Post-Quantum Cryptography VPN Application: Python Implementation

COMP8047 – Major Project

Kuan-Yu (Gray) Chen – A01088589
10-20-2023

Table of Contents

1. Introduction

1.1. Student Background

I am an international student from Taiwan and am currently in the Bachelor of Technology in the Computer Systems program. Before joining the bachelor's program, I completed the Computer Systems Technology Diploma and have two years of programming experience. Currently, I specialize in network security applications development, and I will be using those experiences and knowledge to take advantage of learning new application and encryption protocols.

1.1.1. Education

British Columbia Institute of Technology

- Bachelor of Science, Applied Computer Science — Sep 2021 – Current
  Network Security Applications Development Option
- Computer Systems Technology Diploma – Web and Mobile — Jan 2019 – Dec 2020

University of British Columbia

- Bachelor of Science – Applied Biology — Sep 2017 – Apr 2018

1.1.2. Projects

- Search Engine Website for Recycling for City of Vancouver, BCIT — Jan 2019 – April 2019
- Mobile-friendly Web Game, BCIT — Apr 2019 – May 2019
- Cross-Platform Mobile Application for Voting, BCITSA — Apr 2020 – May 2020
- Mobile Application for Construction, BICC Professionals — Sep 2020 – Dec 2020

1.2. Project Description

Post-quantum cryptography, which can be referred to as quantum-resistant cryptography or quantum-safe cryptography, aims to construct public key cryptosystems that remain secure from attackers with quantum computers. This project's goal is to create a VPN application that integrates post-quantum cryptography, specifically the **CRYSTALS-KYBER Algorithm,** with the **OpenVPN** protocol. By implementing this application, users will be able to use the internet with their data encrypted and secured even after quantum computers are built. This application will consist of two parts: VPN Client and VPN Server. The VPN Client will encrypt the data from the user by applying CRYSTALS-KYBER algorithm and send the traffic through the VPN Tunnel to the VPN Server. On the other side, the VPN Server will listen to communications on specified port to receive data and decrypt the received data. It will also make requests to the internet with encryption while hiding the client's address and geolocation.

1.2.1. Essential Problems

In the development of this Quantum-Resistant VPN project, essential challenges are being addressed to ensure the security and functionality of the VPN service in an era where quantum computing poses a potential threat to conventional cryptographic systems. Key issues include the selection and implementation of quantum-resistant cryptographic algorithms for secure key exchange, data encryption, and digital signatures. A focus is placed on creating a robust key exchange mechanism that prevents the exposure of shared secrets to quantum attacks, considering quantum key distribution as a potential solution. The project also evaluates the VPN's security against quantum attacks, actively seeking to remain secure even when faced with the advent of quantum computing. These challenges form the core of the efforts in building this Quantum-Resistant VPN project that meets the growing demand for secure, quantum-resistant communication.

1.2.2. Goals and Objectives

The problem my application will solve is that users do not have to worry about the security of their data anymore, even when quantum computers are built. With the combination of the current encryption protocol and post-quantum cryptography, the application not only secures the user's data from classical computers but also prevents future threats when quantum computers are built and widely used by attackers. By implementing post-quantum cryptography into my application, the VPN application will allow users to keep their data secure against both quantum and classical computers. The protocol I will use to develop my VPN application is the OpenVPN protocol, which is the one of the most widely used protocols for VPN applications. The algorithm I will use for implementation is the CRYSTALS-KYBER Algorithm, which is one of the algorithms that are selected by the NIST (National Institute of Standards and Technology) for Post-Quantum Cryptography in 2022.

2. Body

2.1. Background

As the internet grows, companies start to move their businesses online, allowing customers to do transactions online. However, websites become vulnerable once their confidential information, such as credit card numbers, passwords, or social insurance numbers, are leaked to cyber criminals. To prevent cyber-attacks, cryptography plays an essential and irreplaceable role in the security of all internet communications.

Nowadays, public key cryptography is widely used for securing data. The most common example of public key cryptography is the security of "HTTPS" (Hypertext Transfer Protocol Secure) web pages. When the customers are making the transactions through the web page, Transport Layer Security (TLS) or Secure Sockets Layer (SSL) protocols are applied to encrypt the exchanged data with their own

algorithms. However, all the public key algorithms are based on complicated mathematical problems, such as discrete logarithms or prime factors of a composite number. Currently, those mathematical problems are almost impossible for classical computers to solve. This means that if a technology is developed to solve those mathematical problems, the data that are encrypted with the problems will no longer be safe.

Currently, scientists and engineers use supercomputers to solve difficult problems. However, most of the supercomputers are very large. They are built with thousands of classical CPU and GPU cores, which makes them so longer “super”. Also, if the problem has a high degree of complexity, such as containing a very large number of variables, supercomputer will often fail to solve, which makes it no longer “super”. The technology that is rising for solving such high complexity problems is called quantum computing. Quantum computing is a rapid-emerging technology that is developed to solve problems that are too complex for classical computers, including supercomputers. The computers that are built based on this technology are called quantum computers.

Fortunately, quantum computing is still a fairly new technology. Therefore, quantum computers have not existed and have not been built in the world yet. However, by the time quantum computers are built, they can easily break most of the problems that are generated by the current encryption algorithms. As a result, all data that is encrypted with the current encryption algorithms is no longer safe. To prevent this happening, quantum-safe cryptography emerged. Quantum-safe cryptography aims to develop algorithms that are safe enough to prevent attacks by both classical and quantum computers. One way of the implementation is to use mathematical techniques such as error correcting codes, lattices, or multivariate equations to develop quantum-safe cryptography (Open Quantum Safe, 2022).

2.2. Project Statement

This project is to develop a quantum resistance VPN application by using the quantum resistance algorithms provided by the Open Quantum Safe Project with the OpenVPN Protocol.

2.3. Possible Alternative Solutions

There are some possible alternative solutions to the project based on the following categories:

- **VPN Protocol**

  There are many selections for VPN Protocol, for example OpenVPN, IKEv2/IPSec, WireGuard, and others. Each protocol serves different purposes and has different security measures.

- **Programming Languages**

  There are many programming languages that we can choose from, including Python, C, C++, Java, and other languages. Each of the languages has their advantages and disadvantages.

- **Quantum Resistance Algorithms**

  According to the Open Quantum Safe (OQS) Project, there are many algorithms that are provided. It is divided into two parts: KEMs (Key encapsulation mechanisms) and Signature schemes. Some of the algorithms provided by OQS project include Kyber, McEliece, Dilithium, and Falcon.

2.4. Chosen Solution

Based on the alternative solutions above, it was decided to use the CRYSTALS-KYBER algorithm to develop a VPN Application with OpenVPN Protocol using Python. The reason for choosing the CRYSTALS-KYBER algorithm is because it is the most recent algorithm that NIST (National Institute of Standards and Technology) had selected for KEMs (Key encapsulation mechanisms). OpenVPN Protocol was selected because it is one of the most widely used VPN Protocols in the world.

2.5. Details of Design and Development

2.5.1. Deliverables

- **VPN Client**

  A client application is delivered with sample configuration and key files. The program is built in a simple user interface by Python.

- **VPN Server**

  A server application is delivered with sample configuration and key files. The program is built in a simple user interface by Python.

- **Setup Package**

  A setup package is delivered to help the user automate the setup process instead of manually installing the required libraries and software.

- **Final Report**

  A detailed report that contains all necessary information about the project is delivered.

2.5.2. Milestones

Each of the following subsections discusses the details of each milestone during the implementation of the project. **Please note that there are some changes to the original proposal in Milestone 2 when I am exploring the post quantum algorithms.**

2.5.2.1. Milestone 1: VPN with OpenVPN Protocol

As I embarked on Milestone 1 of setting up a VPN using the OpenVPN Protocol, I initially felt quite lost. Understanding the protocol turned out to be more challenging and time-consuming than I had expected. However, I was determined to figure things out. One of the first problems I had to overcome was installing all the necessary software and configuring both the client and server components. This was a significant step, marking my progress in learning how the VPN worked. But my journey was far from over.

I soon encountered a new problem: I couldn't get the client and server to connect when they were on different networks with their own unique IP addresses. It was like they were speaking different languages and couldn't understand each other. After some careful investigation, I found the solution. It turned out that I needed to make a few adjustments to my home router. By setting up port forwarding on my router, I allowed the server to communicate beyond my local network. This was a critical step because it meant that the server could now reach out to the wider internet, connecting to the client and making the VPN functional.

Milestone 1 was a learning experience that not only tested my technical skills but also taught me the importance of paying attention to detail when working with VPNs and networks. It reminded me that even the smallest adjustments can have a big impact on achieving a successful connection.

2.5.2.2. Milestone 2: Post Quantum Algorithm Implementation

While exploring the quantum resistance algorithms from the Open Quantum Safe project, I discovered that the project divided the algorithms into two parts: KEMs (Key encapsulation mechanism) and Signature schemes. The algorithm that I chose, Kyber, is one of the KEMs. After spending time understanding how the algorithm and the OpenVPN protocol works, I found **that the OpenVPN protocol does not support custom encryption algorithms without changing the source code**. Changing the source code requires in-depth knowledge of OpenVPN, as well as OpenSSL, since OpenVPN relies on OpenSSL for their encryption algorithms. Also, since the source code of OpenVPN is written in C, it will be out of scope to change my entire program from Python to C as the proposal states that the project will be implemented by Python.

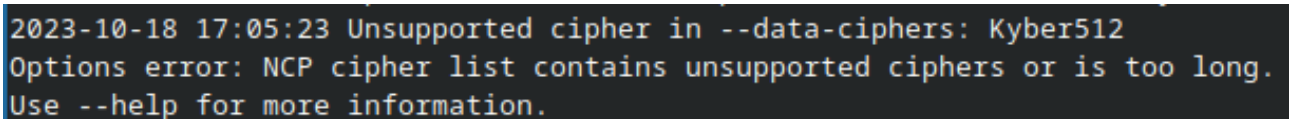


Figure 1: OpenVPN fails to support Kyber

As a result, I switched from the Kyber algorithm (KEMs) to Dilithium algorithm (Signature schemes), which is also from CRYSTALS (Cryptographic Suite for Algebraic Lattices), the same team that developed Kyber. The reason of it is because I think it is possible to use the generated certificates and keys by a post quantum signature schemes algorithm to secure the VPN application. However, the OpenVPN protocol does not accept the certificates that are generated by post quantum algorithms as OpenSSL cannot recognize the type of certificates since they cannot determine the security level of the post quantum algorithms that just came out in the recent years. Also, it might be possible to bypass the check of OpenSSL, but it fails the purpose of the project to provide quantum-resistance security to the current OpenVPN application.

```
OpenSSL: error:0A0000F7:SSL routines::unknown certificate type
Cannot load certificate file server_cert.crt
Exiting due to fatal error
```

Figure 2: OpenSSL Fails to Determine Certificate Type

With both the failures to integrate the KEMs and Signature schemes algorithms with the OpenVPN protocol and the allocated time constraints, I decided to lower the scope and use the post quantum algorithms to build a client-server application to demonstrate that I understood and implemented post-quantum algorithms. The implementation is divided into two parts:

1) Key Encapsulation Mechanism (KEMs): using Kyber512 algorithm for key exchange mechanism.
2) Signature Schemes: using Dilithium3 algorithm to perform digital signature.

The architecture and design of the application is provided in Section 2.5.8. Post Quantum Client-Server Application with an architecture diagram for better understanding.

Even though I spent some time setting up the liboqs library from the OQS project for KYBER algorithm and the oqsprovider for OpenSSL, I eventually got both programs functions normally for the demonstration of both types of quantum resistance algorithms. In Milestone 2, my journey through post-quantum algorithm integration unveiled both challenges and adaptations. The initial thought to incorporate the Kyber Key Encapsulation Mechanism (KEM) into the OpenVPN Protocol gave way to the realization that the fundamental structure of OpenVPN did not readily accommodate custom encryption algorithms. An exploration of Signature Schemes led me to the Dilithium algorithm, offering a potential avenue for quantum-resistant security. However, the resistance of OpenSSL and the OpenVPN Protocol to these innovations underscored the emerging nature of post-quantum cryptography.

In response to these obstacles, the project's scope was refined, focusing on application that demonstrated a functional implementation of both the KEMs and Signature Schemes algorithms. Through persistence and diligent setup of the liboqs library and oqsprovider for OpenSSL, these programs have emerged as showcases of quantum-resistance in action. This milestone reinforced my understanding of post-quantum cryptography and its practical applications, even in the face of compatibility challenges.

2.5.2.3. Milestone 3: GUI Integration

In Milestone 3, my focus shifted to GUI Integration, where I aimed to create a graphical user interface (GUI) for my application using the PyQt library. While this was my first experience with PyQt, I was pleasantly surprised by how quickly I could develop a basic GUI for the application. The initial steps were relatively smooth. I successfully crafted a simple GUI that aligned with the application's requirements. However, the most time-consuming part of this milestone was translating the wireframes I had designed into the actual GUI. This process involved meticulously implementing the visual elements and functionalities outlined in the wireframes.

Despite my best efforts, the constraints of time became a challenge. I couldn't bring every detail from the wireframes into the final GUI. Nonetheless, I managed to ensure that the core functions were fully functional, and users could interact with the application effectively.
Another essential aspect of this milestone was dedicating time to error handling. I wanted to make sure that the program didn't crash unexpectedly. By implementing error-handling mechanisms, I could enhance the overall stability and reliability of the application.

In the end, this milestone was a significant achievement. I successfully created two distinct applications, one for the Client and one for the Server, each equipped with a simple yet functional GUI built with

PyQt. Despite the time constraints and the challenges of translating wireframes into a working interface, I emerged with two applications that were ready for use, enhancing the user experience and the overall functionality of the VPN system.

2.5.3. Installation Manuals

The installation guide requires the following requirements:

- **Linux Machine (Tested on Fedora 36)**
  - If you do not have Linux on your local machine, you will need to download the .iso file of your choice of Linux System (Fedora 36 in the project) and install it on your local machine. For Fedora set up, you can find the .iso file here: https://fedoraproject.org/spins/kde/

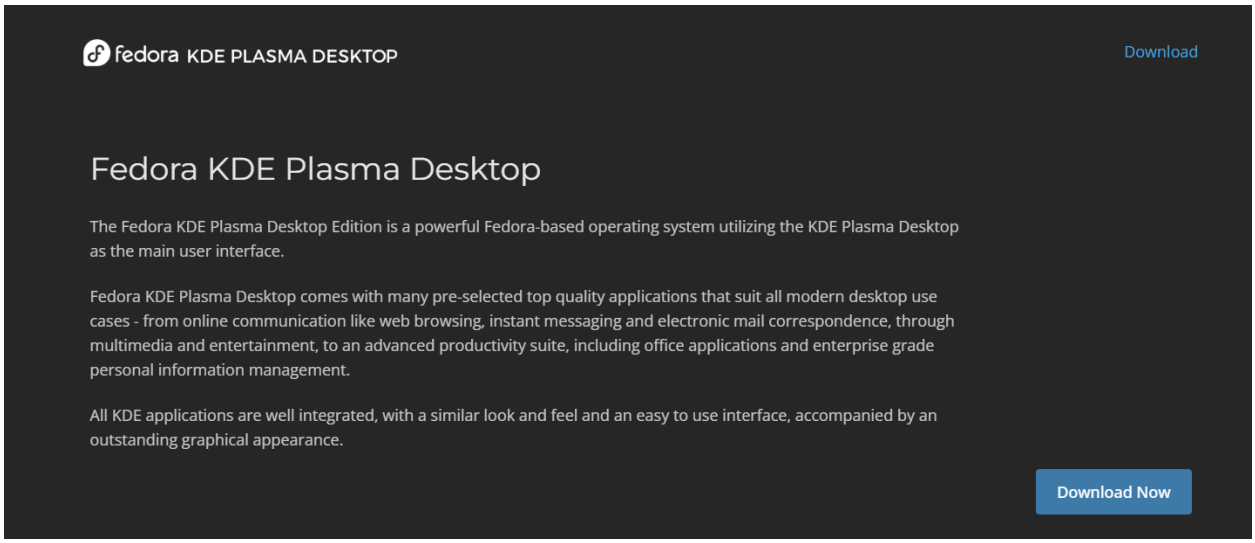


Figure 3: Fedora KDE Plasma Desktop Download Page

- After the .iso file is downloaded, you need to prepare a USB drive and create a bootable image of Linux system to install on your local machine. One of the tools that creates bootable USB drive is Rufus, which can be found in here: https://rufus.ie/en/
- After the USB is bootable, restart your device and boot with the USB to install Linux on your local device.

- **Python Compiler (Tested on Python 3.10.10)**
  - Please check the Python version of your local device before starting the application. You can check the version by entering the command ***python --version***
- **OpenSSL version 3.0.8 and TLS version 1.3**
  - Please check if the OpenSSL and TLS version is the close to the test version. It is recommended to use the same version as the tested version to avoid errors.
- **OpenVPN 2.5.9 (Tested on x86_64 redhat-linux-gnu)**
  - It is important to use an OpenVPN 2.X version instead of OpenVPN 3.X as the source code is written in different languages and functions differently.

The applications require the following libraries:

- **PyQt5, PyQt6**
  - Install the library using the following command: ***pip install PyQt5 PyQt6***
- **OQS-provider**
  - First, clone the Github repository of oqs-provider with the following command:

    git clone https://github.com/open-quantum-safe/oqs-provider.git
  - Run the setup script and test if it is installed in the repository by the following command:

    cd oqs-provider

    ./ scripts/fullbuild.sh

    ./scripts/runtests.sh (Optional)

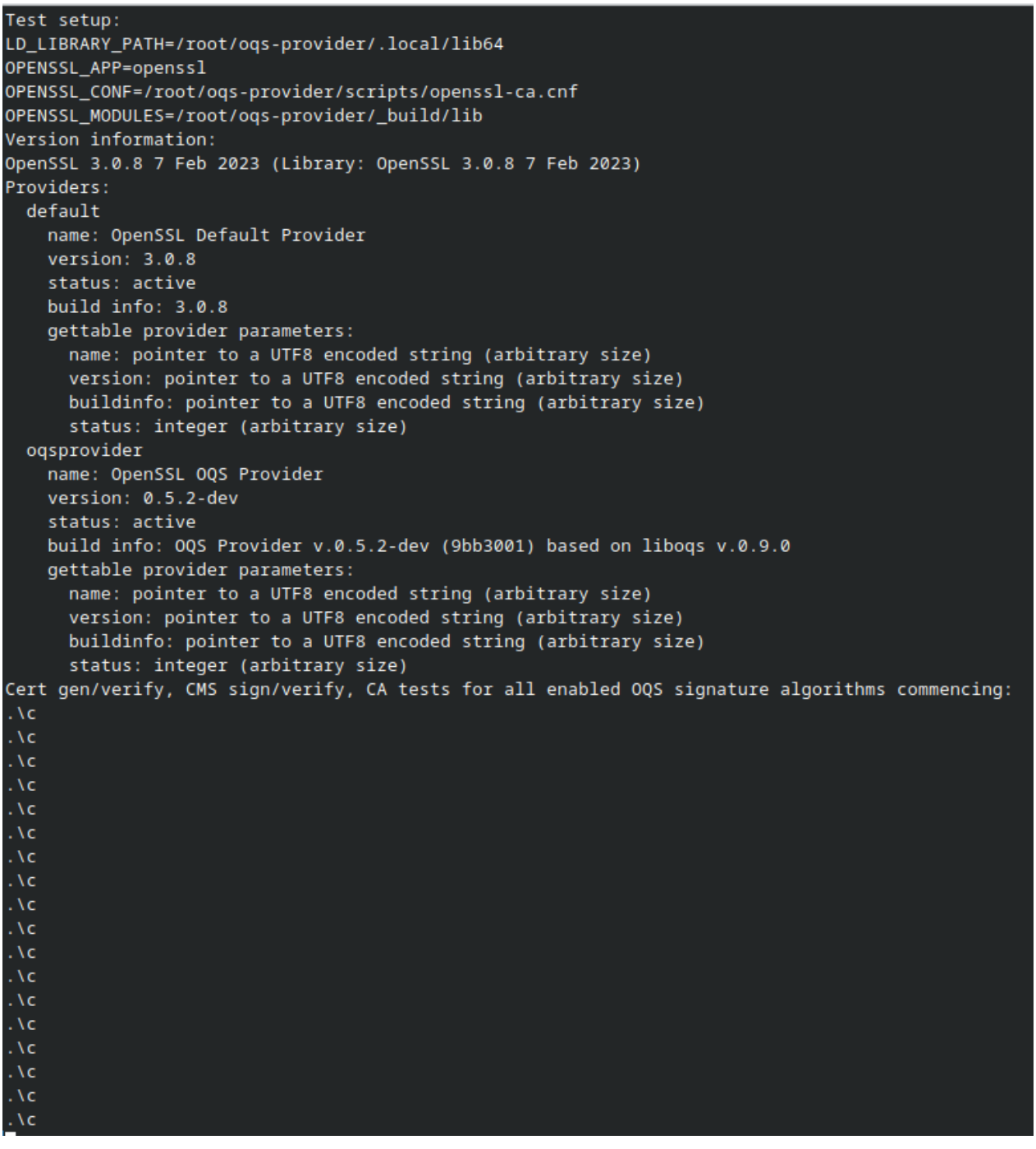


Figure 4: Running Tests on oqs-provider after installation.

- Note: By running the ***fullbuild.sh*** script, the program will also install the liboqs library if the device did not have it installed, which is also a required library for the project

- **liboqs-python**
    - Note: liboqs-python requires the liboqs library to be installed, which is already been installed when installing oqs-provider above

- First, set the LD_LIBRARY_PATH environmental variable point to the path to liboqs library directory. For Fedora 36 users, the variables can be set with this command:

    export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib64

- Second, clone the Github repository of liboqs-python with the following command:

    git clone --depth=1 https://github.com/open-quantum-safe/liboqs-python

- Finally, move into the library directory and install the wrapper with the following command:

    cd liboqs-python

    pip install .

