PQC Document

Introduction

This documentation provides step-by-step instructions for generating SSL certificates, Self signed certificate , and Digital signature using Dilithium-based cryptographic algorithms with OpenSSL. Dilithium is a post-quantum secure digital signature algorithm, designed to protect against attacks from quantum computers.

System Configurations used

Operating System : Windows 10 Pro

Processor : Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz

RAM : 32 GB

Virtual Machine Configurations used

VM Application : VMware Workstation 17 Player

Operating System : Ubuntu 22.04

Processor : 2 Cores

RAM : 4 GB

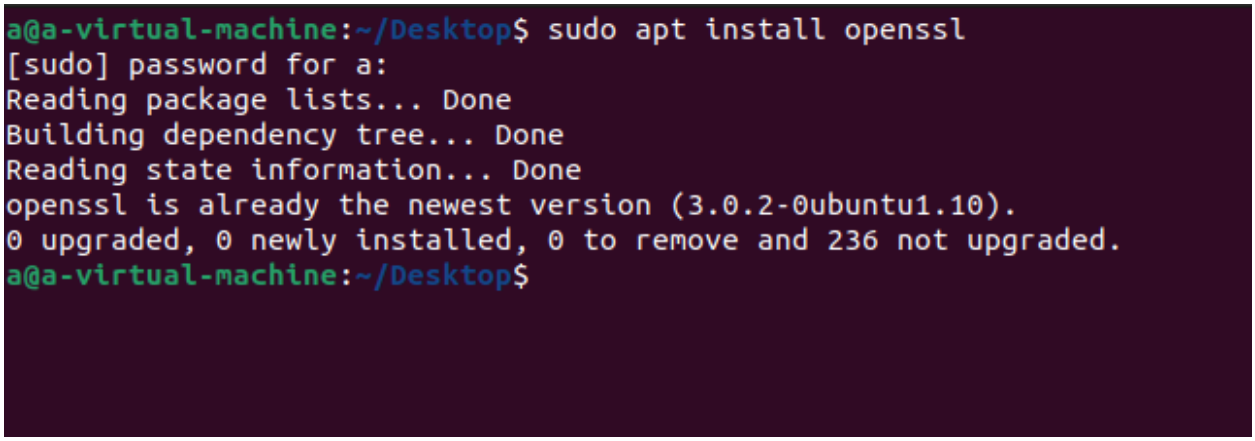
Hard Disk : 20 GB

Prerequisites

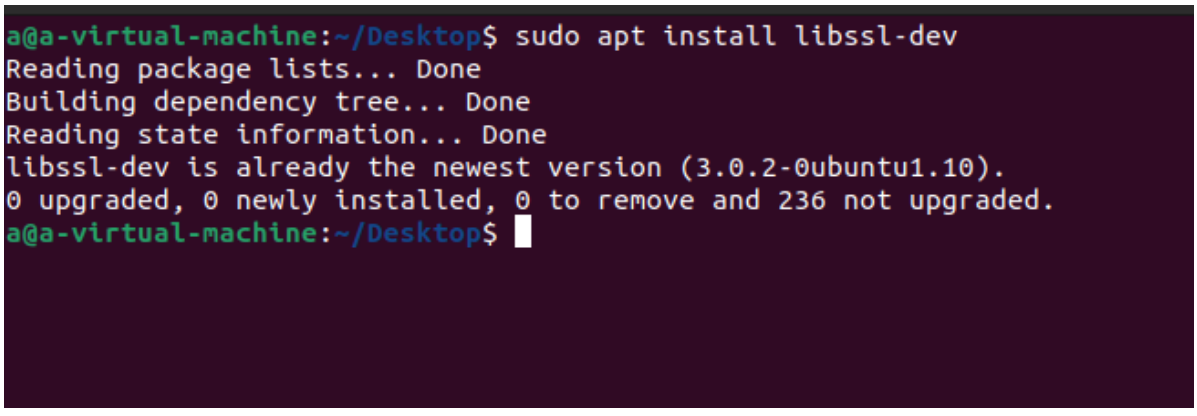
Before proceeding with the certificate generation process, ensure you have the following prerequisites:

1. OpenSSL : Make sure you have OpenSSL (openssl version >= 3.0.0) installed on your system.

   sudo apt install openssl



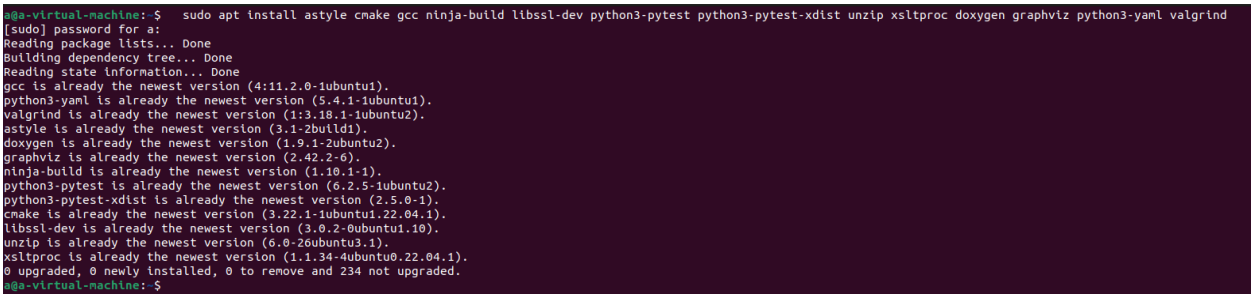
   sudo apt install libssl-dev



2. Open-quantum-safe / Oqs-provider : Ensure that your OpenSSL installation includes support for quantum safe algorithms.

