OpenSSL and Docker hands-on tutorial

Docker is a virtualization tool conceived to facilitate software deployment. The software applications, including the operating system, are packaged into **images** that can be instantiated as **containers**. Docker also allows the creation of virtual networks where the different containers can interact, emulating the behavior of a real network.

We can find repositories with available preconfigured images for different purposes, including cryptography. In this tutorial we will use docker images provided by the Open Quantum Safe project, that can be found in https://hub.docker.com/u/openquantumsafe.

We will use these docker images to study different scenarios:

- Study a web client/server connection (1 and 2).
- Perform TLS handshakes (3 and C). TLS (Transport Layer Security) is the most common protocol to establish authenticated and encrypted sessions on the Internet.
- Create and verify digital certificates and use them to start a server and connect a client to it using TLS1.3 handshakes (4). For this case, you will be able to choose the PQC digital signature algorithm / Key Encapsulation Mechanism (KEM) from all the available ones in OpenSSL with liboqs provider.
- As homework you can create a VPN using PQC signatures.

0. Useful commands

All the Docker images and more can be found in https://hub.docker.com/u/openquantumsafe .

1. For windows users: Before using any of these open Docker Desktop in Windows to launch the application.
2. Useful commands:
    a. **To remove a network:** docker network rm *name network* -> httpd-test
    b. **Remove a container:** docker rm *container id* -> iw23ndsiudy832

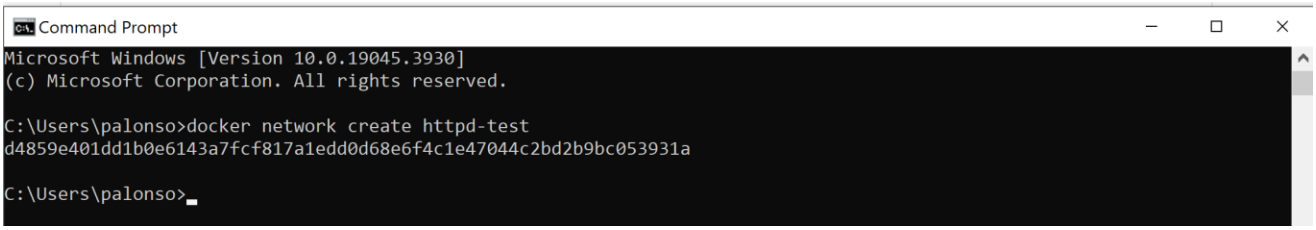
The previous commands need to be used when repeating a test twice. You will see a disclaimer indicating that the network with that name is already on use and you should delete it to restart the commands, and similarly with the container.

1. Client/Server Web: Test with httpd

Link: https://hub.docker.com/r/openquantumsafe/httpd .

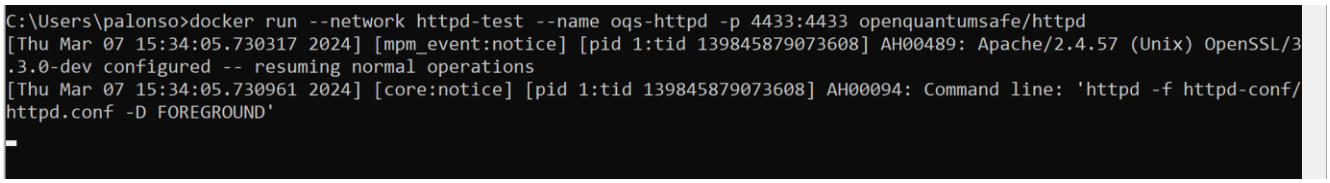
1. First, we create the network. From the command prompt:

```
docker network create httpd-test
```



2. Then, we start the QSC-enabled httpd and listening for quantum-safe crypto protected TLS 1.3 connections on port 4433.

```
docker run --network httpd-test --name oqs-httpd -p 4433:4433 openquantumsafe/httpd
```



