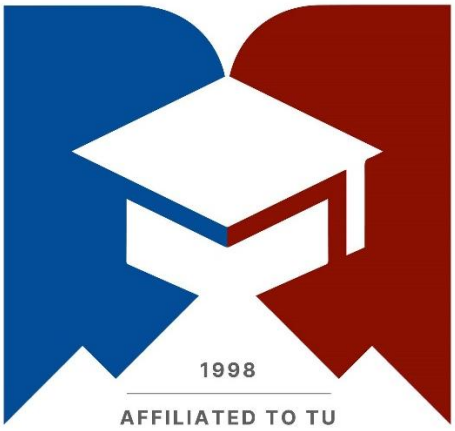
NEW SUMMIT COLLEGE

Santinagar, Kathmandu

(Affiliated to Tribhuvan University)



LABORATORY ASSIGNMENT

REPORT OF

INFORMATION SECURITY (CACS459)

Submitted by:

Name: Anish Chauhan

Roll No: 04

Semester: Eighth

Faculty: Humanities and Social Science

Level: Bachelor

Program: Computer Application

Submitted to:

Instructor: Indra Chaudhary

Submission Date: …….

2025

NEW SUMMIT COLLEGE
BCA 8TH SEMESTER | INFORMATION SECURITY (CACS495)
Laboratory Assignment Log Sheet

Task No.	Title	Signature
Cryptographic Algorithms (based on programming Language)		
1	Implementation of Caesar Cipher	
2	Implementation of Playfair Cipher	
3	Implementation of Rail Fence – Row & Column Transformation Techniques	
4	Implementation of DES Algorithm	
5	Implementation of Primality Testing	
6	Implementation of Euclidean Algorithm	
7	Implementation of RSA Algorithm	
Message Authentication and Hashing Functions (based on programming language)		
8	Implementation of MD5 Hashing Algorithm	
Access Control, Intrusion and Network Security (based on Linux Machine)		
9	Preparing Linux Lab Environment	
10	Users, Groups, Permission	
11	User and Group Administration	
12	Firewall Configuration	
13	Configuring SSH Server to allow/deny root login and allow/deny users login	
14	Configuring SSH Server to allow/deny SSH login from selected hosts only	
15	Configuring SSH Server for direct SSH login by generating and publishing private and public key	
16	Secure Network Copy using “SCP”	
17	Security Enhanced Linux (SE Linux)	
18	Configuring SSL-Enabled Apache (HTTPS) Server (self-signed)	

Lab Program Number: 1

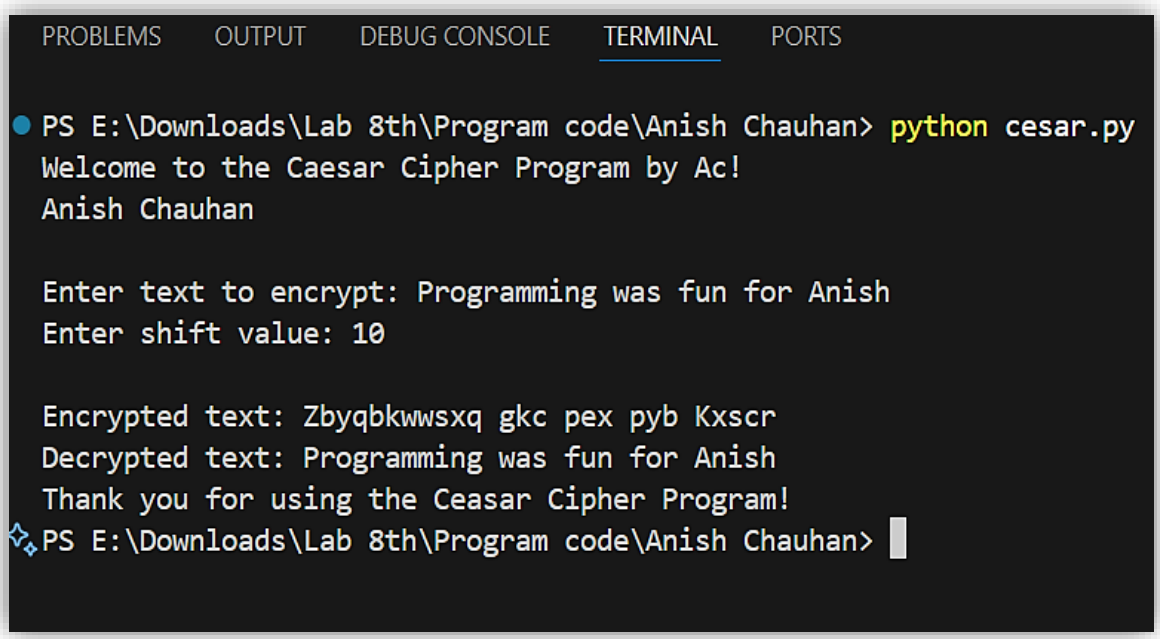
Title: Implementation of Caesar Cipher

A Caesar cipher is one of the simplest and most widely known encryption techniques. It's a type of substitution cipher where each letter in the plaintext is replaced by a letter some fixed number of positions down or up the alphabet. Due to only having 25 possible shifts, it is not secure and is easily broken, often serving as a basic example in cryptography education.

Program:

```
def caesar_cipher(text, shift):
    result = ""
    for char in text:
        if char.isalpha():
            base = ord('A') if char.isupper() else ord('a')
            result += chr((ord(char) - base + shift) % 26 + base)
        else:
            result += char
    return result
print("Welcome to the Caesar Cipher Program by AC!")
print("Anish Chauhan\n")

text = input("Enter text to encrypt: ")
shift = int(input("Enter shift value: "))

# Encryption
encrypted = caesar_cipher(text, shift)
print("\nEncrypted text:", encrypted)

# Decryption
decrypted = caesar_cipher(encrypted, -shift)
print("Decrypted text:", decrypted)
print("Thank you for using the Ceasar Cipher Program!")
```

Output:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS E:\Downloads\Lab 8th\Program code\Anish Chauhan> python cesar.py
Welcome to the Caesar Cipher Program by Ac!
Anish Chauhan
Enter text to encrypt: Programming was fun for Anish
Enter shift value: 10
Encrypted text: Zbyqbkwwsxq gkc pex pyb Kxscr
Decrypted text: Programming was fun for Anish
Thank you for using the Ceasar Cipher Program!
PS E:\Downloads\Lab 8th\Program code\Anish Chauhan>

Lab Program Number: 2

Title: Implementation of Playfair Cipher

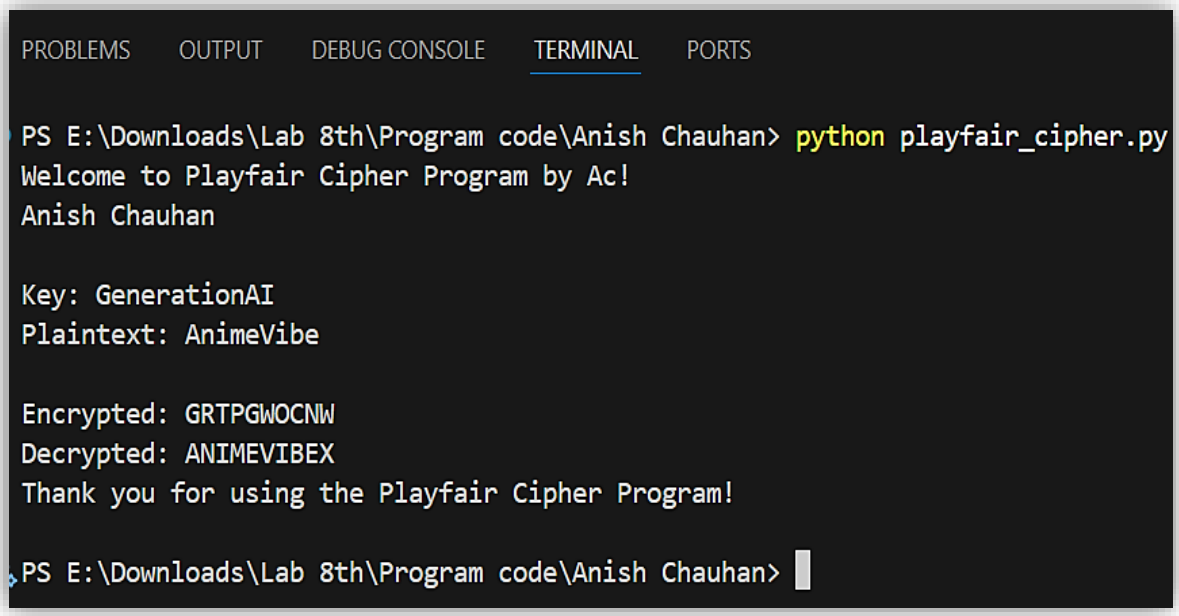
A Playfair cipher is a classical polygraphic substitution cipher that encrypts letters in pairs (digraphs) instead of individually. It uses a 5×5 square matrix based on a secret keyword to determine the substitution. Because it operates on pairs of letters, it significantly increases the number of possible symbols (25 × 25 = 625 digrams) and is therefore much more difficult to break using the simple frequency analysis attacks that easily defeat the Caesar cipher.

Program:

```
def create_matrix(key):
    key = key.upper().replace('J', 'I')
    matrix, used = [], set()
    for c in key + 'ABCDEFGHIKLMNOPQRSTUVWXYZ':
        if c.isalpha() and c not in used:
            used.add(c)
            matrix.append(c)
    return [matrix[i:i+5] for i in range(0, 25, 5)]

def find_pos(matrix, ch):
    for i in range(5):
        for j in range(5):
            if matrix[i][j] == ch:
                return i, j

def playfair(text, key, mode='encrypt'):
    m = create_matrix(key)
    text = text.upper().replace('J', 'I').replace(' ', '')
    pairs, i = [], 0

    # Create digraphs
    while i < len(text):
        a = text[i]
        b = text[i + 1] if i + 1 < len(text) else 'X'
        if a == b: b = 'X'; i += 1
        else: i += 2
```

```
        pairs.append(a + b)

    result = ''
    for a, b in pairs:
        r1, c1 = find_pos(m, a)
        r2, c2 = find_pos(m, b)
        if r1 == r2:
            shift = 1 if mode == 'encrypt' else -1
            result += m[r1][(c1 + shift) % 5] + m[r2][(c2 + shift) % 5]
        elif c1 == c2:
            shift = 1 if mode == 'encrypt' else -1
            result += m[(r1 + shift) % 5][c1] + m[(r2 + shift) % 5][c2]
        else:
            result += m[r1][c2] + m[r2][c1]
    return result

if __name__ == "__main__":
    print("Welcome to Playfair Cipher Program!")
    print("Anish Chauhan\n")

    key = "MONARCHY"
    text = "HELLO"

    encrypted = playfair(text, key, 'encrypt')
    decrypted = playfair(encrypted, key, 'decrypt')

    print(f"Key: {key}")
    print(f"Plaintext: {text}\n")
    print(f"Encrypted: {encrypted}")
    print(f"Decrypted: {decrypted}")
    print("Thank you for using the Playfair Cipher Program! \n")
```

Output:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS E:\Downloads\Lab 8th\Program code\Anish Chauhan> python playfair_cipher.py
Welcome to Playfair Cipher Program by Ac!
Anish Chauhan
Key: GenerationAI
Plaintext: AnimeVibe
Encrypted: GRTPGWOCNW
Decrypted: ANIMEVIBEX
Thank you for using the Playfair Cipher Program!
PS E:\Downloads\Lab 8th\Program code\Anish Chauhan>

Lab Program Number: 3

Title: Implementation of Rail Fence Cipher – Row & Column Transformation Technique

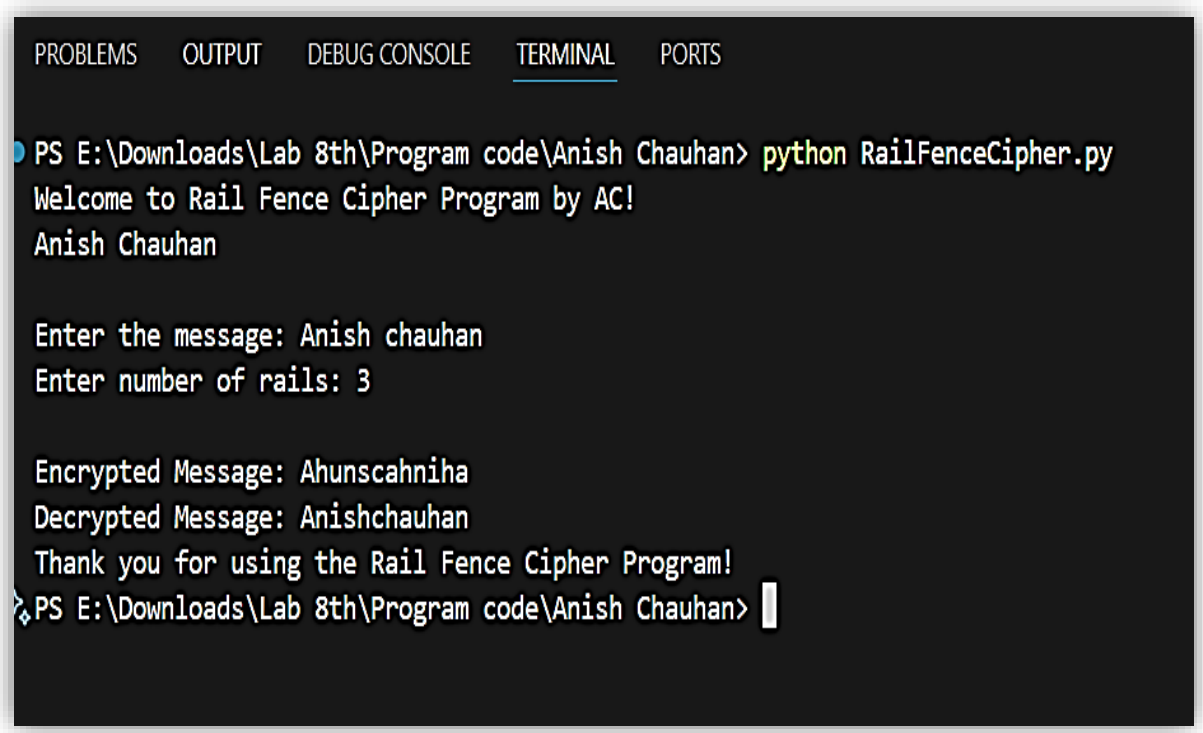
A Rail Fence Cipher is a form of transposition cipher where the letters of a message are written in a zigzag pattern across multiple "rails" (rows) and then read row by row to create the ciphertext. This rearranges the order of characters without changing the letters themselves. The number of rails determines the pattern of encryption and decryption. Although it provides only basic security, it is widely used as a simple example to demonstrate the concept of transposition in cryptography.