   2.1 Install dependencies:

   sudo apt install astyle cmake gcc ninja-build libssl-dev python3-pytest python3-pytest-xdist unzip xsltproc doxygen graphviz python3-yaml valgrind

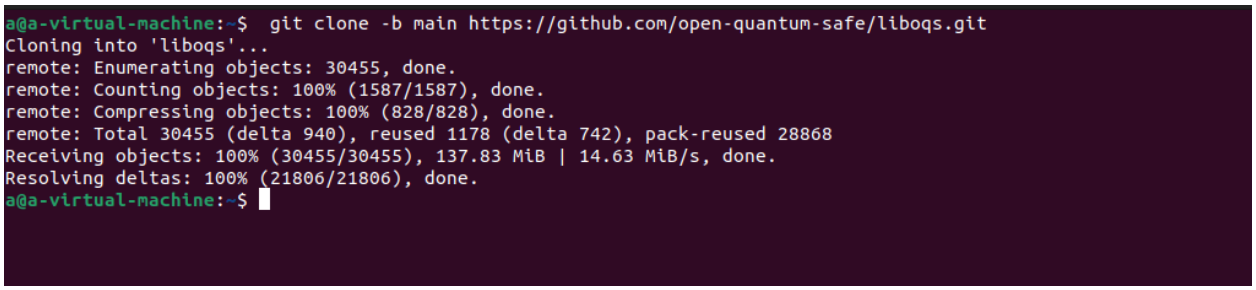


   2.2 Get the source (liboqs) :

   apt install git

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

   cd liboqs



2.3 Build :

mkdir build && cd build

```
a@a-virtual-machine:~$ cd liboqs/
a@a-virtual-machine:~/liboqs$ sudo mkdir build && cd build
a@a-virtual-machine:~/liboqs/build$
```

cmake -GNinja ..

```
a@a-virtual-machine:~/liboqs/build$ sudo cmake -GNinja ..
-- The C compiler identification is GNU 11.3.0
-- The ASM compiler identification is GNU
-- Found assembler: /usr/bin/cc
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Performing Test CC_SUPPORTS_WA_NOEXECSTACK
-- Performing Test CC_SUPPORTS_WA_NOEXECSTACK - Success
-- Performing Test LD_SUPPORTS_WL_Z_NOEXECSTACK
-- Performing Test LD_SUPPORTS_WL_Z_NOEXECSTACK - Success
-- Looking for pthread.h
-- Looking for pthread.h - found
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD - Success
-- Found Threads: TRUE
-- Alg enablement unchanged
-- Found OpenSSL: /usr/lib/x86_64-linux-gnu/libcrypto.so (found suitable version "3.0.2", minimum required is "1.1.1")
-- Looking for aligned_alloc
-- Looking for aligned_alloc - found
-- Looking for posix_memalign
-- Looking for posix_memalign - found
-- Looking for memalign
-- Looking for memalign - found
-- Looking for explicit_bzero
-- Looking for explicit_bzero - found
-- Looking for memset_s
-- Looking for memset_s - not found
-- Found Doxygen: /usr/bin/doxygen (found version "1.9.1") found components: doxygen dot
-- Configuring done
-- Generating done
-- Build files have been written to: /home/a/liboqs/build
a@a-virtual-machine:~/liboqs/build$
```

ninja

```
a@a-virtual-machine:~/liboqs/build$ sudo  ninja
[1226/1226] Linking C executable tests/dump_alg_info
a@a-virtual-machine:~/liboqs/build$
```

2.4 Install package:

ninja install

```
a@a-virtual-machine:~/liboqs/build$ sudo ninja install
[0/1] Install the project...
-- Install configuration: ""
-- Installing: /usr/local/lib/cmake/liboqs/liboqsConfig.cmake
-- Installing: /usr/local/lib/cmake/liboqs/liboqsConfigVersion.cmake
-- Installing: /usr/local/lib/pkgconfig/liboqs.pc
-- Installing: /usr/local/lib/liboqs.a
-- Installing: /usr/local/lib/cmake/liboqs/liboqsTargets.cmake
-- Installing: /usr/local/lib/cmake/liboqs/liboqsTargets-noconfig.cmake
-- Installing: /usr/local/include/oqs/oqs.h
-- Installing: /usr/local/include/oqs/common.h
-- Installing: /usr/local/include/oqs/rand.h
-- Installing: /usr/local/include/oqs/aes.h
-- Installing: /usr/local/include/oqs/sha2.h
-- Installing: /usr/local/include/oqs/sha3.h
-- Installing: /usr/local/include/oqs/sha3x4.h
-- Installing: /usr/local/include/oqs/kem.h
-- Installing: /usr/local/include/oqs/sig.h
-- Installing: /usr/local/include/oqs/kem_bike.h
-- Installing: /usr/local/include/oqs/kem_frodokem.h
-- Installing: /usr/local/include/oqs/kem_ntruprime.h
-- Installing: /usr/local/include/oqs/kem_classic_mceliece.h
-- Installing: /usr/local/include/oqs/kem_hqc.h
-- Installing: /usr/local/include/oqs/kem_kyber.h
-- Installing: /usr/local/include/oqs/sig_dilithium.h
-- Installing: /usr/local/include/oqs/sig_falcon.h
-- Installing: /usr/local/include/oqs/sig_sphincs.h
-- Installing: /usr/local/include/oqs/oqsconfig.h
a@a-virtual-machine:~/liboqs/build$
```

