Project Report

On

VIRTUAL PRIVATE NETWORK (VPN) SERVER MANAGEMENT SYSTEM

Submitted in partial fulfillment of the requirements for the award of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

(Artificial Intelligence & Machine Learning)

by

Ms. G TEJASWINI (22WH1A6608)

Ms. G ANUSHA (22WH1A6641)

Ms. G VAISHNAVI (22WH1A6658)

Ms. B ANITHA (22WH1A6659)

Under the esteemed guidance of

Ms. P Anusha

Assistant Professor, CSE(AI&ML)

Department of Computer Science & Engineering

(Artificial Intelligence & Machine Learning)

BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN

(AUTONOMOUS)

(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)

Accredited by NBA and NAAC with A Grade

Bachupally, Hyderabad – 500090

2024-25

Abstract

A Virtual Private Network (VPN) Server Management System is a comprehensive solution designed to simplify the deployment, configuration, monitoring, and maintenance of VPN servers. This system provides a centralized platform to manage user access, enforce security policies, and monitor server performance in real-time. It includes features such as user authentication, automated certificate generation, bandwidth monitoring, and secure protocol support (e.g., OpenVPN, IPSec). Administrators can create and manage VPN instances with ease, ensuring scalability and adaptability for small to large-scale networks. The system also integrates advanced analytics to provide insights into usage patterns and detect potential security threats. With a focus on user-friendliness, automation, and robust security, this system addresses the growing need for secure, private, and efficient communication in modern digital environments.

Problem Statement

Managing VPN servers in modern networks is a complex task requiring constant monitoring, secure configuration, and seamless user access control. Organizations face challenges in efficiently deploying VPN servers, managing user authentication, and enforcing security policies across diverse networks. Manual management increases the risk of configuration errors, performance bottlenecks, and potential security vulnerabilities. Furthermore, limited insights into usage patterns and server performance hinder effective decision-making. There is a need for a centralized VPN server management system that automates key tasks, provides real-time monitoring, ensures robust security, and simplifies scalability to meet the demands of secure and private communication.

Functional Requirements

1. **User Authentication and Access Control:**
   - Allow user registration and authentication using secure methods (e.g., passwords, two-factor authentication).
   - Manage user roles and permissions to control access levels.
   - Enable blocking or revoking access for specific users.
2. **VPN Server Deployment and Configuration:**
   - Provide automated setup of VPN servers with support for popular protocols like OpenVPN, IPSec, and WireGuard.
   - Allow administrators to configure server settings, such as encryption protocols, ports, and bandwidth limits.
3. **Real-time Monitoring and Analytics:**
   - Monitor server status, bandwidth usage, and connected clients in real time.
   - Generate reports on server performance, user activity, and data usage patterns.
4. **Certificate and Key Management:**
   - Automate the generation, renewal, and revocation of security certificates for secure communication.
   - Provide storage and management for encryption keys.
5. **Policy Management:**
   - Enable administrators to define and enforce security policies, such as allowed IP ranges and device restrictions.
   - Configure split tunneling and traffic prioritization.
6. **Multi-Server and Multi-Region Support:**
   - Allow management of multiple VPN servers across different geographic locations.

- Provide seamless server switching for users based on performance or proximity.

7. **User-Friendly Interface:**
    - Offer an intuitive dashboard for administrators to manage servers, monitor activity, and configure settings.
    - Provide a simple interface for end-users to connect to VPN servers.
8. **Security Features:**
    - Implement advanced encryption standards (AES-256, etc.) for secure data transmission.
    - Detect and mitigate potential security threats such as unauthorized access or DoS attacks.
9. **Scalability and High Availability:**
    - Support the addition of new servers or users without disrupting existing operations.
    - Ensure high availability with failover mechanisms for server outages.
10. **Audit Logs and Notifications:**
    - Maintain detailed logs of user activities and administrative changes for auditing purposes.
    - Send alerts for critical events, such as server downtimes or suspicious login attempts.