3. Finally, we connect to the server by opening a new terminal without closing the first one.

```
docker run --network httpd-test -it openquantumsafe/curl curl -k https://oqs-httpd:4433
```

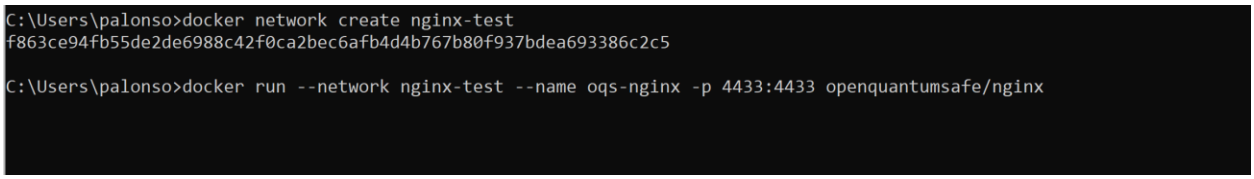
2. Client/Server Web: Launch with nginx

Link: https://hub.docker.com/r/openquantumsafe/nginx .

1. Similarly to the previous example, we first create the network and then we launch it.

```
docker network create nginx-test
```

```
docker run --network nginx-test --name oqs-nginx -p 4433:4433 openquantumsafe/nginx
```



2. From another terminal, we connect to the web server.

`docker run --network nginx-test -it openquantumsafe/curl curl -k https://oqs-nginx:4433`

```
C:\Users\palonso>docker run --network nginx-test -it openquantumsafe/curl curl -k https://oqs-nginx:4433
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Extra:

Both for the httpd or the nginx case we can specify which KEM we use to stablish the connection. When we launch the server and when we connect to that server we can indicate the algorithm to be used. For example, using httpd (similarly with nginx). For the current version the only available KEM is kyber768. If we try the hybrid version first, we see it raises an issue.

`docker run --network httpd-test -it openquantumsafe/curl curl -k https://oqs-httpd:4433 --curves kyber768`

```
C:\Users\palonso>docker run --network httpd-test -it openquantumsafe/curl curl -k https://oqs-httpd:4433 --curves p256_kyber768
curl: (35) error:0A000410:SSL routines::ssl/tls alert handshake failure

C:\Users\palonso>docker run --network httpd-test -it openquantumsafe/curl curl -k https://oqs-httpd:4433 --curves kyber768
<html><body><h1>It works!</h1></body></html>

C:\Users\palonso>
```

3. TLS handhsake: Install openssl with the oqs provider

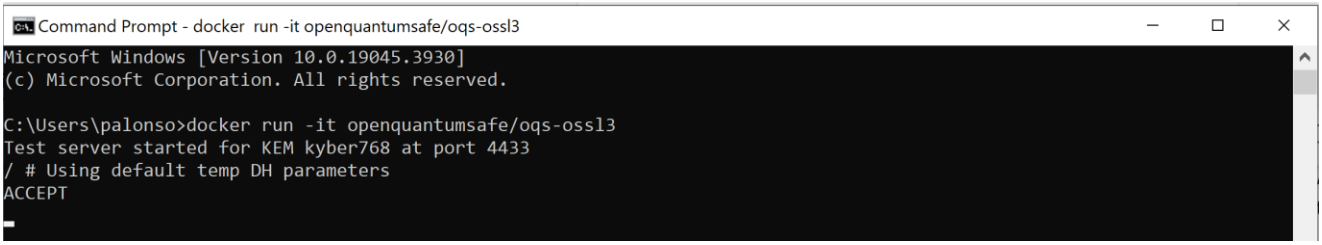
Link: https://hub.docker.com/r/openquantumsafe/oqs-ossl3 .

This provides a ready-to-run build of the current master branch of OpenSSL (3) together with a provider implementing plain and hybrid OQS key exchange mechanisms according to draft-ietf-tls-hybrid-design-00 as well as plain and hybrid OQS signature algorithms for X.509 cert generation, CMS and DGST operations using the OpenSSL command line tools.

What can be done with Open SSL3 OQS provider:

1. Start an OQS-enabled TLS test server:

```
docker run -it openquantumsafe/oqs-ossl3
```



2. Query the built-in test server:

```
openssl s_client -connect localhost –groups kyber768
```

   - Kyber768 is an example of suitable KEM appearing in the list *openssl list -kem-algorithm*. At the end of the document you can find a list of the available KEMs.
   - OpenSSL's `s_client` command can be used to analyze client-server communication, including whether a port is open and if that port can accept an SSL/TLS connection. It is a useful tool for investigating SSL/TLS certificate-based plugins, and for confirming that a line of secure communications is available.
   - `Localhost` is the hostname or the computer that is currently in use to run a program, in which the computer has the role as a virtual server.

4. Advanced – creating your own PKI

We can customize the server as well by modifying the KEM used for the TLS establishment.
We will continue using oqs-ossl3 container. You could even use volumes to create a virtual non-local network to share the public keys and certificates. In this case, we will keep it local for simplicity.

1. Start an OQS-enabled TLS test server (similarly as above):

```
docker run -it openquantumsafe/oqs-ossl3
```

2. To initiate a server we need to generate first a X.509 certificate, which we can customize to be self-signed or part of a Certificate Authorities (CAs) chain. In this example we will only work with a root CA. We will first create the elf-signed root CA certificate. In openssl we create the request to creat the self-signed certificate, create the keys with the desired signing algorithm from the available ones (we used dilithium5 as an example), and the result (-out) is the certificate, that we chose it to last 365 days.

```
openssl req -x509 -new -newkey dilithium5 -keyout dilithium5_CA.key -out dilithium5_CA.crt -nodes -subj "/CN=oqstest CA"
```

```
/ # openssl req -x509 -new -newkey dilithium5 -keyout dilithium5_CA.key -out dilithium5_CA.crt
-nodes -subj "/CN=oqstest CA"
-----
```

3. After creating the root CA we initiate a server with a Certificate Signing Request (CSR), which we need to send to the CA so that it validates it and generate the server's certificate with the algorithm falcon1024.

`openssl req -new -newkey falcon1024 -keyout falcon1024_srv.key -out falcon1024_srv.csr -nodes -subj "/CN=oqstest server"`

```
/ # openssl req -new -newkey falcon1024 -keyout falcon1024_srv.key -out falcon1024_srv.csr
-nodes -subj "/CN=oqstest server"
-----
```

4. Upon reception, the CA validates the CSR and generates a certificate. The CA creates the server's certificate (falcon1024_srv.crt) using its own certificate (dilithium5_CA.crt) and private key (dilithium5_CA.key). The CA creates a serial number if it does not have one already.

`openssl x509 -req -in falcon1024_srv.csr -out falcon1024_srv.crt -CA dilithium5_CA.crt -CAkey dilithium5_CA.key -CAcreateserial -days 365`

```
/ # openssl x509 -req -in falcon1024_srv.csr -out falcon1024_srv.crt -CA dilithium5_CA.crt
-CAkey dilithium5_CA.key -CAcreateserial -days 365
Certificate request self-signature ok
subject=CN=oqstest server
```

5. The server is now ready to initialize with the certificate that the CA has generated. The server can also choose a KEM from the list of PQC KEM algorithms. It is better to indicate the port because by default it will connect to 4433 and if it is already in use you will not be able to test more combinations later.

`openssl s_server -port PORT -cert falcon1024_srv.crt -key falcon1024_srv.key -groups kyber768 -www -tls1_3 &`