2.5 Install Prerequisites for oqs-provider:

These prerequisites include git, cmake, and a C compiler.

cd

apt install cmake build-essential

```
a@a-virtual-machine:~$ sudo apt install cmake build-essential git
[sudo] password for a:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  binutils binutils-common binutils-x86-64-linux-gnu cmake-data cpp-11 dh-elpa-helper dpkg-dev fakeroot g++ g++-11 gcc gcc-11 gcc-11-base git-man libalgorithm-diff-perl libalgorithm-diff-xs-perl
  libalgorithm-merge-perl libasan6 libbinutils libc-dev-bin libc-devtools libc6-dev libcc1-0 libcrypt-dev libctf-nobfd0 libctf0 libdpkg-perl liberror-perl libfakeroot libfile-fcntllock-perl
  libgcc-11-dev libitm1 libjsoncpp25 liblsan0 libnsl-dev libquadmath0 librhash0 libstdc++-11-dev libtirpc-dev libtsan0 libubsan1 linux-libc-dev lto-disabled-list make manpages-dev rpcsvc-proto
Suggested packages:
  binutils-doc cmake-doc ninja-build cmake-format gcc-11-locales debian-keyring g++-multilib g++-11-multilib gcc-11-doc gcc-multilib autoconf automake libtool flex bison gcc-doc gcc-11-multilib
  git-daemon-run | git-daemon-sysvinit git-doc git-email git-gui gitk gitweb git-cvs git-mediawiki git-svn glibc-doc bzr libstdc++-11-doc make-doc
The following NEW packages will be installed:
  binutils binutils-common binutils-x86-64-linux-gnu build-essential cmake cmake-data dh-elpa-helper dpkg-dev fakeroot g++ g++-11 gcc gcc-11 git git-man libalgorithm-diff-perl libalgorithm-diff-xs-perl
  libalgorithm-merge-perl libasan6 libbinutils libc-dev-bin libc-devtools libc6-dev libcc1-0 libcrypt-dev libctf-nobfd0 libctf0 libdpkg-perl liberror-perl libfakeroot libfile-fcntllock-perl
  libgcc-11-dev libitm1 libjsoncpp25 liblsan0 libnsl-dev libquadmath0 librhash0 libstdc++-11-dev libtirpc-dev libtsan0 libubsan1 linux-libc-dev lto-disabled-list make manpages-dev rpcsvc-proto
The following packages will be upgraded:
  cpp-11 gcc-11-base
2 upgraded, 47 newly installed, 0 to remove and 234 not upgraded.
Need to get 75.4 MB of archives.
After this operation, 239 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

2.6 Clone oqs-provider library :

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

cd oqs-provider

2.7 Build and Install:

To build and install oqs-provider, use the standard CMake build sequence. You can specify the location of OpenSSL and liboqs libraries if they are not installed in the system standard locations.

a. If openssl and liboqs are installed in the system standard locations:

cmake -S . -B _build && cmake --build _build && cmake --install _build