Program:

```
def rail_fence_encrypt(message, rails):
    message = message.replace(" ", "")
    fence = [[" for _ in range(len(message))] for _ in range(rails)]

    row, step = 0, 1
    for i, char in enumerate(message):
        fence[row][i] = char
        if row == 0:
            step = 1
        elif row == rails - 1:
            step = -1
        row += step

    encrypted = ''.join(ch for r in fence for ch in r if ch)
    return encrypted

def rail_fence_decrypt(cipher, rails):
    fence = [[" for _ in range(len(cipher))] for _ in range(rails)]
    index, row, step = 0, 0, 1

    # Mark the zigzag pattern
    for i in range(len(cipher)):
        fence[row][i] = '*'
        if row == 0:
            step = 1
        elif row == rails - 1:
            step = -1
        row += step

# Fill the pattern with cipher letters
    for r in range(rails):
```

```
        for c in range(len(cipher)):
            if fence[r][c] == '*' and index < len(cipher):
                fence[r][c] = cipher[index]
                index += 1

    # Read plaintext in zigzag order
    result, row, step = '', 0, 1
    for i in range(len(cipher)):
        result += fence[row][i]
        if row == 0:
            step = 1
        elif row == rails - 1:
            step = -1
        row += step

    return result

print("Welcome to Rail Fence Cipher Program by AC!")
print("Anish Chauhan\n")

message = input("Enter the message: ")
rails = int(input("Enter number of rails: "))

encrypted = rail_fence_encrypt(message, rails)
decrypted = rail_fence_decrypt(encrypted, rails)

print("\nEncrypted Message:", encrypted)
print("Decrypted Message:", decrypted)
print("\nThank you for using the Rail Fence Cipher Program!")
```

Output:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS E:\Downloads\Lab 8th\Program code\Anish Chauhan> python RailFenceCipher.py
Welcome to Rail Fence Cipher Program by AC!
Anish Chauhan
Enter the message: Anish chauhan
Enter number of rails: 3
Encrypted Message: Ahunscahniha
Decrypted Message: Anishchauhan
Thank you for using the Rail Fence Cipher Program!
PS E:\Downloads\Lab 8th\Program code\Anish Chauhan>

Lab Program Number: 4

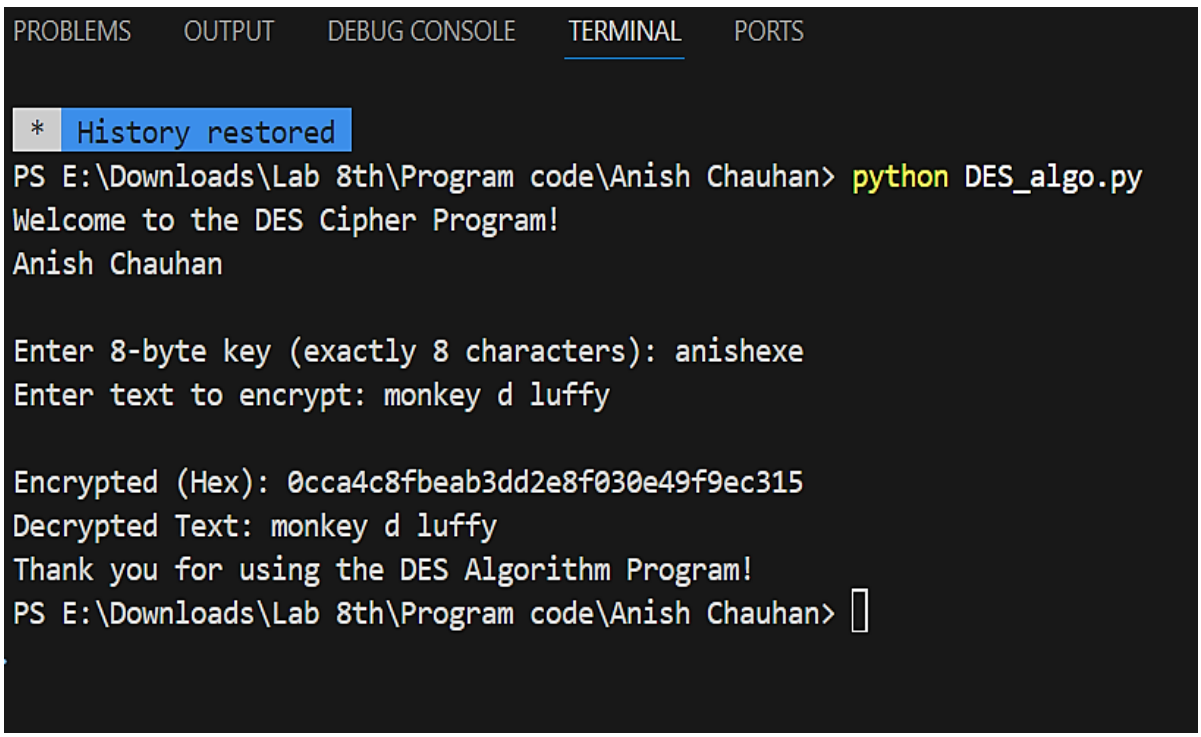
Title: Implementation of DES Algorithm

The Data Encryption Standard (DES) is a symmetric-key block cipher used for the encryption and decryption of digital data. It operates on 64-bit blocks of plaintext using a 56-bit key and applies a series of complex permutations and substitutions to secure the message. DES uses the same key for both encryption and decryption, making it efficient for short, secure data transmission. Although it has been largely replaced by stronger algorithms like AES, DES remains an important foundation in the study of modern cryptography.

Program:

```
from pyDes import des, ECB, PAD_PKCS5
import binascii

print("Welcome to the DES Cipher Program!")
print("Anish Chauhan\n")

# Input key and data
key = input("Enter 8-byte key (exactly 8 characters): ")
data = input("Enter text to encrypt: ")

# Create DES cipher object
cipher = des(key, ECB, padmode=PAD_PKCS5)

# Encrypt and decrypt
encrypted = cipher.encrypt(data)
decrypted = cipher.decrypt(encrypted)

# Display results
print("\nEncrypted (Hex):", binascii.hexlify(encrypted).decode())
print("Decrypted Text:", decrypted.decode())
print("Thank you for using the DES Algorithm Program!")
```

Output:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
* History restored
PS E:\Downloads\Lab 8th\Program code\Anish Chauhan> python DES_algo.py
Welcome to the DES Cipher Program!
Anish Chauhan
Enter 8-byte key (exactly 8 characters): anishexe
Enter text to encrypt: monkey d luffy
Encrypted (Hex): 0cca4c8fbeab3dd2e8f030e49f9ec315
Decrypted Text: monkey d luffy
Thank you for using the DES Algorithm Program!
PS E:\Downloads\Lab 8th\Program code\Anish Chauhan>

Lab Program Number: 5

Title: Implementation of Primality Testing

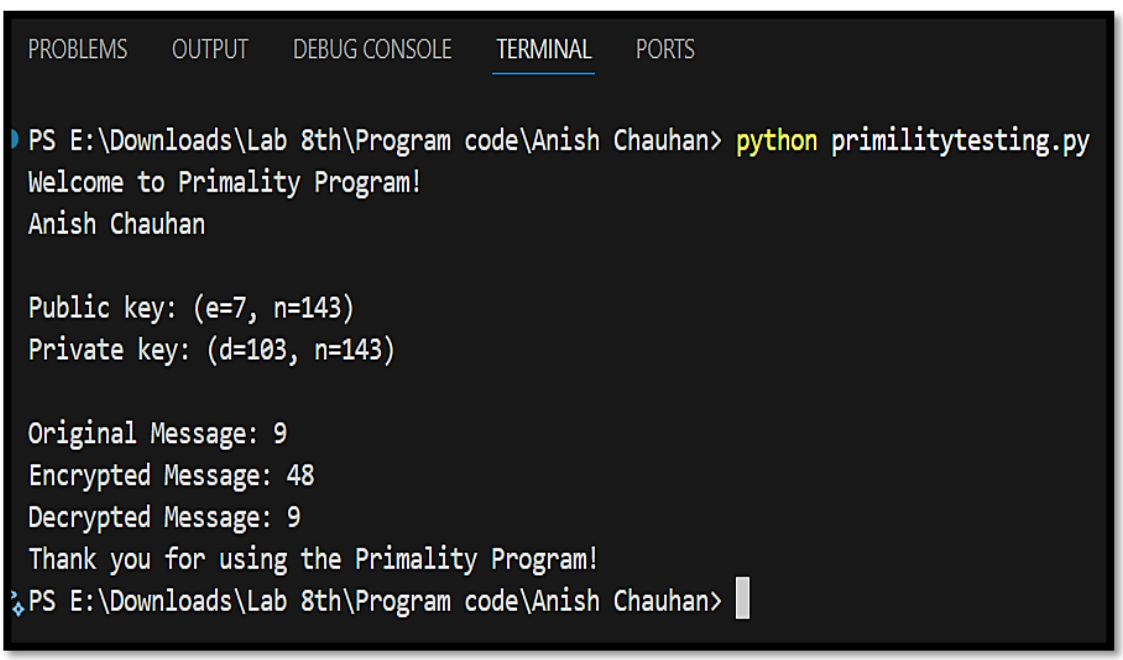
Primality Testing is a process used to determine whether a given number is a prime number or not. A prime number is one that is greater than 1 and divisible only by 1 and itself. In cryptography, primality testing plays a vital role in algorithms such as RSA, where large prime numbers are used to generate secure public and private keys. Efficient primality tests ensure both security and performance in modern encryption systems.

Program:

```
def is_prime(n):
    if n <= 1:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False
    for i in range(3, int(n**0.5)+1, 2):
        if n % i == 0:
            return False
    return True

# Sample primes
p = 11
q = 13

if not is_prime(p) or not is_prime(q):
    print("p or q is not prime, can't proceed with RSA!")
else:
    n = p * q
    phi = (p - 1) * (q - 1)
    e = 7  # Public exponent
    d = pow(e, -1, phi)  # Private exponent

    print("Welcome to Primality Program!")
```

```
print("Anish Chauhan\n")

print(f"Public key: (e={e}, n={n})")
print(f"Private key: (d={d}, n={n})\n")

message = 9
encrypted = pow(message, e, n)
decrypted = pow(encrypted, d, n)

print("Original Message:", message)
print("Encrypted Message:", encrypted)
print("Decrypted Message:", decrypted)
print("Thank you for using the Primality Program!")
```

Output:



Lab Program Number: 6

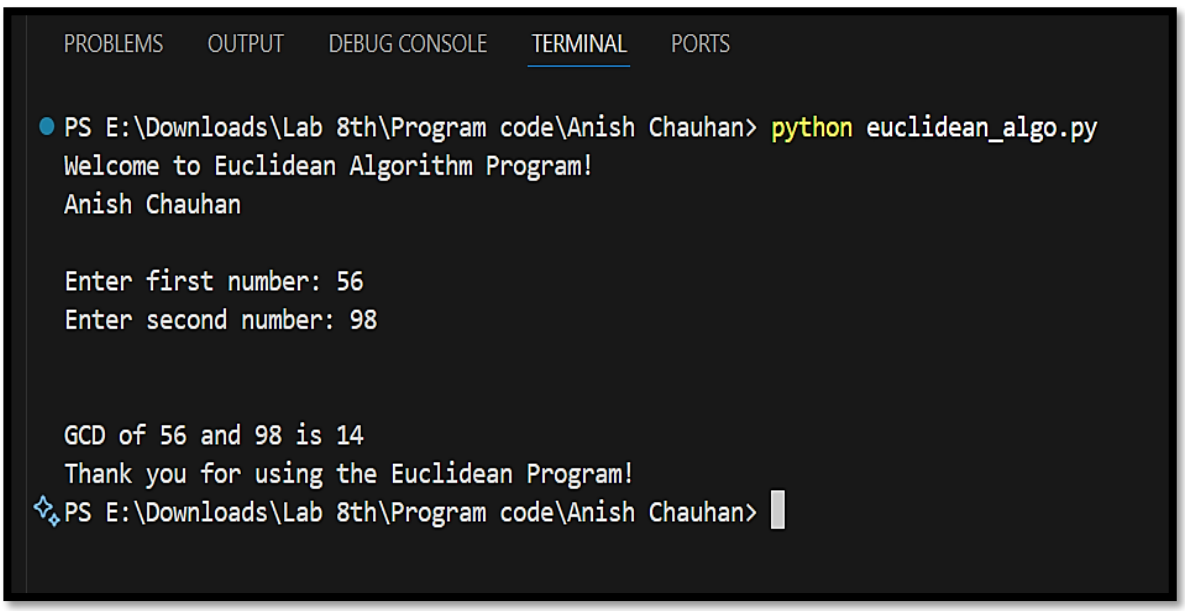
Title: Implementation of Euclidean Algorithm

The Euclidean Algorithm is one of the oldest and most efficient methods for finding the Greatest Common Divisor (GCD) of two numbers. It works by repeatedly replacing the larger number with the remainder when it is divided by the smaller number, continuing until the remainder becomes zero. The last non-zero remainder is the GCD. This algorithm is widely used in cryptography and number theory, particularly in RSA key generation.

Program:

```
def euclidean_gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a


print("Welcome to Euclidean Algorithm Program!")
print("Anish Chauhan\n")

num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))
print("\n")

gcd = euclidean_gcd(num1, num2)
print(f"GCD of {num1} and {num2} is {gcd}")
print("Thank you for using the Euclidean Program!")
```

Output:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS E:\Downloads\Lab 8th\Program code\Anish Chauhan> python euclidean_algo.py
Welcome to Euclidean Algorithm Program!
Anish Chauhan
Enter first number: 56
Enter second number: 98
GCD of 56 and 98 is 14
Thank you for using the Euclidean Program!
PS E:\Downloads\Lab 8th\Program code\Anish Chauhan>

Lab Program Number: 7

Title: Implementation of RSA Algorithm

The RSA Algorithm is a widely used public-key cryptographic system that enables secure data transmission. It uses two keys: a public key for encryption and a private key for decryption. The algorithm is based on the mathematical difficulty of factoring large prime numbers. RSA ensures confidentiality, authenticity, and data integrity, making it a fundamental part of modern secure communication.

Program:

```
import math, random
def is_prime(n):
    if n < 2: return False
    for i in range(2, int(math.sqrt(n))+1):
        if n % i == 0: return False
    return True

def generate_prime(min_val, max_val):
    p = random.randint(min_val, max_val)
    while not is_prime(p):
        p = random.randint(min_val, max_val)
    return p

def mod_inverse(e, phi):
    for d in range(3, phi):
        if (d * e) % phi == 1: return d
    raise ValueError("mod_inverse does not exist")

def generate_keypair():
    p, q = generate_prime(100, 1000), generate_prime(100, 1000)
    while p == q: q = generate_prime(100, 1000)
    n, phi = p*q, (p-1)*(q-1)
    e = random.randint(3, phi-1)
    while math.gcd(e, phi) != 1: e = random.randint(3, phi-1)
    d = mod_inverse(e, phi)
```

```
    return (e, n), (d, n)

def encrypt(pub, plaintext):
    e, n = pub
    return [pow(ord(c), e, n) for c in plaintext]

def decrypt(priv, ciphertext):
    d, n = priv
    return ''.join([chr(pow(c, d, n)) for c in ciphertext])

if __name__ == "__main__":
    pub, priv = generate_keypair()
    print("Welcome to the RSA Cipher Program!")
    print("Anish Chauhan\n")

    print(f"Public key: {pub}")
    print(f"Private key: {priv}\n")

    text = input("Enter the text to encrypt: \n")
    if not text: print("Error: No text provided.")
    else:
        print(f"Plaintext: {text}")
        encrypted_msg = encrypt(pub, text)
        print(f"Encrypted: {encrypted_msg}\n")
        decrypted_msg = decrypt(priv, encrypted_msg)
        print(f"Decrypted: {decrypted_msg}")
        print("Thank you for using the RSA Cipher Program!")
```

Output:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS E:\Downloads\Lab 8th\Program code\Anish Chauhan> python rsa_algo.py
Welcome to the RSA Cipher Program!
Anish Chauhan
Public key: (154967, 260233)
Private key: (44903, 260233)
Enter the text to encrypt:
Naruto Uzumaki
Plaintext: Naruto Uzumaki
Encrypted: [227227, 238081, 95874, 16889, 17412, 126194, 34079, 42505, 113018, 16889, 185880, 238081, 208683, 73515]
Decrypted: Naruto Uzumaki
Thank you for using the RSA Cipher Program!
PS E:\Downloads\Lab 8th\Program code\Anish Chauhan>

Lab Program Number: 8

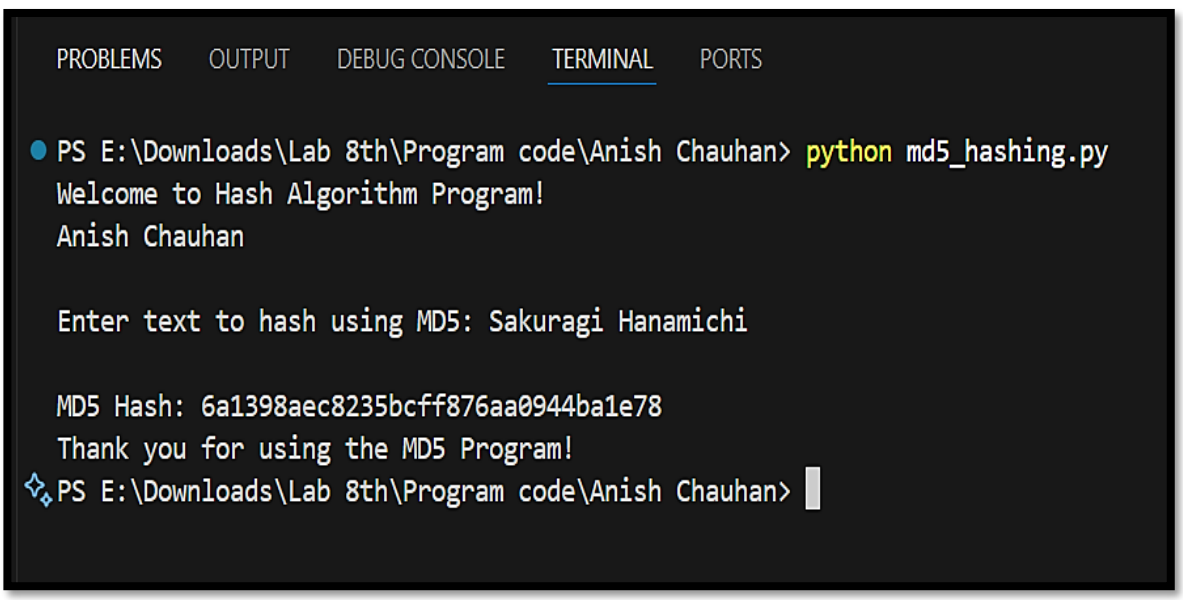
Title: Implementation of MD5 Hashing Algorithm

The MD5 (Message Digest 5) algorithm is a widely used cryptographic hash function that produces a 128-bit (16-byte) hash value. It takes an input (such as text) and returns a fixed-size hexadecimal string, often used to verify data integrity. However, due to vulnerabilities, MD5 is no longer considered secure for cryptographic purposes but remains useful for checksums and data verification.