- You can test if the library is configured and installed with the following scripts:

    python3 liboqs-pythonn/examples/kem.py

    python3 liboqs-pythonn/examples/sig.py

    python3 liboqs-pythonn/examples/rand.py

```
sh-5.2# python3 examples/kem.py
/usr/local/lib/python3.10/site-packages/oqs/oqs.py:67: UserWarning: Please install liboqs-python using setup.py
  warnings.warn("Please install liboqs-python using setup.py")
/usr/local/lib/python3.10/site-packages/oqs/oqs.py:74: UserWarning: liboqs version 0.9.0 differs from liboqs-python version None
  warnings.warn("liboqs version {} differs from liboqs-python version {}".format(oqs_version(), oqs_python_version()))
liboqs version: 0.9.0
liboqs-python version: None
Enabled KEM mechanisms:
['BIKE-L1', 'BIKE-L3', 'BIKE-L5', 'Classic-McEliece-348864',
 'Classic-McEliece-348864f', 'Classic-McEliece-460896',
 'Classic-McEliece-460896f', 'Classic-McEliece-6688128',
 'Classic-McEliece-6688128f', 'Classic-McEliece-6960119',
 'Classic-McEliece-6960119f', 'Classic-McEliece-8192128',
 'Classic-McEliece-8192128f', 'HQC-128', 'HQC-192', 'HQC-256', 'Kyber512',
 'Kyber768', 'Kyber1024', 'sntrup761', 'FrodoKEM-640-AES', 'FrodoKEM-640-SHAKE',
 'FrodoKEM-976-AES', 'FrodoKEM-976-SHAKE', 'FrodoKEM-1344-AES',
 'FrodoKEM-1344-SHAKE']

Key encapsulation details:
{'claimed_nist_level': 1,
 'is_ind_cca': True,
 'length_ciphertext': 768,
 'length_public_key': 800,
 'length_secret_key': 1632,
 'length_shared_secret': 32,
 'name': 'Kyber512',
 'version': 'https://github.com/pq-crystals/kyber/commit/74cad307858b61e434490c75f812cb9b9ef7279b'}
b'\x81\x94\x15\xa4fH0\xd9\x03!VG\xa0\x9c\xadI\x80\xc9\xd2:\x19\xbd\x04,\x82A\x8c\xb1\x840\xfd$\x95\xe5\x11:\x93\x11\x1e\xd6\x12\x
'\xa8pkpZ\xa6\x85\xcc4\x08V\xa6dgX\x14\x8cIxy\xc9I\xdcI\xa3\xf8G\xb2\x87\x8a\xc7\x02CBz\xc9,[\xa3\xd0\x990I\xc2\xab\xc6{H\xa9\x89
xa8#\x9c\xa2\x83l}\xfb\xb1G\x98\xbbL<f3\xb1\x04\x072W\x82la\x9a\xf5\x98<\x80VAH?\x8e\xb75\x82\x9c\xc5\x04\xf8\x91~\x1c6\xcf\x95\x
d%\xac\xb3\xf7\xad\xa7j\x84U`3\xa5\x95\x00}\xf4C\xaeD+\xe8\x04o\xd4\x1a\xb9V\xa3\x8c\xf8\xd4\xbb6\x8b~\xd7\xdako\xd9Y^\xfbN\xbe\x
5\xe6\xb5\x1d2x\x9b\xd5\x1a\x02\xdfeD\xa5\x10P\x96(s9\xe5\x98\xe2\x97h\xba\x0b\x8bY\x1b\x84\xea\x08\xaf^\xb3 \x0eW\xa4*,\x97\x89\
\xd6\x19\x81\xfc6\x19\x91\x13i3\xa8\x17v\x17\x9c\x90\x99\x00\x82\x0e\x8dy\xb2\xe3h\x9dK<\xab\xf9*D\xc3\x16P\xe0\xb5\x7f\xf5\x9b|\
xb3b\xd9E\xc2\x0c\xf3\xcd\x12\xb7>V\xd9\x11KW\x92\x17\x88\xb4%r\x88hD\x92\xd93\xb9\xae\xcb\x86,\xf4s2\xd4 I;s\xbes-\x13\x99\x9a\x
C\x16\x06q\x98<\x1d\xc5K\x18e\xf2\x0f\x7f\x9aT\x00\xf9\xc4\xbb\x97H\xa2\xc2y`|\x97\x12\x81W2\x8bG\xa3\x88\x912;=u\xc7\xc7\xd8|\xa
fBl{\xe5\x14\x90T\xbf\x7f\xe0\xc1/\xa3u\tv\nT\xe7\xba\x9ffTC\x10\xae\x87@U\x03\xfa\xc4\xbd0\x00\x14k\x85\x95XG\xab\xb2\x8e\x87\xc
c5\x0ecP\xc7\x18d\x05K\xfa1\x05p\x14\xb7Be\xe1\xb2\x1e\xf3\x0b\xa0s8g\xb0[(\x10\xa6\xa2]\x93\xb0\x89\x16\x95\xe9\x90\xaf\xe5\x12\
7\xf4\xe0\xa5g\xbd\xaf\xc1\xa4\x06O\x96\x90r'

Shared secretes coincide: True
```

Figure 5: Sample script output on liboqs library after installation.

2.5.4. Setup Guide

2.5.4.1. Setting up OpenVPN

- To start with, we need to setup OpenVPN on the **server** machine first.
- Open the terminal and execute the bash script openvpn-install.sh that is contained in the deliverable program inside the Setup Package with the following command:

  ./setup_package/openvpn-install.sh

  Note: the original script is written by https://github.com/Nyr/openvpn-install, it is modified to optimize the configuration for this project
- Follow the instructions in the script file. First, choose the IP Address you are using for your server.

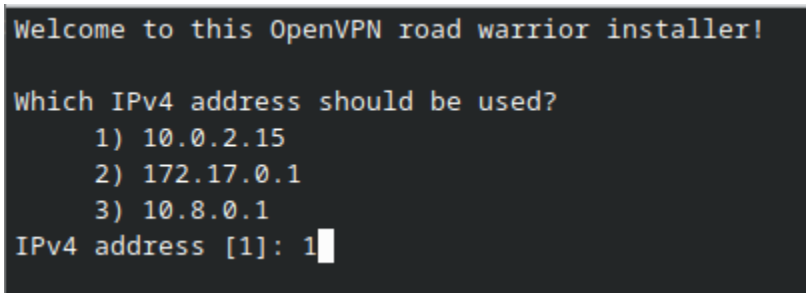


Figure 6: Configuring Local IP Address for VPN Server

- Enter your public IP address or hostname so that the outside world can connect to your server.

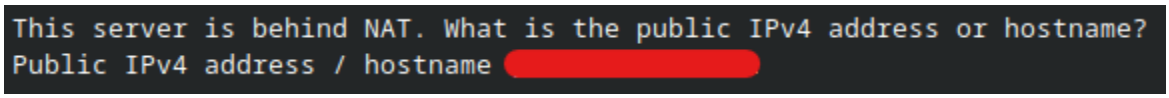


Figure 7: Configuring Public IP Address for VPN Server

Note: A virtual machine is used for the demonstration of the setup guide

- Afterward, choose UDP for the protocol.

```
Which protocol should OpenVPN use?
   1) UDP (recommended)
   2) TCP
Protocol [1]: 1
```

Figure 8: Configuring Internet Protocol for VPN Server

- Choose the port number you will like the VPN server to listen to. For demonstrating, the default port number 1194 are chosen.

```
What port should OpenVPN listen to?
Port [1194]:
```

Figure 9: Configuring Port Number for VPN Server

- Select a DNS server for the clients. You can choose any of those, the demonstration will be using the current system resolvers.

```
Select a DNS server for the clients:
   1) Current system resolvers
   2) Google
   3) 1.1.1.1
   4) OpenDNS
   5) Quad9
   6) AdGuard
DNS server [1]:
```

Figure 10: Configuring DNS for VPN Server

- Enter your first client's name. The demonstration will use the default (client).

```
Enter a name for the first client:
Name [client]:

OpenVPN installation is ready to begin.
Press any key to continue...
```

Figure 11: Configuring client configuration file for VPN Server

- Afterward, press any key and the script will install all the necessary libraries, as well as generating keys and configuration files, for OpenVPN.

```
Certificate created at:
* /etc/openvpn/server/easy-rsa/pki/issued/client.crt

Notice
------
Inline file created:
* /etc/openvpn/server/easy-rsa/pki/inline/client.inline

* Using SSL: openssl OpenSSL 3.0.8 7 Feb 2023 (Library: OpenSSL 3.0.8 7 Feb 2023)

* Using Easy-RSA configuration: /etc/openvpn/server/easy-rsa/pki/vars
Using configuration from /etc/openvpn/server/easy-rsa/pki/b9757fc7/temp.ba294177

Notice
------
An updated CRL has been created.
CRL file: /etc/openvpn/server/easy-rsa/pki/crl.pem
2023-10-18 22:01:51 WARNING: Using --genkey --secret filename is DEPRECATED.  Use --genkey secret filename instead.
success
success
success
success
success
success
Created symlink /etc/systemd/system/multi-user.target.wants/openvpn-server@server.service → /usr/lib/systemd/system/openvpn-server@.service.

Finished!

The client configuration is available in: /root/client.ovpn
New clients can be added by running this script again.
```

Figure 12: Sample Output after setting up OpenVPN

- After the installation is complete, the terminal shows the location of the client's VPN configuration file. Go to the directory and copy the configuration file, ***client.ovpn***, to the client's device.

2.5.4.2. Port Forwarding on Home Router (Optional)

- If you wish to let your client connect to your server from the outside world, you will need to setup port forwarding on your home router on your server's device to allow outside connections.
- To start with, we need the server's device connects to the router (either through LAN or Wi-Fi)
- Afterward, you need to access the router through the gateway. You can find the gateway by the following command: ***ip r***

```
sh-5.2# ip r
default via 192.168.0.1 dev enp0s3 proto dhcp src 192.168.0.37 metric 100
```

Figure 13: Finding Network Gateway

- Go to your browser and type the gateway in. In the demonstration, the gateway is 192.168.0.1. Routers might have different login page than the demonstration.

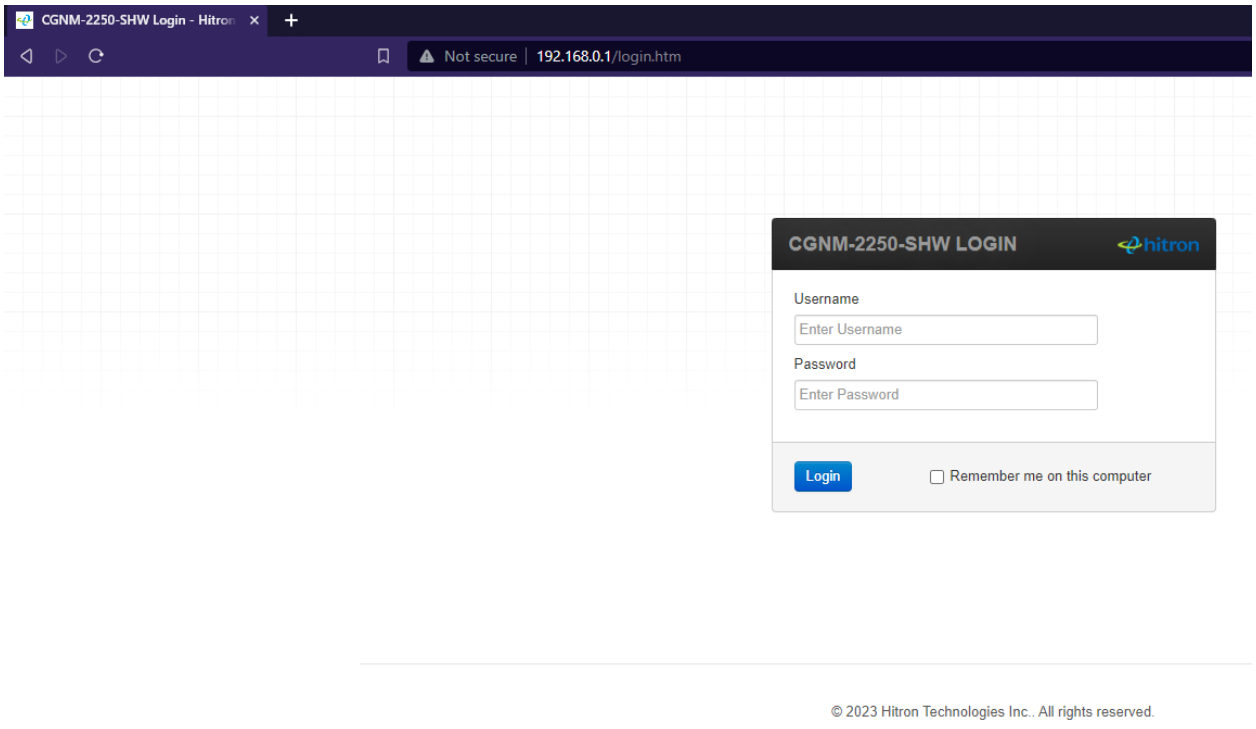


Figure 14: Entering Router IP address / Gateway

- Login with the credentials for the router. If you do not know the credentials, it is often written on the home router.

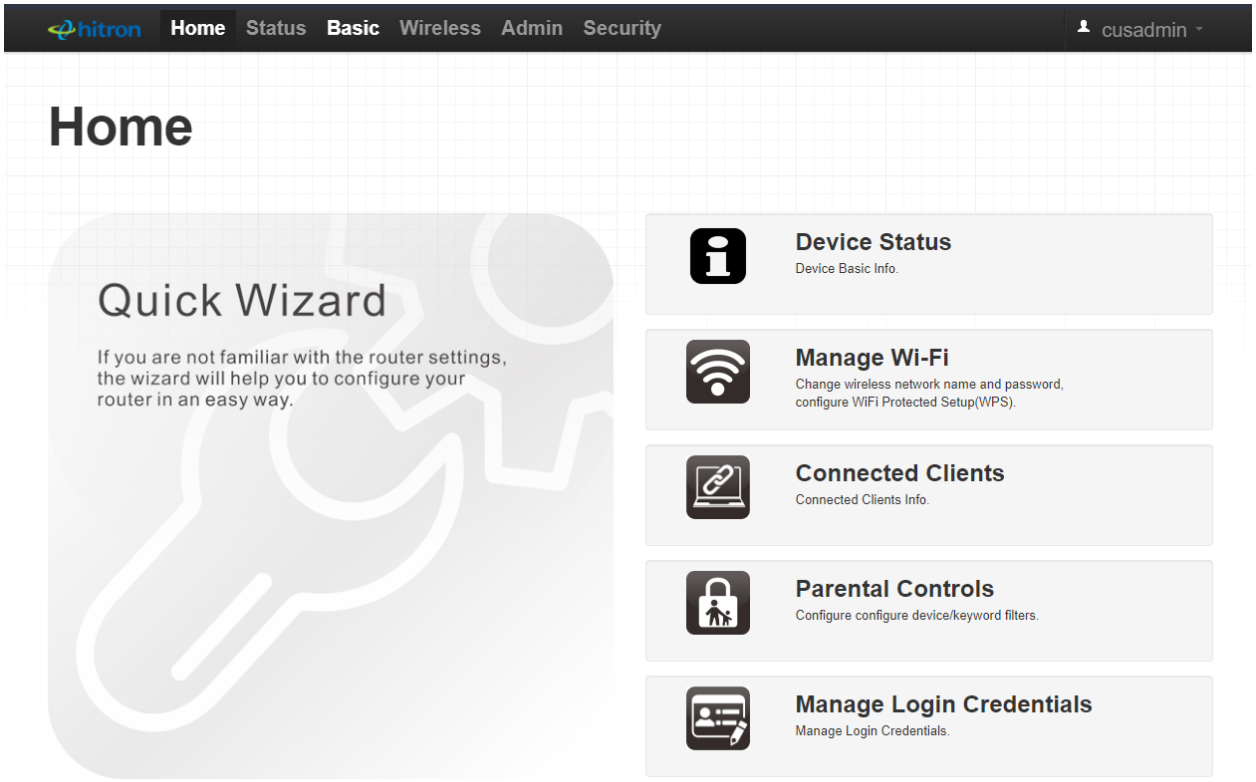


Figure 15: Log Into Router Settings

- Once you are logged in, find the option ***Port Forwarding***. In the demonstration, it is under **Basic** > **Port Forwarding**.

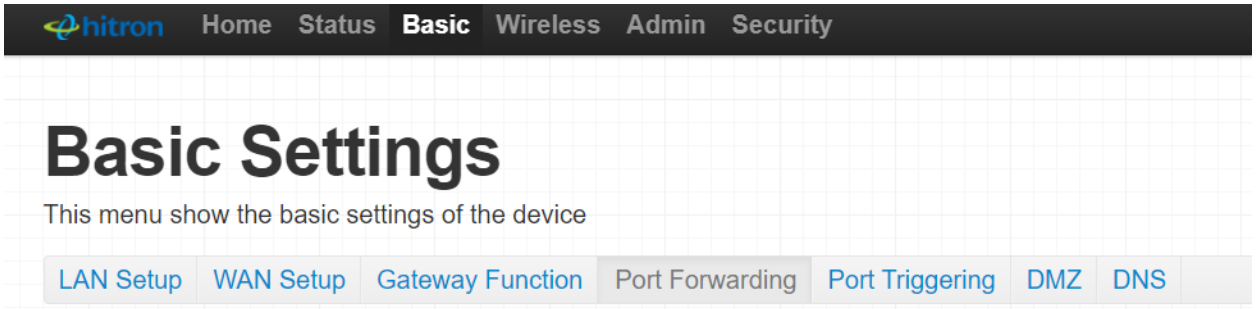


Figure 16: Finding the Port Forwarding Option

- Depending on the router, find the place to add the port forwarding rule for your server.

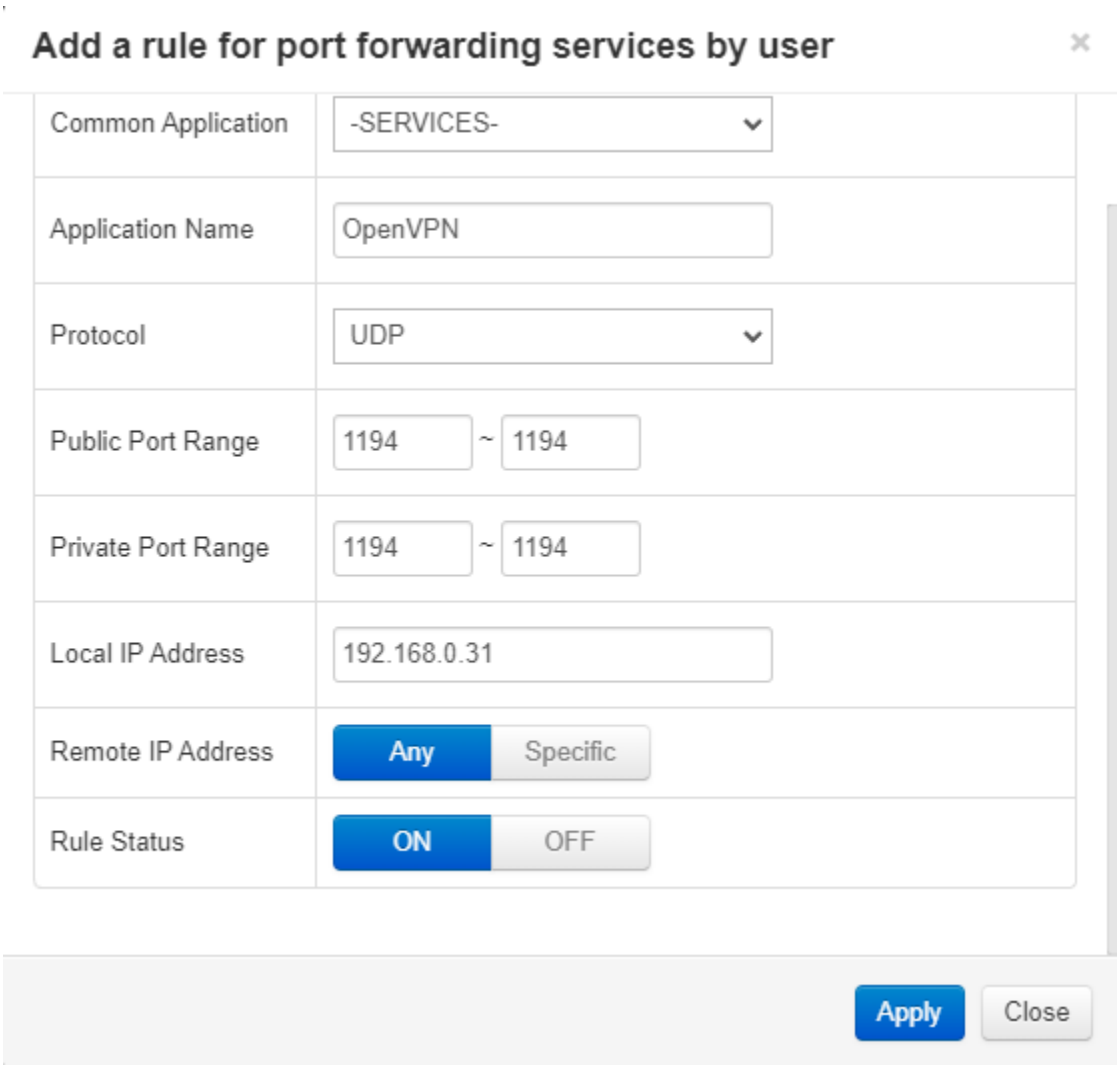


Figure 17: Adding Port Forwarding Rule to Server

- Finally, save the changes to the rules and restart the router.
- After the port forwarding rule is applied, your VPN server will be able to accept client connections from the outside world.