```
/ # openssl s_server -cert falcon1024_srv.crt -key falcon1024_srv.key -groups kyber768 -www
-tls1_3 &
/ # Using default temp DH parameters
486B914E5E7F0000:error:80000062:system library:BIO_bind:Address in use:crypto/bio/bio_sock2.
c:240:calling bind()
486B914E5E7F0000:error:10000075:BIO routines:BIO_bind:unable to bind socket:crypto/bio/bio_s
ock2.c:242:
   0 items in the session cache
   0 client connects (SSL_connect())
   0 client renegotiates (SSL_connect())
   0 client connects that finished
   0 server accepts (SSL_accept())
   0 server renegotiates (SSL_accept())
   0 server accepts that finished
   0 session cache hits
   0 session cache misses
   0 session cache timeouts
   0 callback cache hits
   0 cache full overflows (128 allowed)
```

6. Finally, we introduce a similar command to the previous section to connect the client to our server. In this case it is a bit more complex because we have created our own keys and certificates and it is essential to indicate where one can find them. The client uses the CA's certificate to validate the server's certificate and guarantee a connection to the right one. If we were not using local Dockers, we would introduce the IP where the variable localhost is written.

`openssl s_client -connect localhost:PORT -groups kyber768 -CAfile dilithium5_CA.crt &`

```
/ # openssl s_client -connect localhost -groups kyber768 -CAfile dilithium5_CA.crt &
/ # Connecting to 127.0.0.1
CONNECTED(00000003)
Can't use SSL_get_servername
depth=0 CN=localhost
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 CN=localhost
verify error:num=21:unable to verify the first certificate
verify return:1
depth=0 CN=localhost
verify return:1
---
Certificate chain
 0 s:CN=localhost
   i:CN=oqstest CA
   a:PKEY: UNDEF, 192 (bit); sigalg: dilithium3
   v:NotBefore: Aug 15 15:35:56 2023 GMT; NotAfter: Aug 14 15:35:56 2024 GMT
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIVaDCCCHOgAwIBAgIUfVKd4zkJQojxVGmrQKGGiCYCIREwDQYLKwYBBAECggsH
BgUwFTETMBEGA1UEAwwKb3FzdGVzdCBDQTAeFw0yMzA4MTUxNTM1NTZaFw0yNDA4
```

You can choose any signature algorithm and KEM available and observe how these algorithms modify the network traffic in the output of the statistics after the connection of the client to the local network.

A. List of available signature algorithms in openssl

```
/ # openssl list -signature-algorithms
  { 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, 1.3.14.3.2.13
, 1.3.14.3.2.27, DSA, DSA-old, DSA-SHA, DSA-SHA1, DSA-SHA1-old, dsaEnc
ryption, dsaEncryption-old, dsaWithSHA, dsaWithSHA1, dsaWithSHA1-old }
 @ default
  { 1.3.101.112, ED25519 } @ default
  { 1.3.101.113, ED448 } @ default
  { 1.2.156.10197.1.301, SM2 } @ 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
```

B. List of available KEMs in openssl

Command Prompt - docker run -it openquantumsafe/oqs-ossl3