Program:

```
import hashlib

print("Welcome to Hash Algorithm Program!")
print("Anish Chauhan\n")

text = input("Enter text to hash using MD5: ")
md5_hash = hashlib.md5(text.encode()).hexdigest()

print("\nMD5 Hash:", md5_hash)
print("Thank you for using the MD5 Program!")
```

Output:



Lab Program Number: 9

Title: Preparing Linux lab Environment

VMware Workstation: A desktop hypervisor application for Windows (and Linux) that lets you run virtual machines (VMs) on your computer. Each VM behaves like a separate computer where you can install an operating system such as AlmaLinux.

AlmaLinux is a free and open-source Linux distribution developed by the AlmaLinux OS Foundation. It is designed to be a binary-compatible replacement for Red Hat Enterprise Linux (RHEL), making it ideal for enterprise servers, cloud environments, and educational labs. After Red Hat shifted CentOS to a rolling-release model (CentOS Stream), AlmaLinux emerged as a stable, community-driven alternative that provides long-term support, regular security updates, and reliability for both developers and system administrators.

1. **Installing VMware Workstation:**
   - Download the VMware Workstation installer from the official VMware website.
   - Double-click the setup file and follow the on-screen instructions.
   - Accept the license agreement and click Next until installation completes.
   - Click Finish when done and open VMware Workstation.

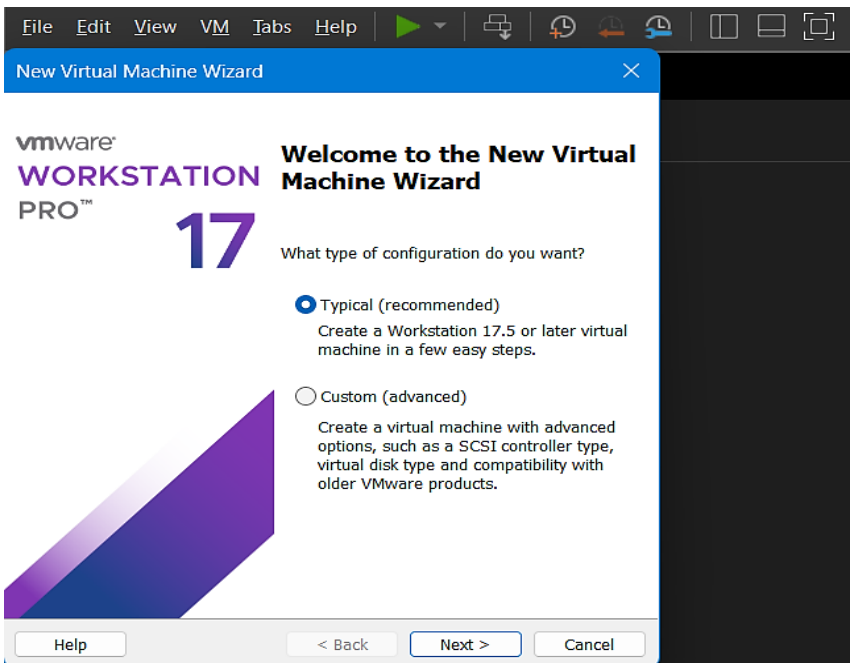


Fig: VMware installation screen

2. Creating a Virtual Machine for AlmaLinux:

- Open VMware Workstation then Click Create a New Virtual Machine.
- Choose Typical (recommended) then click on Next.
- Select Installer disc image file (ISO). Browse and select the downloaded AlmaLinux.
- Allocate at least 2 GB RAM and 20 GB disk space.

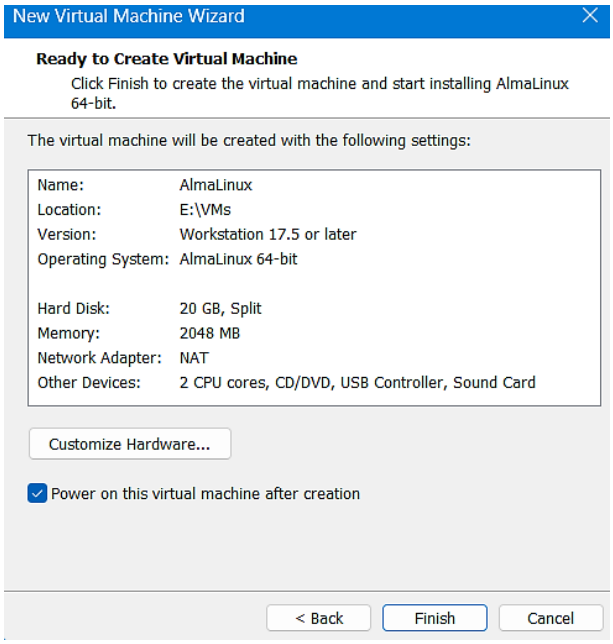


Fig: Creating new VM wizard

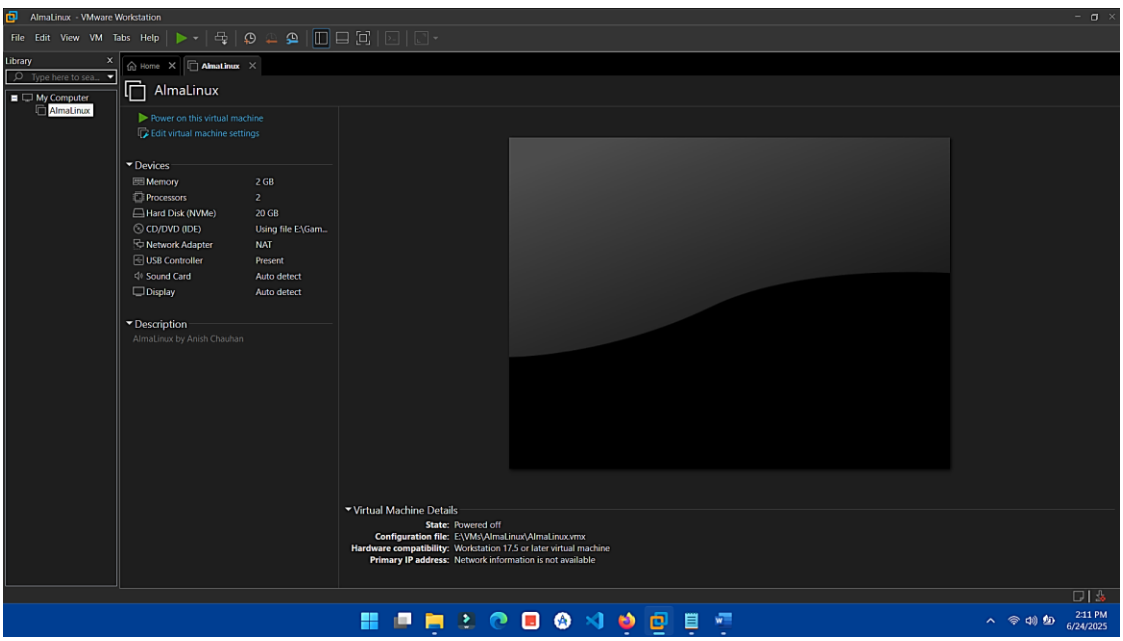
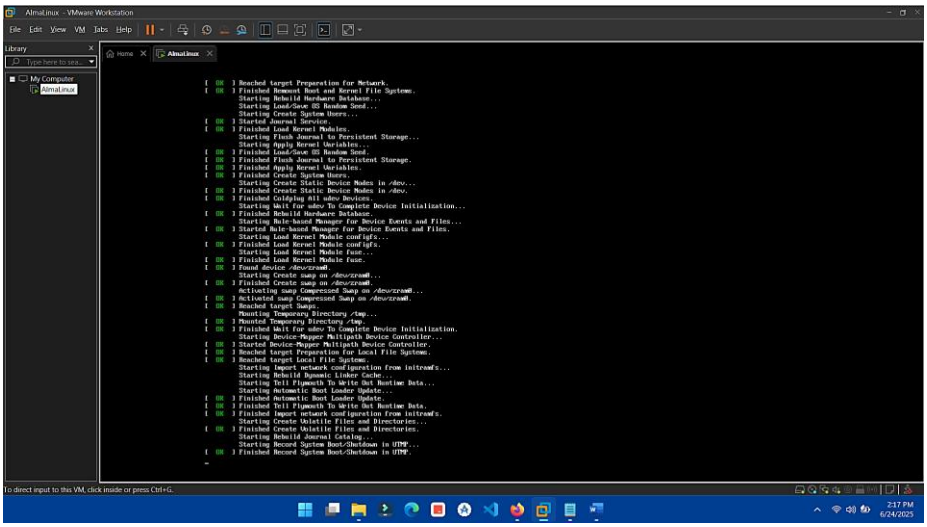


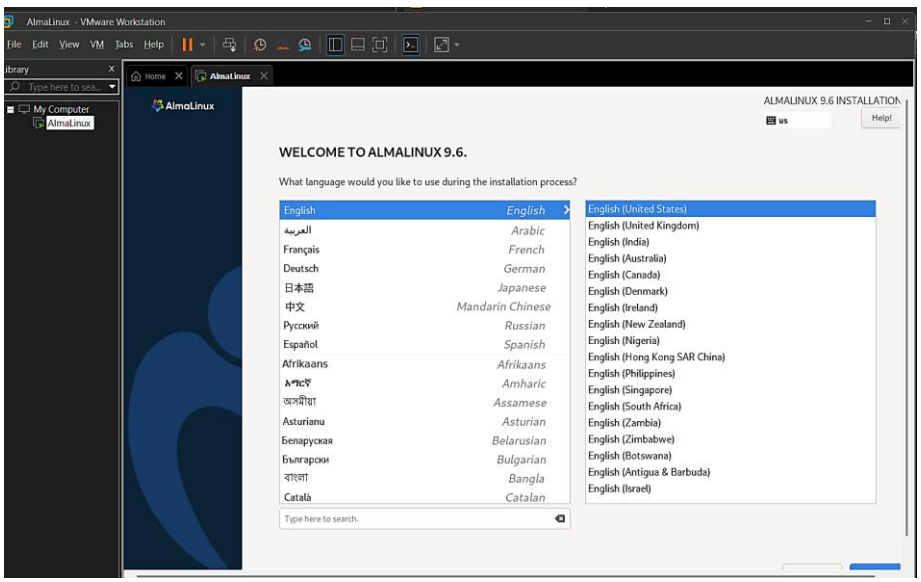
Fig: AlmaLinux on VM

3. Installing AlmaLinux on the Virtual Machine:

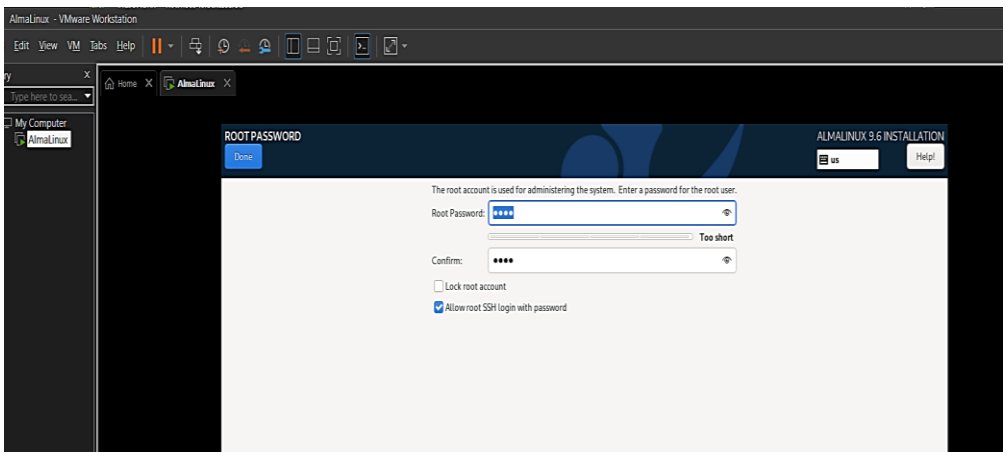
- Power on the new VM.



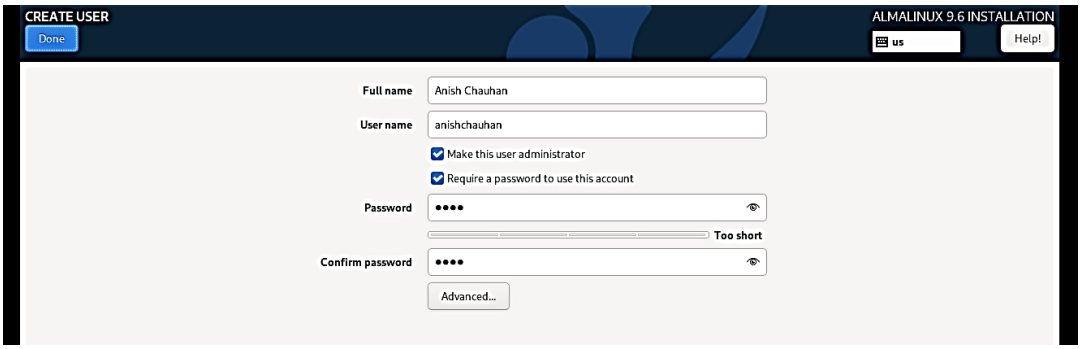
- Choose preferred language and keyboard layout.

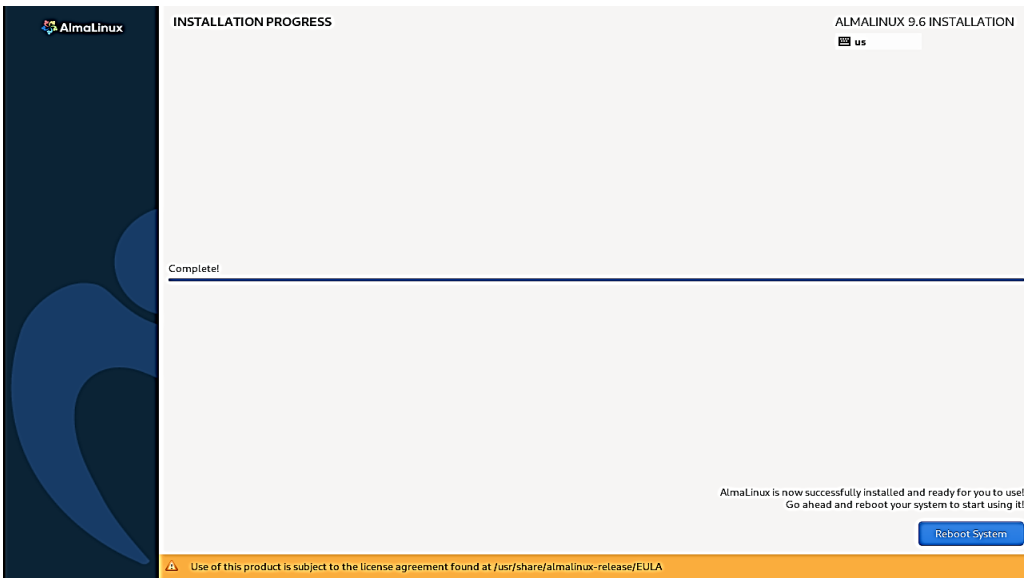


- Set Root Password and create a User Account.