2.5.4.3. OQS-Provider configuration on OpenSSL

- After we installed OQS-Provider in the previous section (2.5.3. Installation Manuals), we need to configure OpenSSL to use the library as a provider.
- To check if OQS-Provider is installed, we can use the following command:

openssl list -providers

```
sh-5.2# openssl list -providers
Providers:
  default
    name: OpenSSL Default Provider
    version: 3.0.8
    status: active
  oqsprovider
    name: OpenSSL OQS Provider
    version: 0.5.2-dev
    status: active
```

Figure 18: OpenSSL list all the providers

- We can also list out the available signature and KEM algorithms with the following command:

openssl list -signature-algorithms -provider oqsprovider

openssl list -kem-algorithms -provider oqsprovider

```
sh-5.2# openssl list -signature-algorithms -provider oqsprovider
  { 1.2.840.113549.1.1.1, 2.5.8.1.1, RSA, rsaEncryption } @ default
  { 1.2.840.10040.4.1, 1.2.840.10040.4.3, 1.3.14.3.2.12, 1.3.14.3.2.1
tion-old, dsaWithSHA, dsaWithSHA1, dsaWithSHA1-old } @ default
  { 1.3.101.112, ED25519 } @ default
  { 1.3.101.113, ED448 } @ default
  ECDSA @ default
  HMAC @ default
  SIPHASH @ default
  POLY1305 @ default
  CMAC @ default
  dilithium2 @ oqsprovider
  p256_dilithium2 @ oqsprovider
  rsa3072_dilithium2 @ oqsprovider
  dilithium3 @ oqsprovider
  p384_dilithium3 @ oqsprovider
  dilithium5 @ oqsprovider
  p521_dilithium5 @ oqsprovider
  falcon512 @ oqsprovider
  p256_falcon512 @ oqsprovider
  rsa3072_falcon512 @ oqsprovider
  falcon1024 @ oqsprovider
  p521_falcon1024 @ oqsprovider
  sphincssha2128fsimple @ oqsprovider
  p256_sphincssha2128fsimple @ oqsprovider
  rsa3072_sphincssha2128fsimple @ oqsprovider
  sphincssha2128ssimple @ oqsprovider
  p256_sphincssha2128ssimple @ oqsprovider
  rsa3072_sphincssha2128ssimple @ oqsprovider
  sphincssha2192fsimple @ oqsprovider
  p384_sphincssha2192fsimple @ oqsprovider
  sphincsshake128fsimple @ oqsprovider
  p256_sphincsshake128fsimple @ oqsprovider
  rsa3072_sphincsshake128fsimple @ oqsprovider
```

Figure 19: OpenSSL list signature algorithms with OQS-provider

```
sh-5.2# openssl list -kem-algorithms -provider oqsprovider
  { 1.2.840.113549.1.1.1, 2.5.8.1.1, RSA, rsaEncryption } @ default
  frodo640aes @ oqsprovider
  p256_frodo640aes @ oqsprovider
  x25519_frodo640aes @ oqsprovider
  frodo640shake @ oqsprovider
  p256_frodo640shake @ oqsprovider
  x25519_frodo640shake @ oqsprovider
  frodo976aes @ oqsprovider
  p384_frodo976aes @ oqsprovider
  x448_frodo976aes @ oqsprovider
  frodo976shake @ oqsprovider
  p384_frodo976shake @ oqsprovider
  x448_frodo976shake @ oqsprovider
  frodo1344aes @ oqsprovider
  p521_frodo1344aes @ oqsprovider
  frodo1344shake @ oqsprovider
  p521_frodo1344shake @ oqsprovider
  kyber512 @ oqsprovider
  p256_kyber512 @ oqsprovider
  x25519_kyber512 @ oqsprovider
  kyber768 @ oqsprovider
  p384_kyber768 @ oqsprovider
  x448_kyber768 @ oqsprovider
  x25519_kyber768 @ oqsprovider
  p256_kyber768 @ oqsprovider
  kyber1024 @ oqsprovider
  p521_kyber1024 @ oqsprovider
  bikel1 @ oqsprovider
  p256_bikel1 @ oqsprovider
  x25519_bikel1 @ oqsprovider
  bikel3 @ oqsprovider
  p384_bikel3 @ oqsprovider
  x448_bikel3 @ oqsprovider
  bikel5 @ oqsprovider
  p521_bikel5 @ oqsprovider
  hqc128 @ oqsprovider
  p256_hqc128 @ oqsprovider
  x25519_hqc128 @ oqsprovider
  hqc192 @ oqsprovider
  p384_hqc192 @ oqsprovider
  x448_hqc192 @ oqsprovider
  hqc256 @ oqsprovider
  p521_hqc256 @ oqsprovider
```

Figure 20: OpenSSL list KEM algorithms with OQS-provider

- In Figure 19 and 20, it shows that the algorithms from the OQS-provider are available for OpenSSL.
- This can be the end for setting up OQS-provider, but we need to add the -provider flag every time we are executing the commands for OpenSSL. The following instructions show how OpenSSL can use the post-quantum algorithms without having to add the provider flag.
- To enable the provider automatically, we need to go to the configuration file for OpenSSL. For Fedora 36 users, the configuration file is located in: ***/etc/ssl/openssl.cnf***
- In the configuration file, we need to insert a few lines at around line 60-70:

oqsprovider = oqsprovider_sect

[oqsprovider_sect]

activate = 1

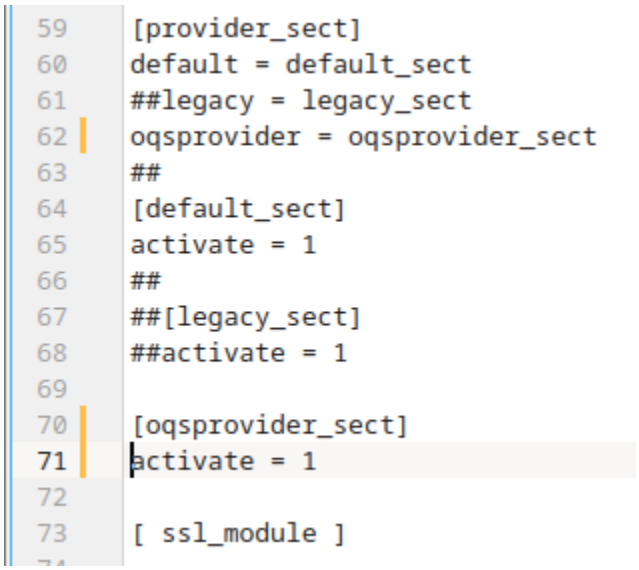


Figure 21: OpenSSL Configuration for OQS-provider

- After adding those lines, save the configuration file and execute the following command to make sure OpenSSL automatically loads the provider when the command is called:

openssl list -signature-algorithms

openssl list kem-algorithms

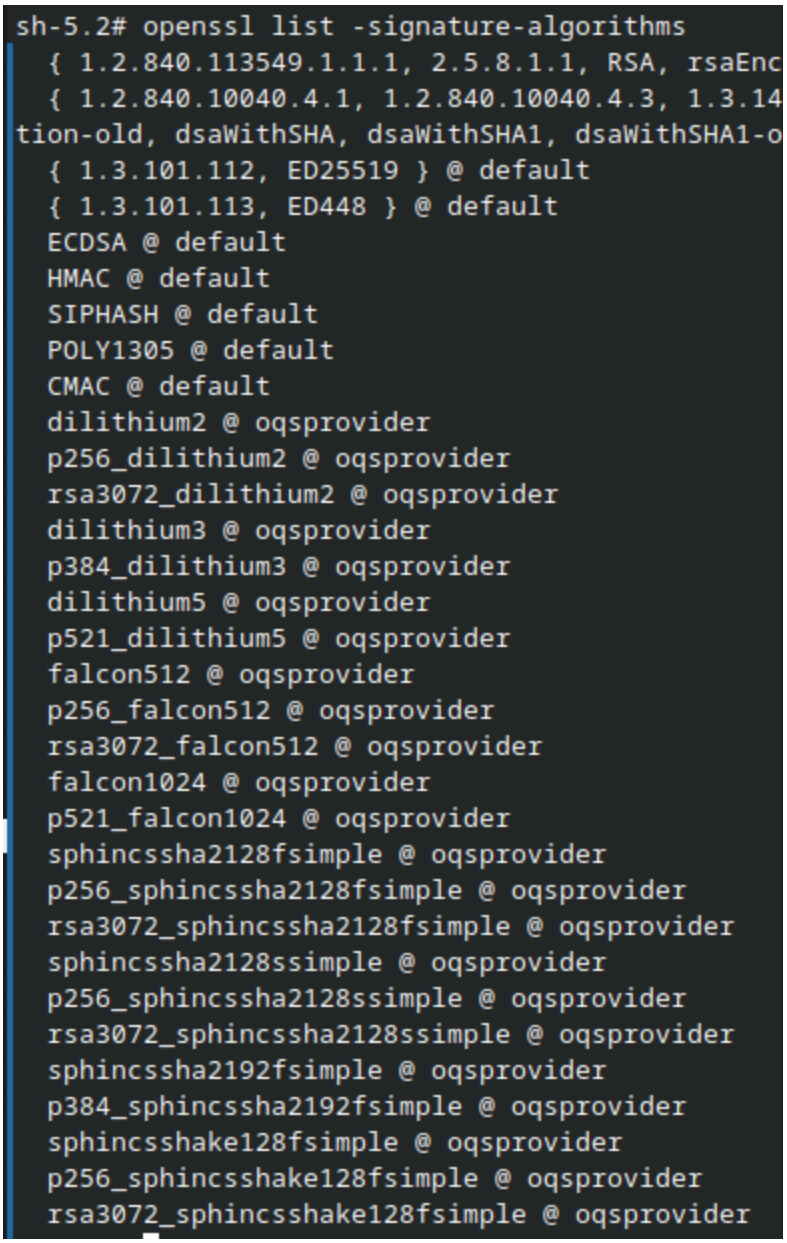


Figure 22: OpenSSL list signature algorithms without provider

- Finally, Figure 22 shows that OpenSSL automatically loads the OQS-provider when the command is called.

2.5.5. User Manuals

2.5.5.1. VPN application (OpenVPN Protocol)

Server

- If you follow the Setup Guide to install OpenVPN in Section 2.5.4.1., you should have your server installed and configured.
- Go ahead and make sure that the server configuration file is in your machine, as well as the certificates and key files. For Fedora 36 users, the server configuration file is in:

/etc/openvpn/server/server.conf.

- To access the server application, it is **required** that all the following files are in ***/etc/openvpn/server***:
  - server.conf
  - ca.crt
  - server.crt
  - server.key
  - dh.pem
  - tc.key
  - crl.pem

- If one of the files is missing, the VPN application will not start. If you wish to change the location of those files, you can go inside the server configuration file and edit the location of each file.

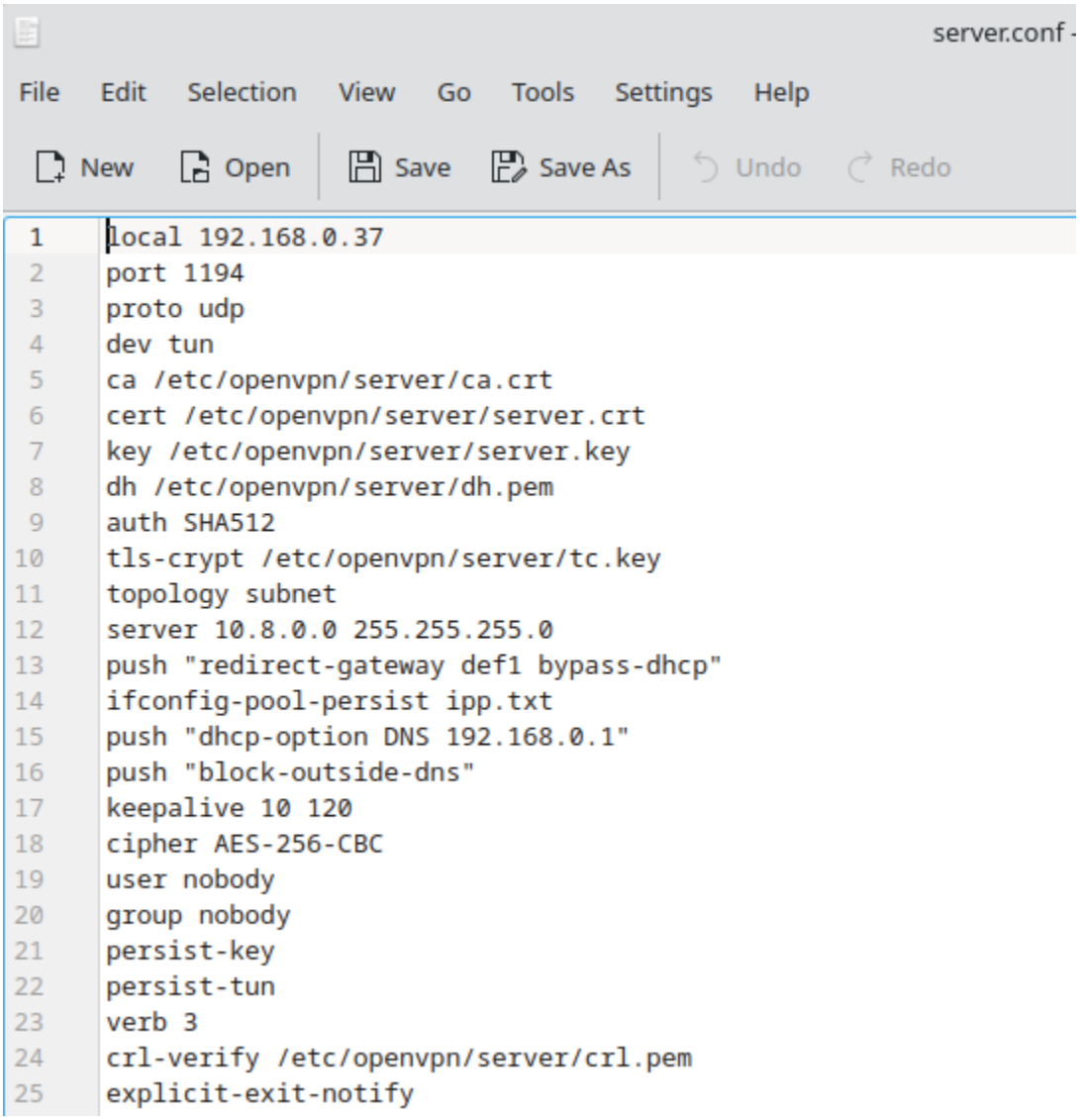


Figure 23: VPN Server Configuration File

- Once the configuration is done, you may start the server program with the following command:

python3 openvpn_server.py

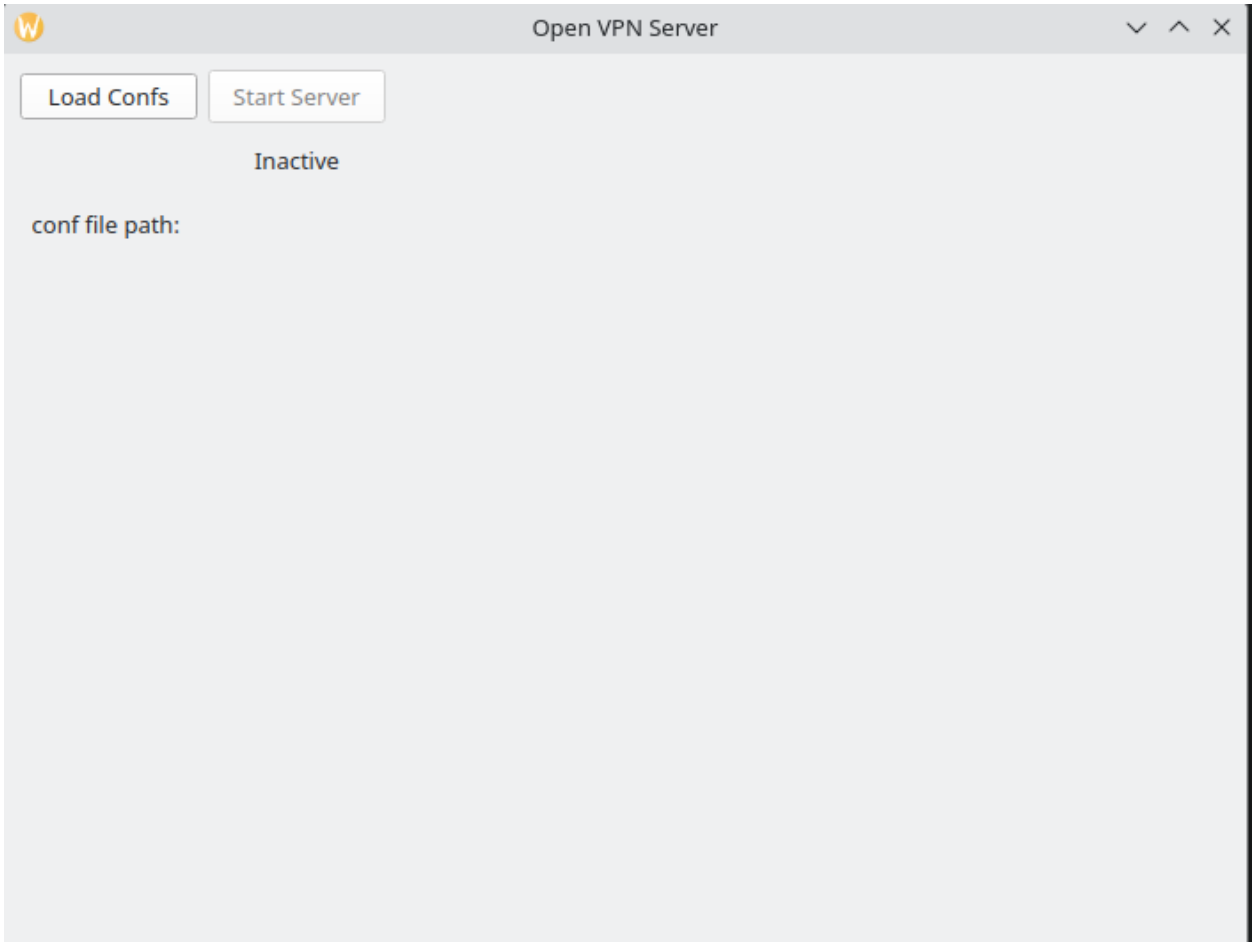


Figure 24: VPN Server GUI

- The GUI of the program will show up after executing the program. Click the "**Load Confs"** button on the **top left** corner of the program and select the configuration file for your OpenVPN Server. It will be in ***/etc/openvpn/server/*** if you installed it with the script.

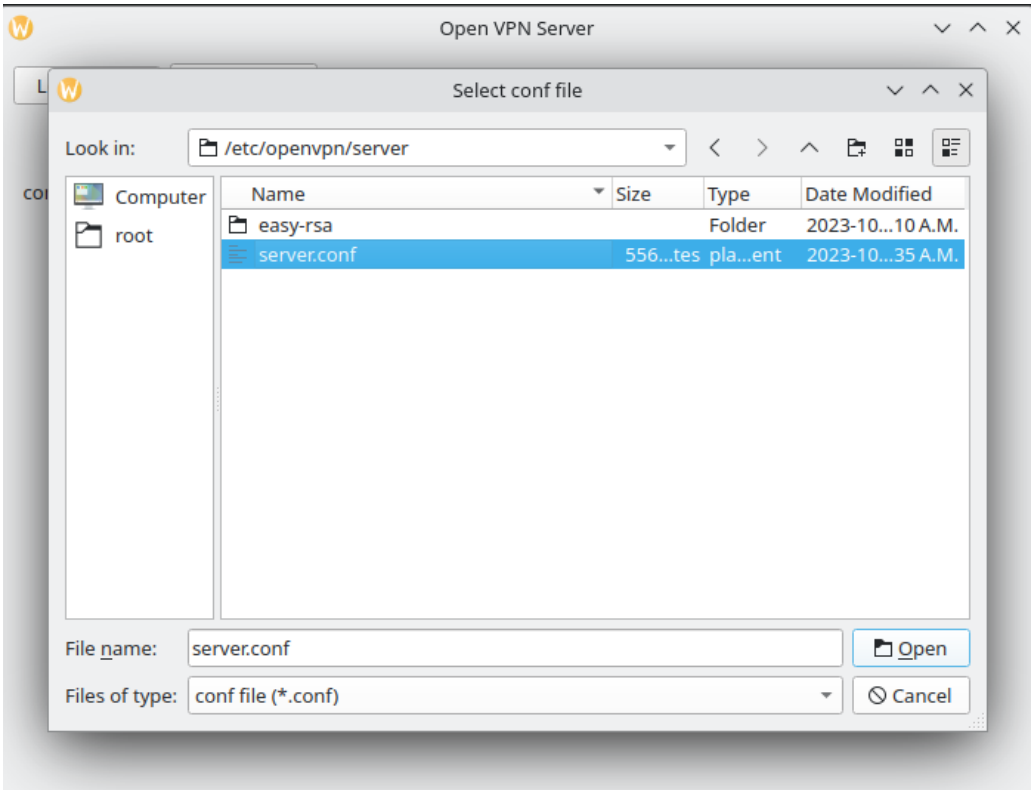


Figure 25: VPN Server- Select Configuration File

- Afterward, the GUI will show the configuration file path. And it is now ready to start the server.

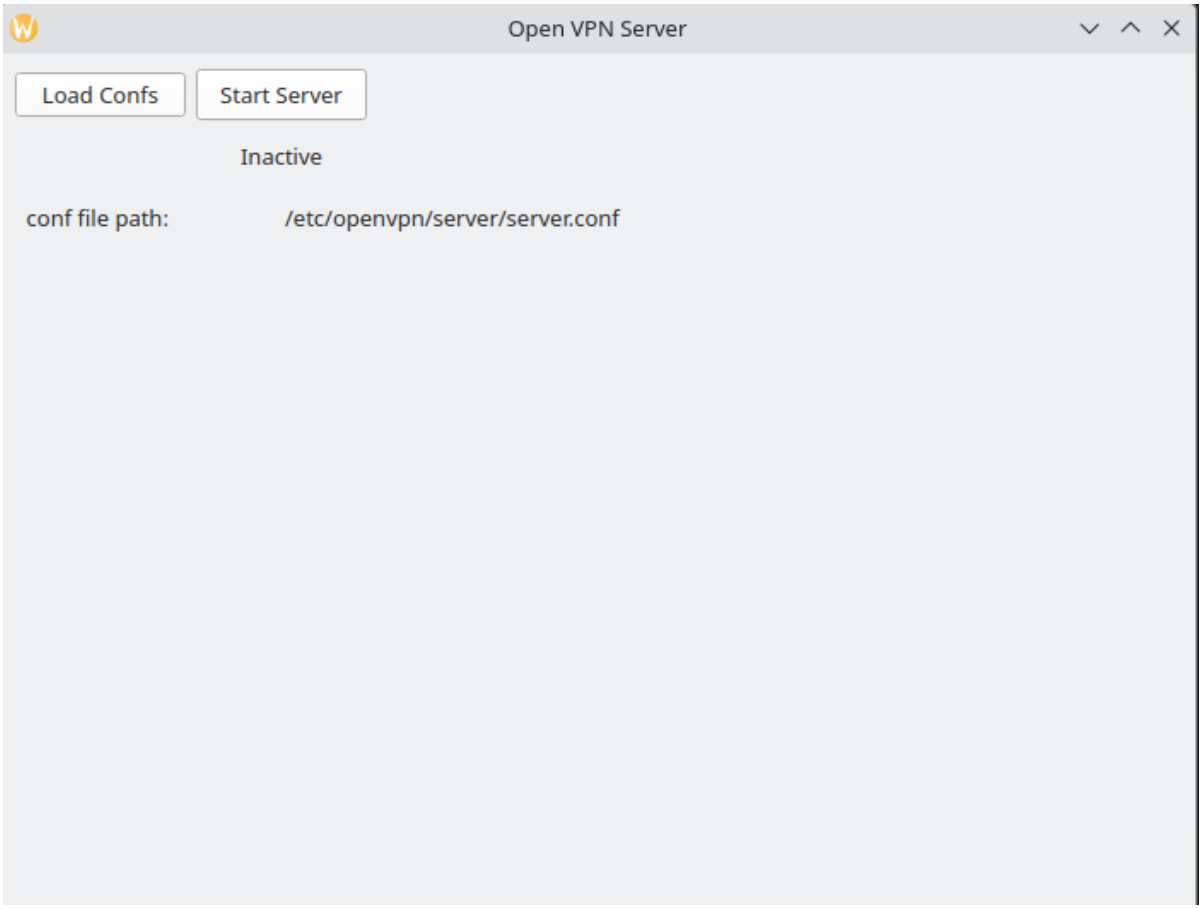


Figure 26: VPN Server- Ready to Start

- Finally, click the “**Start Server”** button to start your own OpenVPN Server!