```
a@a-virtual-machine:~/Downloads/oqs-provider-main$ sudo su
root@a-virtual-machine:/home/a/Downloads/oqs-provider-main# cmake -S . -B _build && cmake --build _build && cmake --install _build
-- Creating Release build
-- Build will store public keys in PKCS#8 structures
-- Build will not include external encoding library for SPKI/PKCS#8
-- liboqs found: Include dir at /usr/local/include;/usr/local/include/oqs
fatal: not a git repository (or any of the parent directories): .git
-- Building commit  in /home/a/Downloads/oqs-provider-main
-- Configuring done
-- Generating done
-- Build files have been written to: /home/a/Downloads/oqs-provider-main/_build
Consolidate compiler generated dependencies of target oqsprovider
[ 39%] Built target oqsprovider
Consolidate compiler generated dependencies of target oqs_test_signatures
[ 50%] Built target oqs_test_signatures
Consolidate compiler generated dependencies of target oqs_test_kems
[ 60%] Built target oqs_test_kems
Consolidate compiler generated dependencies of target oqs_test_groups
[ 75%] Built target oqs_test_groups
Consolidate compiler generated dependencies of target oqs_test_tlssig
[ 89%] Built target oqs_test_tlssig
Consolidate compiler generated dependencies of target oqs_test_endecode
[100%] Built target oqs_test_endecode
-- Install configuration: ""
-- Installing: /usr/lib/x86_64-linux-gnu/ossl-modules/oqsprovider.so
root@a-virtual-machine:/home/a/Downloads/oqs-provider-main#
```

b. If openssl and/or liboqs have been installed to custom locations, use the OPENSSL_ROOT_DIR and liboqs_DIR CMake defines or environment variables:

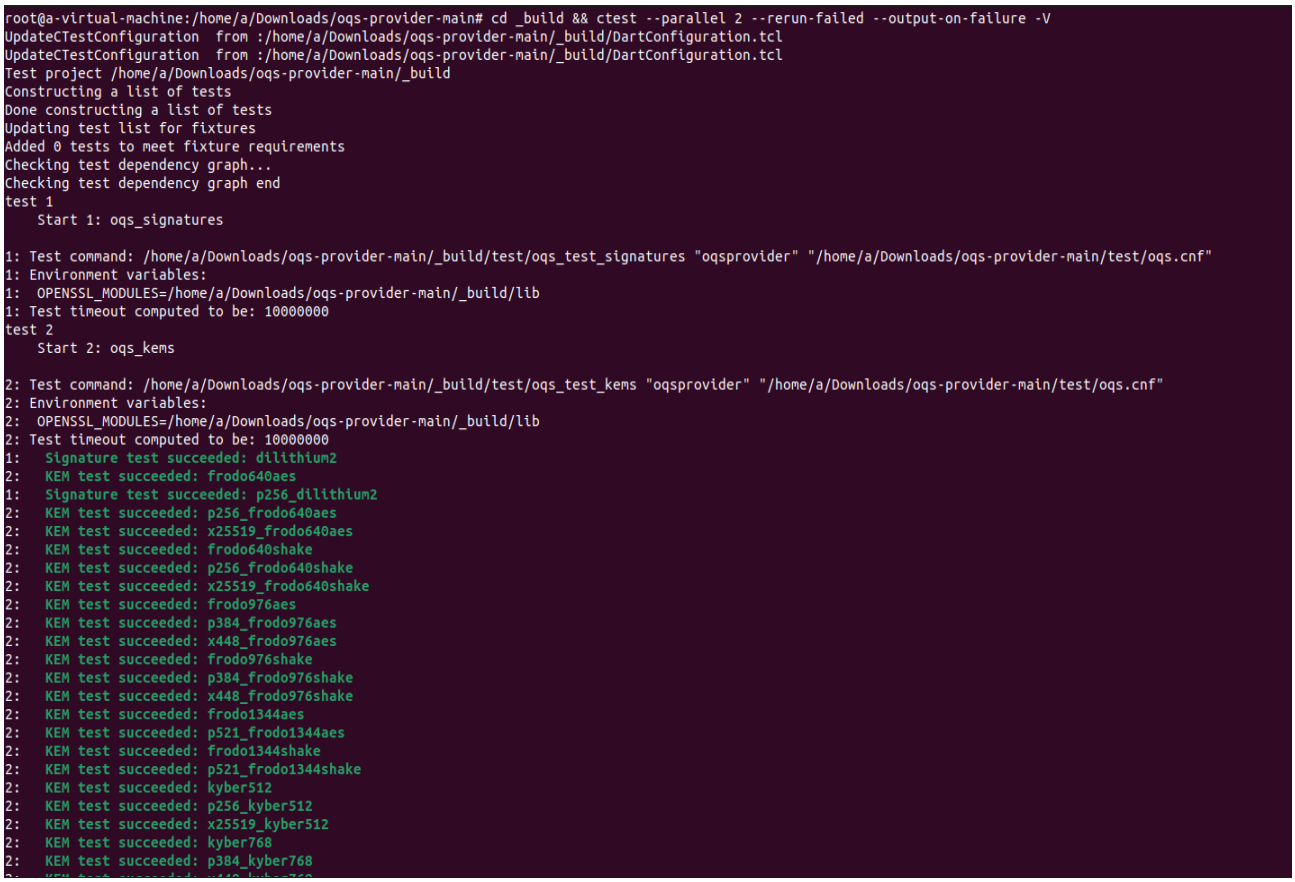
liboqs_DIR=../liboqs cmake -DOPENSSL_ROOT_DIR=/opt/openssl3 -S . -B _build && cmake --build _build && cmake --install _build

Replace /opt/openssl3 with the actual path to the OpenSSL installation if it is not in the system standard location.

2.8 Test the Build:

Standard ctest can be used to validate correct operation in build directory _build

cd _build && ctest --parallel 2 --rerun-failed --output-on-failure -V



2.9 Activation:

Step 1 :

a. Use -provider option:

Most OpenSSL commands accept the -provider option followed by the name of the provider to be activated. For oqs-provider, you can use it like this:

```
openssl list -signature-algorithms -provider oqsprovider
```

```
a@a-virtual-machine:~$ openssl list -signature-algorithms -provider oqsprovider
  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
a@a-virtual-machine:~$
```

The above command will list all quantum-safe signature algorithms made available for OpenSSL use by the oqs-provider

b. Use -provider-path option (if provider is not installed in the system location):

If the oqs-provider binary is not installed in the system location (lib/ossl-modules in the main OpenSSL installation tree), you can specify the location using the -provider-path option. For example:

```
openssl list -signature-algorithms -provider-path
/path/to/oqsprovider_binary
```

Replace /path/to/oqsprovider_binary with the actual path to the oqs-provider binary.

Step 2:

a. Edit the openssl.cnf file:

Locate the OpenSSL configuration file (openssl.cnf). The location of this file may vary depending on your system and OpenSSL installation. Common locations include /etc/ssl/openssl.cnf and /usr/lib/ssl/openssl.cnf.

(Or you can run : opensssl version -d)

b. Add activation instructions:

Open the openssl.cnf file in a text editor with administrative privileges. Add the following lines to the file:

1. Add the following lines below [provider_sect] keep the existing lines :

   legacy = legacy_sect

   oqsprovider = oqsprovider_sect

2. Add the following lines below [default_sect] keep the existing lines :

   activate = 1

3. Add the following lines if the header exists merge the the following with the existing body part :

   [legacy_sect]

   activate = 1

   [oqsprovider_sect]

   activate = 1

   [openssl_init]

```
ssl_conf = ssl_sect

[ssl_sect]

system_default = system_default_sect


[system_default_sect]

CipherString = DEFAULT:@SECLEVEL=2

Groups = kyber768:kyber1024
```

Save and close the openssl.cnf file.

Sample openssl.cnf :

```
#
# OpenSSL example configuration file.
# See doc/man5/config.pod for more info.
#
# This is mostly being used for generation of certificate requests,
# but may be used for auto loading of providers

# Note that you can include other files from the main configuration
# file using the .include directive.
#.include filename

# This definition stops the following lines choking if HOME isn't
# defined.
HOME                    = .

 # Use this in order to automatically load providers.
openssl_conf = openssl_init

# Comment out the next line to ignore configuration errors
config_diagnostics = 1

# Extra OBJECT IDENTIFIER info:
```

```
# oid_file       = $ENV::HOME/.oid
oid_section = new_oids

# To use this configuration file with the "-extfile" option of the
# "openssl x509" utility, name here the section containing the
# X.509v3 extensions to use:
# extensions             =
# (Alternatively, use a configuration file that has only
# X.509v3 extensions in its main [= default] section.)

[ new_oids ]
# We can add new OIDs in here for use by 'ca', 'req' and 'ts'.
# Add a simple OID like this:
# testoid1=1.2.3.4
# Or use config file substitution like this:
# testoid2=${testoid1}.5.6

# Policies used by the TSA examples.
tsa_policy1 = 1.2.3.4.1
tsa_policy2 = 1.2.3.4.5.6
tsa_policy3 = 1.2.3.4.5.7

# For FIPS
# Optionally include a file that is generated by the OpenSSL fipsinstall
```

```
# application. This file contains configuration data required by the OpenSSL

# fips provider. It contains a named section e.g. [fips_sect] which is

# referenced from the [provider_sect] below.

# Refer to the OpenSSL security policy for more information.

# .include fipsmodule.cnf

[openssl_init]

providers = provider_sect

ssl_conf = ssl_sect

# List of providers to load

[provider_sect]

default = default_sect

legacy = legacy_sect

[default_sect]

activate = 1

[legacy_sect]

activate = 1

# The fips section name should match the section name inside the

# included fipsmodule.cnf.
```

```
# fips = fips_sect

# If no providers are activated explicitly, the default one is activated implicitly.
# See man 7 OSSL_PROVIDER-default for more details.
#
# If you add a section explicitly activating any other provider(s), you most
# probably need to explicitly activate the default provider, otherwise it
# becomes unavailable in openssl.  As a consequence applications depending on
# OpenSSL may not work correctly which could lead to significant system
# problems including inability to remotely access the system.
[default_sect]
# activate = 1


####################################################################
[ ca ]
default_ca      = CA_default            # The default ca section

####################################################################
[ CA_default ]
```

```
dir             = ./demoCA              # Where everything is kept
certs           = $dir/certs            # Where the issued certs are kept
crl_dir         = $dir/crl              # Where the issued crl are kept
database        = $dir/index.txt        # database index file.
#unique_subject = no                    # Set to 'no' to allow creation of
                                        # several certs with same subject.
new_certs_dir   = $dir/newcerts         # default place for new
certs.

certificate     = $dir/cacert.pem       # The CA certificate
serial          = $dir/serial           # The current serial number
crlnumber       = $dir/crlnumber        # the current crl number
                                        # must be commented out to leave a V1
CRL
crl             = $dir/crl.pem          # The current CRL
private_key     = $dir/private/cakey.pem# The private key

x509_extensions = usr_cert              # The extensions to add to the cert

# Comment out the following two lines for the "traditional"
# (and highly broken) format.
name_opt        = ca_default            # Subject Name options
cert_opt        = ca_default            # Certificate field options
```

```
# Extension copying option: use with caution.
# copy_extensions = copy

# Extensions to add to a CRL. Note: Netscape communicator chokes on V2 CRLs
# so this is commented out by default to leave a V1 CRL.
# crlnumber must also be commented out to leave a V1 CRL.
# crl_extensions        = crl_ext

default_days  = 365                   # how long to certify for
default_crl_days= 30                  # how long before next CRL
default_md    = default               # use public key default MD
preserve      = no                    # keep passed DN ordering

# A few difference way of specifying how similar the request should look
# For type CA, the listed attributes must be the same, and the optional
# and supplied fields are just that :-)
policy        = policy_match