- Click Begin Installation, then Reboot when complete.





4. Assigning Hostname to the Linux Machine:

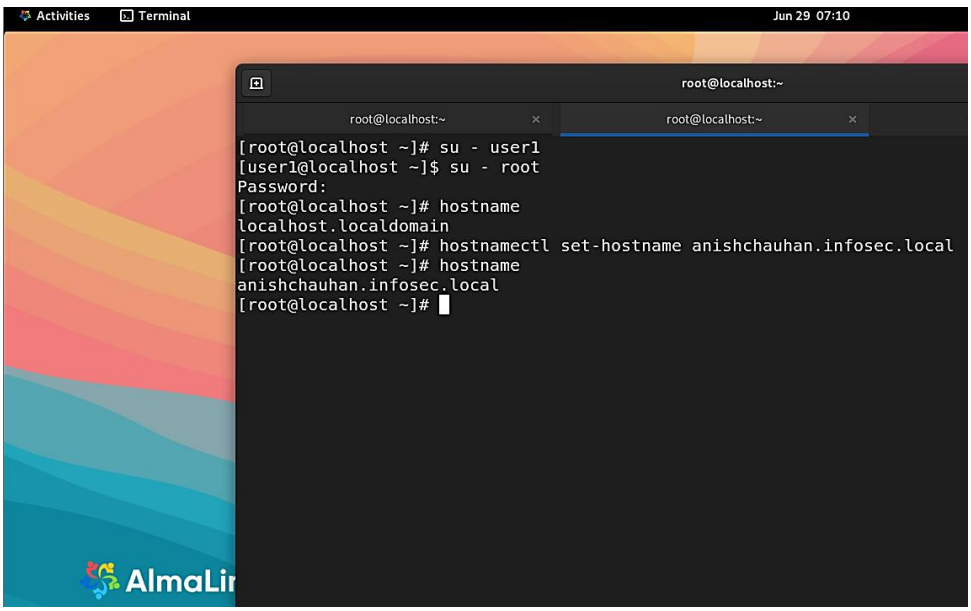


Fig: Hostname set using terminal

5. **Verifying the Configuration:**

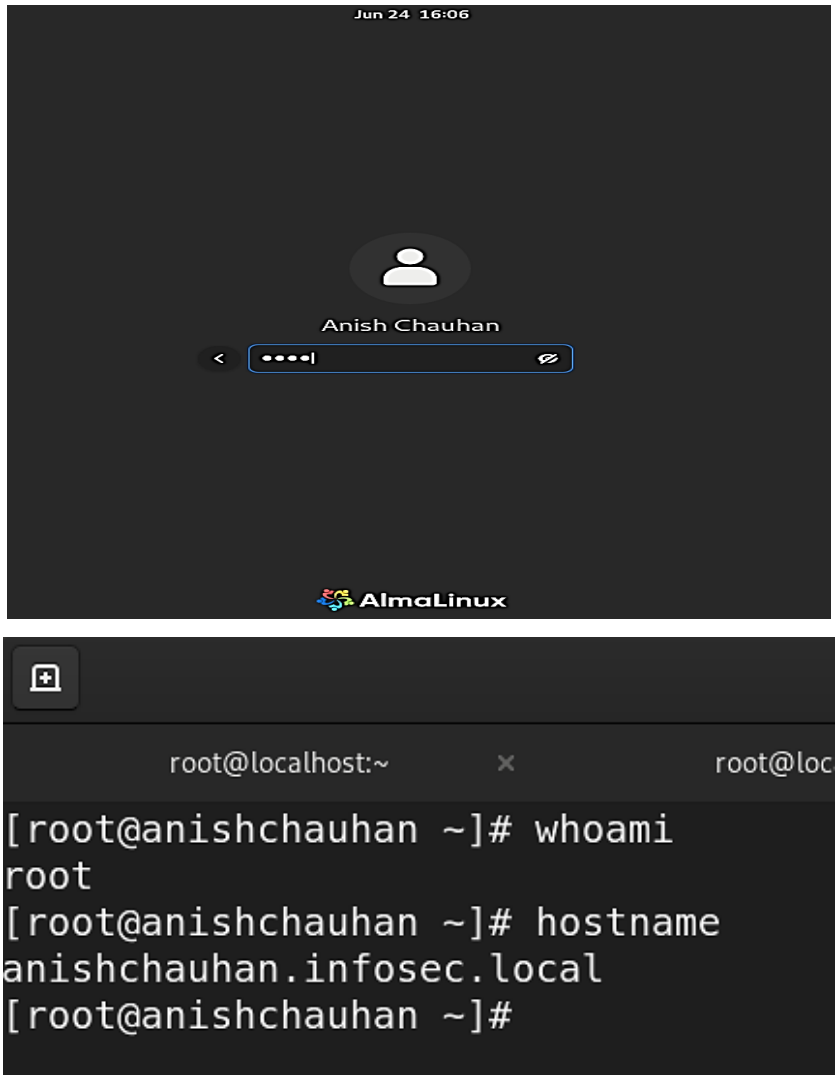


Fig: Verification of hostname and OS

Conclusion:

We successfully prepared a Linux lab environment using VMware Workstation and AlmaLinux. The steps included installing VMware, creating a virtual machine, installing the Linux OS, and setting the hostname. This setup provides a safe environment to practice Linux commands, networking, and cybersecurity tasks.

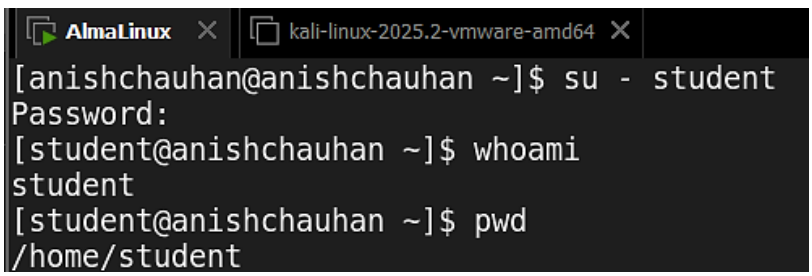
Lab Program Number: 10

Title: Users, group Permission.

In Linux, users and groups control who can access or modify files and directories. Permissions define what actions (read, write, execute) each user or group can perform. This lab demonstrates how to create a new user, make directories and files, and set permissions using the chmod command.

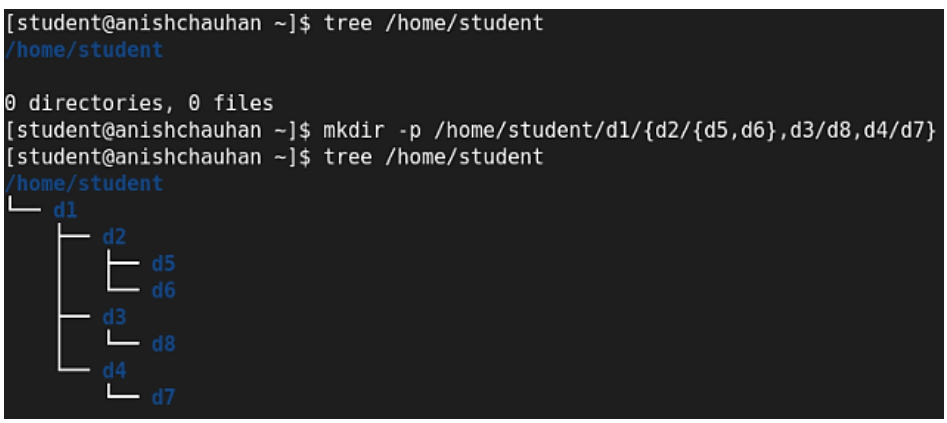
Creating a Linux User and Directory Structure:

1. **Creating a User Student:**
   - We create a new user called student to work within this lab.
   - Use command: **useradd -m student**
2. **Switching to student User:**
   - As logged as the student user, check the current working directory.
   - Use command: **su – student, whoami, pwd**.



Creating Directories and Files:

1. **Creating Directory Structure:**
   - We will create the directory structure as shown using the mkdir command.
   - Command**: mkdir -p /home/student/d1/d2/{d5,d6}**
     /home/student/d1/{d3/d8,d4/d7}



2. Creating Files:

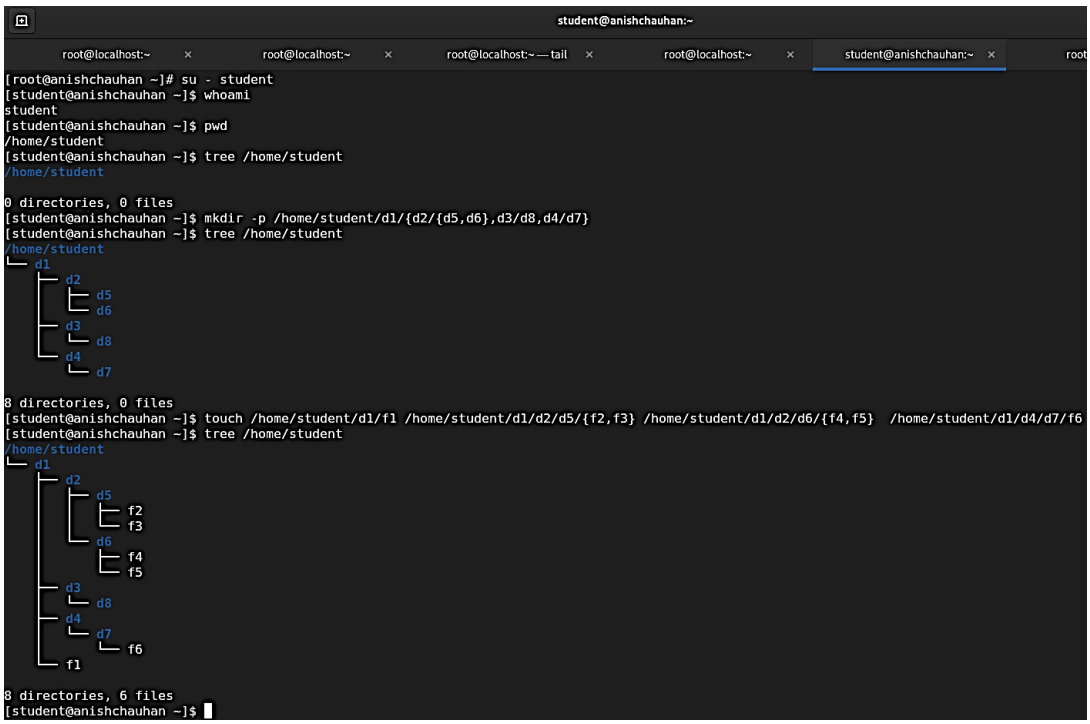
- Creating necessary files inside each of these directories.
- Command:

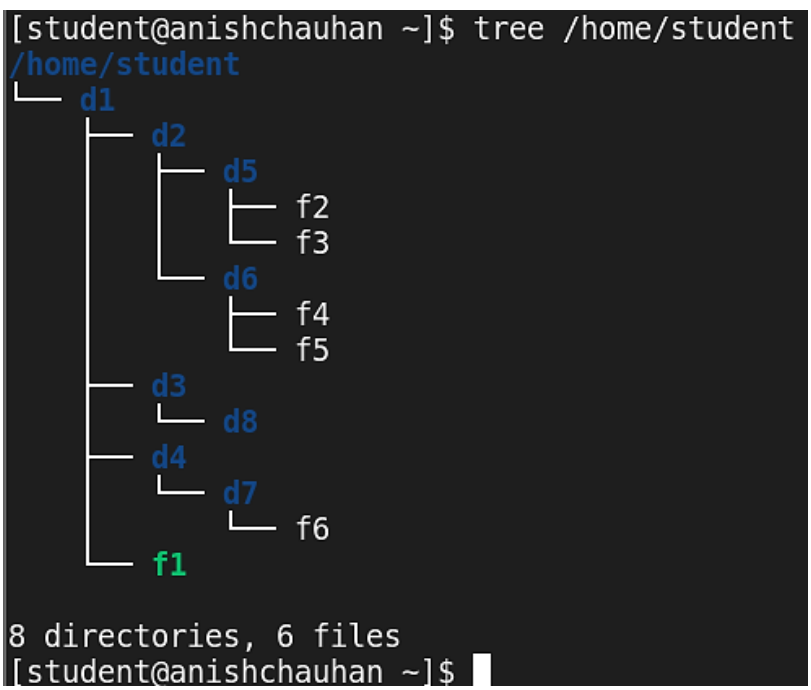
**touch /home/student/d1/f1 **

**/home/student/d1/d2/d5/{f2,f3} **

**/home/student/d1/d2/d6/{f4,f5} **

/home/student/d1/d4/d7/f6





Setting Permissions on Files and Directories:

a. File f1:

- Owner has full permission, Group has read and execute, Others have read-only.
- Command:

chmod 754 /home/student/d1/f1,

ls -l /home/student/d1/f1

```
[student@anishchauhan ~]$ chmod 754 /home/student/d1/f1
[student@anishchauhan ~]$ ls -l /home/student/d1/f1
-rwxr-xr--. 1 student student 0 Jun 29 07:52 /home/student/d1/f1
```

Fig: Verification of file f1

- **rwx** = Owner (student) has read, write, execute.
- **r-x** = Group (student) has read and execute.
- **r--** = Others have read-only.

b. File f2:

- Owner and Group have read and write; Others have no permission.
- Command:

chmod 660 /home/student/d1/d2/d5/f2

ls -l /home/student/d1/d2/d5/f2

```
[student@anishchauhan ~]$ chmod 660 /home/student/d1/d2/d5/f2
[student@anishchauhan ~]$ ls -l /home/student/d1/d2/d5/f2
-rw-rw----. 1 student student 0 Jun 29 07:52 /home/student/d1/d2/d5/f2
[student@anishchauhan ~]$
```

Fig: Verification of file f2

- **rw-** = Owner (student) can read and write, cannot execute.
- **rw-** = Group members can read and write, cannot execute.
- --- = other has no permission.

c. **Directory d3:**

- All users have full permission.
- Command:

chmod 777 /home/student/d1/d3

ls -ld /home/student/d1/d3

```
[student@anishchauhan ~]$ chmod 660 /home/student/d1/d2/d5/f2
[student@anishchauhan ~]$ ls -l /home/student/d1/d2/d5/f2
-rw-rw----. 1 student student 0 Jun 29 07:52 /home/student/d1/d2/d5/f2
[student@anishchauhan ~]$ chmod 777 /home/student/d1/d3
[student@anishchauhan ~]$ ls -ld /home/student/d1/d3
drwxrwxrwx. 3 student student 16 Jun 29 07:48 /home/student/d1/d3
[student@anishchauhan ~]$
```

Fig: Verification of directory d3

- **d** = This indicates it is a directory.
- **rwx** = Owner (student) can read, write, execute.
- **rwx** = Group member can read, write, execute.
- **rwx** = Everyone else can rea, write, execute.

Conclusion:

In this lab, we successfully created a new Linux user named student and switched to that user. We set up the required directory structure under /home/student and created all the necessary files inside these directories. Specific permissions were applied to individual files and directories using the chmod command, allowing us to control access for the owner, group members, and others. This practical exercise enhanced our understanding of Linux file system permissions, user management, and access control in a secure environment.

Lab Program Number: 11

Title: Users, Group Administration.

In Linux, users and groups are used to manage access control and file permissions efficiently. A user represents an individual account, while a group is a collection of users who share similar permissions. Using groups, administrators can easily assign and control access for multiple users in different departments or roles.

In this lab, we will create users, groups, and departmental directories. We will also assign ownership and permissions so that only specific users and groups can access their respective directories.