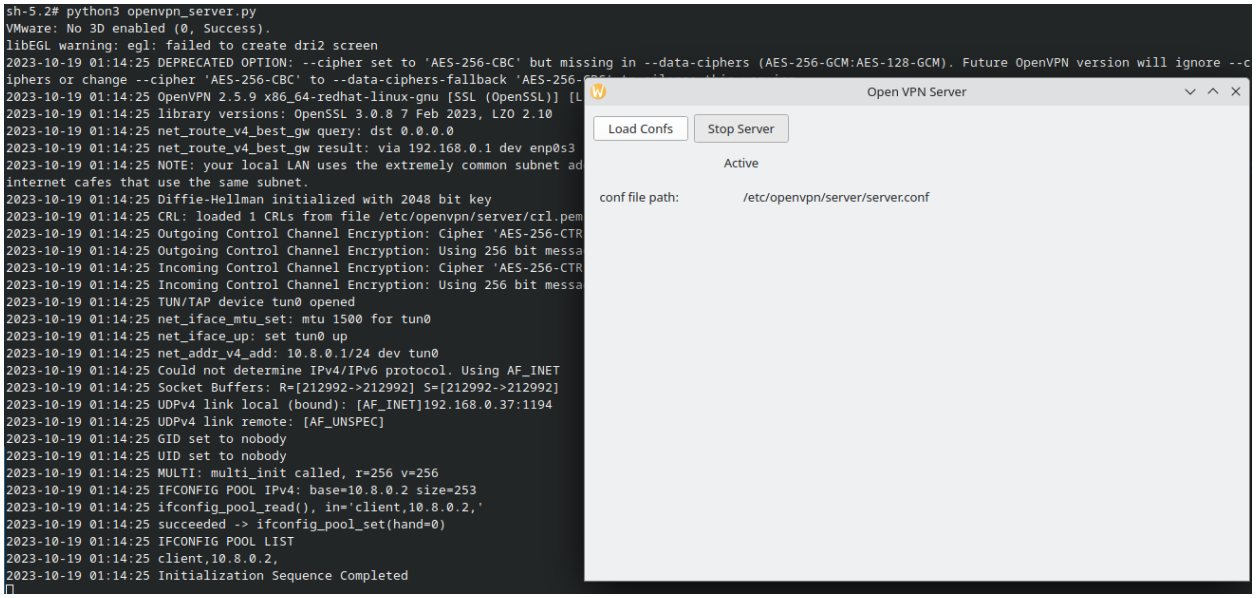


Figure 27: VPN Server- Start Server

- You can also click the stop server button at any time to stop your VPN Server after starting it.

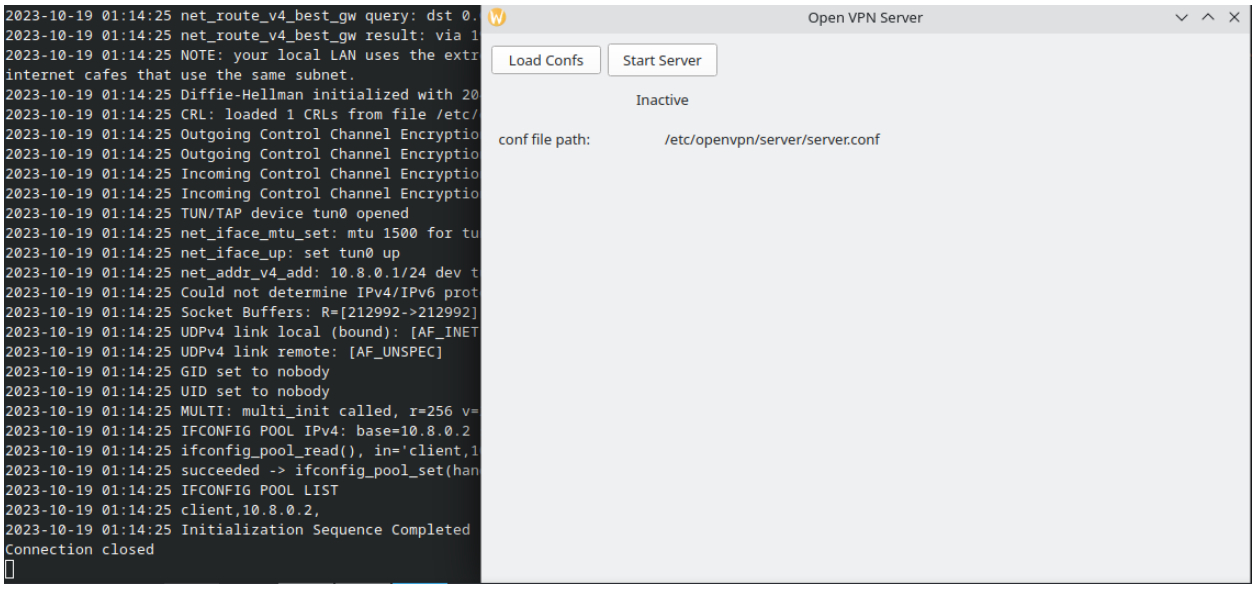


Figure 28: VPN Server- Stop Server

Client

- To access the client application, it is required to contain the following files in your client local device:
  - client1.ovpn
- Also, if your client device does not have OpenVPN installed, you will need to install it before starting the program. You can either install by following the Setup Guide (Section 2.5.4.) or manually type the following command:

dnf install openvpn

- Once OpevVPN is installed on the device, the client program is ready to start by typing the following command:

python3 openvpn_client.py

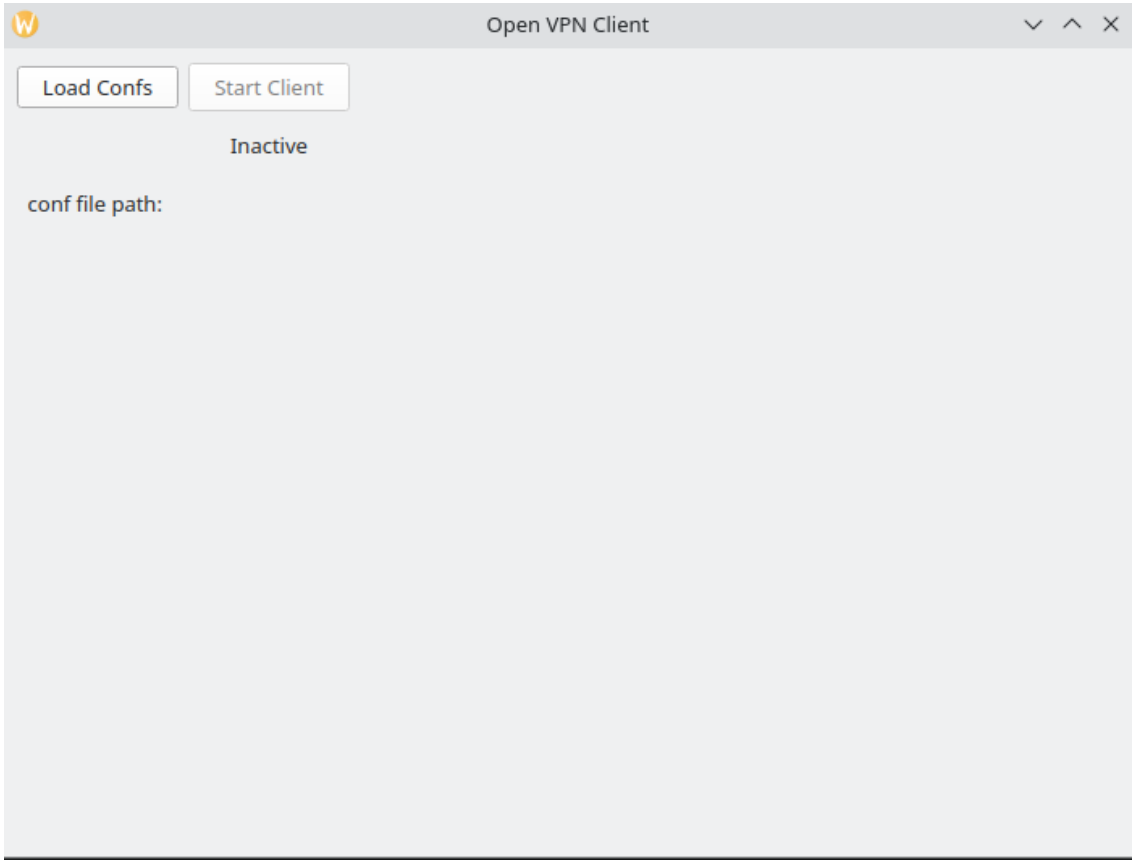


Figure 29: VPN Client GUI

- The GUI of the program will show up after executing the program. Click the "**Load Confs"** button on the **top left** corner of the program and select the configuration file for the OpenVPN client.

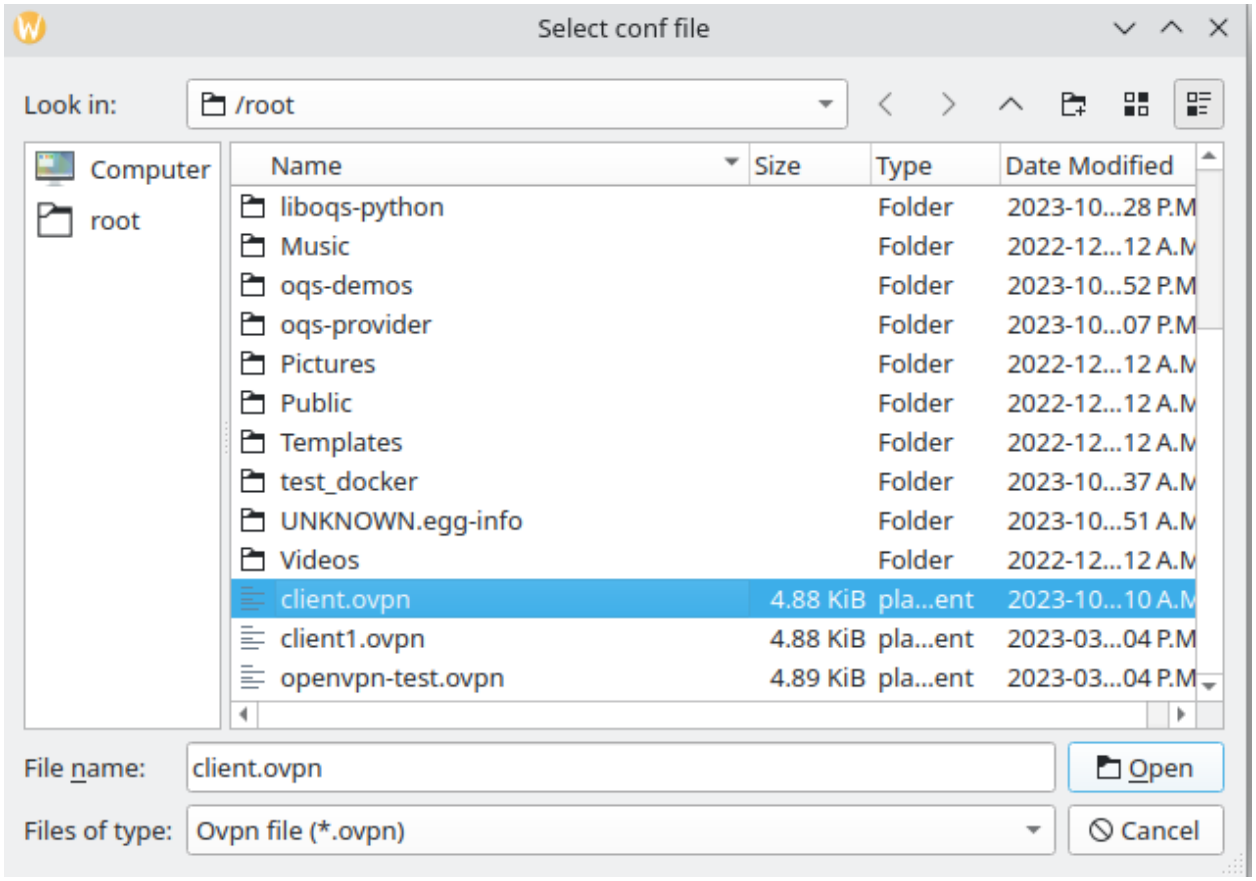


Figure 29: VPN Client- Select Configuration File

- Afterward, the GUI will show the configuration file path. And it is now ready to start the connection.

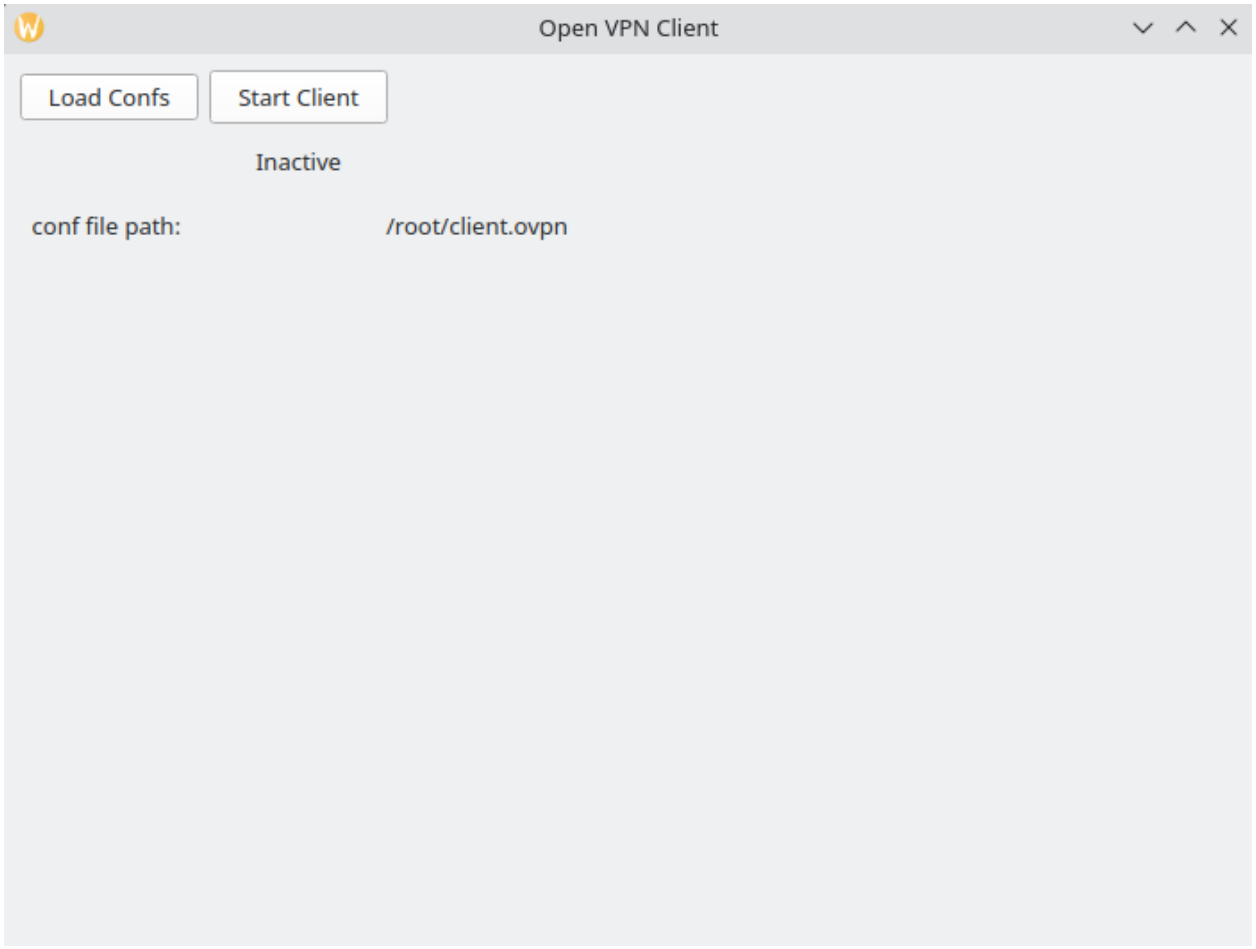


Figure 30: VPN Client- Ready to Start

- Finally, click the “**Start Server”** button to start your own OpenVPN Client!

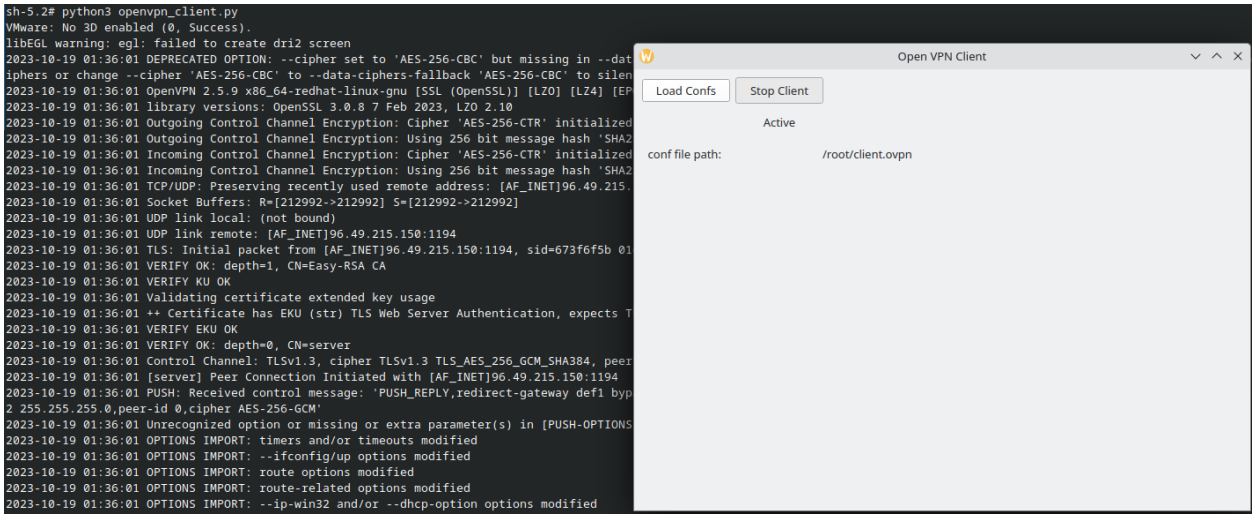


Figure 31: VPN Client- Start Client

- In Figure 31, we can see that the client successfully connects to the server.

- You can also click the “**Stop Client”** button at any time to stop your VPN Client after starting it.

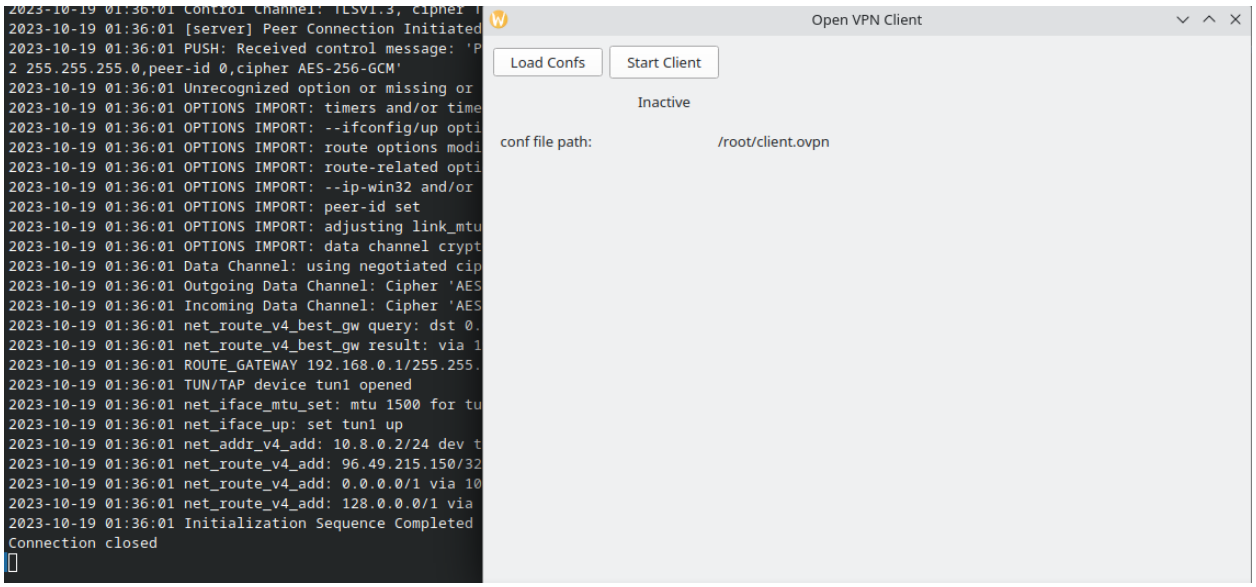


Figure 32: VPN Client- Stop Client

2.5.5.2. Post Quantum Algorithm Client-Server Application

Client (pq_client.py)

- To access the client application, it is required to have liboqs-python library installed on the client device. If not, you can follow the instructions in Section 2.5.3. to get it installed.
- The application requires the following arguments:
  - **-s, --server**: the server's internal IP address
  - **-p, --port**: port number
  - **-o, --option**: KEMs or Signature Schemes algorithm, enter only **KEM** or **SIG/signature**
- Make sure to start the server before starting the client.
- Once everything is ready, execute the following command to start the client:

python pq_client.py -s SERVER_IP -p PORT_NUMBER -o KEM

python pq_client.py -s SERVER_IP -p PORT_NUMBER -o SIG

```
pq_client.py  pq_server.py
sh-5.2# python pq_client.py -s 192.168.0.41 -p 123 -o KEm
/usr/local/lib/python3.10/site-packages/oqs/oqs.py:67: UserWarning: Please install liboqs-python using setup.p
y
  warnings.warn("Please install liboqs-python using setup.py")
/usr/local/lib/python3.10/site-packages/oqs/oqs.py:74: UserWarning: liboqs version 0.9.0 differs from liboqs-p
ython version None
  warnings.warn("liboqs version {} differs from liboqs-python version {}".format(oqs_version(), oqs_python_ver
sion()))
====================Performing Key Exchange using Kyber512====================
Generating Keypair...
Sending Public Key to the server...
Received ciphertext from the server.
Decapsulating the ciphertext from the server with my pulic key...
The shared secret by the client is: b'\xfbb>\xfb\xf9a\xae#B\xe4\x88\x15Y\xc5\xe4yP\xeb\xe4Ve\xf5BF\xe83\xd8\x0
1@\xb1\xa9\xa9'
====================End of Key Exchange (Kyber512)====================
```

```
sh-5.2# python pq_client.py -s 192.168.0.41 -p 123 -o sig
/usr/local/lib/python3.10/site-packages/oqs/oqs.py:67: UserWarning: Please install liboqs-python using setup.p
y
  warnings.warn("Please install liboqs-python using setup.py")
/usr/local/lib/python3.10/site-packages/oqs/oqs.py:74: UserWarning: liboqs version 0.9.0 differs from liboqs-p
ython version None
  warnings.warn("liboqs version {} differs from liboqs-python version {}".format(oqs_version(), oqs_python_ver
sion()))
====================Performing Digital Signature using Dilithium3====================
Received the data from the server.
Verifying the signature...
Signature verified successfully.
====================End of Key Exchange (Dilithium3)====================
```

Figure 33: Post-Quantum Application: Sample Output for Client

Server (pq_server.py)

- To access the client application, it is required to have liboqs-python library installed on the client device. If not, you can follow the instructions in Section 2.5.3. to get it installed.
- The application requires the following arguments:
  - **-s, --server**: the server's internal IP address
  - **-p, --port**: port number
  - **-o, --option**: KEMs or Signature Schemes algorithm, enter only **KEM** or **SIG/signature**
- Once everything is ready, execute the following command to start the server:

python pq_server.py -s SERVER_IP -p PORT_NUMBER -o KEM

python pq_server.py -s SERVER_IP -p PORT_NUMBER -o SIG

```
[root@fedora PQ_Test]# python pq_server.py -s 192.168.0.41 -p 123 -o KEm
/usr/local/lib/python3.11/site-packages/oqs/oqs.py:75: UserWarning: liboqs version 0.9.0 differs from liboqs-p
ython version 0.8.0
  warnings.warn("liboqs version {} differs from liboqs-python version {}".format(oqs_version(), oqs_python_ver
sion()))
Client ('192.168.0.37', 39458) is connected to the server.
====================Performing Key Exchange using Kyber====================
Encapsulating the server's secret using the client's public key...
Sending ciphertext to the client...
Ciphertext sent!
The shared secret by the server is: b'\xfbb>\xfb\xf9a\xae#B\xe4\x88\x15Y\xc5\xe4yP\xeb\xe4Ve\xf5BF\xe83\xd8\x0
1@\xb1\xa9\xa9'
====================End of Key Exchange (Kyber)====================
```

```
[root@fedora PQ_Test]# python pq_server.py -s 192.168.0.41 -p 123 -o sig
/usr/local/lib/python3.11/site-packages/oqs/oqs.py:75: UserWarning: liboqs version 0.9.0 differs from liboqs-p
ython version 0.8.0
  warnings.warn("liboqs version {} differs from liboqs-python version {}".format(oqs_version(), oqs_python_ver
sion()))
Client ('192.168.0.37', 43684) is connected to the server.
====================Performing Digital Signature using Dilithium3====================
Generating the signer's keys...
Signing the message...
Sending data to the client...
====================End of Digital Signature (Dilithium3)====================
```

Figure 34: Post-Quantum Application: Sample Output for Server

2.5.6. System Diagram (VPN Application)

The following diagram is the system diagram for the VPN application with OpenVPN Protocol.