```
/ # openssl list -kem-algorithms
  { 1.2.840.113549.1.1.1, 2.5.8.1.1, RSA, rsaEncryption } @ default
  { 1.2.840.10045.2.1, EC, id-ecPublicKey } @ default
  { 1.3.101.110, X25519 } @ default
  { 1.3.101.111, X448 } @ 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
/ #
```

C. Testing different sizes

Install wireshark: https://www.wireshark.org/download.html .

Wireshark is a powerful tool to study the traffic in a network. There are many variables that can be studied. In this test we will study the time to perform the full connection, the bits sent per user and the number of packages exchanged. On top of that, we will compare the conversations in each case and we will observe that the algorithm used can change the values significantly.

1. Download the network capture files from insert.
2. Open in wireshark both documents*.

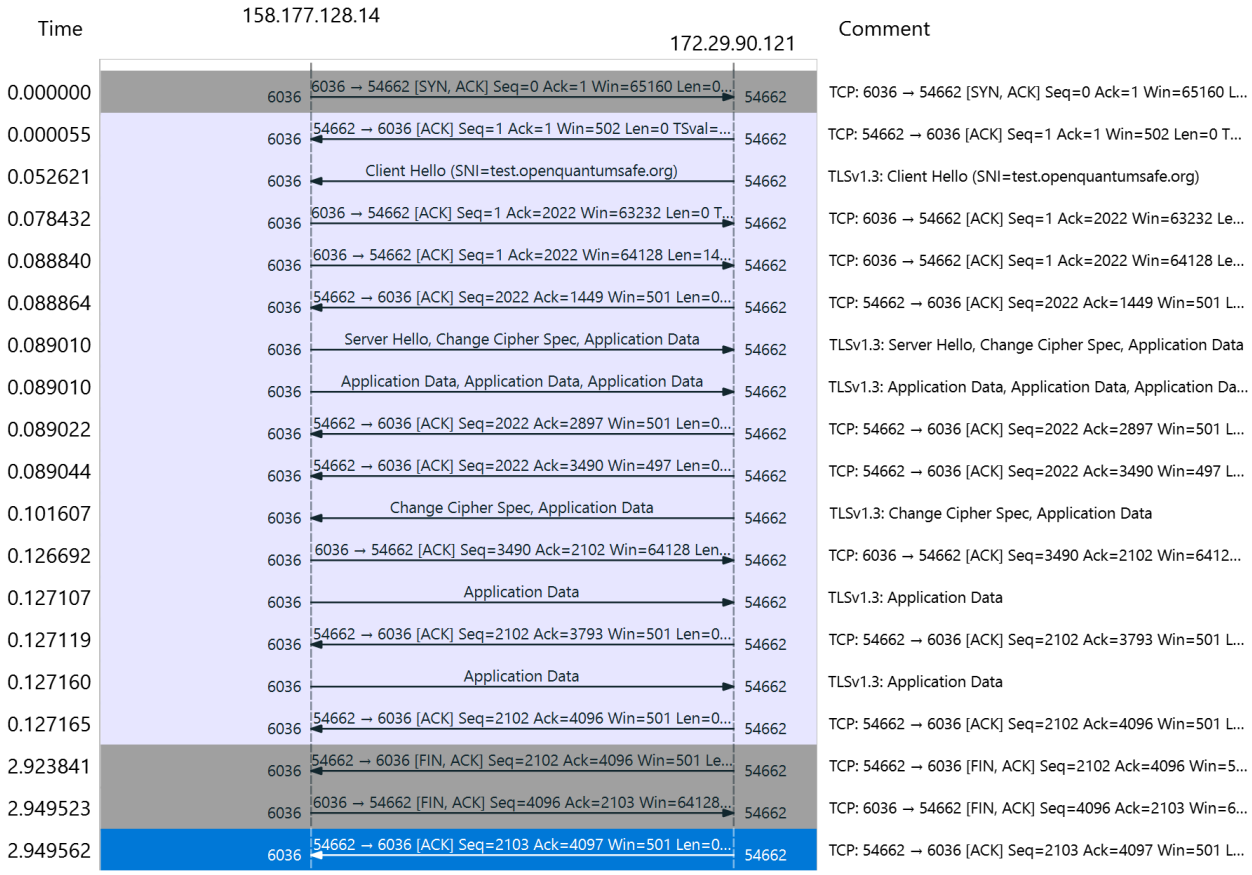
No.	Time	Source	Info	Destination	Protocol	Length
1	0.000000	158.177.128.14	6036 → 54662 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=3408017666 TSecr=2480858807 WS=128	172.29.90.121	TCP	74
2	0.000055	172.29.90.121	54662 → 6036 [ACK] Seq=1 Ack=1 Win=502 Len=0 TSval=2480858833 TSecr=3408017666	158.177.128.14	TCP	66
3	0.052621	172.29.90.121	Client Hello (SNI=test.openquantumsafe.org)	158.177.128.14	TLSv1.3	2087
4	0.078432	158.177.128.14	6036 → 54662 [ACK] Seq=1 Ack=2022 Win=63232 Len=0 TSval=3408017745 TSecr=2480858886	172.29.90.121	TCP	66
5	0.088840	158.177.128.14	6036 → 54662 [ACK] Seq=1 Ack=2022 Win=64128 Len=1448 TSval=3408017754 TSecr=2480858886 [TCP segment of a reassembled PDU]	172.29.90.121	TCP	1514
6	0.088864	172.29.90.121	54662 → 6036 [ACK] Seq=2022 Ack=1449 Win=501 Len=0 TSval=2480858922 TSecr=3408017754	158.177.128.14	TCP	66
7	0.089010	158.177.128.14	Server Hello, Change Cipher Spec, Application Data	172.29.90.121	TLSv1.3	1514
8	0.089010	158.177.128.14	Application Data, Application Data, Application Data	172.29.90.121	TLSv1.3	659
9	0.089022	172.29.90.121	54662 → 6036 [ACK] Seq=2022 Ack=2897 Win=501 Len=0 TSval=2480858922 TSecr=3408017754	158.177.128.14	TCP	66
10	0.089044	172.29.90.121	54662 → 6036 [ACK] Seq=2022 Ack=3490 Win=497 Len=0 TSval=2480858922 TSecr=3408017755	158.177.128.14	TCP	66