A. Create groups for each department:

- We need to create three groups: "production, marketing, and sales".
- Commands**:**

```
sudo groupadd production
sudo groupadd marketing
sudo groupadd sales
```

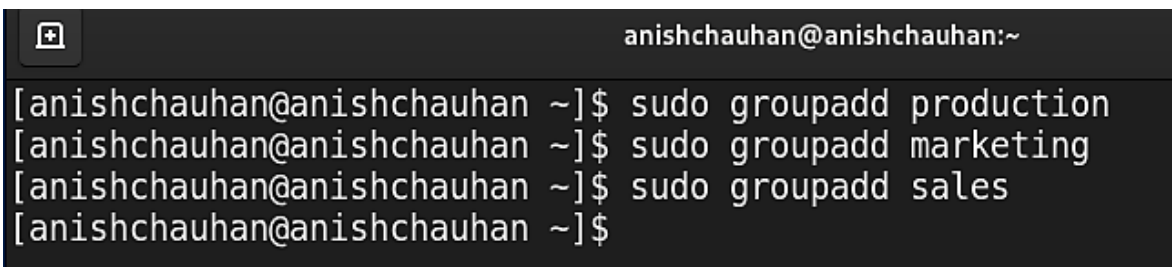


Fig: Groups created successfully

B. Create user accounts and assign them to groups:

- Create user1 and user2 and assign them to the production group.
- Create user3 and user4 and assign them to the marketing group.
- Create user5 and user6 and assign them to the sales group.
- Create manager and boss and assign them to the management group.
- Command:

```
# Production group users
sudo useradd -m -G production user1
sudo useradd -m -G production user2
# Marketing group users
sudo useradd -m -G marketing user3
```

sudo useradd -m -G marketing user4

 # Sales group users

sudo useradd -m -G sales user5

sudo useradd -m -G sales user6

 # Manager and Boss users

sudo useradd -m manager

sudo useradd -m boss

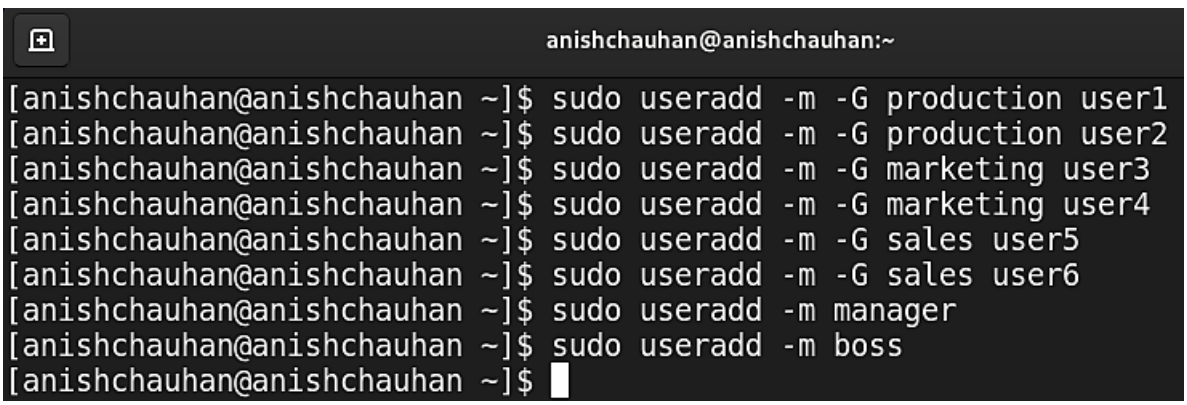


Fig: Users created and assigned to groups

C. Create common directories foe each department:

- Create directories under the root folder for each department.
- Commands:

  sudo mkdir /root/production

  sudo mkdir /root/marketing

  sudo mkdir /root/sales

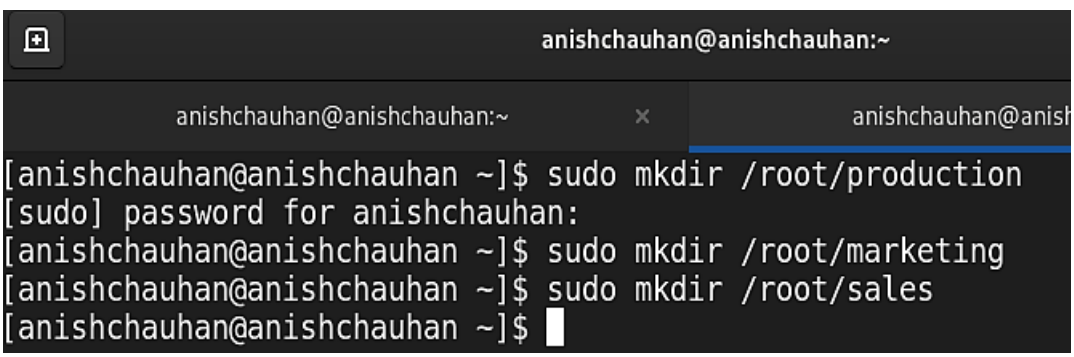


Fig: Department directories created

D. Change ownership of directories:

- Now we make boss the owner of all directories and assign each respective group as the group owner.
- Command:

```
sudo chown boss:production /root/production
sudo chown boss:marketing /root/marketing
sudo chown boss:sales /root/sales
```

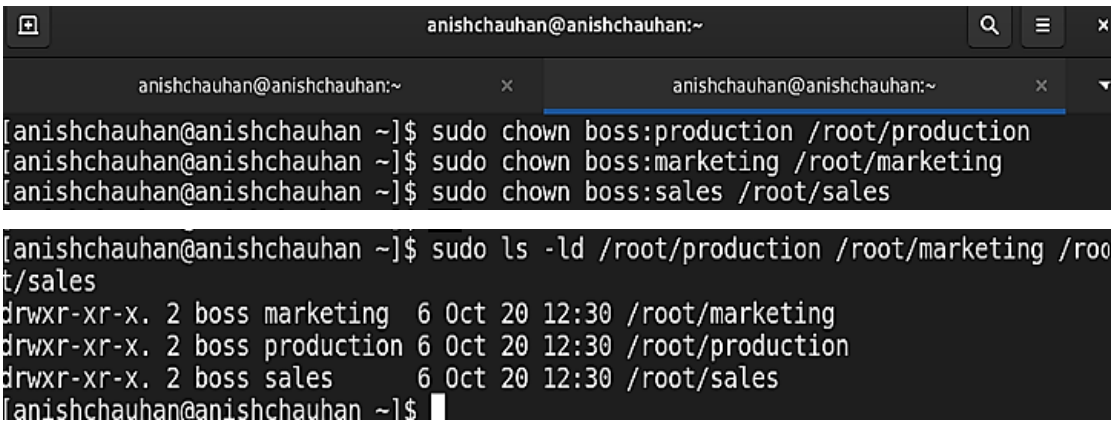


Fig: Ownership changed for departmental directories

E. Set permissions for each group directory:

- We give full permission (read, write, execute) to owner (boss) and group members, and no permission to others.
- Command:

```
sudo chmod 770 /root/production
sudo chmod 770 /root/marketing
sudo chmod 770 /root/sales
```

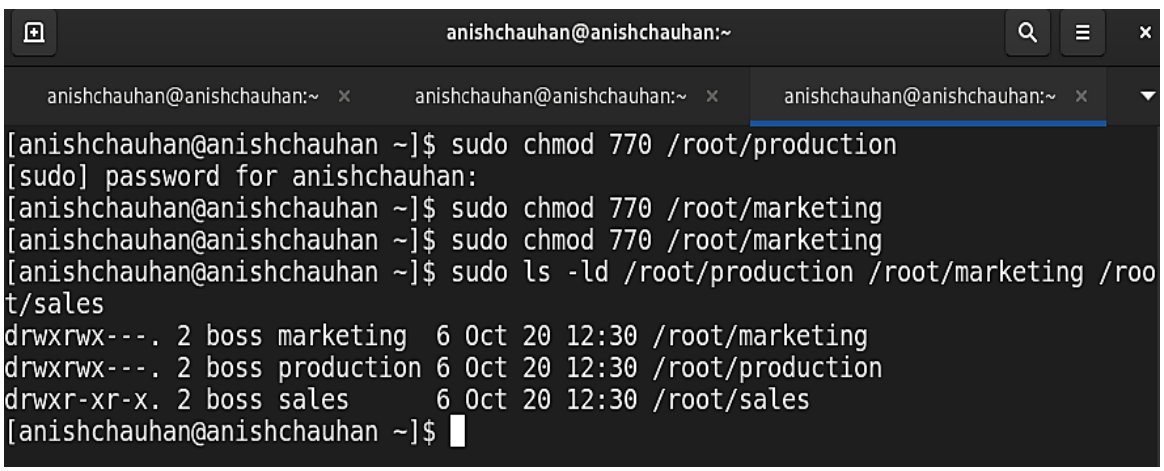


Fig: Permissions set for group directories

Lab Program Number: 12

Title: Firewall Configuration.

A **firewall** is a network security system that acts as a barrier between a trusted internal network and untrusted external networks such as the internet. It monitors and filters both incoming and outgoing network traffic based on predefined security rules. The main purpose of a firewall is to protect the system from unauthorized access, attacks, or data leaks by allowing only safe and necessary communication.

In Linux, the **firewalld** service provides a dynamic way to manage firewall rules. It supports network zones, service-based filtering, and runtime or permanent configurations without needing to restart the firewall. Administrators use firewalld to allow or block specific services (like HTTP, SMTP) and ports (like 25, 80, 110) for better control and system protection

Firewall Configuration:

a. Install firewalld package, start and enable firewall services:

- We must first install the firewalld package, start it, and enable it to run automatically at boot.
- Commands:

```
yum -y install firewalld
systemctl status firewalld
systemctl start firewalld
systemctl enable firewalld
systemctl enable --now firewalld
systemctl status firewalld
```

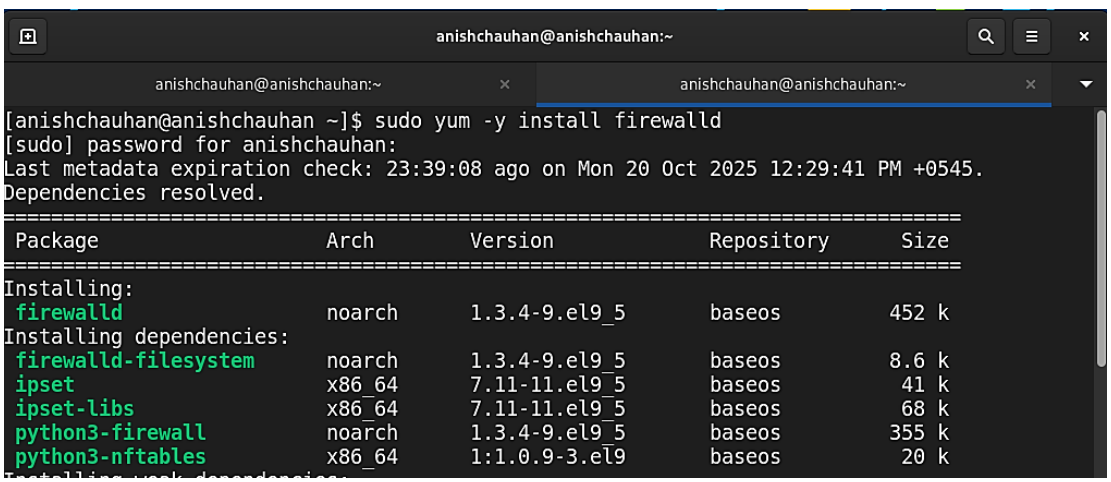


Fig: Installation of firewalld

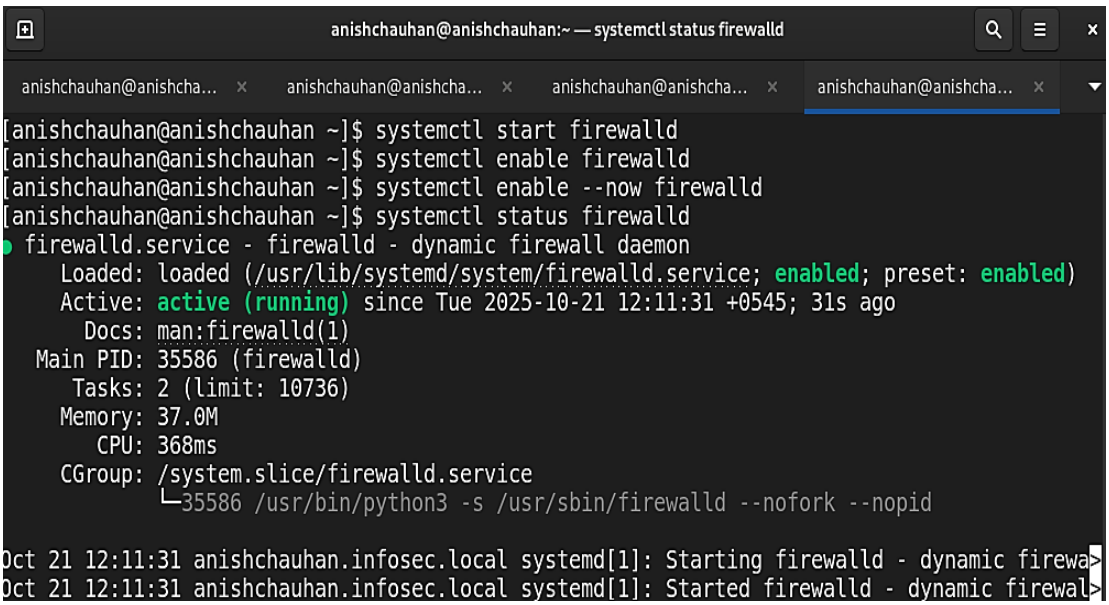


Fig: Installation and enabling of firewalld service

b. Add services and ports to allow packets through the firewall:

- Allow Service: http, smtp.
- Allow Ports: 25/tcp, 25/udp, 110/tcp.
- Commands:

```
firewall-cmd --permanent --add-service=http
firewall-cmd --permanent --add-service=smtp
firewall-cmd --permanent --add-port=25/tcp
firewall-cmd --permanent --add-port=25/udp
firewall-cmd --permanent --add-port=110/tcp
```

```
[anishchauhan@anishchauhan ~]$ sudo firewall-cmd --permanent --add-service=http
Warning: ALREADY_ENABLED: http
success
[anishchauhan@anishchauhan ~]$ firewall-cmd --permanent --add-service=smtp
success
[anishchauhan@anishchauhan ~]$ firewall-cmd --permanent --add-port=25/tcp
success
[anishchauhan@anishchauhan ~]$ firewall-cmd --permanent --add-port=25/udp
success
[anishchauhan@anishchauhan ~]$ firewall-cmd --permanent --add-port=110/tcp
success
```

Fig: Added services and ports in firewall

```
[anishchauhan@anishchauhan ~]$ sudo firewall-cmd --list-all
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: ens160
  sources:
  services: cockpit dhcpv6-client http smtp ssh
  ports: 110/tcp 25/tcp 25/udp
  protocols:
  forward: yes
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

Fig: Allowed services and ports successfully

c. Remove services and ports to block packets through the firewall:

- Remove service: smtp.
- Remove ports: 25/tcp, 25/udp.
- Commands:

firewall-cmd --permanent --remove-service=smtp

firewall-cmd --permanent --remove-port=25/tcp

firewall-cmd --permanent --remove-port=25/udp

```
anishchauha... ×  anishchauha... ×  anishchauha... ×  anishchauha... ×  anishchauha... ×  anishchauha
[anishchauhan@anishchauhan ~]$ sudo firewall-cmd --permanent --remove-service=smtp
[sudo] password for anishchauhan:
success
[anishchauhan@anishchauhan ~]$ sudo firewall-cmd --permanent --remove-port=25/tcp
success
[anishchauhan@anishchauhan ~]$ sudo firewall-cmd --permanent --remove-port=25/udp
success
[anishchauhan@anishchauhan ~]$ firewall-cmd --reload
success
[anishchauhan@anishchauhan ~]$ sudo firewall-cmd --list-all
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: ens160
  sources:
  services: cockpit dhcpv6-client http ssh
  ports: 110/tcp
  protocols:
  forward: yes
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
[anishchauhan@anishchauhan ~]$
```

Fig: Removed SMTP service and blocked ports

d. Verify firewall status and configuration:

- Commands:

  firewall-cmd --reload

  firewall-cmd --state

  firewall-cmd --list-all