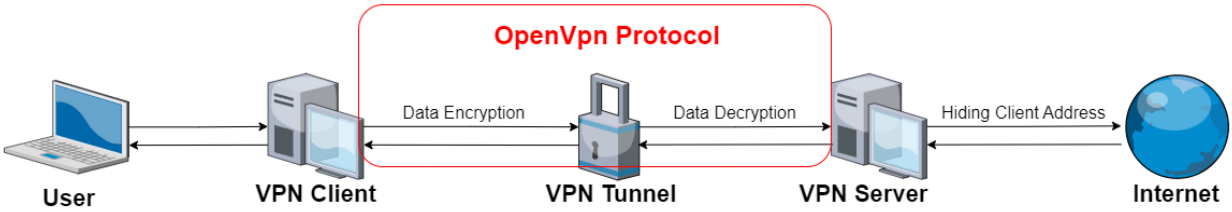


Figure 35: System Diagram of VPN Application

The application is consisting of two parts:

1. **VPN Client**: VPN Client runs on the user's device. It encrypts the data from the user by applying the selected cipher provided by OpenVPN and sends the traffic through VPN Tunnel to the VPN Server.
2. **VPN Server**: VPN Server runs on the server's device. It listens to communications on specified port and receive data. Afterward, it decrypts the received data and makes requests to the internet while hiding the client's address and geolocation.

The application is developed with the OpenVPN protocol and encrypted by the selected cipher provided by OpenSSL. The OpenVPN protocol handles both the certificate signing and verification, as well as the key exchange process during the traffic, to make the entire connection secure.

2.5.7. Software Architecture Diagram (VPN Application)

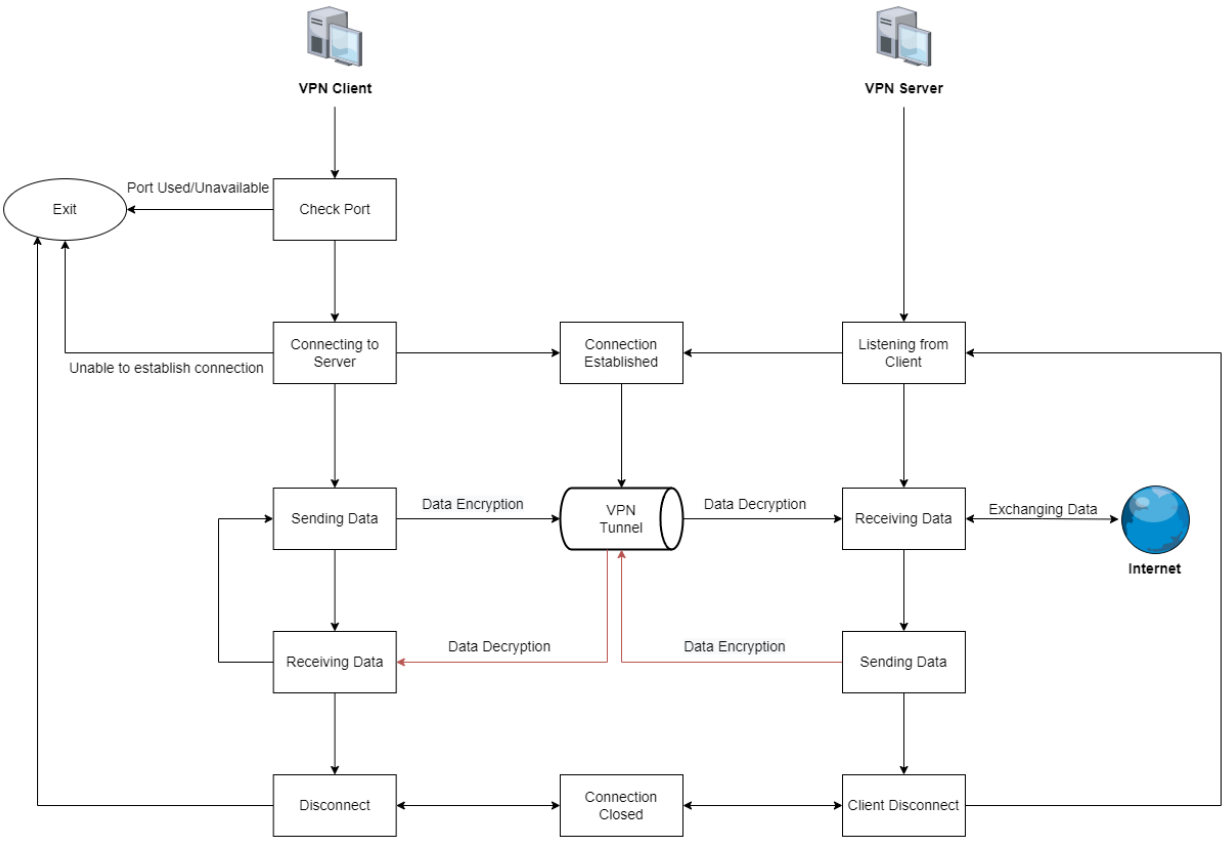


Figure 36: Software Architecture Diagram of VPN Application

The architecture of the software is shown in the figure above. The client starts by checking if the port is either used or unavailable. If it is, the program will exit. Otherwise, it continues to the next step to connect to the server. If it is unable to connect to the server on the port, it will exit. The server, at the same time, listens to connections on the same port. If the server receives requests from the client, the server verifies the certificate and responds to the client if it is valid. Once the connection is established, the client and the server form a VPN tunnel for data transmissions. Data encryption and decryption are done by both ends with their agreement of the encryption ciphers. The server receives the requests from the client and exchange data with the internet. Afterward, the server encrypts the data that it

receives from the internet and sends it back to the client. After the client disconnects, the connection is closed.

2.5.8. Software Architecture Diagram (Post Quantum Client-Server Application)

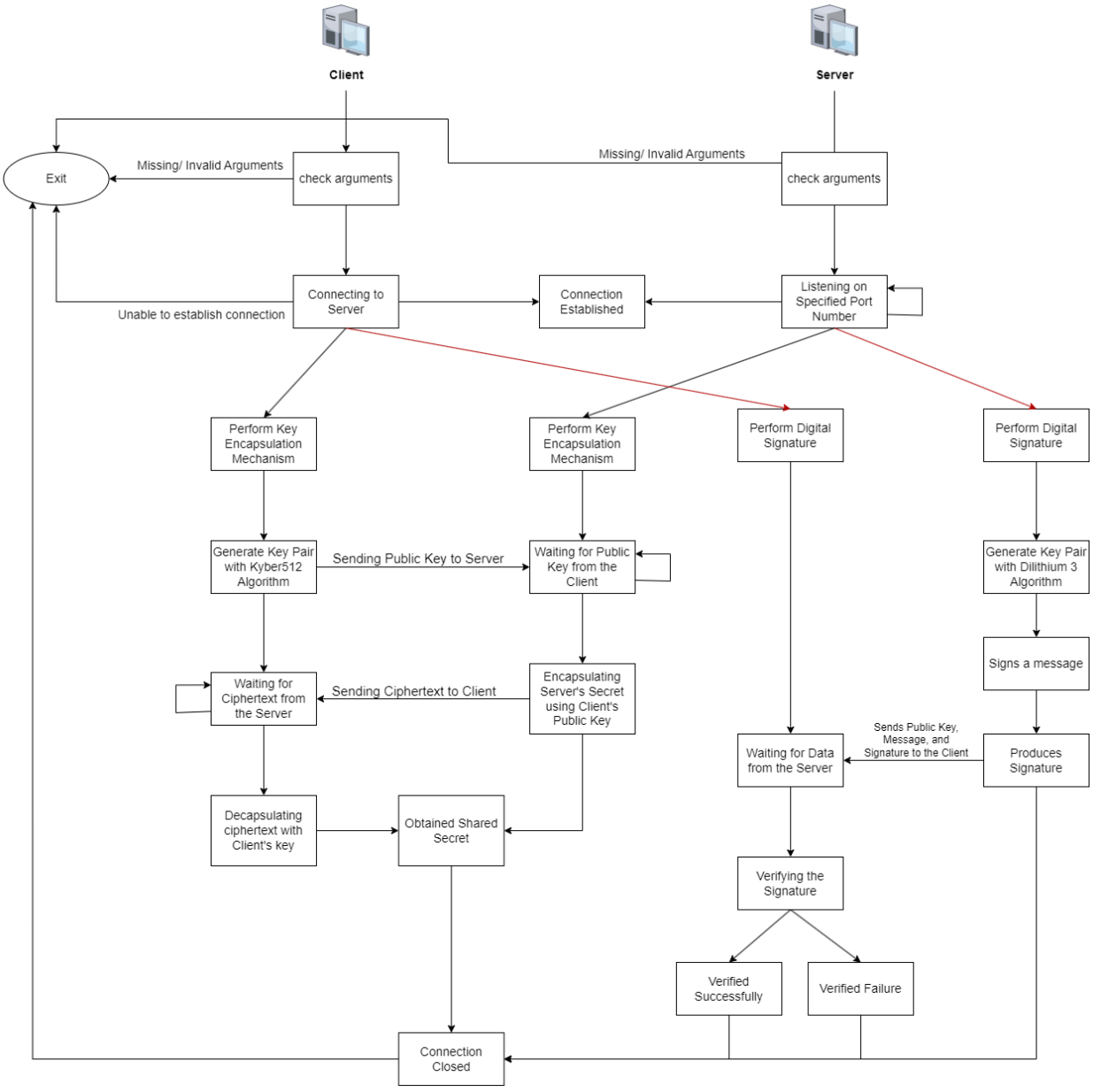


Figure 37: Software Architecture Diagram of Post Quantum Client-Server Application

Architecture

For the architecture, the application is a client-server program, which the client connects to the server that is listening on a specific port to perform communications. The application performs two operations: Key Exchange using KEMs algorithm and Digital Signature using Signature Schemes. Kyber512 is used in Key Exchange, and Dilithium3 is used in Digital Signature.

For Key Exchange, the client first generates its keys (both public and private) and sends its public key to the server. Once the server receives the public key from the client, the server encapsulates the server's secret by using the client's public key to generate a ciphertext. After encapsulation, the server sends the generated ciphertext back to the client. The client then decapsulates the ciphertext with the client's private key to get the secret from the server.

For Digital Signatures, after the client connects to the server, the server first generates the server's keys (both public and private). It then signs a message with the server's private key to produce a certificate. Afterward, the server sends all data (public key, message, and signature) together to the client with delimiter at the end of each data. The client then splits the data into separate parts by identifying the delimiter. Finally, the client verifies the data and determines whether it is valid or not.

Kyber and Dilithium

Kyber and Dilithium are post-quantum cryptographic algorithms, specifically designed to resist attacks from quantum computers. These algorithms serve different purposes within the realm of public-key cryptography:

- Kyber (Key Encapsulation Mechanism):

  Kyber is a key encapsulation mechanism (KEM) designed to provide secure key exchange. It's used to establish a shared secret key between two parties while protecting the confidentiality and integrity of the key. The core of Kyber is based on lattice-based cryptography, specifically using the Learning With Errors (LWE) problem as a foundation. This problem is believed to be hard for both classical and quantum computers.

  Kyber generates public and private key pairs. The public key can be safely shared with others, while the private key is kept secret. When two parties want to establish a shared secret, one party uses the recipient's public key to encapsulate a random secret key, and the recipient uses their private key to decapsulate the shared secret.

- Dilithium (Signature Scheme):

  Dilithium is a post-quantum digital signature scheme. Its primary purpose is to provide secure digital signatures, ensuring the authenticity and integrity of data. Like Kyber, Dilithium is also based on lattice-based cryptography, specifically targeting the hardness of the Ring Learning With Errors (Ring-LWE) problem. This problem is resistant to attacks by both classical and quantum computers.

  Dilithium generates a public and private key pair for digital signatures. To sign a message, the private key holder uses their private key to create a signature for the message. The signature can be verified by anyone using the corresponding public key to ensure the message's authenticity and integrity.

2.5.9. System Use Case Diagram (VPN Application)

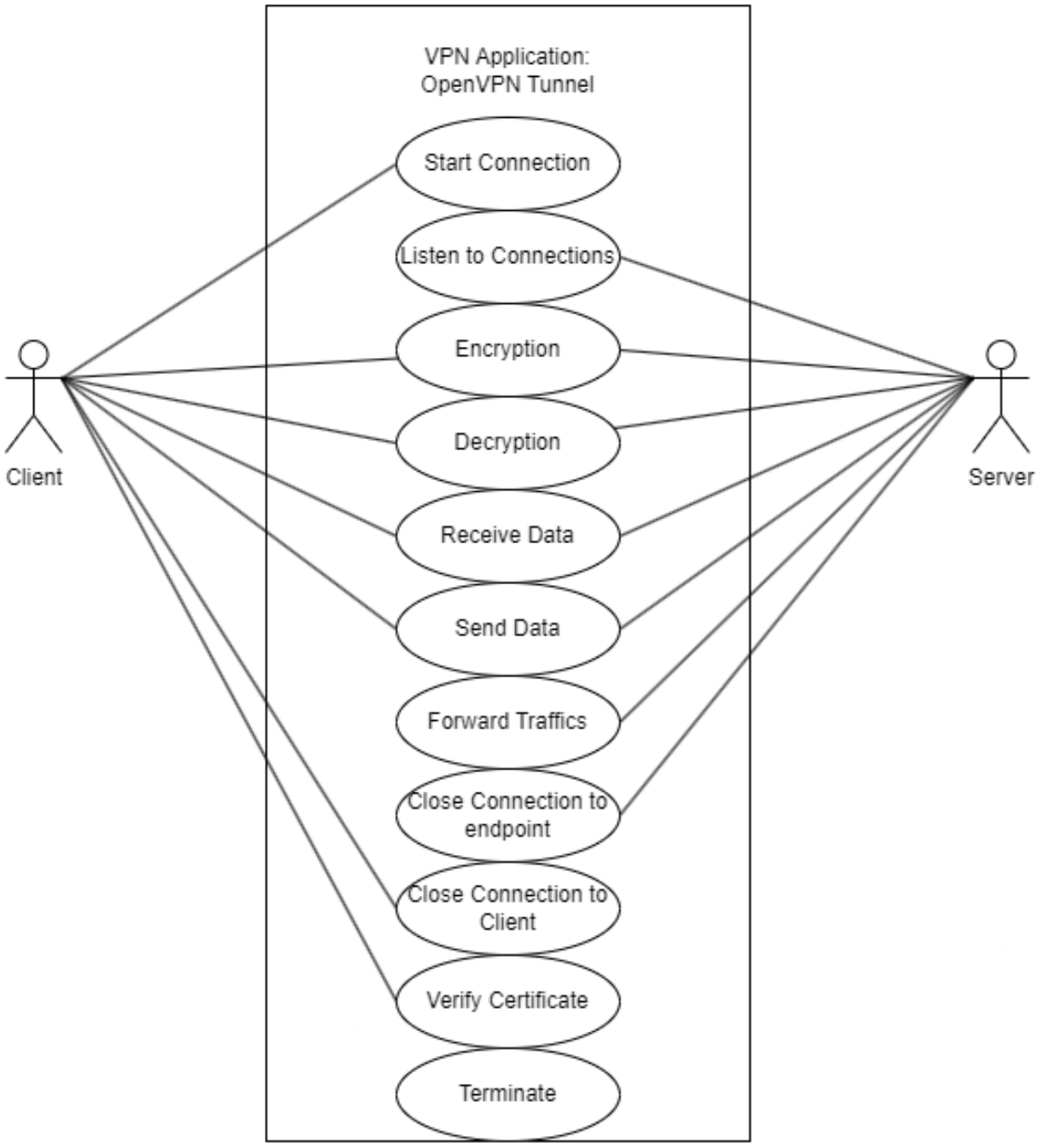


Figure 38: Use Case Diagram of VPN Application

2.5.10. Network Diagram of VPN Application

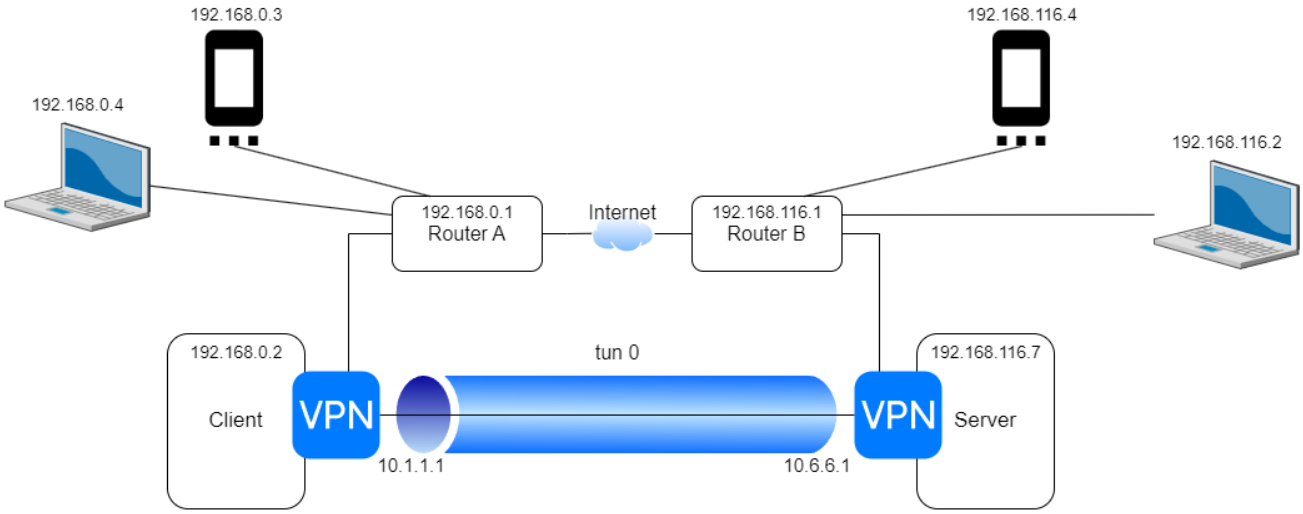


Figure 39: Network Diagram of VPN Application

The Network Diagram shows how the VPN is involved in the entire network.

- Only the Client (192.168.0.2) and the Server (192.168.116.7) have the VPN turned on.
- Both the client and the server require encryption with the AES-256 algorithm in CBC mode. The AES algorithm uses a 256-bit key.
- Both the client and the server require authentication with the SHA-2 algorithm. The SHA-2 algorithm uses a 512-bit key.
- Both the client and the server use shared security associations.

2.5.11. GUI UML Diagram

The figure below shows the UML Diagram for the GUI of the server and the client program. The GUI is built with the PyQt library, which contains several objects for the screen and the elements. It starts with the main window at the bottom that is responsible for the main layout elements such as the buttons and the displayed text. On top of that are the widgets that are responsible for displaying the selection of the config files.