11	0.101607	172.29.90.121	Change Cipher Spec, Application Data	158.177.128.14	TLSv1.3	146
12	0.126692	158.177.128.14	6036 → 54662 [ACK] Seq=3490 Ack=2102 Win=64128 Len=0 TSval=3408017793 TSecr=2480858935	172.29.90.121	TCP	66
13	0.127107	158.177.128.14	Application Data	172.29.90.121	TLSv1.3	369
14	0.127119	172.29.90.121	54662 → 6036 [ACK] Seq=2102 Ack=3793 Win=501 Len=0 TSval=2480858960 TSecr=3408017793	158.177.128.14	TCP	66
15	0.127160	158.177.128.14	Application Data	172.29.90.121	TLSv1.3	369
16	0.127165	172.29.90.121	54662 → 6036 [ACK] Seq=2102 Ack=4096 Win=501 Len=0 TSval=2480858960 TSecr=3408017793	158.177.128.14	TCP	66
17	2.923841	172.29.90.121	54662 → 6036 [FIN, ACK] Seq=2102 Ack=4096 Win=501 Len=0 TSval=2480861757 TSecr=3408017793	158.177.128.14	TCP	66
18	2.949523	158.177.128.14	6036 → 54662 [FIN, ACK] Seq=4096 Ack=2103 Win=64128 Len=0 TSval=3408020616 TSecr=2480861757	172.29.90.121	TCP	66
19	2.949562	172.29.90.121	54662 → 6036 [ACK] Seq=2103 Ack=4097 Win=501 Len=0 TSval=2480861782 TSecr=3408020616	158.177.128.14	TCP	66

***Disclaimer**: If the protocol is detected as X11 instead of TLS1.3, go to Edit > Preferences... > Protocols > X11, and modify the ports to any other range that is not the one in which the connection is being established.

3. Study the bandwidth and number of packets exchanged: Statistics > Protocol Hierarchy > Transport Layer Security

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s	PDUs
▾ Frame	100.0	19	100.0	7458	20 k	0	0	0	19
▾ Ethernet	100.0	19	3.6	266	721	0	0	0	19
▾ Internet Protocol Version 4	100.0	19	5.1	380	1030	0	0	0	19
▾ Transmission Control Protocol	100.0	19	91.3	6812	18 k	13	1872	5077	19
Transport Layer Security	31.6	6	97.3	7258	19 k	6	3981	10 k	8

4. By accessing to Statistics > Flow Graph, we can observe the directions of the packets.



All the screenshot from Wireshark correspond to the outputs when opening the file `ecdsap256p521_kyber1024_ONLINE_OQS.pcap`. You will see great differences when loading the other example file.

D. VPN: setting up openVPN

Link: https://hub.docker.com/r/openquantumsafe/openvpn .

Follow the guidelines. Configuration depends on the operating system. For windows define the variables as `set variable=variable_value` and call them with `%variable%.`