```
[anishchauhan@anishchauhan ~]$ firewall-cmd --state
running
[anishchauhan@anishchauhan ~]$ firewall-cmd --list-all
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: ens160
  sources:
  services: cockpit dhcpv6-client http ssh
  ports: 110/tcp
  protocols:
  forward: yes
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
[anishchauhan@anishchauhan ~]$
```

Fig: Verification of firewall status

Conclusion:

In this lab, we successfully installed and configured the firewalld service on a Linux system. We started and enabled the firewall to ensure it runs automatically, allowed specific services and ports to permit network access, and removed unwanted ones to enhance security. Finally, we verified the firewall configuration and learned how Linux firewalls help protect systems from unauthorized access while maintaining secure network communication.

Lab Program Number: 13

Title: Configuring SSH Server to allow/deny root login and allow/deny user login.

SSH (Secure Shell) is a secure network protocol used for remote system login, command execution, and file transfer. It ensures encryption between client and server, preventing unauthorized access. To protect the system further, administrators can control SSH access by managing root login permissions and defining which users are allowed or denied access through the SSH configuration file located at /etc/ssh/sshd_config.

Firewalld is also used to allow SSH traffic by enabling the SSH service port. Together, these configurations strengthen system security and ensure only trusted users can access the machine.

Configuring SSH Server to Allow/Deny Root Login and User Login:

a. **Start and enable firewall services:**

- Before configuring SSH access, ensure the firewall is active to manage incoming SSH traffic.
- Commands:

  systemctl start firewalld

  systemctl enable firewalld

  firewall-cmd --list-all

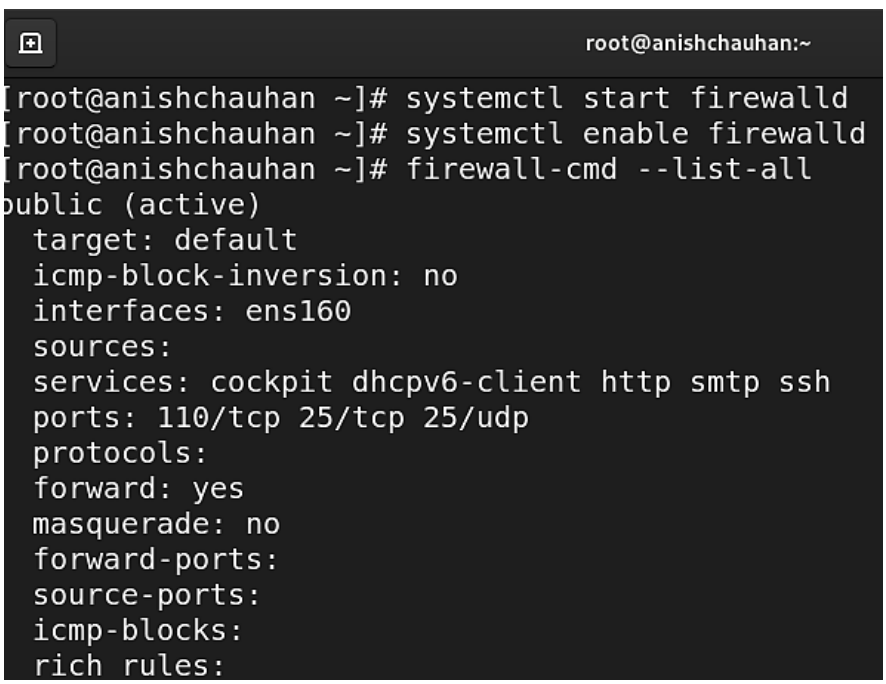


Fig: Starting and enabling of Firewall

b. Add required services to the firewall:

- Allow important services like SMTP and HTTP permanently.
- Command:

  firewall-cmd --permanent --add-service={smtp,http}

  firewall-cmd --list-all

Fig: Allowed SMTP and HTTP services

c. Install OpenSSH Server package:

- The server package is needed to run SSH services.
- Command:

  yum -y install openssh-server

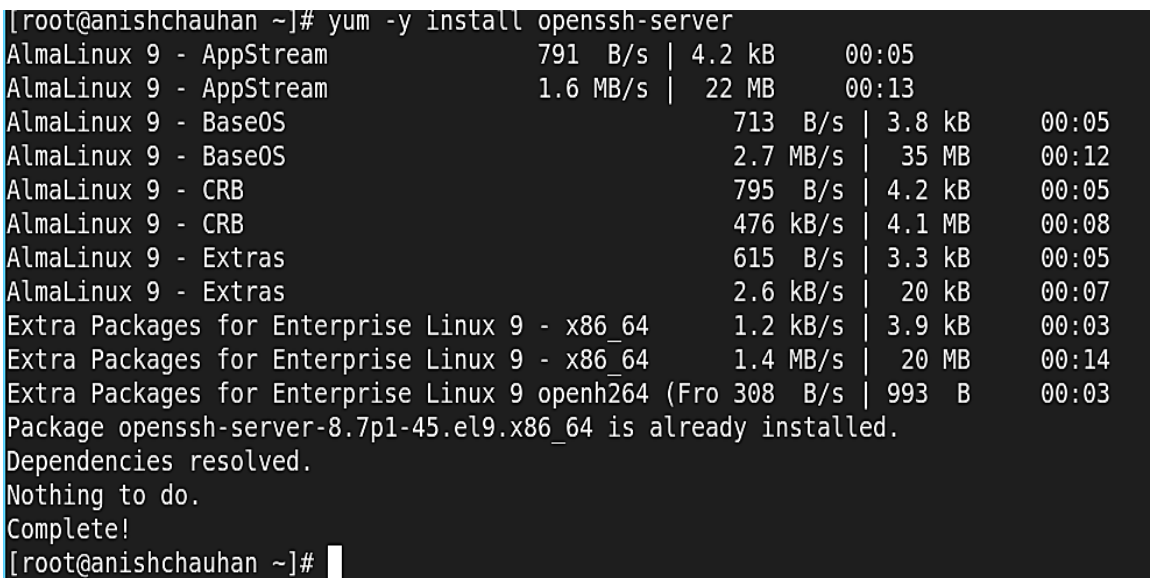


Fig: Installation of OpenSSH server

d. Allow SSH service through the firewall:

- Enable SSH service so remote users can connect.
- Command:

  firewall-cmd --permanent --add-service=ssh

  firewall-cmd --reload

  firewall-cmd --list-all

```
[root@anishchauhan ~]# firewall-cmd --permanent --add-service=ssh
Warning: ALREADY_ENABLED: ssh
success
[root@anishchauhan ~]# firewall-cmd --reload
success
[root@anishchauhan ~]#
```

Fig: SSH service added to firewall

e. Check network routes:

- Used to verify routing and network paths.
- Command:

  route -n

  ip route

```
[root@anishchauhan ~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.126.2   0.0.0.0         UG    100    0        0 ens160
192.168.126.0   0.0.0.0         255.255.255.0   U     100    0        0 ens160
[root@anishchauhan ~]# ip route
default via 192.168.126.2 dev ens160 proto dhcp src 192.168.126.128 metric 100
192.168.126.0/24 dev ens160 proto kernel scope link src 192.168.126.128 metric 100
[root@anishchauhan ~]#
```

Fig: Network routing table output

f. Access SSH configuration directory:

- SSH settings are stored in /etc/ssh.
- Command:

  cd /etc/ssh

  ls

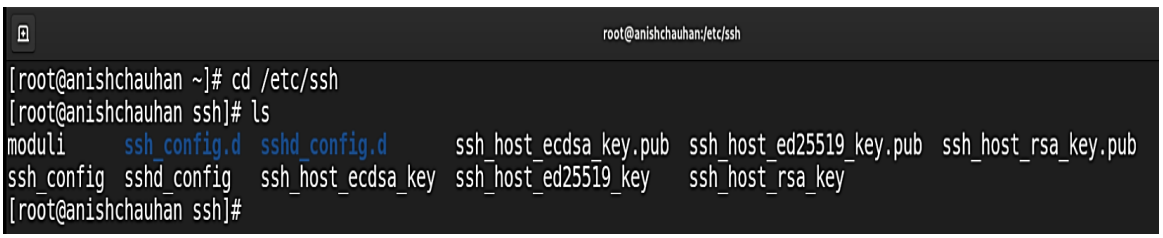


Fig: SSH configuration files listed

g. Configure Root Login (Allow/Deny):

- The directive PermitRootLogin controls whether root is allowed to log in through SSH.
- Command:

  vim sshd_config

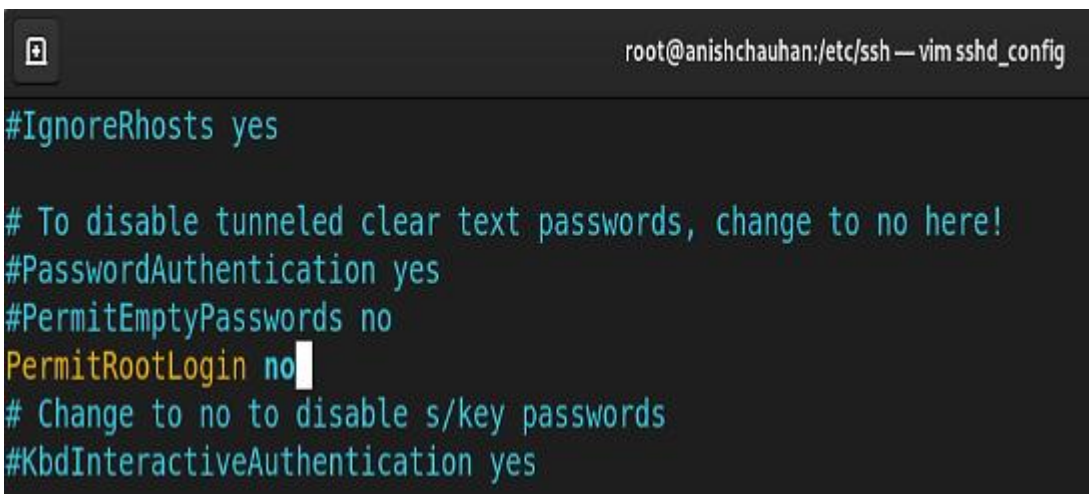


Fig: Editing PermitRootLogin directive

h. Allow or Deny Users:

- Command:

  vim sshd_config

  AllowUsers alice bob

  DenyUsers david

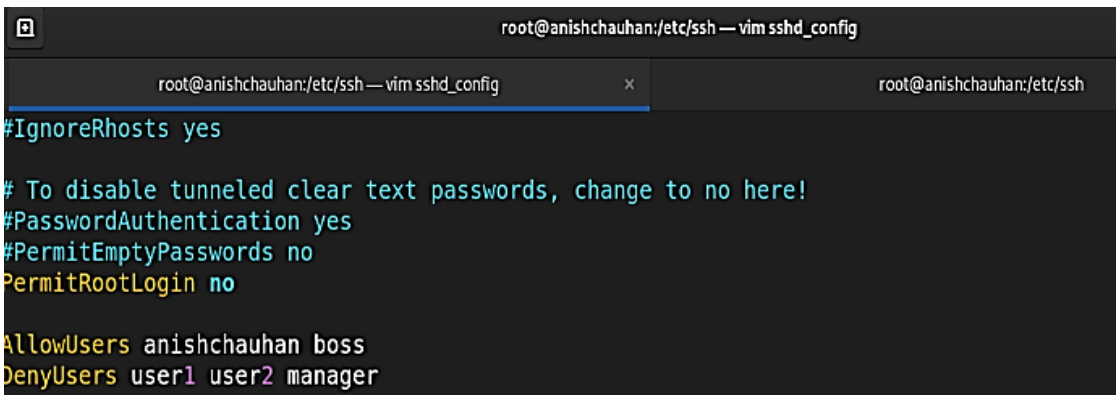


Fig: Adding AllowUsers and DenyUsers directives

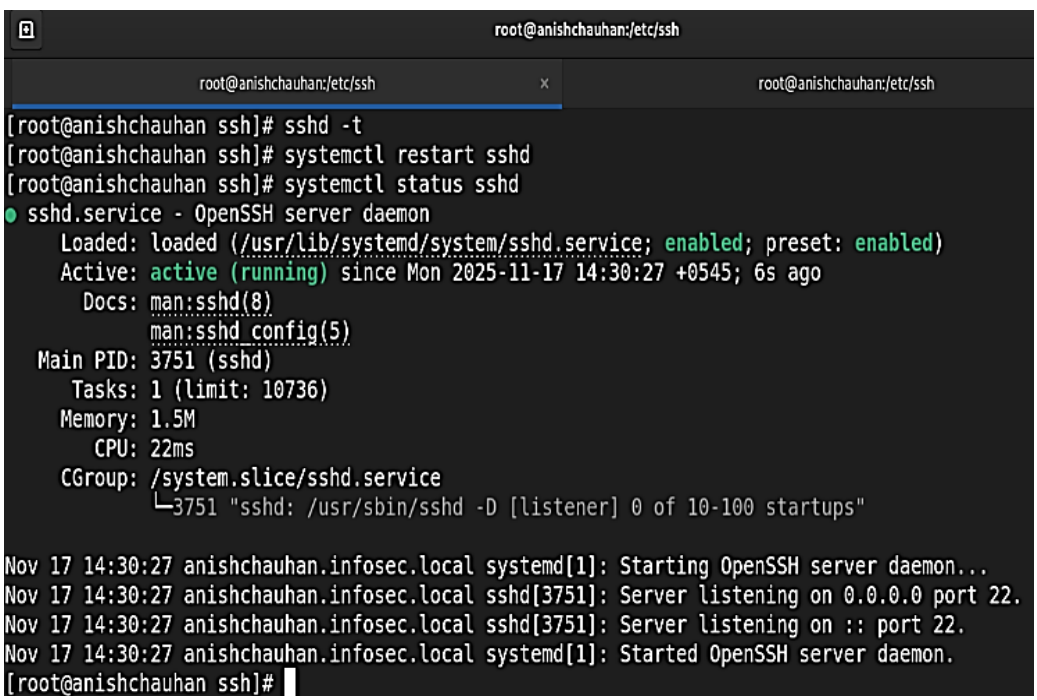
i. Verify configuration:

- Commands:

  sshd -t

  systemctl restart sshd

  systemctl status sshd



Lab Program Number: 14

Title: Configuring SSH Server to allow/deny SSH login from selected hosts only.

a. Configure OpenSSH server to deny all hosts except the host (192.168.10.10).

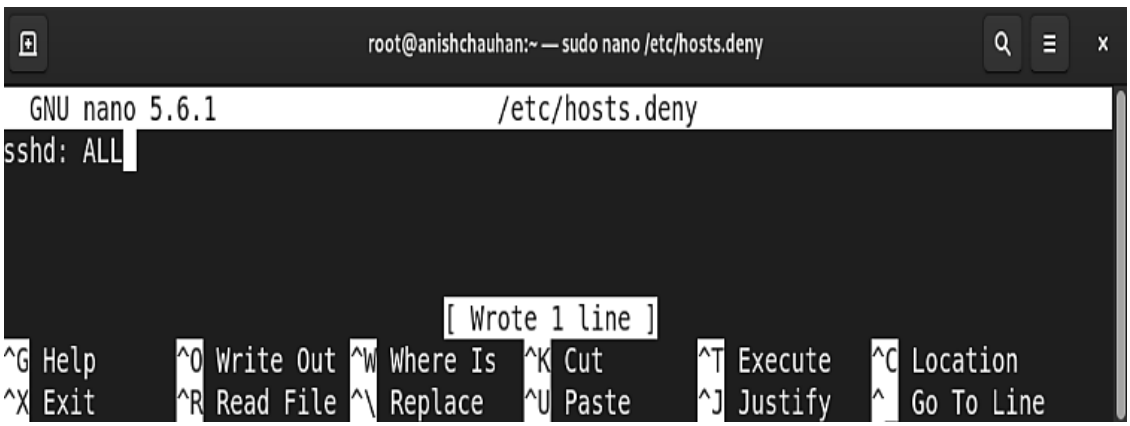


Fig: Configuring SSH server to deny all hosts

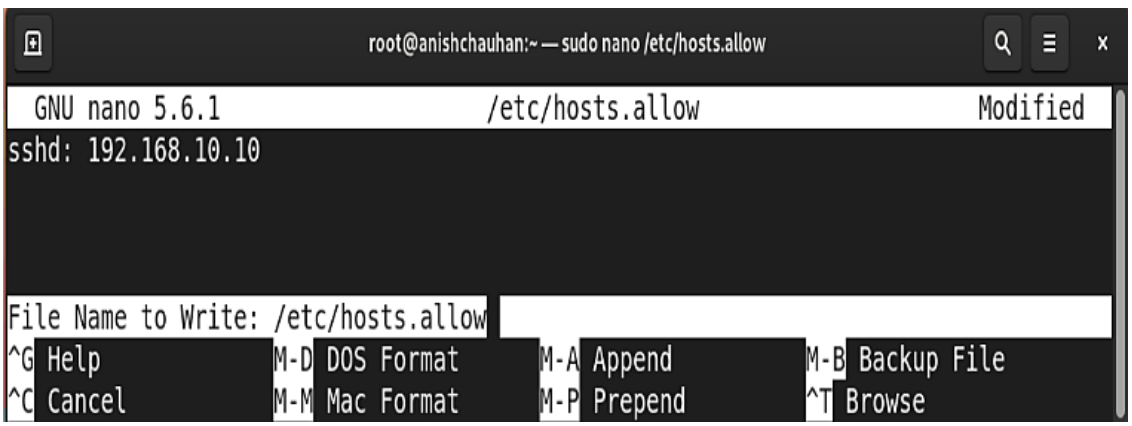
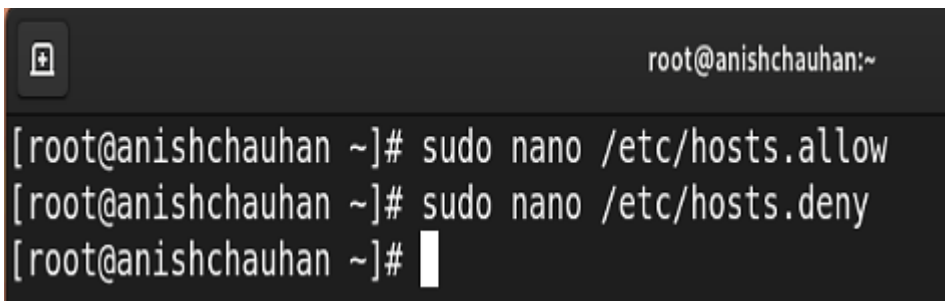


Fig: Commands for allowing or denying SSH login



Lab Program Number: 15

Title: Configuring SSH Server to allow/deny SSH login from selected hosts only.

a. Generate SSH keypair in localhost.

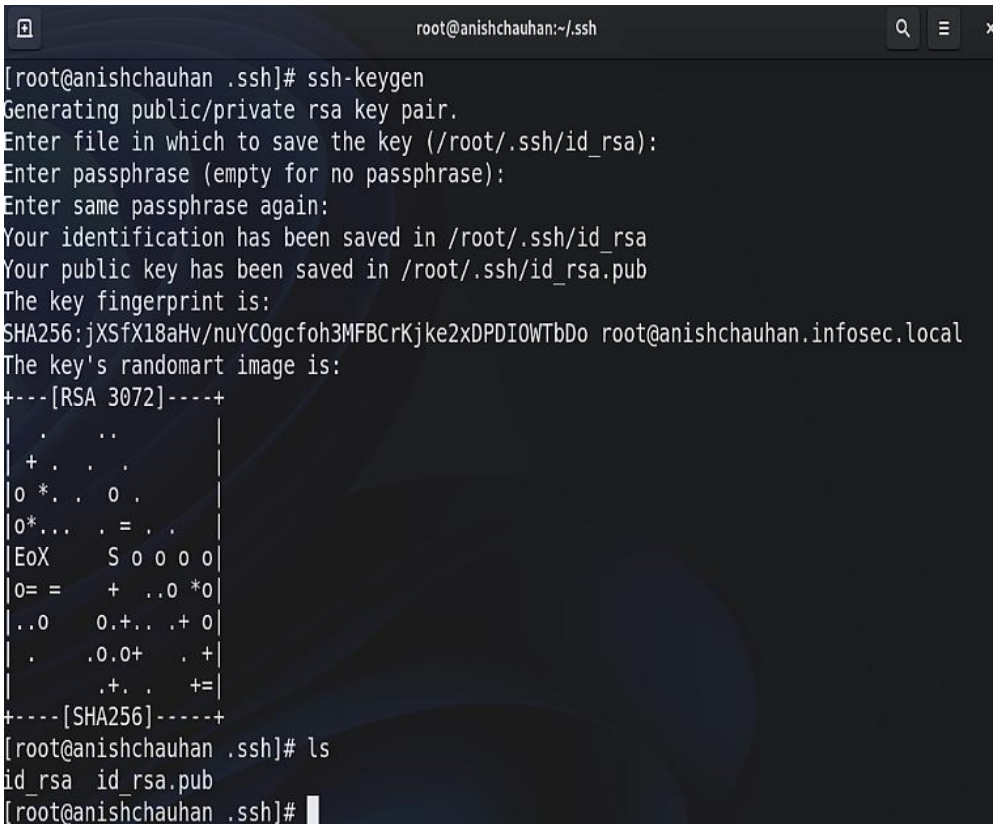


Fig: Generating public and private keys

b. Generate SSH keypair in localhost.

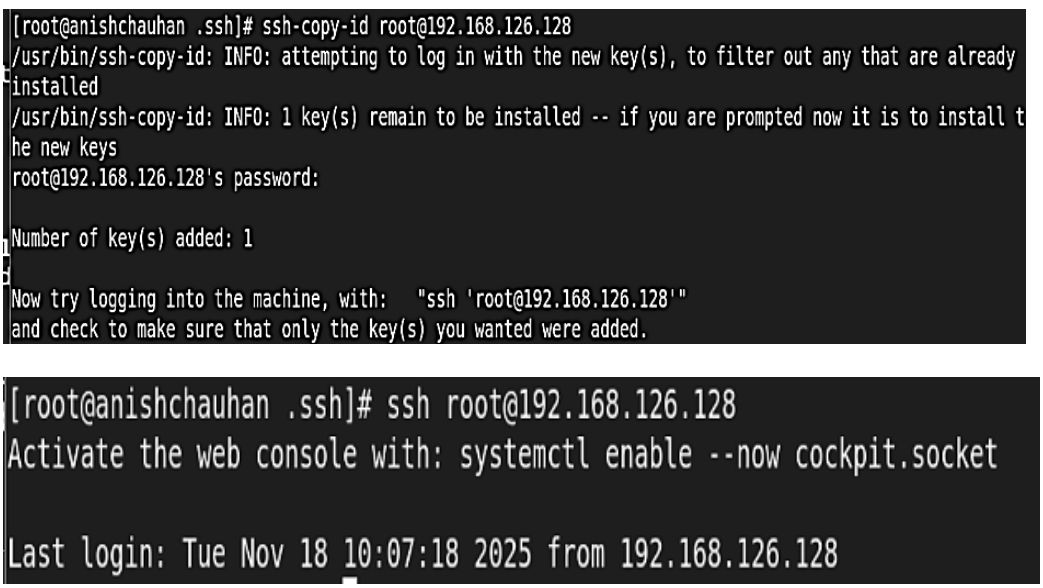


Fig: Direct login without password

Lab Program Number: 16

Title: Secure Network Copy using "SCP"

A. Create files on remote system (192.168.126.128):