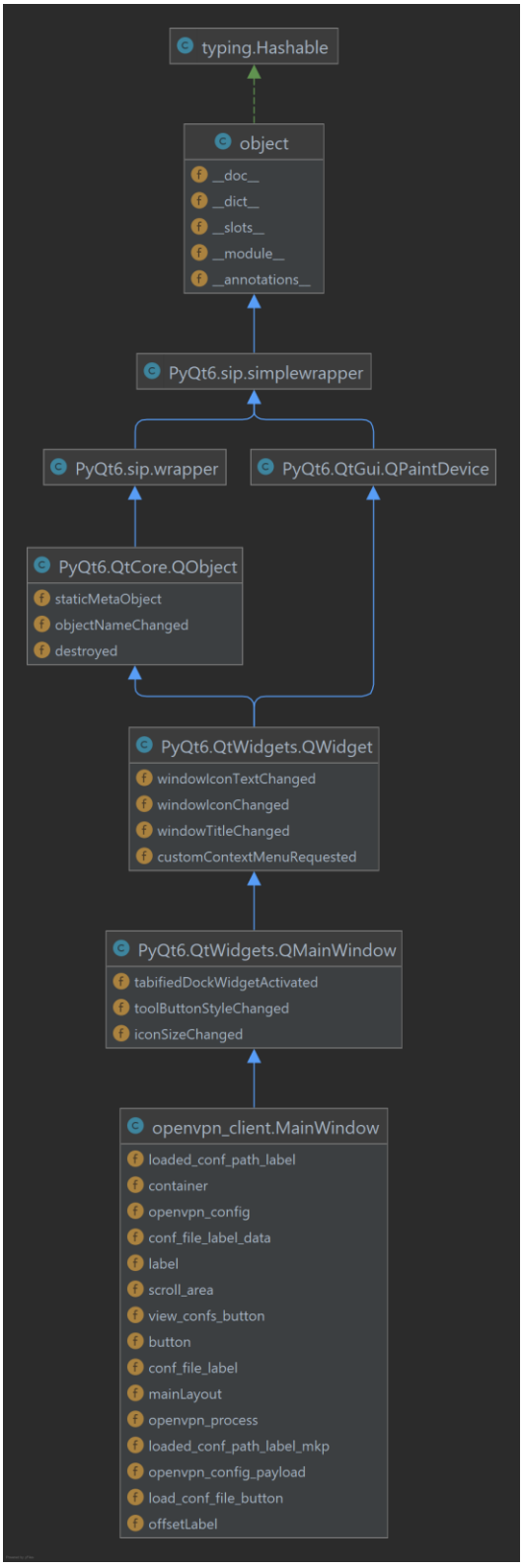


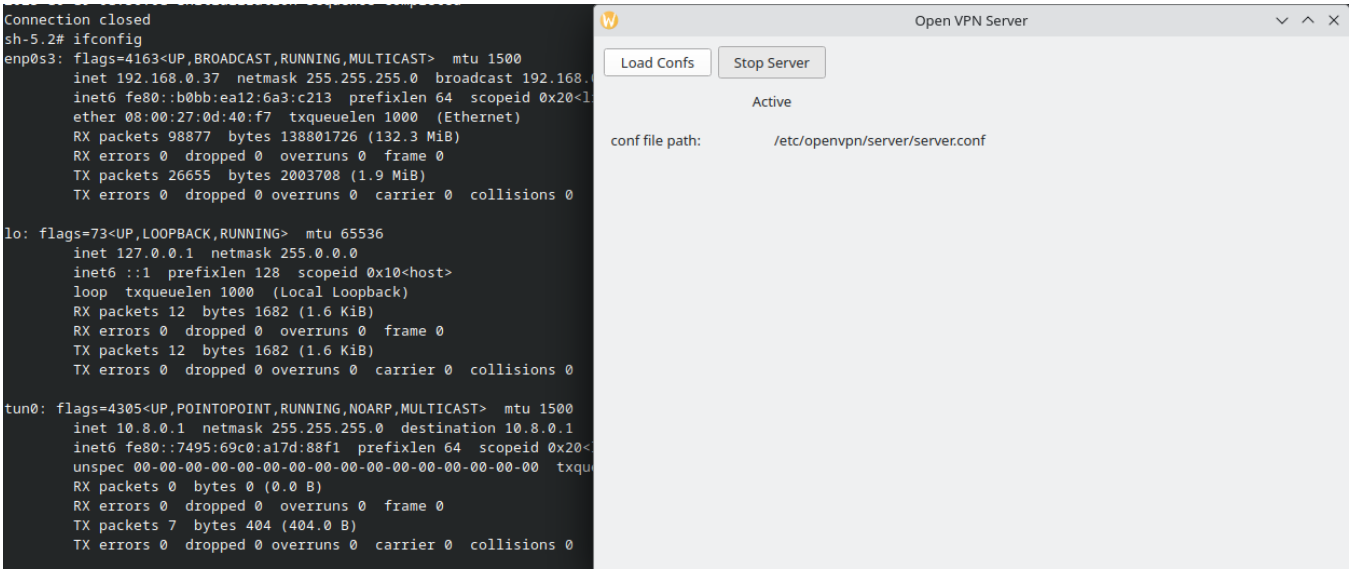
Figure 40: GUI UML Diagram

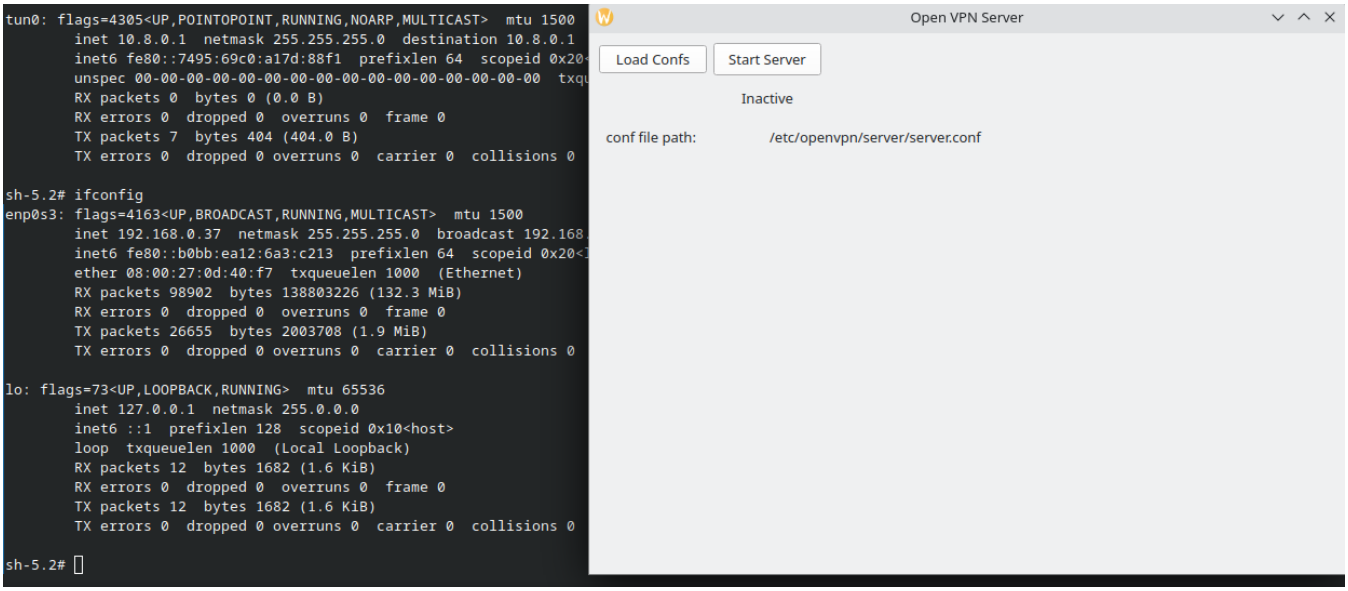
2.6. Testing Details and Results

2.6.1. VPN with OpenVPN Protocol

Test Case Number	Description	Status
#1	The server's VPN tunneling functions normally. Pass: The tunnel is opened when the server turns on the VPN, and the tunnel is closed when the server turns off. Fail: The tunnel is either closed while server is on or opened while the server is off.	Pass
#2	The client's VPN tunneling functions normally. Pass: The tunnel is opened when the client connects to the server, and the tunnel is closed when the client does not connect to the server. Fail: The tunnel is either closed while client is connected to the server or opened while the client is not connected to the server.	Pass
#3	Traffic is forwarded through the tunnel to the server. Pass: The server receives the network request. Fail: The server does not receive the network request.	Pass
#4	The client's address and geolocation are hidden. Pass: The IP address on the internet is the server's address. Fail: The IP address on the internet is the client's address.	Pass
#5	Two or more clients can secure their addresses and geolocations. Pass: The IP addresses on the internet for all clients are all the server's address. Fail: One or none of the IP addresses are the server's address.	Pass
#6	Data is secured with encryption. Pass: The tokens are all encrypted. Fail: The tokens are not encrypted.	Pass

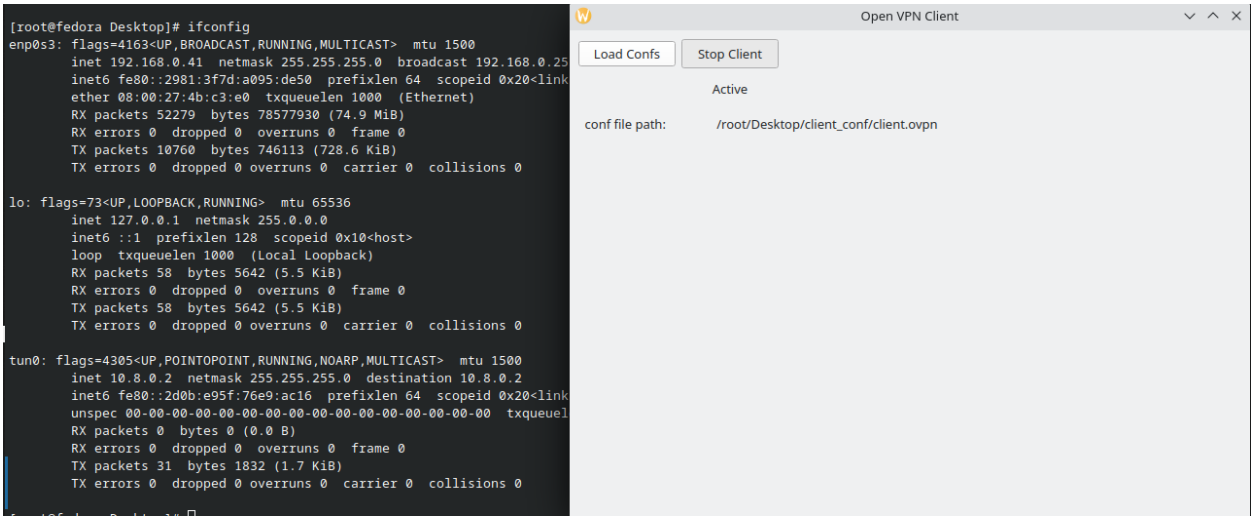
Test Case #1: The server's VPN tunneling functions normally.

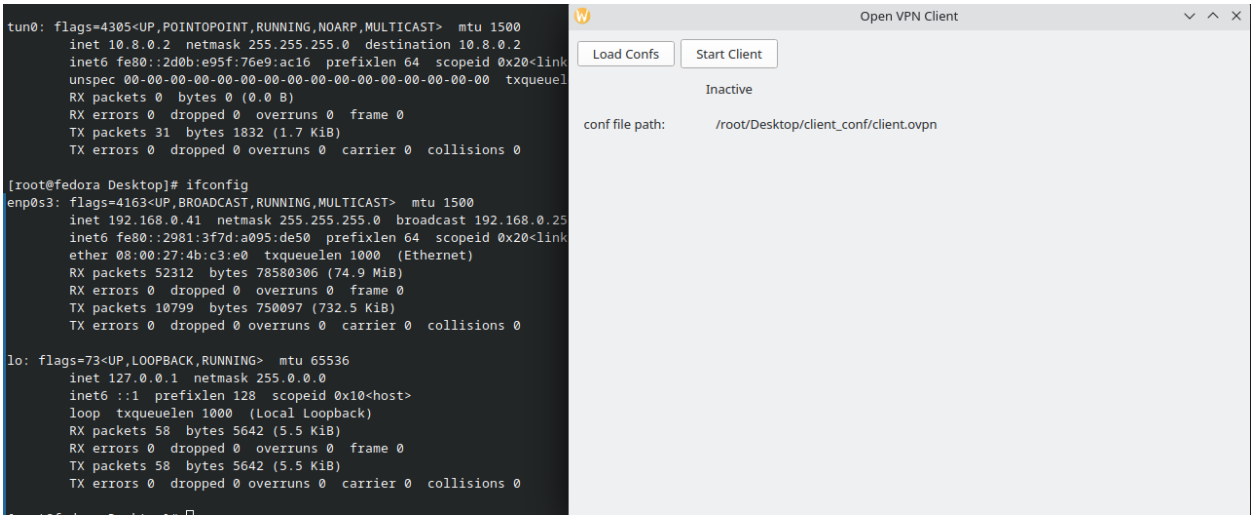




The tunnel is created while the server is on, and the tunnel is closed after the server is turned off.

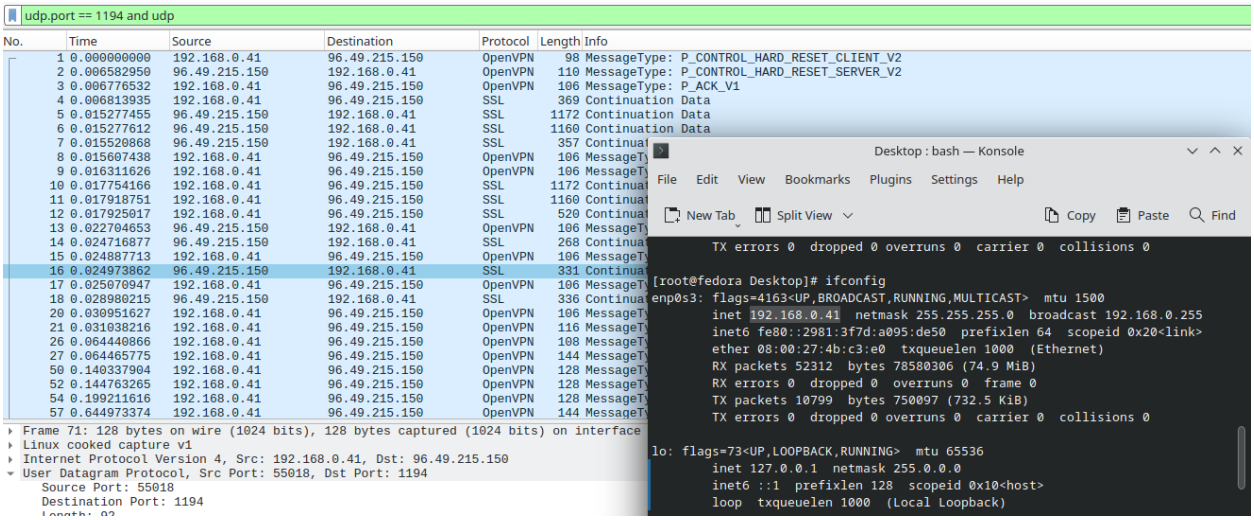
Test Case #2: The client's VPN tunneling functions normally.

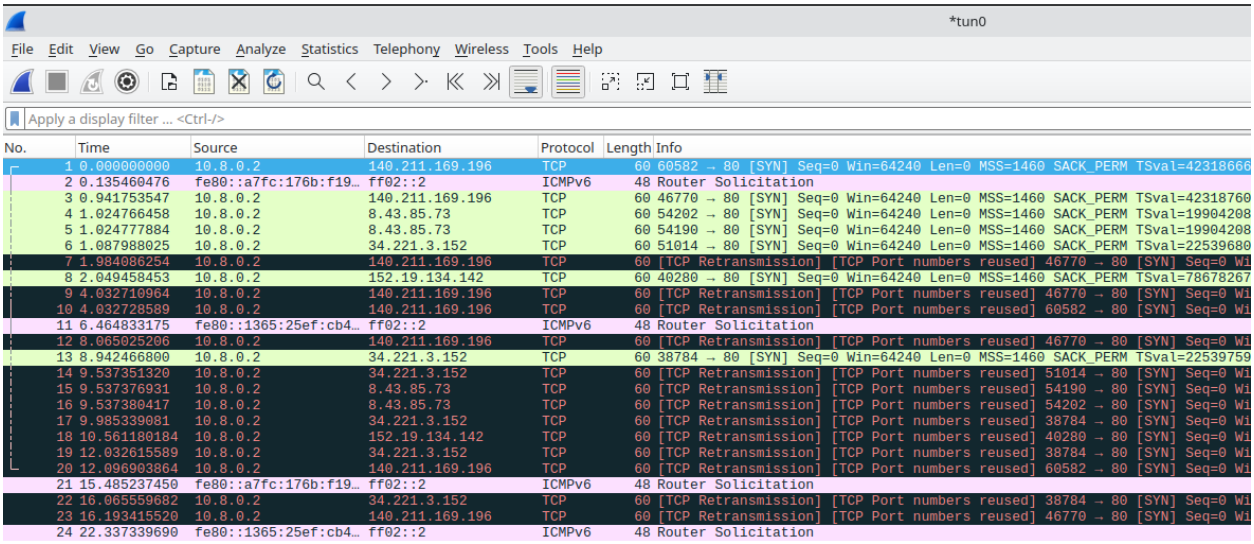




The tunnel is opened when the client connects to the server, and the tunnel is closed when the client does not connect to the server.

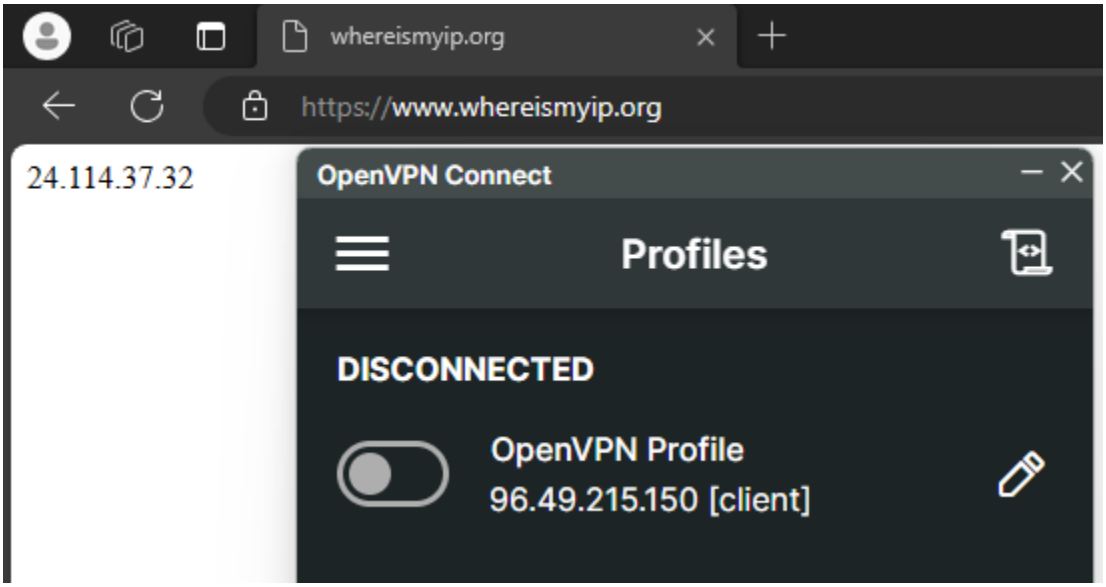
Test Case #3: Traffic is forwarded through the tunnel to the server.

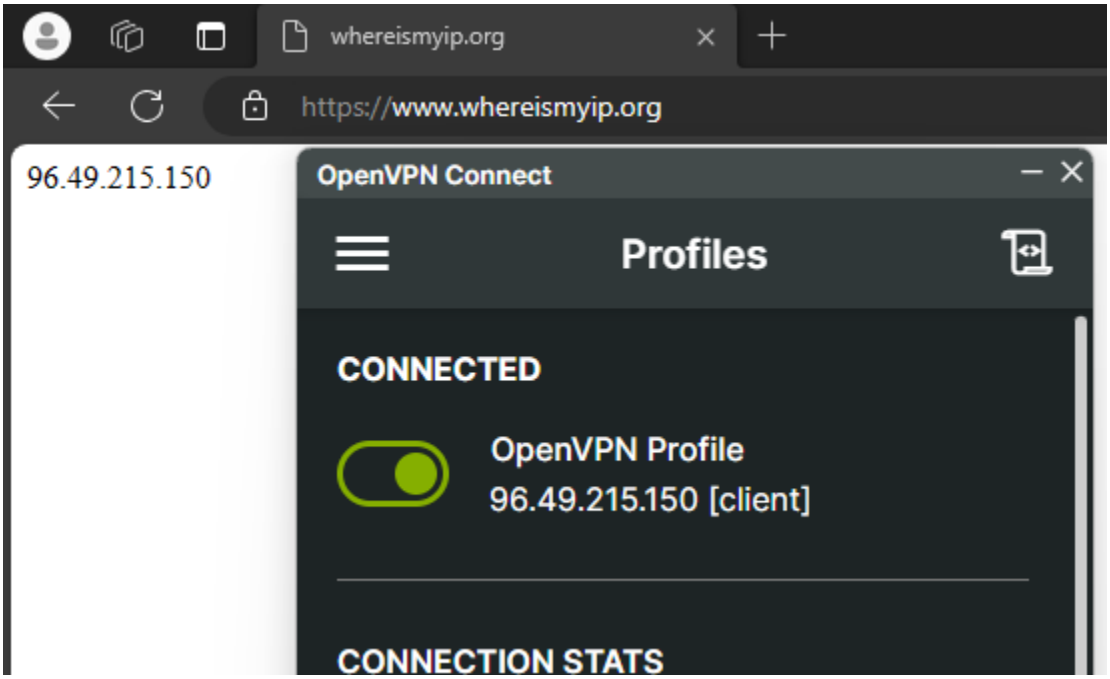




The package capture shows that the client (local ip: 192.168.0.41) is talking to the server (public ip: 96.49.215.150) while the connection is on. The other package capture captures the tunnel directly.