# For the CA policy
[ policy_match ]
countryName           = match
stateOrProvinceName         = match
organizationName    = match
```

```
organizationalUnitName     = optional
commonName                 = supplied
emailAddress               = optional

# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types.
[ policy_anything ]
countryName            = optional
stateOrProvinceName        = optional
localityName           = optional
organizationName       = optional
organizationalUnitName     = optional
commonName                 = supplied
emailAddress           = optional

####################################################################
####
[ req ]
default_bits           = 2048
default_keyfile        = privkey.pem
distinguished_name = req_distinguished_name
attributes             = req_attributes
x509_extensions    = v3_ca         # The extensions to add to the self signed
```

cert

Passwords for private keys if not present they will be prompted for

input_password = secret

output_password = secret

This sets a mask for permitted string types. There are several options.

default: PrintableString, T61String, BMPString.

pkix : PrintableString, BMPString (PKIX recommendation before 2004)

utf8only: only UTF8Strings (PKIX recommendation after 2004).

nombstr : PrintableString, T61String (no BMPStrings or UTF8Strings).

MASK:XXXX a literal mask value.

WARNING: ancient versions of Netscape crash on BMPStrings or UTF8Strings.

string_mask = utf8only

req_extensions = v3_req # The extensions to add to a certificate request

[req_distinguished_name]

countryName = Country Name (2 letter code)

countryName_default = AU

countryName_min = 2

countryName_max = 2

stateOrProvinceName = State or Province Name (full name)

stateOrProvinceName_default = Some-State

localityName = Locality Name (eg, city)

0.organizationName = Organization Name (eg, company)

0.organizationName_default = Internet Widgits Pty Ltd

we can do this but it is not needed normally :-)

#1.organizationName = Second Organization Name (eg, company)

#1.organizationName_default = World Wide Web Pty Ltd

organizationalUnitName = Organizational Unit Name (eg, section)

#organizationalUnitName_default =

commonName = Common Name (e.g. server FQDN or YOUR name)

commonName_max = 64

emailAddress = Email Address

emailAddress_max = 64

SET-ex3 = SET extension number 3

```
[ req_attributes ]
challengePassword               = A challenge password
challengePassword_min                   = 4
challengePassword_max                   = 20

unstructuredName                = An optional company name

[ usr_cert ]

# These extensions are added when 'ca' signs a request.

# This goes against PKIX guidelines but some CAs do it and some software
# requires this to avoid interpreting an end user certificate as a CA.

basicConstraints=CA:FALSE

# This is typical in keyUsage for a client certificate.
# keyUsage = nonRepudiation, digitalSignature, keyEncipherment

# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer
```

```
# This stuff is for subjectAltName and issuerAltname.
# Import the email address.
# subjectAltName=email:copy
# An alternative to produce certificates that aren't
# deprecated according to PKIX.
# subjectAltName=email:move

# Copy subject details
# issuerAltName=issuer:copy

# This is required for TSA certificates.
# extendedKeyUsage = critical,timeStamping

[ v3_req ]

# Extensions to add to a certificate request

basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment

[ v3_ca ]


# Extensions for a typical CA
```

```
# PKIX recommendation.

subjectKeyIdentifier=hash

authorityKeyIdentifier=keyid:always,issuer

basicConstraints = critical,CA:true

# Key usage: this is typical for a CA certificate. However since it will
# prevent it being used as an test self-signed certificate it is best
# left out by default.
# keyUsage = cRLSign, keyCertSign

# Include email address in subject alt name: another PKIX recommendation
# subjectAltName=email:copy
# Copy issuer details
# issuerAltName=issuer:copy

# DER hex encoding of an extension: beware experts only!
# obj=DER:02:03
# Where 'obj' is a standard or added object
# You can even override a supported extension:
```

```
# basicConstraints= critical, DER:30:03:01:01:FF

[ crl_ext ]

# CRL extensions.
# Only issuerAltName and authorityKeyIdentifier make any sense in a CRL.

# issuerAltName=issuer:copy
authorityKeyIdentifier=keyid:always

[ proxy_cert_ext ]
# These extensions should be added when creating a proxy certificate

# This goes against PKIX guidelines but some CAs do it and some software
# requires this to avoid interpreting an end user certificate as a CA.

basicConstraints=CA:FALSE

# This is typical in keyUsage for a client certificate.
# keyUsage = nonRepudiation, digitalSignature, keyEncipherment

# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer
```

```
# This stuff is for subjectAltName and issuerAltname.
# Import the email address.
# subjectAltName=email:copy
# An alternative to produce certificates that aren't
# deprecated according to PKIX.
# subjectAltName=email:move

# Copy subject details
# issuerAltName=issuer:copy

# This really needs to be in place for it to be a proxy certificate.
proxyCertInfo=critical,language:id-ppl-anyLanguage,pathlen:3,policy:foo

####################################################################
####
[ tsa ]

default_tsa = tsa_config1      # the default TSA section

[ tsa_config1 ]