Non-Functional Requirements

1. **Performance:**
   - The system should support at least 1000 concurrent users per server without degradation in performance.
   - Ensure low-latency connections, with a target response time of less than 50ms for user authentication.
2. **Scalability:**
   - The system must scale to accommodate an increasing number of servers and users without significant architectural changes.
   - Support the addition of up to 50 new servers across multiple regions with minimal downtime.
3. **Security:**
   - Ensure end-to-end encryption using industry-standard protocols such as AES-256 and TLS 1.3.
   - Protect against vulnerabilities like data breaches, unauthorized access, and DoS attacks.
   - Provide regular security patches and updates.
4. **Reliability and Availability:**
   - Achieve 99.99% uptime for VPN services with failover mechanisms for server crashes or outages.
   - Implement automated recovery processes to minimize downtime during failures.
5. **Usability:**
   - The system should have an intuitive interface for administrators and users, requiring minimal training.
   - Provide clear documentation and support for onboarding new users and administrators.
6. **Compatibility:**

- Ensure compatibility with various operating systems (Windows, macOS, Linux) and mobile platforms (iOS, Android).
- Support integration with third-party tools like Active Directory for user management.

7. **Maintainability:**
   - Enable easy updates and upgrades to the system without disrupting active users.
   - Provide modular code architecture for ease of maintenance and debugging.
8. **Compliance:**
   - Adhere to data protection regulations like GDPR, HIPAA, or ISO 27001 for handling sensitive user data.
   - Implement measures to ensure secure storage and transmission of personal information.
9. **Efficiency:**
   - Optimize resource usage to minimize CPU, memory, and bandwidth consumption on servers.
   - Ensure the system performs efficiently even during peak traffic hours.
10. **Extensibility:**
    - The system should be optimized for cost, balancing performance, scalability, and resource usage without incurring excessive infrastructure or operational costs.

Source Code

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

// User structure
typedef struct {
    char username[50];
    char password[50];
    int connected; // 0 = disconnected, 1 = connected
} User;

// Function declarations
void authenticate(User *user);
void connect_vpn(User *user);
void disconnect_vpn(User *user);
void encrypt_data(const char *data);
void decrypt_data(const char *data);

// Main function
int main() {
    User user = {"", "", 0}; // Initialize user
    int choice;
```

```
char data[256];

printf("=== VPN Server Management System ===\n");

while (1) {
   printf("\nMenu:\n");
   printf("1. Authenticate\n");
   printf("2. Connect to VPN\n");
   printf("3. Disconnect from VPN\n");
   printf("4. Encrypt Data\n");
   printf("5. Decrypt Data\n");
   printf("6. Exit\n");
   printf("Enter your choice: ");
   scanf("%d", &choice);

   switch (choice) {
   case 1:
      authenticate(&user);
      break;
   case 2:
      connect_vpn(&user);
      break;
   case 3:
      disconnect_vpn(&user);
```

```
                break;
            case 4:
                printf("Enter data to encrypt: ");
                scanf(" %[^\n]", data);
                encrypt_data(data);
                break;
            case 5:
                printf("Enter data to decrypt: ");
                scanf(" %[^\n]", data);
                decrypt_data(data);
                break;
            case 6:
                printf("Exiting VPN Server Management System...\n");
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}

// Authenticate user
void authenticate(User *user) {
    char username[50];
    char password[50];

    printf("Enter username: ");
    scanf("%s", username);
    printf("Enter password: ");
    scanf("%s", password);

    // For simplicity, assume username "admin" and password "password"
    if (strcmp(username, "admin") == 0 && strcmp(password, "password") == 0)
{
        strcpy(user->username, username);
        strcpy(user->password, password);
        printf("Authentication successful!\n");
    } else {
        printf("Authentication failed. Try again.\n");
    }
}