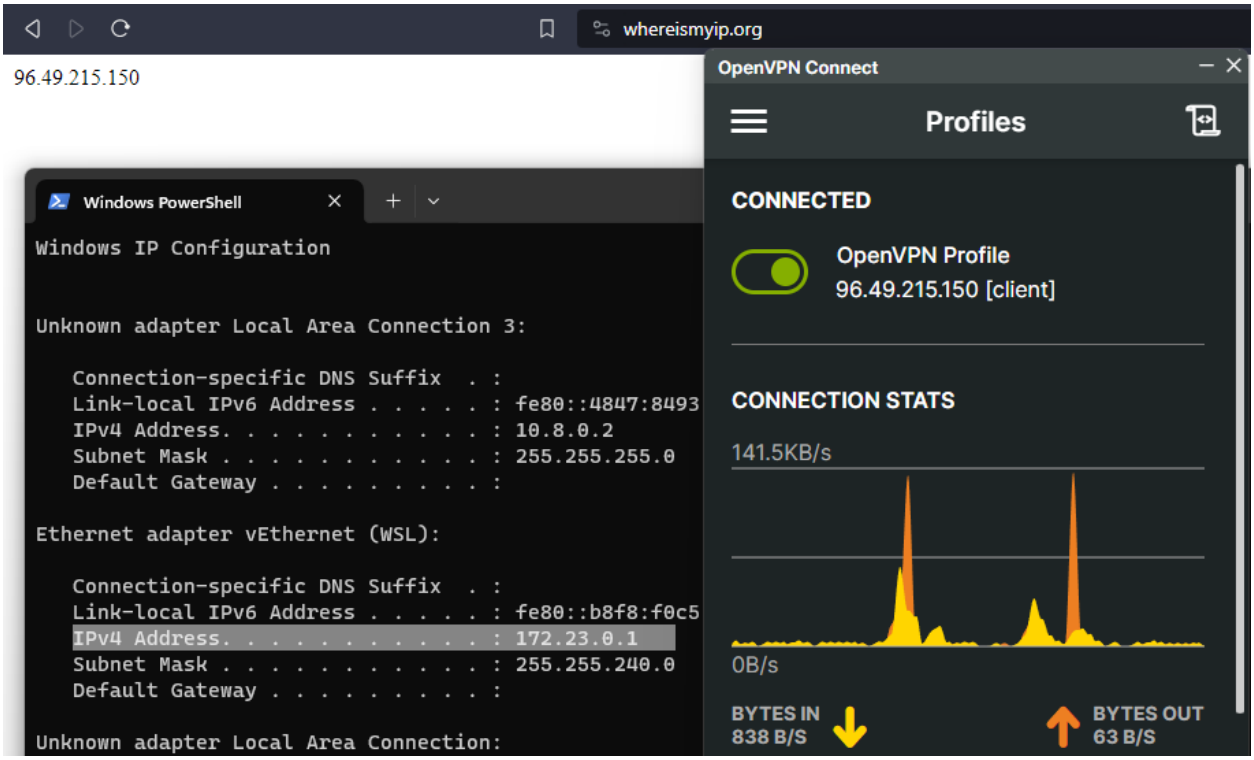
Test Case #4: The client's address and geolocation are hidden.

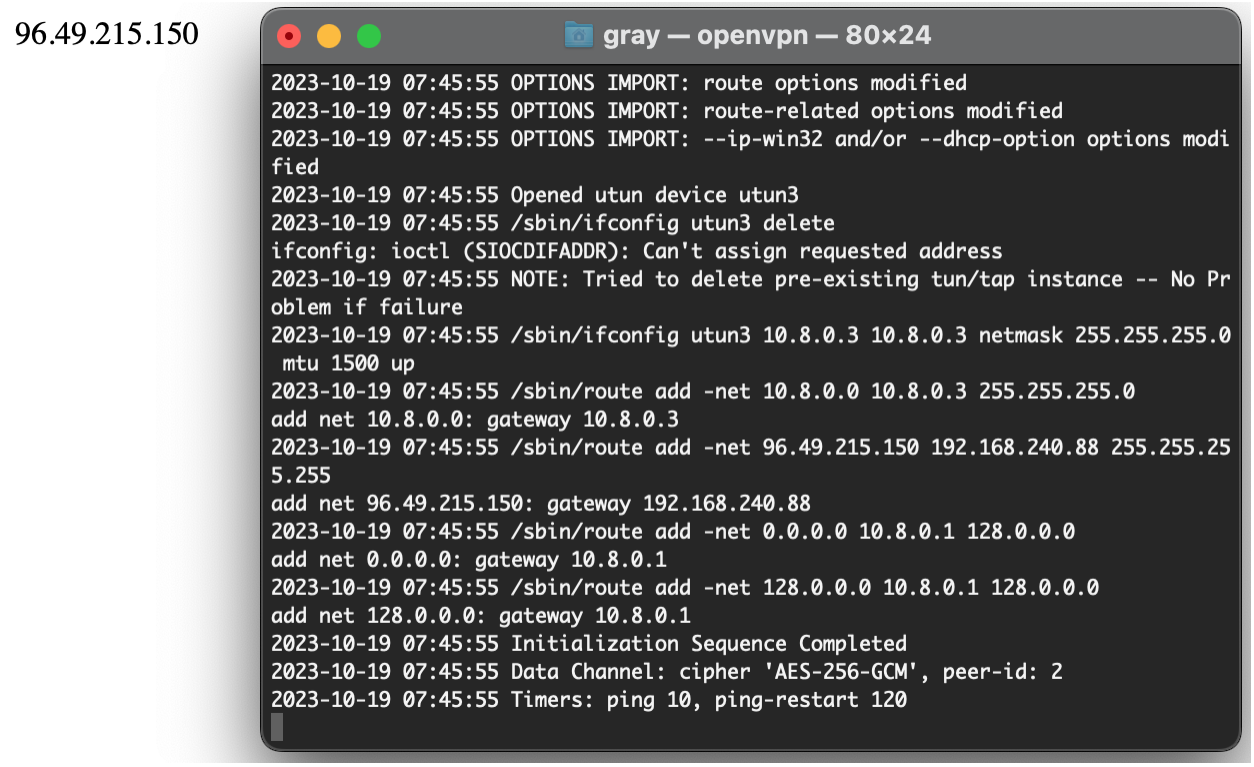




The IP address on the internet is the server's address (client is on cellular data).

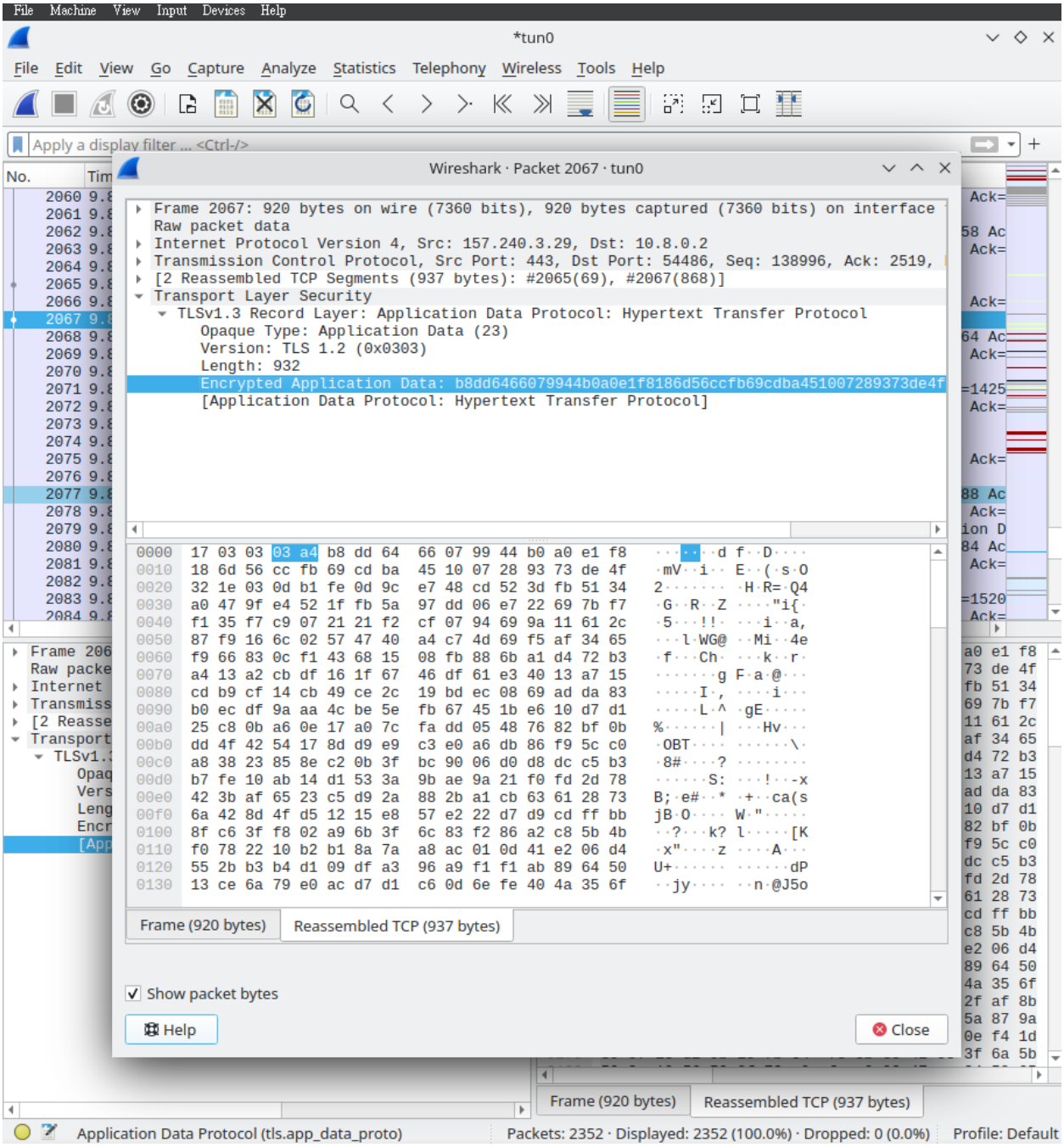
Test Case #5: Two or more clients can secure their addresses and geolocations.





The IP addresses on the internet for both clients are all the server's address (both clients are on cellular data: 24.114.38.90)

Test Case #6: Data is secured with encryption.



The capture shows that the traffic in the tunnel contains encrypted tokens.

2.6.2. Post Quantum Algorithm Client-Server Application

Test Case Number	Description	Status
#1	The server sends a secret that is encapsulated with the Kyber algorithm. Pass: The shared secret is the same on the client and the server's end. Fail: The shared secret is different on the client and the server's end.	Pass
#2	The client verifies a signature that was encrypted by the Dilithium algorithm by using the server's public key. Pass: The client successfully verifies the signature with the server's public key. Fail: The client fails to verify the signature with the server's public key.	Pass
#3	Data is secured with encryption. Pass: The tokens are all encrypted. Fail: The tokens are not encrypted.	Pass

Test Case #1: The server sends a secret that is encapsulated with the Kyber algorithm.

```
sh-5.2# python pq_client.py -s 192.168.0.41 -p 123 -o KEm
/usr/local/lib/python3.10/site-packages/oqs/oqs.py:67: UserWarning: Please install liboqs-python using setup.p
y
  warnings.warn("Please install liboqs-python using setup.py")
/usr/local/lib/python3.10/site-packages/oqs/oqs.py:74: UserWarning: liboqs version 0.9.0 differs from liboqs-p
ython version None
  warnings.warn("liboqs version {} differs from liboqs-python version {}".format(oqs_version(), oqs_python_ver
sion()))
====================Performing Key Exchange using Kyber512====================
Generating Keypair...
Sending Public Key to the server...
Received ciphertext from the server.
Decapsulating the ciphertext from the server with my pulic key...
The shared secret by the client is: b'\xfbb>\xfb\xf9a\xae#B\xe4\x88\x15Y\xc5\xe4yP\xeb\xe4Ve\xf5BF\xe83\xd8\x0
1@\xb1\xa9\xa9'
====================End of Key Exchange (Kyber512)====================
```

```
[root@fedora PQ_Test]# python pq_server.py -s 192.168.0.41 -p 123 -o KEm
/usr/local/lib/python3.11/site-packages/oqs/oqs.py:75: UserWarning: liboqs version 0.9.0 differs from liboqs-p
ython version 0.8.0
  warnings.warn("liboqs version {} differs from liboqs-python version {}".format(oqs_version(), oqs_python_ver
sion()))
Client ('192.168.0.37', 39458) is connected to the server.
====================Performing Key Exchange using Kyber====================
Encapsulating the server's secret using the client's public key...
Sending ciphertext to the client...
Ciphertext sent!
The shared secret by the server is: b'\xfbb>\xfb\xf9a\xae#B\xe4\x88\x15Y\xc5\xe4yP\xeb\xe4Ve\xf5BF\xe83\xd8\x0
1@\xb1\xa9\xa9'
====================End of Key Exchange (Kyber)====================
```

The shared secret is the same on the client and the server's end.

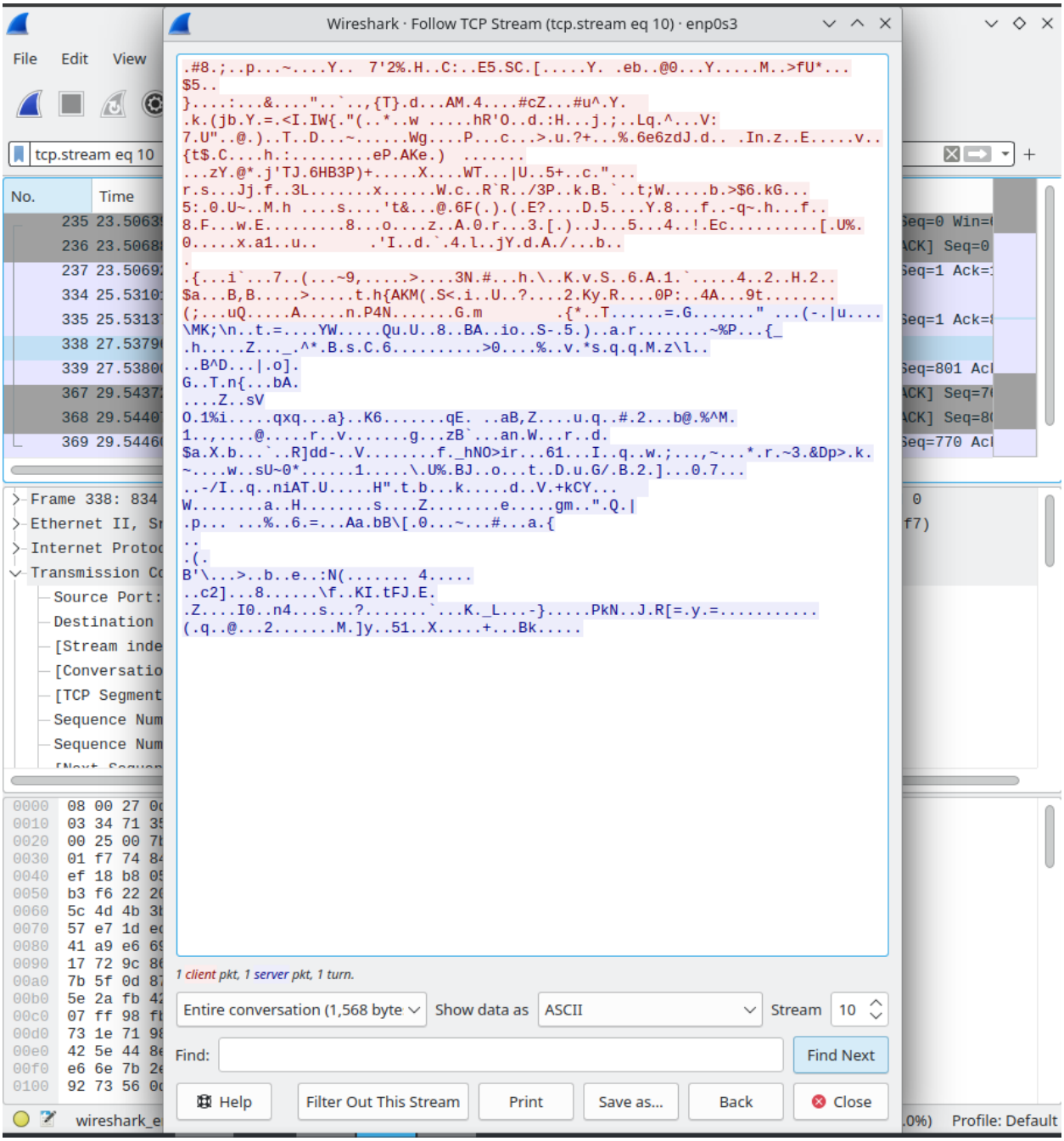
Test Case #2: The client verifies a signature that was encrypted by the Dilithium algorithm by using the server's public key.

```
[root@fedora PQ_Test]# python pq_server.py -s 192.168.0.41 -p 123 -o sig
/usr/local/lib/python3.11/site-packages/oqs/oqs.py:75: UserWarning: liboqs version 0.9.0 differs from liboqs-p
ython version 0.8.0
  warnings.warn("liboqs version {} differs from liboqs-python version {}".format(oqs_version(), oqs_python_ver
sion()))
Client ('192.168.0.37', 43684) is connected to the server.
====================Performing Digital Signature using Dilithium3====================
Generating the signer's keys...
Signing the message...
Sending data to the client...
====================End of Digital Signature (Dilithium3)====================
```

```
sh-5.2# python pq_client.py -s 192.168.0.41 -p 123 -o sig
/usr/local/lib/python3.10/site-packages/oqs/oqs.py:67: UserWarning: Please install liboqs-python using setup.p
y
  warnings.warn("Please install liboqs-python using setup.py")
/usr/local/lib/python3.10/site-packages/oqs/oqs.py:74: UserWarning: liboqs version 0.9.0 differs from liboqs-p
ython version None
  warnings.warn("liboqs version {} differs from liboqs-python version {}".format(oqs_version(), oqs_python_ver
sion()))
====================Performing Digital Signature using Dilithium3====================
Received the data from the server.
Verifying the signature...
Signature verified successfully.
====================End of Key Exchange (Dilithium3)====================
```

The client successfully verifies the signature with the server's public key.

Test Case #3: Data is secured with encryption.

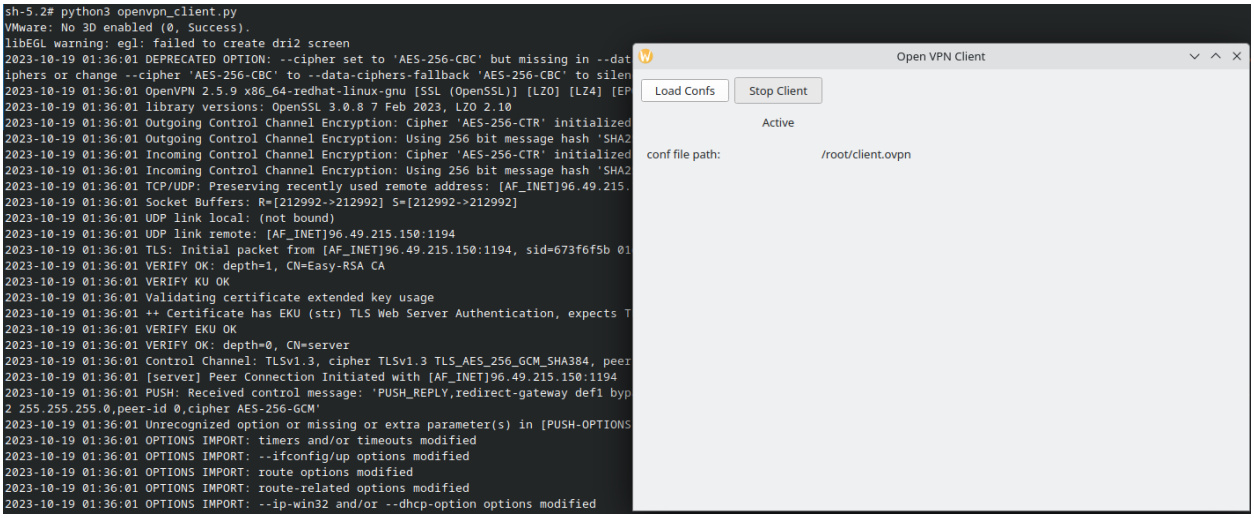


The tokens are all encrypted.

2.6.3. GUI Integration

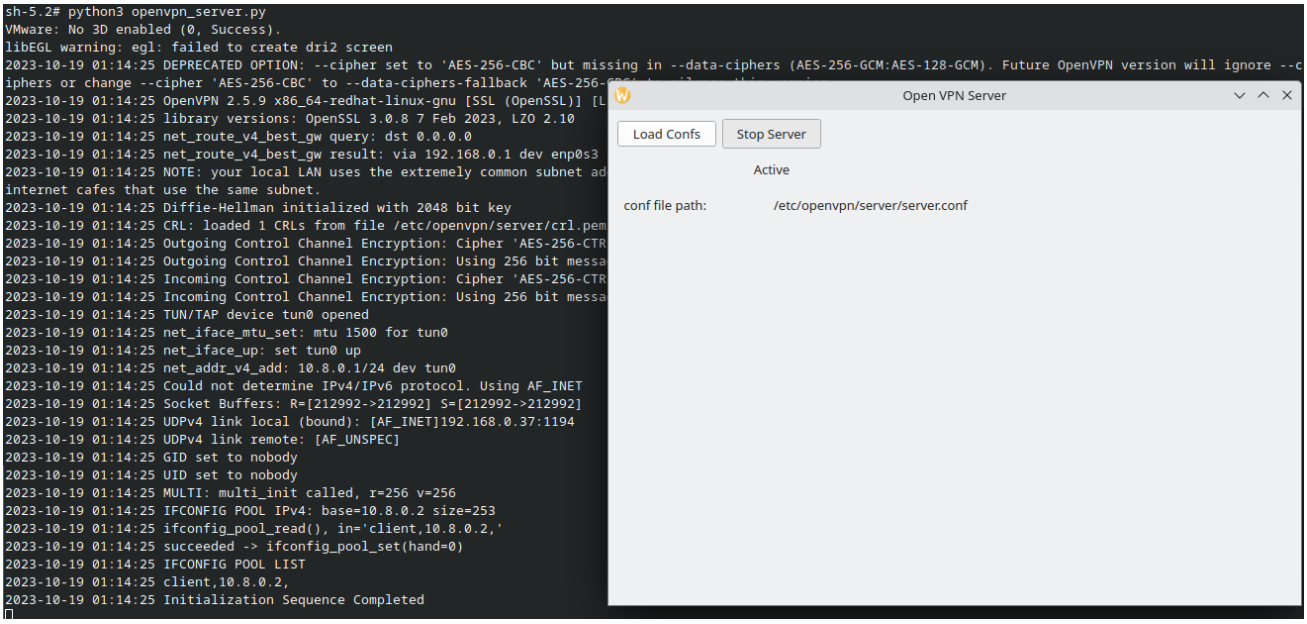
Test Case Number	Description	Status
#1	Client can connect to the server without crashing. Pass: GUI successfully responds to the client and connects to the server. Fail: GUI does not respond to the client or crashes.	Pass
#2	Server can start the service without crashes. Pass: GUI successfully responds to the server and starts the application. Fail: GUI does not respond to the server or crashes if port is currently used.	Pass
#3	Client selects other files that are not with the extension .ovpn. Pass: GUI does not allow the client to select files that do not end in .ovpn extension. Fail: GUI allows the client to select files that do not end in .ovpn extension.	Pass
#4	Server selects other files that are not with the extension .conf. Pass: GUI does not allow the server to select files that do not end in .conf extension. Fail: GUI allows the server to select files that do not end in .conf extension.	Pass
#5	Traffic is collected in a log file. Pass: All traffic is written in a log file. Fail: Traffic is not logged in a file.	Pass

Test Case #1: Client can connect to the server without crashing.