# These are used by the TSA reply generation only.
dir                = ./demoCA              # TSA root directory
```

```
serial          = $dir/tsaserial        # The current serial number
(mandatory)

crypto_device   = builtin               # OpenSSL engine to use for
signing

signer_cert     = $dir/tsacert.pem      # The TSA signing certificate
                                        # (optional)

certs           = $dir/cacert.pem       # Certificate chain to include in reply
                                        # (optional)

signer_key      = $dir/private/tsakey.pem # The TSA private key (optional)

signer_digest  = sha256                 # Signing digest to use. (Optional)

default_policy  = tsa_policy1           # Policy if request did not specify
it
                                        # (optional)

other_policies  = tsa_policy2, tsa_policy3     # acceptable policies
(optional)

digests     = sha1, sha256, sha384, sha512  # Acceptable message digests
(mandatory)

accuracy        = secs:1, millisecs:500, microsecs:100        # (optional)

clock_precision_digits  = 0   # number of digits after dot. (optional)

ordering        = yes   # Is ordering defined for timestamps?
                        # (optional, default: no)

tsa_name        = yes   # Must the TSA name be included in the reply?
                        # (optional, default: no)

ess_cert_id_chain     = no    # Must the ESS cert id chain be included?
                        # (optional, default: no)
```

```
ess_cert_id_alg                 = sha1 # algorithm to compute certificate
                                # identifier (optional, default: sha1)

[insta] # CMP using Insta Demo CA
# Message transfer
server = pki.certificate.fi:8700
# proxy = # set this as far as needed, e.g., http://192.168.1.1:8080
# tls_use = 0
path = pkix/

# Server authentication
recipient = "/C=FI/O=Insta Demo/CN=Insta Demo CA" # or set srvcert or
issuer
ignore_keyusage = 1 # potentially needed quirk
unprotected_errors = 1 # potentially needed quirk
extracertsout = insta.extracerts.pem

# Client authentication
ref = 3078 # user identification
secret = pass:insta # can be used for both client and server side

# Generic message options
cmd = ir # default operation, can be overridden on cmd line with, e.g., kur
```

```
# Certificate enrollment
subject = "/CN=openssl-cmp-test"
newkey = insta.priv.pem
out_trusted = insta.ca.crt
certout = insta.cert.pem

[pbm] # Password-based protection for Insta CA
# Server and client authentication
ref = $insta::ref # 3078
secret = $insta::secret # pass:insta

[signature] # Signature-based protection for Insta CA
# Server authentication
trusted = insta.ca.crt # does not include keyUsage digitalSignature

# Client authentication
secret = # disable PBM
key = $insta::newkey # insta.priv.pem
cert = $insta::certout # insta.cert.pem
[ir]
cmd = ir
[cr]
cmd = cr
```

```
[kur]

# Certificate update

cmd = kur

oldcert = $insta::certout # insta.cert.pem


[rr]

# Certificate revocation

cmd = rr

oldcert = $insta::certout # insta.cert.pem

[ssl_sect]

system_default = system_default_sect

[system_default_sect]

CipherString = DEFAULT:@SECLEVEL=2

Groups = kyber768:kyber1024

[provider_sect]

default = default_sect

oqsprovider = oqsprovider_sect

[oqsprovider_sect]

activate = 1
```

==========

Available quantum-safe/PQ KEM algorithms : https://github.com/open-quantum-safe/oqs-provider/blob/main/README.md#kem-algorithms

1. Generating Dilithium-based SSL Certificates

Generating the Root Certificate (Certificate Authority)

Step 1: Generate the private key for the Certificate Authority (CA):

openssl genpkey -algorithm <dilithium3> -out key_CA.key

```
a@a-virtual-machine:~/Desktop/cert$ sudo openssl genpkey -algorithm dilithium3 -out key_CA.key
a@a-virtual-machine:~/Desktop/cert$
```

Step 2: Create the self-signed Root Certificate (CA Certificate):

openssl req -x509 -new -newkey <dilithium3> -key key_CA.key -out Certificate_CA.crt -nodes -subj "/CN=My CA" -days 365 -config /usr/lib/ssl/openssl.cnf

```
a@a-virtual-machine:~/Desktop/cert$ openssl req -x509 -new -newkey dilithium3 -key key_CA.key -out Certificate_CA.crt -nodes -days 365 -config /usr/lib/ssl/openssl.cnf
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IN
State or Province Name (full name) [Some-State]:KA
Locality Name (eg, city) []:BLR
Organization Name (eg, company) [Internet Widgits Pty Ltd]:CDAC
Organizational Unit Name (eg, section) []:RISE
Common Name (e.g. server FQDN or YOUR name) []:*.cdac.in
Email Address []:cdac@cdac.in
a@a-virtual-machine:~/Desktop/cert$
```

Generating the Server Certificate

Step 1: Generate the private key for the server certificate:

openssl genpkey -algorithm <dilithium3> -out private.key

```
a@a-virtual-machine:~/Desktop/cert$ openssl genpkey -algorithm dilithium3 -out private.key
a@a-virtual-machine:~/Desktop/cert$
```

Step 2: Create a certificate signing request (CSR) for the server certificate:

openssl req -new -newkey <dilithium3> -key private.key -out Certificate.csr -nodes -subj "/CN=test server" -config /usr/lib/ssl/openssl.cnf -extensions v3_req

