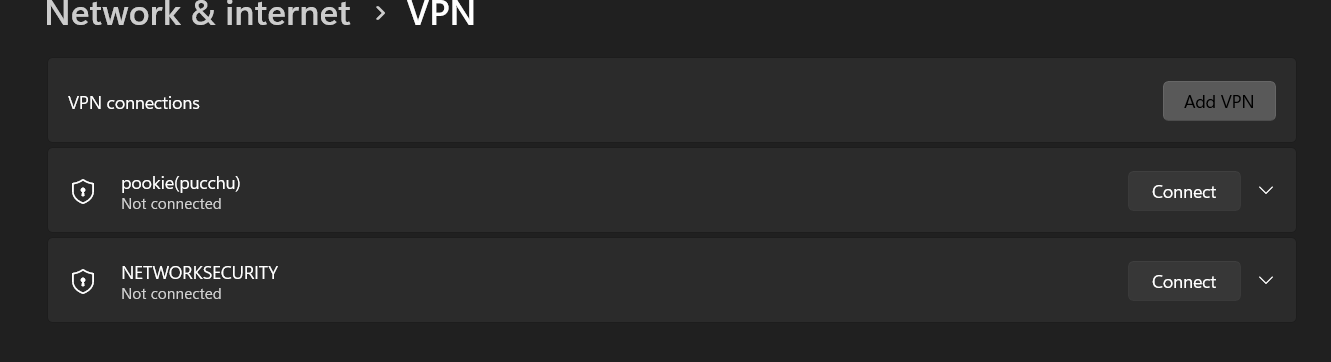
1. **Open VPN Settings in Windows 11:**
   * Click on the **Start** menu and go to **Settings**.
   * Select **Network & Internet** from the list of options.
   * In the left-hand menu, click on **VPN**.



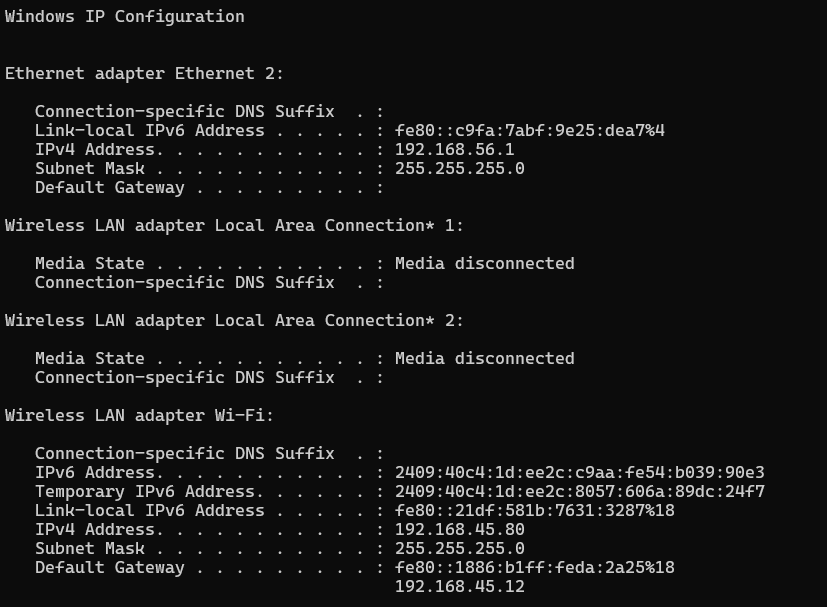
1. **Add a New VPN Connection:**
   * Click the **Add VPN** button to start the configuration.
   * In the window that appears, enter the following details:
     + **VPN Provider:** Windows (built-in)
     + **Connection Name:** Any custom name, such as "VPNBook"
     + **Server Name or Address:** Enter the VPN server address (e.g., **US2.vpnbook.com**).
     + **VPN Type:** Choose **PPTP** (Point to Point Tunneling Protocol).
     + **Type of Sign-in Info:** Select **Username and Password**.
     + **Username and Password:** Enter the username and password provided by VPNBook.
2. **Save the VPN Connection:**
   * After entering all the details, click **Save**.

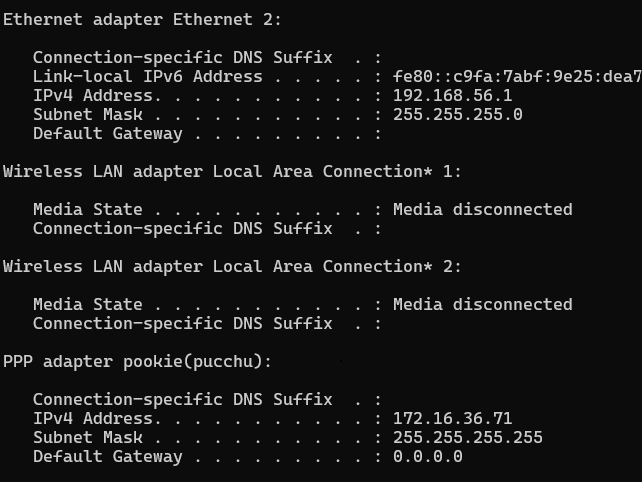


1. **Connect to the VPN:**
   * In the VPN settings, you should now see the newly created VPN connection.
   * Click on the VPN connection (e.g., VPNBook) and then click **Connect** to establish the VPN connection.