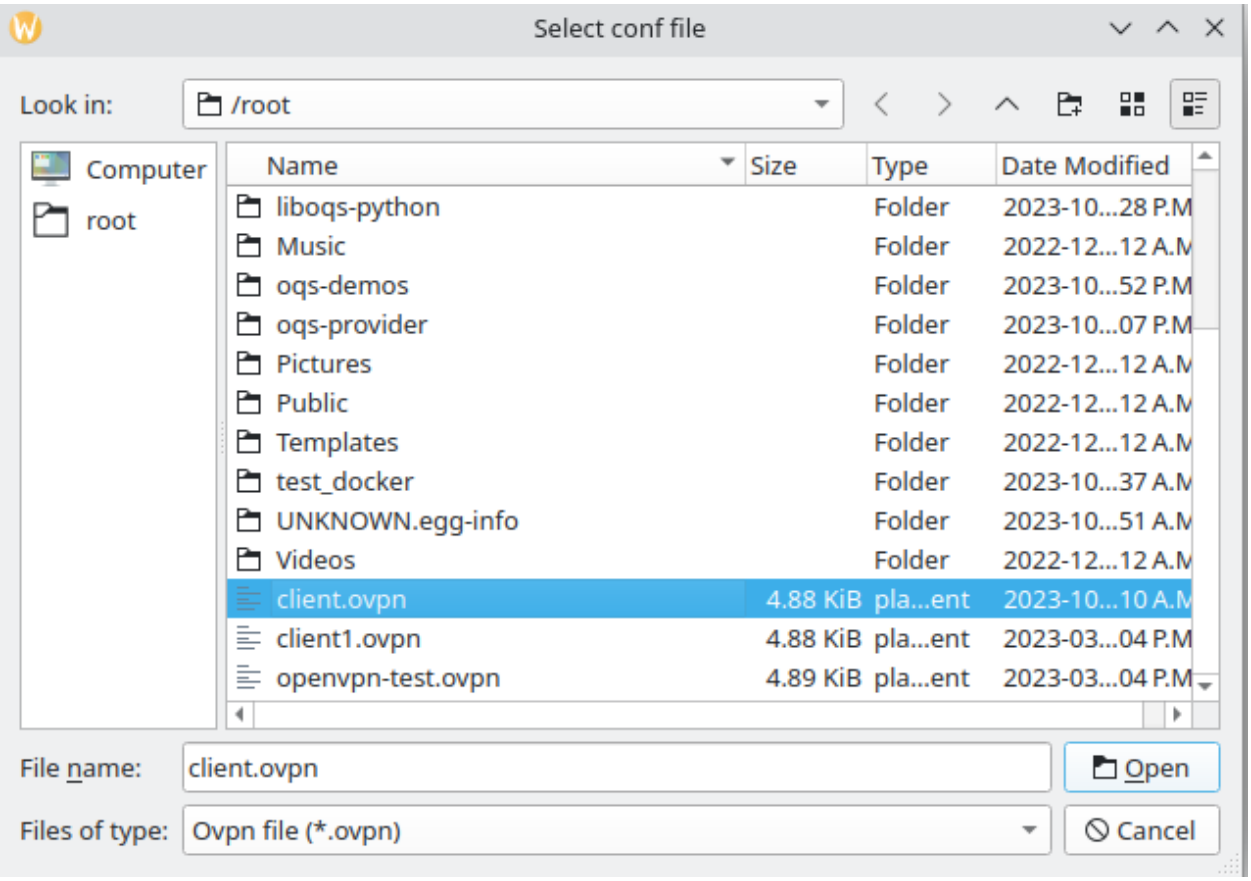
GUI successfully responds to the client and connects to the server.

Test Case #2: Server can start the service without crashes.



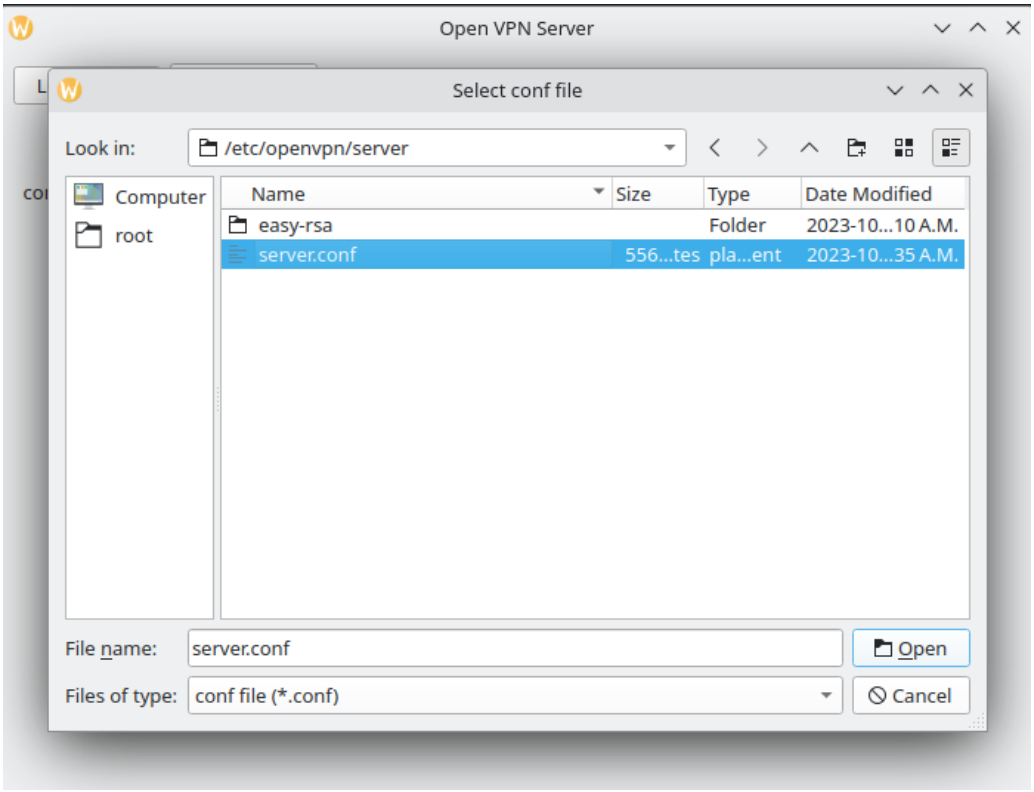
GUI successfully responds to the server and starts the application.

Test Case #3: Client selects other files that are not with the extension .ovpn.



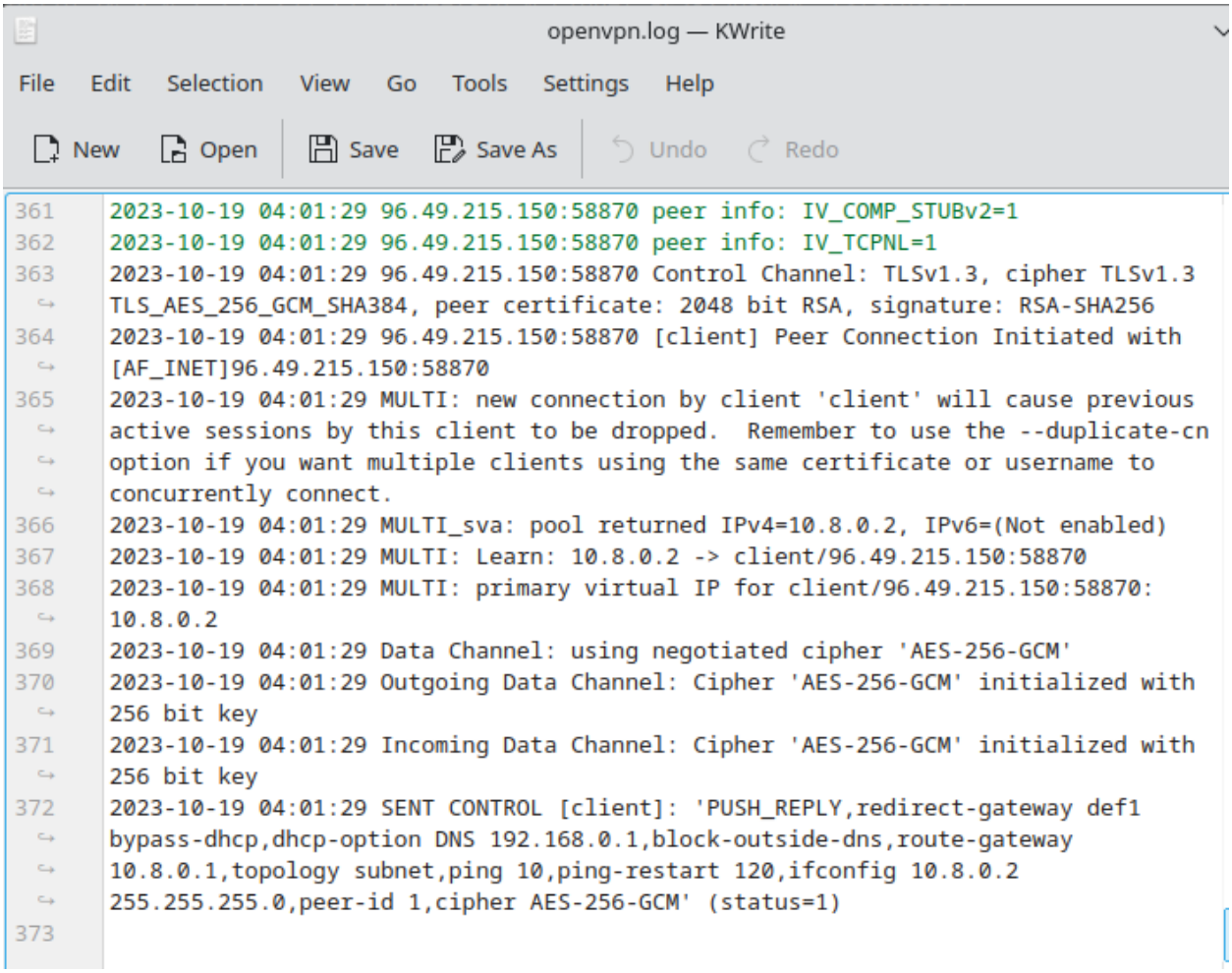
GUI does not allow the client to select files that do not end in .ovpn extension.

Test Case #4: Server selects other files that are not with the extension .conf.



GUI does not allow the server to select files that do not end in .conf extension.

Test Case #5: Traffic is collected in a log file.



All traffic is written in a log file when the VPN is configured to capture logs in a separate file.

2.7. Implications of Implementation

Implications of the implementation on my Quantum-Resistant VPN project are multifaceted. Initially, my limited knowledge of the quantum realm and post-quantum cryptography posed a significant challenge. I dedicated a substantial amount of time to researching and exploring the libraries offered by the Open Quantum Safe Project, aiming to ensure that the VPN remains secure against emerging quantum threats. However, this venture necessitated a pivot in my project's scope. I discovered that the chosen VPN protocol did not inherently support quantum-safe algorithms, compelling me to adapt my vision to a smaller scale. This unexpected shift led to a notable deviation from the anticipated timeline, particularly during Milestone 2, which entailed the intricate exploration of integration methods for the VPN application outlined in Milestone 1.

Despite these challenges, I was pleased to find that the development of the Graphical User Interface (GUI) progressed more swiftly than initially anticipated. Notably, the application's design accommodates additional requirements, including cross-platform support (Windows, Linux, Mac, and mobile devices) for VPN clients. Furthermore, I seized the opportunity to independently implement another post-quantum algorithm, Dilithium, which stands as a testament to the project's commitment to cutting-edge security. This additional implementation enhances the project's versatility and positions it favorably for future developments in post-quantum cryptography and quantum-resistant security solutions.

2.8. Innovation

The Open Quantum Safe (OQS) project that aims to create quantum-resistant cryptography was started in late 2016, which is a relatively new concept in cryptography, so its implementation of it should be considered innovative as it is considered a future leading-edge algorithm. My application not only implements cryptography but also integrates the encryption algorithms with the SSL protocol that is used in OpenVPN. The implementation creates an entirely new VPN application that is well-prepared for future attacks from both classical and quantum computers.

One of the similar products to my application is Mullvad VPN. It is a VPN application that also implements post-quantum cryptography. It uses WireGuard as its communication protocol and classic McEliece as its post-quantum encryption algorithms (Mullvad, 2022). Since WireGuard is written in around 4000 to 5000 lines, it does not provide much freedom when it comes to encryption and security compared to OpenVPN, which is written in at least 70,000 lines. If we are to implement a new algorithm in our application, WireGuard would have more restrictions when it comes to development.

2.9. Complexity

The complexity of this project depends on the newly introduced cryptography and the integration with an existing VPN protocol. Because post-quantum cryptography is relatively new, it will be constantly changing when the developer implements it. Since the liboqs library, which is an open-source library for quantum-safe cryptographic algorithms, only provides algorithms that are accepted by NIST, students will need to implement the entire encryption process manually, including setting up keys and ciphertext and validating keys, by using the CRYSTALS-KYBER algorithm. Diploma students will not be able to solve this problem because they have no knowledge of cryptography and very limited knowledge of VPN protocols. This project requires enough knowledge of networking and application protocols to be able to complete it. Also, for the OpenVPN protocol, students will need to know how SSL (Secure Socket Layer) protocol works, as well as the knowledge of encryption.

2.10. Research in New Technologies

The following is a list of new technologies that I researched or implemented:

- **Modern Cryptology Knowledge**: Before delving into cryptography, I completed a cryptology course at BCIT to establish a fundamental understanding of the subject. This course introduced me to contemporary encryption algorithms such as AES, Triple DES, RSA, and more. It provided the cornerstone for my cryptography knowledge, proving invaluable as I frequently encountered these modern encryption algorithms during my exploration of new technologies.
- **Post Quantum Cryptography**: This is an entirely novel field of study. I invested a substantial amount of time comprehending its nuances, including how it differs from modern cryptography. I also delved into the extensive libraries offered by the Open Quantum Safe project. While this was a valuable opportunity to explore this emerging field of cryptology, time constraints necessitated that I move forward with the project's completion.

- **OpenSSL**: OpenSSL plays a pivotal role in this project since the OpenVPN protocol relies on it for encryption and hashing algorithms. Throughout the project, I successfully generated quantum-resistant certificates using the Dilithium3 algorithm via OpenSSL with the OQS provider. However, I encountered an issue when the OpenVPN protocol failed to recognize this relatively new certificate type.
- **GUI Library**: Exploring the PyQt library proved to be a rewarding experience, primarily due to its clear documentation. This library offers comprehensive support for a wide range of elements and includes prebuilt basic layouts. For instance, implementing a file selection window is straightforward compared to other GUI libraries. Moreover, it facilitates the application of filters for file extensions with minimal code. However, PyQt's products tend to share a similar appearance, which presented challenges in aligning them with my design vision, ultimately necessitating some adjustments. Overall, PyQt is an excellent choice for initiating a basic application with a user-friendly interface.
- **VPN Protocol**: During the project, I delved into the intricacies of the OpenVPN protocol. Significant time was dedicated to configuring my application in tandem with this protocol, which boasts a wide array of configuration options. However, a thorough understanding of firewall and iptables rules is imperative when configuring specific options, particularly when routing all client traffic through the server. Customizing the protocol proved to be a challenge without delving into its source code, and I ultimately ran out of time for integrating post-quantum algorithms into the protocol.

2.11. Future Enhancements

The following list outlines several enhancements that can be implemented in the future for this project:

- **Integration of Post-Quantum Cryptography into OpenVPN Protocol**: This will involve making changes to the source code in C to seamlessly integrate quantum-resistant algorithms, fortifying the project's security.
- **Cross-Platform Functionality for the VPN Server**: Expanding the VPN server's compatibility to run efficiently across various operating systems, ensuring accessibility for a wider range of users.
- **Support for Multiple Protocols**: Enhancing the application's versatility by enabling it to handle diverse protocols beyond OpenVPN, accommodating a broader spectrum of user needs.
- **Selective Traffic Forwarding**: Implementing the capability to route network traffic solely from specific applications, offering fine-grained control and network efficiency.
- **Detection of Intrusion Attempts**: Integrating advanced security measures to identify and thwart potential attacks, including the detection of port scanning or spoofing attempts.
- **User Preferences and Configuration Saving**: Facilitating a more user-friendly experience by enabling the application to save user preferences and configurations, streamlining the setup and operation process.
- **Performance Optimization**: Focusing on improving performance to reduce packet loss or latency when connecting to multiple clients, ensuring a seamless and responsive user experience.

2.12. Timeline and Milestones

Task	Estimated Hours	Total per Milestone	Actual Hours
Project Initialization			
Setup GitHub Repository	1 hours	5 hours	1 hour
Setup New Project (Pycharm)	1 hours		1 hour
Import Libraries (liboqs, OpenVPN)	3 hours		10 hours
Milestone 1: **VPN with OpenVPN Protocol**			
Researching basic VPN implementation with OpenVPN Protocol	15 hours	100 hours	20 hours
Planning and Gathering Requirements	10 hours		15 hours
Designing architecture of the VPN application	15 hours		15 hours
Implementation	30 hours		60 hours
Writing and Running Tests	20 hours		20 hours
Documentation	10 hours		10 hours
Milestone 2: **Post Quantum Algorithm Implementation**			
Researching and learning about encryption algorithms	30 hours	175 hours	30 hours
Exploring liboqs library	15 hours		30 hours
Exploring CRYSTALS-KYBER Algorithm	15 hours		30 hours
Designing architecture	15 hours		15 hours
Implementation	60 hours		120 hours
Writing and Running Tests	30 hours		30 hours
Documentation	10 hours		10 hours

Milestone 3: GUI Integration			
Exploring PyQt library	10 hours	85 hours	10 hours
Designing/Drawing wireframes for both applications (Client and Server)	5 hours		5 hours
Building basic elements for home/main page of the application	10 hours		10 hours
Integrating functions into GUI	30 hours		10 hours
Writing and Running Tests	20 hours		10 hours
Documentation	10 hours		10 hours
Milestone 4: Final Documentation			
Finishing and revising the Final Report	20 hours	30 hours	30 hours
Gathering files for deliverables	10 hours		10 hours
Total:		395 hours	512 hours

Milestone	Completion Date
Milestone 1: *VPN with OpenVPN Protocol*	March 20, 2023
Milestone 2: Post Quantum Algorithm Implementation	August 24, 2023
Milestone 3: GUI Integration	September 30, 2023
Milestone 4: Final Documentation	October 18, 2023

As highlighted in the development process, Milestone 2 required a considerably greater amount of time than initially anticipated. The complexity of integrating post-quantum cryptography into the project, particularly within the confines of Python, was a significant factor. The liboqs-python library, an essential component of the Open Quantum Safe Project, presented limitations in terms of functionality. This, in turn, posed challenges in implementing the desired quantum-resistant algorithms seamlessly. The

cumulative effect of these challenges resulted in an extension of the project's timeline, surpassing the initial estimated deadlines outlined in the project proposal. Despite these hurdles, the project persevered, adapting to unforeseen circumstances and gaining valuable insights into the intricacies of quantum-resistant security solutions.

3. Conclusion

Unlike the class assignments and projects, this project aims to extend on the development of VPN application, which is a perfect project for a BTech student in the Network Security Applications Development Option. It involves implementing VPN with a well-known and well-used protocol, OpenVPN. Also, it requires the understanding of cryptography to implement the algorithms of encryption for post-quantum purpose. It is a great opportunity to gain knowledge of cryptography as it is the last piece for network security.

3.7. Lessons Learned

This project has been a journey of profound learning and growth, providing valuable insights into the realm of network security and encryption. Among the key takeaways are:

- Comprehensive Knowledge Gain: One of the most significant achievements of this project has been the acquisition of in-depth knowledge in the field of security. Before embarking on this endeavor, I had limited understanding of both modern encryption techniques and the intricacies of quantum-resistant encryption. The project offered a steep learning curve that allowed me to grasp the foundations and complexities of encryption, thus expanding my expertise.
- Quantum Resistance Awareness: The exploration of quantum-resistant encryption was particularly enlightening. This emerging field not only demands a deep dive into cryptographic concepts but also underscores the criticality of fortifying our digital infrastructure against the potential advent of

quantum computing. It has served as a reminder of the constant need to stay ahead of the curve in network security.

- Network Security Significance: In an age where the internet has become the focal point of modern life, the project underscored the paramount importance of network security. The internet's pivotal role in our daily interactions highlights the necessity of safeguarding sensitive data and communications, making security expertise increasingly essential.
- VPN Protocol Understanding: The journey into VPN protocols has provided invaluable insights into the creation and management of secure communication channels. Understanding the intricacies of VPNs empowers us to build our own security mechanisms and tailor them to meet the unique needs and expectations of the digital world.

In conclusion, this project has not only expanded my horizons in the security domain but has also reinforced the urgency of staying at the forefront of evolving security paradigms. The knowledge and skills gained are not only applicable to this project but have broader implications for ensuring the privacy and security of digital communications in an interconnected world.

3.8. Closing Remarks

This project stands as an exceptional opportunity within the CST BSc Network Security Applications Development Option. It not only reinforces the core principles taught throughout the program but also extends the learning experience, equipping students with the knowledge and skills necessary for a seamless transition into the industry. The understanding gained and the lessons learned hold immeasurable value for students specializing in this field.

4. Appendix

4.7. Approved Proposal

https://drive.google.com/file/d/136lMKuBrjESpZ38zfFUo3iF0rt35ZzeU/view?usp=share_link

4.8. Project Supervisor Approvals

5. References

Barker, E. B., & Roginsky, A. L. (2015). Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths. https://doi.org/10.6028/nist.sp.800-131ar1

Chen, L., Jordan, S., Liu, Y.-K., Moody, D., Peralta, R., Perlner, R., & Smith-Tone, D. (2016). Report on post-quantum cryptography. https://doi.org/10.6028/nist.ir.8105

Computer Security Division, I. T. L. (n.d.). *Post-quantum cryptography: CSRC*. CSRC. Retrieved October 26, 2022, from https://csrc.nist.gov/projects/post-quantum-cryptography

Crystals. Kyber. Retrieved October 26, 2022, from https://pq-crystals.org/kyber/

IBM. What is quantum computing? Retrieved November 7, 2022, from https://www.ibm.com/topics/quantum-computing

Mullvad. Experimental post-quantum safe VPN tunnels - blog. Mullvad VPN. Retrieved November 7, 2022, from https://mullvad.net/en/blog/2022/7/11/experimental-post-quantum-safe-vpn-tunnels/

Open-Quantum-Safe. *Open-quantum-safe/liboqs-python: Python 3 bindings for liboqs*. GitHub. Retrieved October 26, 2022, from https://github.com/open-quantum-safe/liboqs-python

Post-quantum cryptography VPN. Microsoft Research. (2020, September 14). Retrieved October 26, 2022, from https://www.microsoft.com/en-us/research/project/post-quantum-crypto-vpn/

Post-quantum cryptography. Open Quantum Safe. Retrieved September 28, 2022, from https://openquantumsafe.org/post-quantum-crypto.html

6. Change Log

2023.10.20 – Version 1

- Initial Submission