// Connect to VPN
void connect_vpn(User *user) {
    if (user->connected) {
        printf("User %s is already connected to the VPN.\n", user->username);
    } else if (strlen(user->username) > 0) {
        user->connected = 1;
        printf("User %s connected to the VPN.\n", user->username);
    } else {
        printf("Please authenticate first.\n");
    }
}

// Disconnect from VPN
void disconnect_vpn(User *user) {
    if (user->connected) {
        user->connected = 0;
        printf("User %s disconnected from the VPN.\n", user->username);
    } else {
        printf("User is not connected to the VPN.\n");
    }
}

// Simulate encryption
void encrypt_data(const char *data) {
    printf("Encrypting data: ");
    for (int i = 0; data[i] != '\0'; i++) {
        printf("%c", data[i] + 3); // Simple Caesar cipher
    }
    printf("\n");
```

```
}

// Simulate decryption
void decrypt_data(const char *data) {
   printf("Decrypting data: ");
   for (int i = 0; data[i] != '\0'; i++) {
      printf("%c", data[i] - 3); // Reverse Caesar cipher
   }
   printf("\n");
}
```

Compilation and Execution :

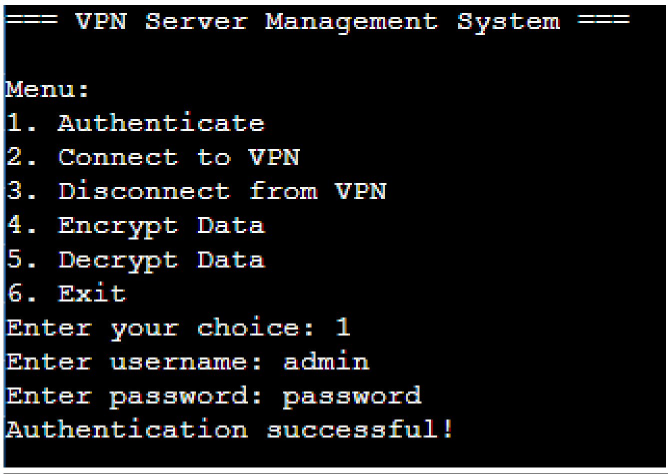
Compile :

```
gcc virtual_private_network.c
```

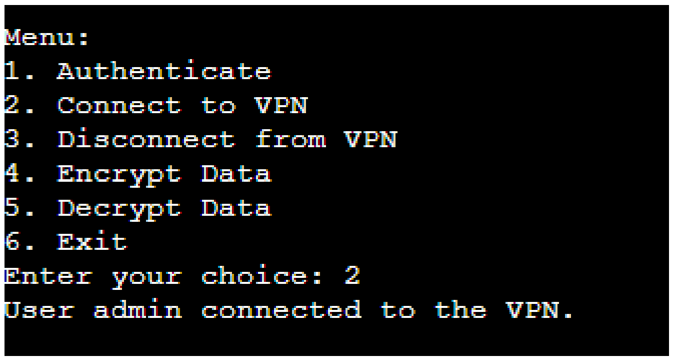
Run the executable code:

```
./a.out
```

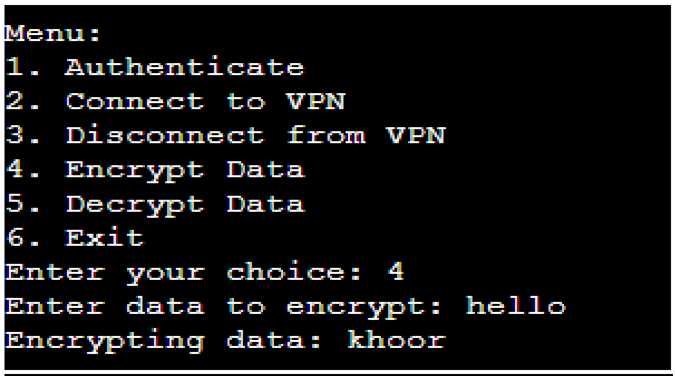
Output:



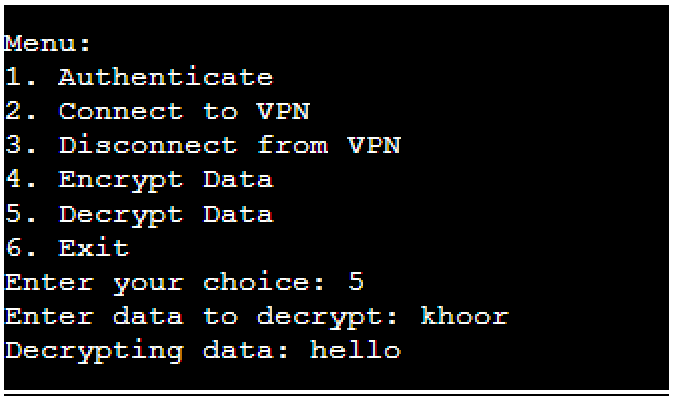
```
=== VPN Server Management System ===