```
ssh root@192.168.126.128
cd /root
echo "This is Anish_OP file on remote host" > Anish_OP.md
ls -l
tree
```

Fig: File on the remote system

B. Create file on the local system (/home/student):

```
cd /home/student
echo "This is Chauhan file on local system" > Chauhan.md
ls -l
tree
```

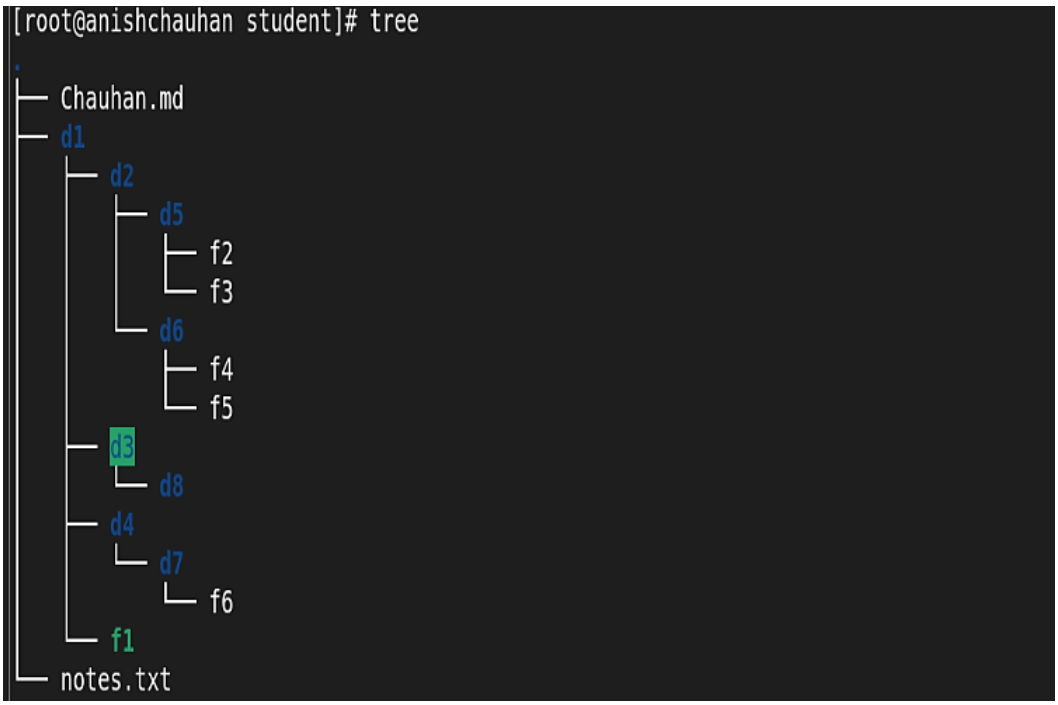


Fig: File on the local system

C. Copy Remote file into Local system:

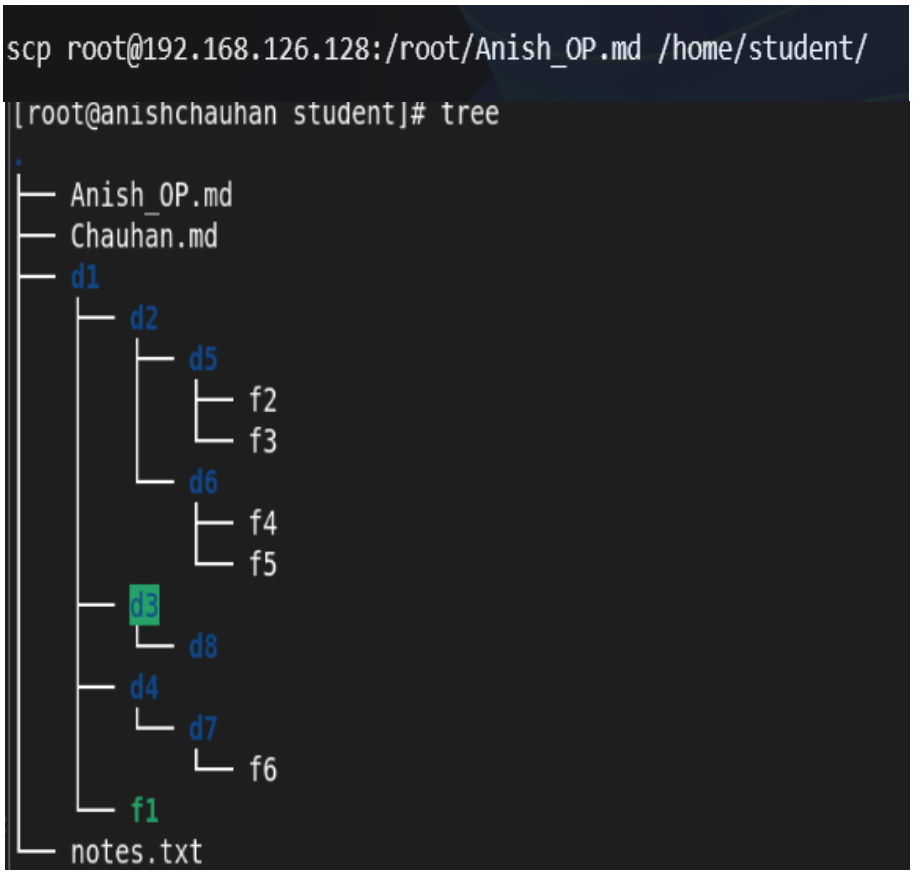


Fig: Copying file on the local system

D. Copy Local file into Remote system:

```
scp /home/student/Chauhan.md root@192.168.126.128:/root/
```

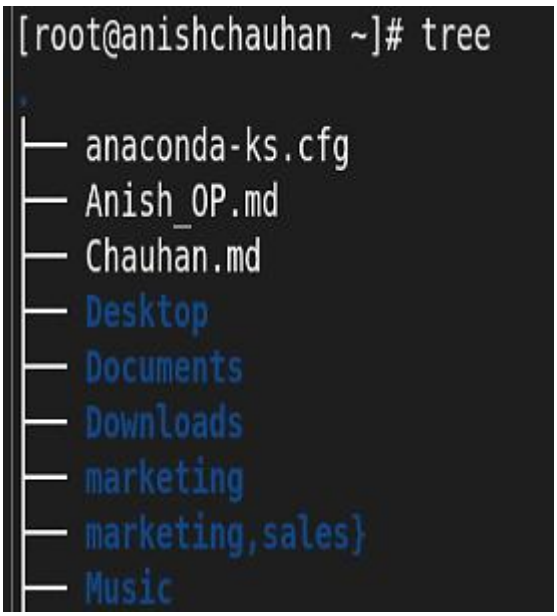


Fig: Copying file on the Remote system

Lab Program Number: 17

Title: Security Enhanced Linux (SE Linux)

Security Enhanced Linux (SE Linux) is a security architecture integrated into Linux systems that provides mandatory access controls (MAC). It restricts programs, processes, and users to access only the files and resources they are explicitly allowed to, adding an extra layer of security beyond traditional Linux permissions.

A. Checking SE Linux status:

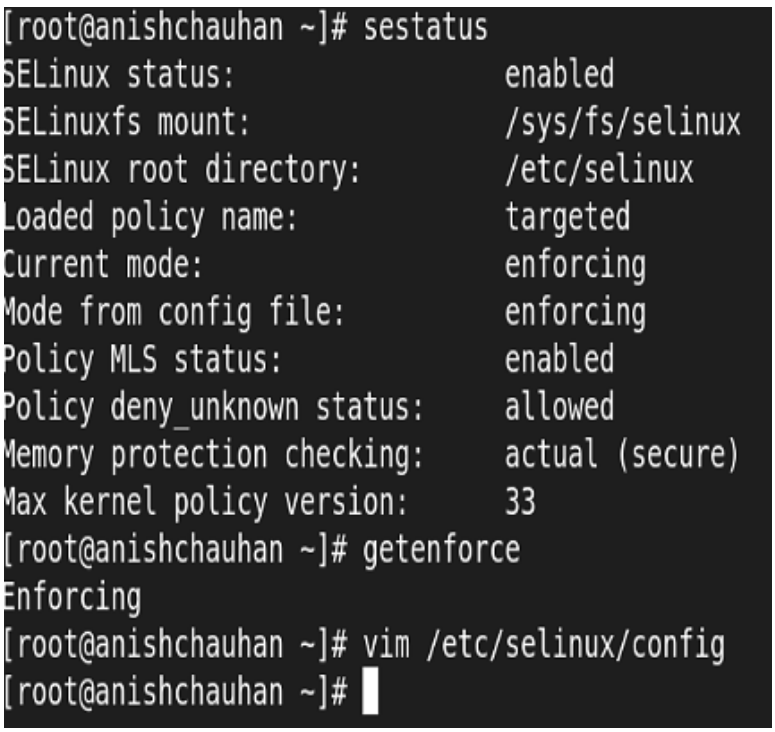


Fig: Checking SE Linux Status

B. Configure the server to enable (Enforcing) SE Linux:

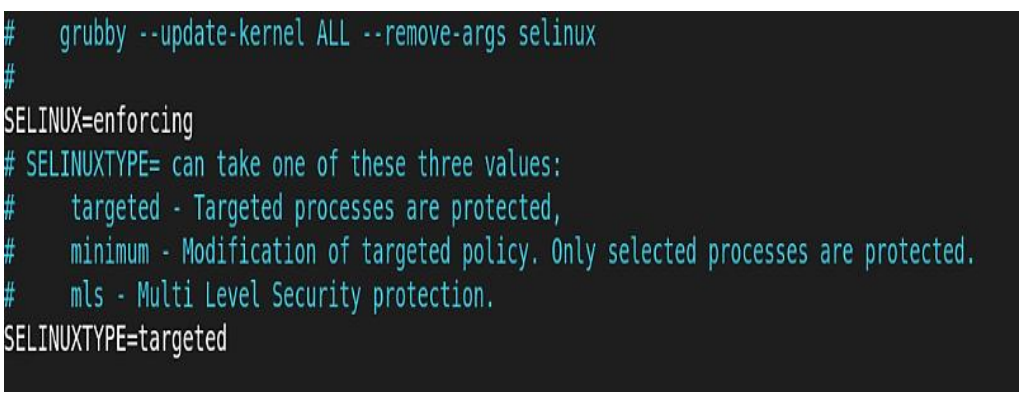


Fig: Enabling SE Linux

Lab Program Number: 18

Title: Configuring SSL-Enabled Apache (HTTPS) Server (self-signed):

SSL (Secure Socket Layer) provides encrypted communication between a web server and clients, ensuring data privacy and integrity. In this lab, we will configure Apache HTTP server with HTTPS using a self-signed certificate**.**

A. Install required package for HTTPS server:

Command: yum -y install httpd mod_ssl openssl

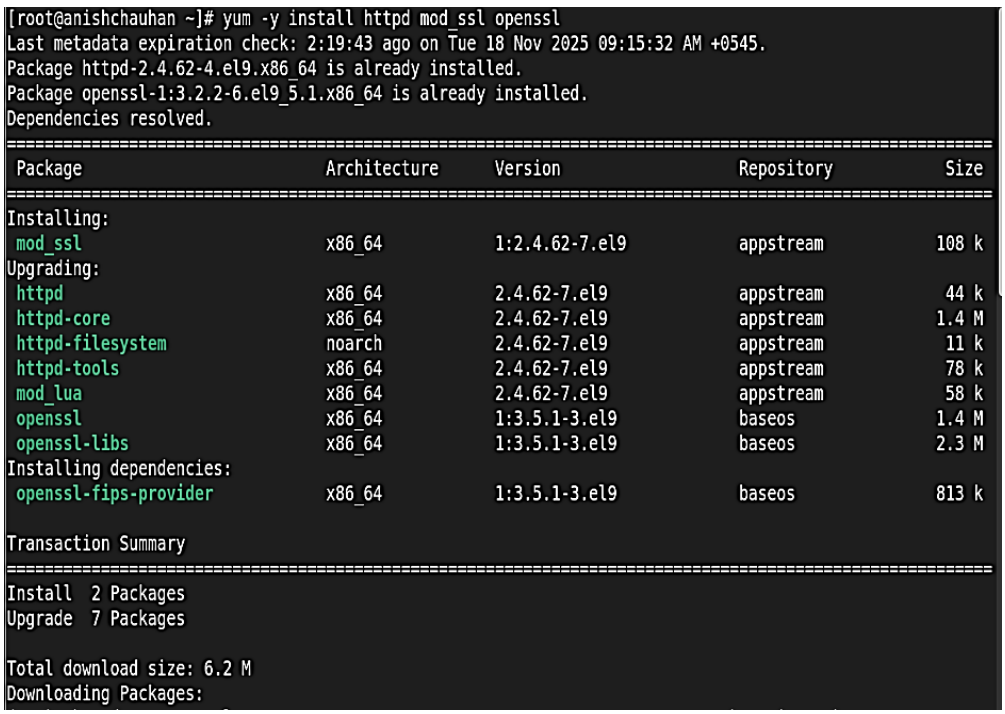


Fig: Installing required httpd, mod_ssl and openssl packages

B. Allowing https (port 443) packets to enter through the Firewall.

Fig: Allowing https through port 443

C. Start and enable Apache web service:

- Command:

  systemctl start httpd

  systemctl enable httpd

  systemctl status httpd

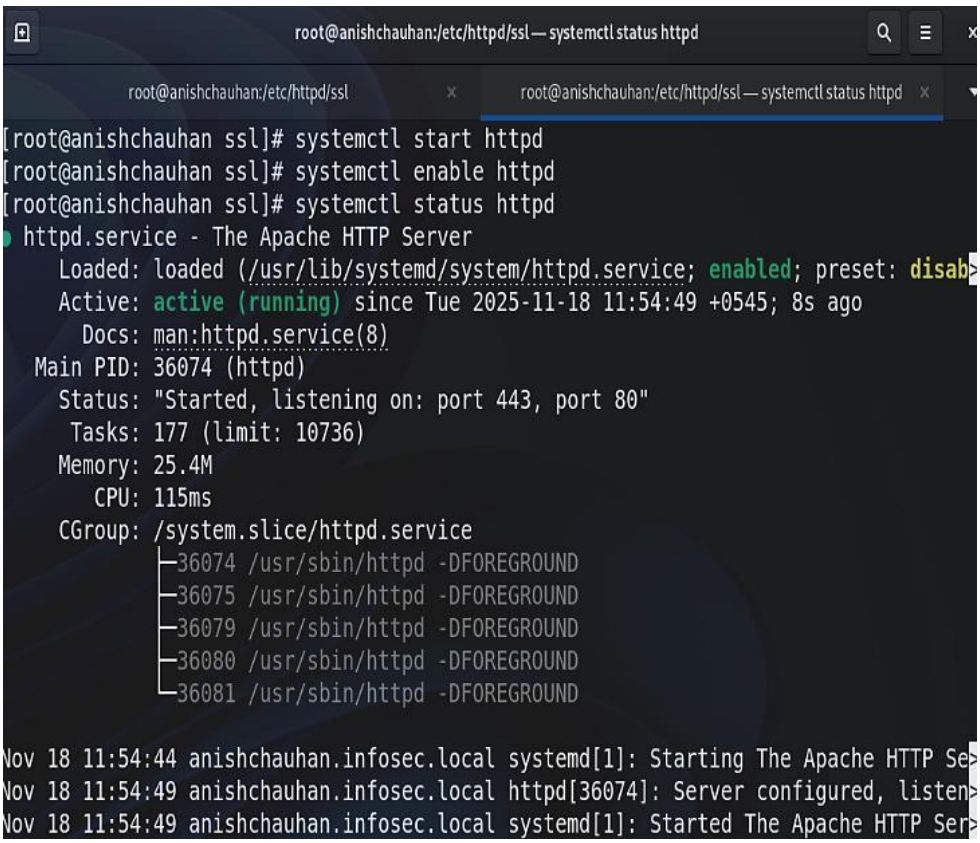


Fig: Starting and enabling web service

D. Generating self-signed key and cert files using openssl.

- Command:

  mkdir -p /etc/httpd/ssl

  cd /etc/httpd/ssl

  openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout server.key -out server.crt

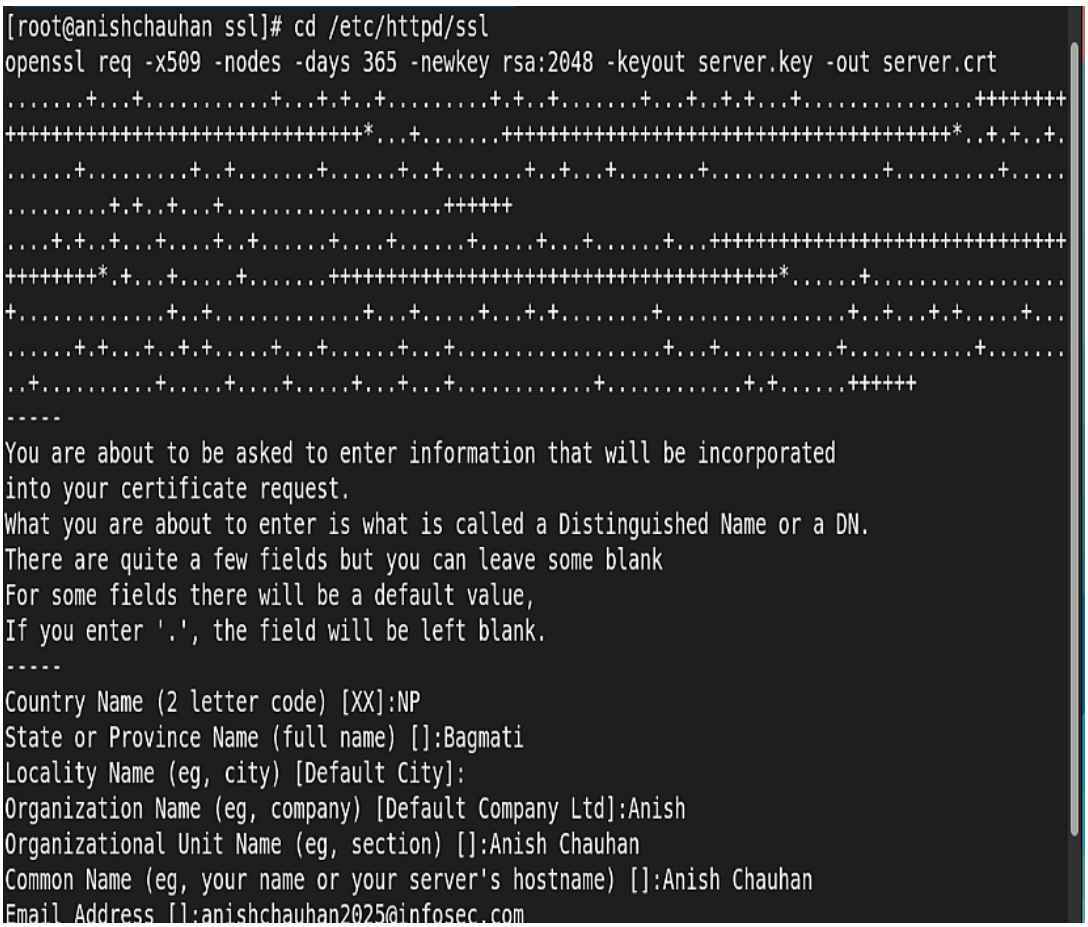


Fig: Generating self-signed key and cert files using openssl

E. Configure Web server to listen from port 443 and set DocumentRoot to "var/www/html", locate the required key and cert files.

- Command: vim /etc/httpd/conf.d/ssl.conf

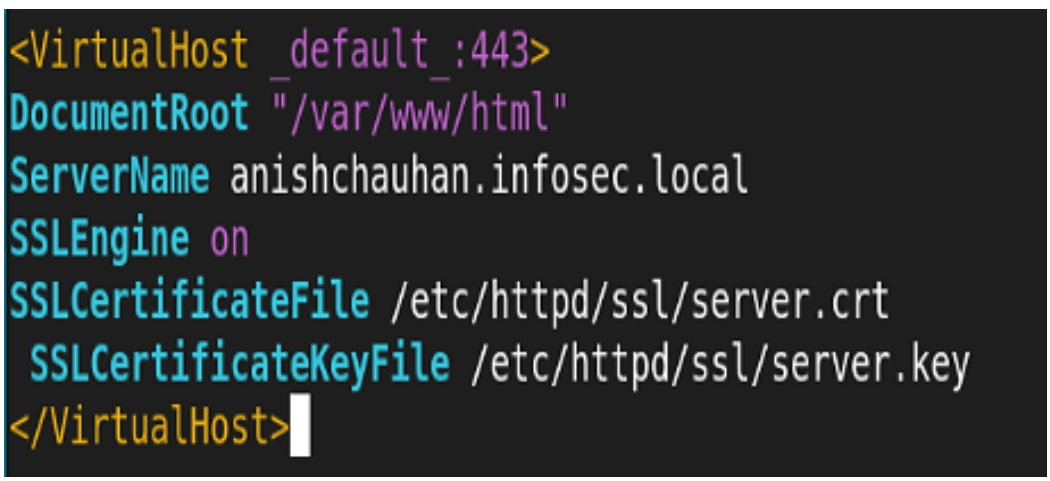


Fig: Configured web server

F. Host a web page called index.html on web server named anishchauhan.infosec.local.

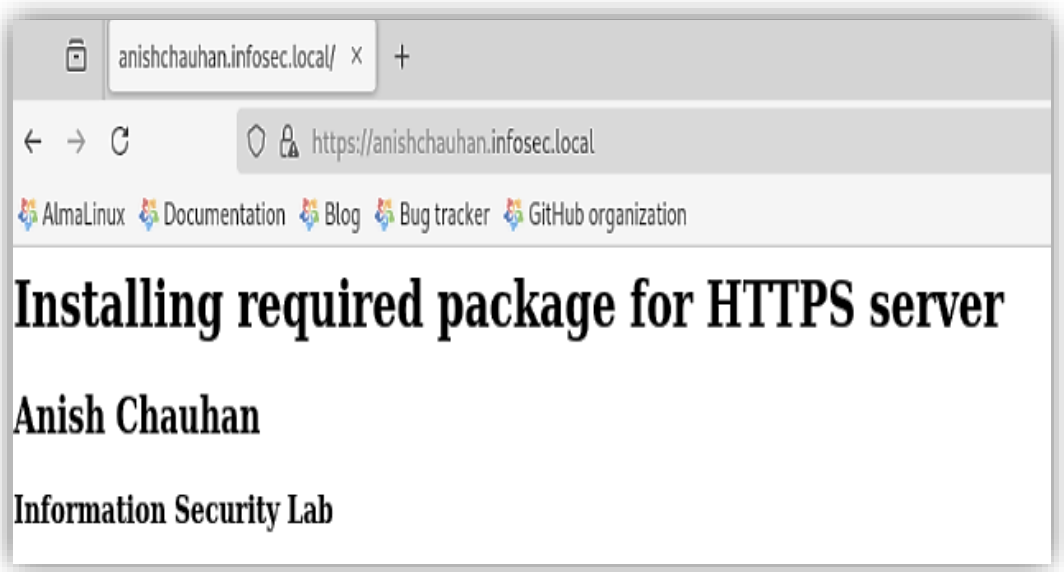


Fig: Hosted index.html on web server

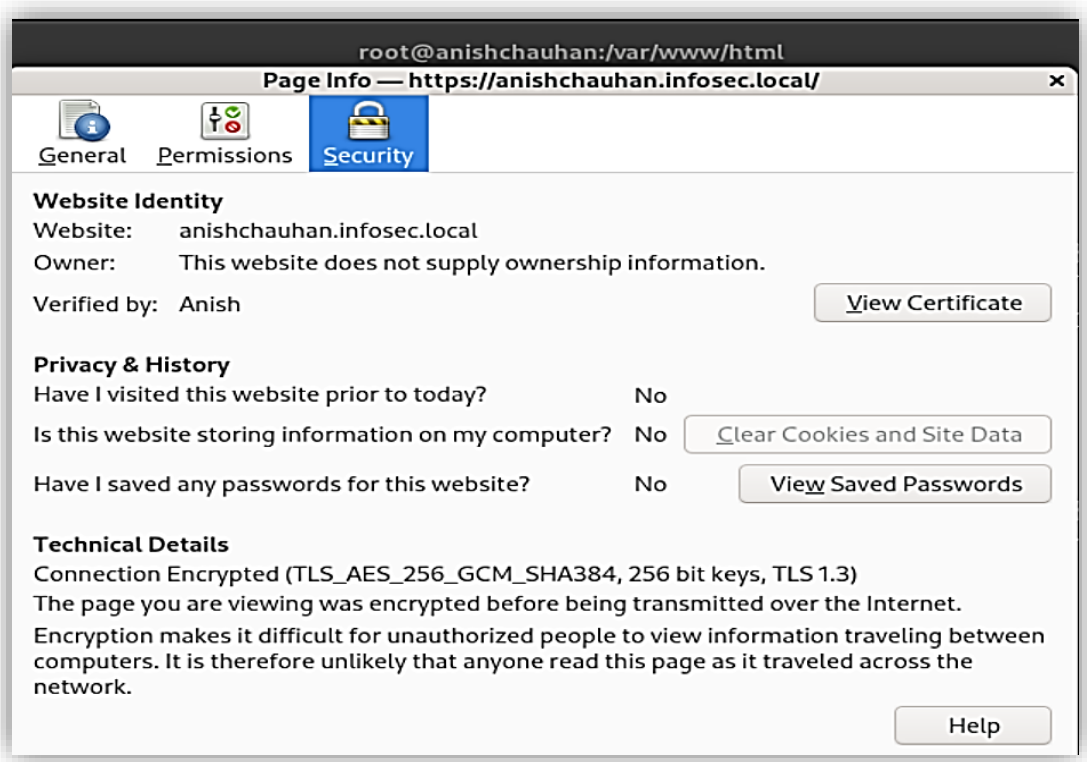


Fig: Certificate of the web page