1. **Verify the VPN Connection:**
   * Once connected, you can check your IP address by visiting any "What is my IP" website to confirm that your traffic is routed through the VPN.





**Experiment :- 6**

**To study of different software vulnerabilities:-**

**1. Buffer Overflow**

A buffer overflow occurs when a program writes more data to a buffer than it can hold, potentially overwriting adjacent memory. This can lead to arbitrary code execution.

**Example:** In a C program, if a buffer is defined to hold 10 characters but the program allows 20 characters to be written, the excess can overwrite critical data. An attacker can exploit this to inject malicious code.

**Code:-**

#include <iostream>

#include <cstring>

using namespace std;

void safeFunction()

{

char buffer[10];

cout << "Enter some text (max 9 characters): ";

cin.getline(buffer, sizeof(buffer)); // Safe input

cout << "You entered: " << buffer << std::endl;

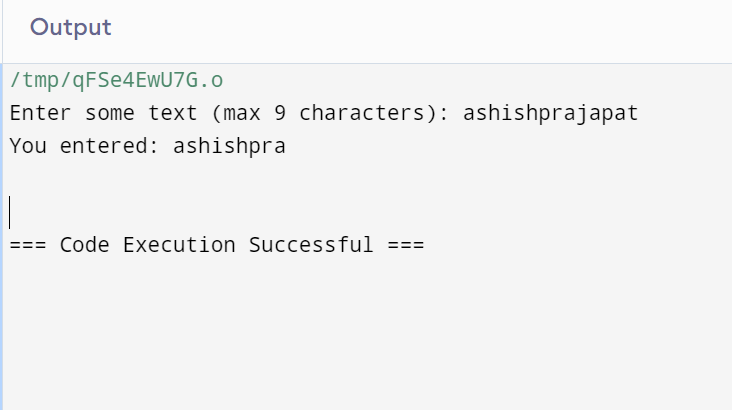
}

int main() {

safeFunction();

return 0;

}



**2. SQL Injection**

SQL Injection occurs when an application includes untrusted input in a SQL query without proper validation or escaping. This can allow attackers to manipulate the database.

**Example:** A web application that uses user input to construct a SQL query like:

**sql**

**code:-**

**SELECT \* FROM users WHERE username = 'user\_input';**

**If user\_input is admin' --, the query becomes:**

**sql**

**code:-**

**SELECT \* FROM users WHERE username = 'admin' --';**

This can grant access to unauthorized data.

**3. Cross-Site Scripting (XSS)**

XSS vulnerabilities allow attackers to inject malicious scripts into web pages viewed by other users. This can lead to session hijacking, defacement, or redirection to malicious sites.

**Example:** A comment field on a website that doesn't sanitize input could allow an attacker to submit:

**<script>alert('Hacked!');</script>**

When another user views the comments, the script runs in their browser.

**4. Cross-Site Request Forgery (CSRF)**

CSRF tricks a user into executing unwanted actions on a web application where they're authenticated. This can manipulate user data or perform actions without their consent.

**Example:** If a user is logged into a banking site, an attacker could send them an email with a link that automatically transfers money without their knowledge if they click it.

**5. Insecure Direct Object References (IDOR)**

IDOR occurs when an application exposes internal implementation objects, allowing attackers to bypass authorization and access or manipulate data they shouldn't.

**Example:** If a URL contains a user ID, such as /account/12345, an attacker could try changing it to /account/12346 to access another user's account.

**6. Security Misconfiguration**

This vulnerability arises from default settings or incomplete setups that can be exploited. It can occur at any level of an application stack.

**Example:** Leaving default credentials (like admin/admin) for a web application or not properly configuring security settings on cloud services can lead to unauthorized access.

**7. Sensitive Data Exposure**

Applications that do not properly protect sensitive data can expose it to attackers. This includes not encrypting data at rest or in transit.

**Example:** An application storing passwords in plain text instead of using strong hashing algorithms allows attackers to retrieve and misuse user passwords if the database is compromised.

**8. Broken Authentication and Session Management**

Weak authentication mechanisms can lead to unauthorized access. This includes issues like predictable session tokens or lack of account lockout policies.

**Example**: An application that allows unlimited login attempts with predictable usernames can be brute-forced to gain access.

**9. Using Components with Known Vulnerabilities**

Using outdated libraries or frameworks with known security flaws can make an application vulnerable.

**Example:** A web application using an outdated version of a library known for XSS vulnerabilities could expose it to attacks if the library is not updated.

**10. Insufficient Logging and Monitoring**

Without proper logging and monitoring, detecting attacks or breaches becomes difficult, prolonging the time it takes to respond to incidents.

**Example:** An application that fails to log failed login attempts may not detect a brute-force attack until significant damage has been done.