Overview

This tutorial will take you through the process of downloading all the source code, setting up a virtual environment to run the code, and setting up Mininet to test the application on a number of different topologies.

This tutorial is slightly longer than building from Docker as you will need to set up all the environment variables, download dependencies, and clone the source code.

Prerequisites

This tutorial assumes that you have a Virtual Machine running Ubuntu 20.04 or above. Any VM hosting software should work (UTM, Parallels for Mac, VirtualBox, etc.), so to start off this tutorial, make sure you build your own Ubuntu 20.04+ machine.

Once you have your VM configured, we will need to install Go version 1.22 or above on this machine. Simply follow the download and build instructions on the Go development website to install the correct version of Go. You can always check what version you have by running:

```
go version
```

Finally, we use pkg-config to configure certain dependencies, to install this, simply run:

```
sudo apt-get install pkg-config
```

Installing OpenSSL (v3.3), LibOQS, OQS-Provider

Initiating a Workspace

Our qs509 library depends on OpenSSL (v3.3), LibOQS, and OQS-Provider. We will be installing our specific version of each of these dependencies inside of a new folder named "quantumsafe" which will be used as our workspace for the rest of the tutorial. To create this folder and install some initial dependencies, run the following:

```
# If you are not running as root you might need to use "sudo apt" instead
sudo apt update
sudo apt -y install git build-essential perl cmake autoconf libtool zlib1g-dev

export WORKSPACE=~/quantumsafe
export BUILD_DIR=$WORKSPACE/build # this will contain all the build artifacts
mkdir -p $BUILD_DIR/lib64
ln -s $BUILD_DIR/lib64 $BUILD_DIR/lib
```

Now, you should have a folder named "quantumsafe" at the root of your machine. Inside should be a folder named "build".

Installing OpenSSL

We need to install specifically OpenSSL version 3.3 in order to have support for post-quantum algorithms. The following can be run to set up and configure OpenSSL v3.3:

```
cd $WORKSPACE

git clone https://github.com/openssl/openssl.git
cd openssl

./Configure \
  --prefix=$BUILD_DIR \
  no-ssl no-tls1 no-tls1_1 no-afalgeng \
  no-shared threads -lm

make -j $(nproc)
make -j $(nproc) install_sw install_ssldirs
```

Now, inside your quantumsafe folder, you should have a new directory, openssl.

Installing LibOQS

LibOQS is used to provide us with some more functionality in post-quantum cryptography. To install and configure this package, run the following:

```
cd $WORKSPACE

git clone https://github.com/open-quantum-safe/liboqs.git
cd liboqs

mkdir build && cd build

cmake \
  -DCMAKE_INSTALL_PREFIX=$BUILD_DIR \
  -DBUILD_SHARED_LIBS=ON \
  -DOQS_USE_OPENSSL=OFF \
  -DCMAKE_BUILD_TYPE=Release \
  -DOQS_BUILD_ONLY_LIB=ON \
  -DOQS_DIST_BUILD=ON \
  ..

make -j $(nproc)
make -j $(nproc) install
```

After running, you should have a liboqs directory inside your quantumsafe folder.

Installing OQS-Provider

OQS-Provider is used to provide many different post-quantum algorithms for us to use, and can be installed by running the following:

```
cd $WORKSPACE

git clone https://github.com/open-quantum-safe/oqs-provider.git
cd oqs-provider

liboqs_DIR=$BUILD_DIR cmake \
  -DCMAKE_INSTALL_PREFIX=$WORKSPACE/oqs-provider \
  -DOPENSSL_ROOT_DIR=$BUILD_DIR \
  -DCMAKE_BUILD_TYPE=Release \
  -S . \
  -B _build
cmake --build _build

# Manually copy the lib files into the build dir
cp _build/lib/* $BUILD_DIR/lib/

# We need to edit the openssl config to use the oqsprovider
sed -i "s/default = default_sect/default = default_sect\noqsprovider =
oqsprovider_sect/g" $BUILD_DIR/ssl/openssl.cnf &&
sed -i "s/\[default_sect\]/\[default_sect\]\nactivate =
1\n\[oqsprovider_sect\]\nactivate = 1\n/g" $BUILD_DIR/ssl/openssl.cnf
```