Menu:
1. Authenticate
2. Connect to VPN
3. Disconnect from VPN
4. Encrypt Data
5. Decrypt Data
6. Exit
Enter your choice: 1
Enter username: admin
Enter password: password
Authentication successful!
```



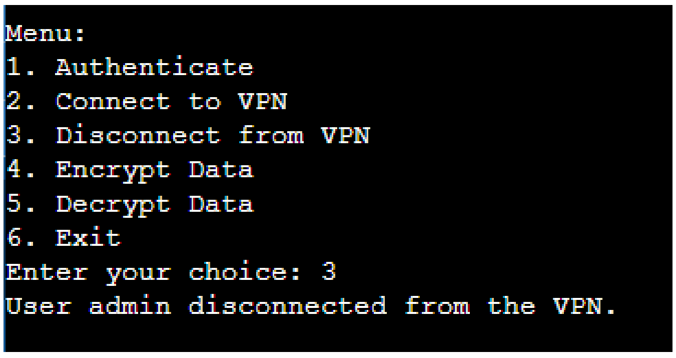
```
Menu:
1. Authenticate
2. Connect to VPN
3. Disconnect from VPN
4. Encrypt Data
5. Decrypt Data
6. Exit
Enter your choice: 2
User admin connected to the VPN.
```



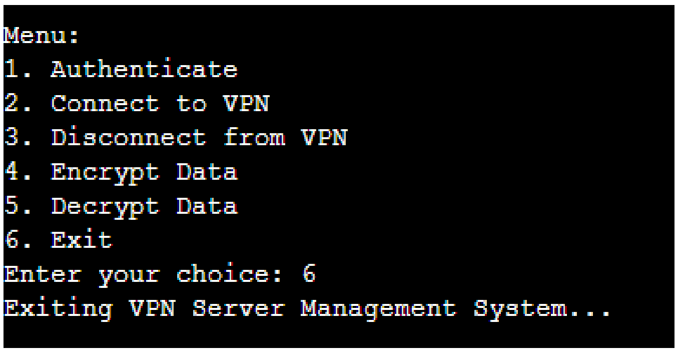
```
Menu:
1. Authenticate
2. Connect to VPN
3. Disconnect from VPN
4. Encrypt Data
5. Decrypt Data
6. Exit
Enter your choice: 4
Enter data to encrypt: hello
Encrypting data: khoor
```



```
Menu:
1. Authenticate
2. Connect to VPN
3. Disconnect from VPN
4. Encrypt Data
5. Decrypt Data
6. Exit
Enter your choice: 5
Enter data to decrypt: khoor
Decrypting data: hello
```



```
Menu:
1. Authenticate
2. Connect to VPN
3. Disconnect from VPN
4. Encrypt Data
5. Decrypt Data
6. Exit
Enter your choice: 3
User admin disconnected from the VPN.
```



```
Menu:
1. Authenticate
2. Connect to VPN
3. Disconnect from VPN
4. Encrypt Data
5. Decrypt Data
6. Exit
Enter your choice: 6
Exiting VPN Server Management System...
```