After running, you should have an oqs-provider directory in your quantumsafe folder. We have a couple more environment variables to export, so run the following:

```
export OPENSSL_CONF=$BUILD_DIR/ssl/openssl.cnf
export OPENSSL_MODULES=$BUILD_DIR/lib
```

Now, we can test and make sure everything was installed properly, run:

```
$BUILD_DIR/bin/openssl list -providers -verbose -provider oqsprovider
```

If everything was done correctly, you should see oqsprovider listed as one of the providers for your OpenSSL, an example output is attached below:

```
parallels@ubuntu-linux-22-04-02-desktop:~/quantumsafe$ $BUILD_DIR/bin/openssl list -providers -verbose -provider oqsprovider
Providers:
  default
    name: OpenSSL Default Provider
    version: 3.3.0
    status: active
    build info: 3.3.0-dev
    gettable provider parameters:
      name: pointer to a UTF8 encoded string (arbitrary size)
      version: pointer to a UTF8 encoded string (arbitrary size)
      buildinfo: pointer to a UTF8 encoded string (arbitrary size)
      status: integer (arbitrary size)
  oqsprovider
    name: OpenSSL OQS Provider
    version: 0.5.4-dev
    status: active
    build info: OQS Provider v.0.5.4-dev (66ee770) based on liboqs v.0.10.0
    gettable provider parameters:
      name: pointer to a UTF8 encoded string (arbitrary size)
      version: pointer to a UTF8 encoded string (arbitrary size)
      buildinfo: pointer to a UTF8 encoded string (arbitrary size)
      status: integer (arbitrary size)
```

Installing LibOQS-Go

LibOQS-Go is a wrapper for the C-based LibOQS. It offers limited functionality for LibOQS, and is used by our client/server application only for Key Exchange.

To install LibOQS-Go:

```
cd $WORKSPACE

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

We need to tell LibOQS-Go where our build files are for liboqs, so we need to adjust only the first two lines of the file:

```
$WORKSPACE/liboqs-go/.config/liboqs-go.pc
```

Adjust only the first two lines so they match you liboqs build directory:

```
LIBOQS_INCLUDE_DIR=~/quantumsafe/liboqs/build/include
LIBOQS_LIB_DIR=~/quantumsafe/liboqs/build/lib
```

Now, we want to rename this file so pkg-config knows these directories are for liboqs rather than liboqs-go.

```
cd $WORKSPACE/liboqs-go/.config

mv liboqs-go.pc liboqs.pc
```

Finally, we need to export a couple of environment variables so our programs know where to find everything.

```
export LD_LIBRARY_PATH=~/quantumsafe/liboqs/build/lib
export PKG_CONFIG_PATH=~/quantumsafe/liboqs-go/.config
```

At this point, everything should be installed and ready for our client and server applications.

Installing qs509, Client, and Server

We are going to create a new folder in our quantumsafe directory named “ProjectCode” where we will store the source code for our project.

```
cd $WORKSPACE
mkdir ProjectCode
cd ProjectCode
```

Once here, we can clone each repository:

```
git clone https://github.com/CSCE482QuantumCryptography/qs509.git
git clone https://github.com/CSCE482QuantumCryptography/server.git
git clone https://github.com/CSCE482QuantumCryptography/client.git
```

And voila! Everything is installed to run our client and server application locally!

Before running anything, let's copy all of our environment variables into the ~/.bashrc file so we can make sure that we have them for any new terminals.

Open the file ~/.bashrc and add the following lines to the very bottom of the file:

```
export OPENSSL_CONF=$BUILD_DIR/ssl/openssl.cnf
export OPENSSL_MODULES=$BUILD_DIR/lib
export WORKSPACE=~/quantumsafe
export BUILD_DIR=$WORKSPACE/build
export LD_LIBRARY_PATH=~/quantumsafe/liboqs/build/lib
export PKG_CONFIG_PATH=~/quantumsafe/liboqs-go/.config
```

Note, each of these should match what you have done before when setting up each of the different dependencies.

Running Client/Server App

In your terminal, move to the server code and run it with:

```
cd $WORKSPACE/ProjectCode/server
go run *.go
```

In another terminal:

```
cd $WORKSPACE/ProjectCode/client
go run *.go
```

Now, you should have a running tunnel!

If you encounter an error when running the server or client that looks like:

```
parallels@ubuntu-linux-22-04-02-desktop:~/quantumsafe/ProjectCode/server$ go run *go
exit status 1
panic: exit status 1

goroutine 1 [running]:
main.main()
        /home/parallels/quantumsafe/ProjectCode/server/server.go:23 +0x30c
exit status 2
```

That means you likely need to source your .bashrc file:

```
source ~/.bashrc
```

For a more in-depth explanation on how to run the client/server application, refer to the Running Our Program documentation!

To continue on to configuring and running our mininet simulations, go to the next page.

Installing Mininet

To install mininet, navigate back to the root of your machine. Here, you will clone the mininet source code and setup scripts:

```
git clone https://github.com/mininet/mininet
```

Now all that is left is to run the setup script that will install mininet, python3, wireshark, and all necessary dependencies:

```
mininet/util/install.sh -a
```

To test that everything was installed correctly, run:

```
sudo mn --switch ovsbr --test pingall
```

An example output is shown below:

```
parallels@ubuntu-linux-22-04-02-desktop:~$ sudo mn --switch ovsbr --test pingall
[sudo] password for parallels:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 0.143 seconds
```

Running Our Custom Topologies

We have provided a number of custom topologies to test our client/server applications. To get these topologies, clone our topology repository in the ProjectCode directory configured during the previous tutorial.

```
cd $WORKSPACE/ProjectCode
git clone https://github.com/CSCE482QuantumCryptography/mininet
```

Once cloned, you need to generate executable files for both the client and the server that Mininet can run.

```
cd $WORKSPACE/ProjectCode/server
go build

cd $WORKSPACE/ProjectCode/client
go build
```

Now, any of our custom topologies can be run by using the following command:

```
sudo python3 <mininet_topo.py>
```

An example is shown here:

```
parallels@ubuntu-linux-22-04-02-desktop:~/quantumsafe/ProjectCode$ sudo python3 ../mininet/basic_topo.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s3
*** Adding links:
(h1, s3) (s3, h2)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s3 ...
*** Starting CLI:
mininet>
```

Once your topology is set up, you can see the different hosts and links between each. You can navigate the hosts to the files for the client and server applications and run them individually.

An example is shown below:

```
parallels@ubuntu-linux-22-04-02-desktop:~/quantumsafe/ProjectCode$ sudo python3 ../mininet/basic_topo.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s3
*** Adding links:
(h1, s3) (s3, h2)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s3 ...
*** Starting CLI:
mininet> h1 cd server
mininet> h2 cd client
mininet> h1 ./server -src 10.0.0.1:9080 &
mininet> h2 ./client -dst 10.0.0.1:9080
-----

Certificate request self-signature ok
subject=CN=test server

Client Certificate Size:  7481
Reading Server Certificate!
Server cert size:  7481
Verified Server Certificate!
Writing my certificate to server!


KEM details:
Name: Kyber512
Version: https://github.com/pq-crystals/kyber/commit/74cad307858b61e434490c75f812cb9b9ef7279b
Claimed NIST level: 1
Is IND_CCA: true
Length public key (bytes): 800
Length secret key (bytes): 1632
Length ciphertext (bytes): 768
Length shared secret (bytes): 32

Sending public kyber key to server!
Received shared secret from server!
IV Sent: [166 166 124 82 15 134 185 178 43 240 12 126 126 89 70 5]
Text to send (q to exit): Hi server!
Encrypted Data Written: [108 117 221 168 95 28 93 73 118 158] 10
Text to send (q to exit): q
Closing connection with the server!
```

A couple things to note:

- The hosts in a mininet configuration are given an IP address to use, so you must tell the client where to dial and the server where to listen, see the image above for an example
- Any mininet problems can be solved by checking out the documentation provided by mininet
- All custom flags can be used in Mininet, please check out the Running Our Program documentation for more information