```
a@a-virtual-machine:~/Desktop/cert$ openssl req -new -newkey dilithium3 -key private.key -out Certificate.csr -nodes -config /usr/lib/ssl/openssl.cnf -extensions v3_req
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IN
State or Province Name (full name) [Some-State]:KA
Locality Name (eg, city) []:BLR
Organization Name (eg, company) [Internet Widgits Pty Ltd]:CDAC
Organizational Unit Name (eg, section) []:RISE
Common Name (e.g. server FQDN or YOUR name) []:*.in
Email Address []:cdac@cdac.in

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:Test@123
An optional company name []:cdac
a@a-virtual-machine:~/Desktop/cert$
```

Signing the Server Certificate

Step 1: Sign the server certificate using the previously generated root certificate and key:

openssl x509 -req -in Certificate.csr -out Certificate.crt -CA Certificate_CA.crt -CAkey key_CA.key -CAcreateserial -days 365 -extfile /usr/lib/ssl/openssl.cnf -extensions v3_req

```
a@a-virtual-machine:~/Desktop/cert$ openssl x509 -req -in Certificate.csr -out Certificate.crt -CA Certificate_CA.crt -CAkey key_CA.key -CAcreateserial -days 365 -extfile /usr/lib/ssl/openssl.cnf -extens
ions v3_req
Certificate request self-signature ok
subject=C = IN, ST = KA, L = BLR, O = CDAC, OU = RISE, CN = *.in, emailAddress = cdac@cdac.in
a@a-virtual-machine:~/Desktop/cert$
```

Verifying the Server Certificate

To verify the server certificate's details, run the following command:

```
openssl x509 -text -noout -in Certificate.crt
```

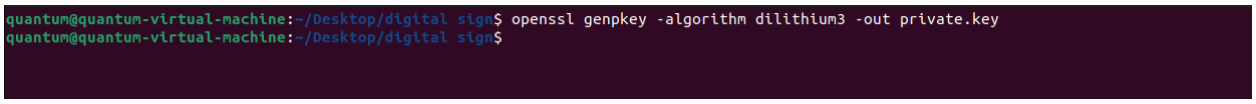
```
a@a-virtual-machine:~/Desktop/cert$ openssl x509 -text -noout -in Certificate.crt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            6a:06:5e:d9:7b:f3:c2:60:d5:40:ee:82:cd:2d:f6:dc:7e:95:41:59
        Signature Algorithm: dilithium3
        Issuer: C = IN, ST = KA, L = BLR, O = CDAC, OU = RISE, CN = *.cdac.in, emailAddress = cdac@cdac.in
        Validity
            Not Before: Jul 25 11:51:58 2023 GMT
            Not After : Jul 24 11:51:58 2024 GMT
        Subject: C = IN, ST = KA, L = BLR, O = CDAC, OU = RISE, CN = *.in, emailAddress = cdac@cdac.in
        Subject Public Key Info:
            Public Key Algorithm: dilithium3
                dilithium3 public key:
                PQ key material:
                    d6:9b:b1:5d:46:d0:70:95:64:fc:2c:cc:bd:7a:b3:
                    7c:19:36:30:c8:03:6c:73:18:02:83:00:cc:29:0a:
                    26:58:d7:7a:09:92:70:18:9e:6f:b5:b8:a2:6e:cf:
                    8a:56:7e:44:27:ab:76:46:83:ad:d3:87:56:31:10:
                    bd:6b:47:4b:f3:a5:98:5d:4b:b8:ee:cc:c7:e5:cc:
                    81:c0:b1:a2:99:5c:0e:f3:bf:e0:a1:35:e0:21:a5:
                    ab:50:aa:93:1e:e5:df:e4:96:91:e7:c0:25:39:a1:
                    b6:7e:94:92:c3:54:90:c3:da:ad:23:be:2f:08:59:
                    bd:98:d5:14:18:2d:b1:af:e2:65:d4:6f:46:b3:6a:
                    36:b0:1f:cb:96:83:32:ea:65:a7:16:fa:cf:ef:9d:
                    fb:6c:9a:74:aa:9c:05:d9:48:5f:92:b5:de:e7:d1:
                    e5:4e:1b:45:ce:60:99:56:d9:0c:9e:df:8a:8a:b5:
                    d3:3d:2c:30:1f:0a:11:8f:2a:01:bf:b8:54:ca:4e:
                    b4:bf:0a:4c:9f:9b:9d:bc:8c:b1:2e:cf:44:71:bc:
                    43:44:79:09:61:da:11:6f:03:77:49:b4:f8:97:96:
                    b3:73:e6:39:66:b1:1e:44:27:b2:fe:57:a9:7e:54:
                    74:d6:92:a2:be:d3:c7:b2:ae:8c:01:be:77:45:73:
                    bf:3d:ad:98:d0:56:30:48:73:aa:b3:93:8a:ab:d0:
                    d3:d1:02:0a:08:cc:09:e5:ce:c0:30:bd:b2:2d:ac:
                    30:eb:02:8f:db:ad:a7:68:71:da:20:c8:df:a5:e4:
                    a7:16:2f:9d:40:eb:51:46:64:f1:97:55:97:ac:b8:
                    24:aa:43:81:a9:c2:6d:4c:c3:f6:7b:22:6b:2c:97:
```

2.Digital Signing

Create private key :

To create a private key, we will be using the following command :

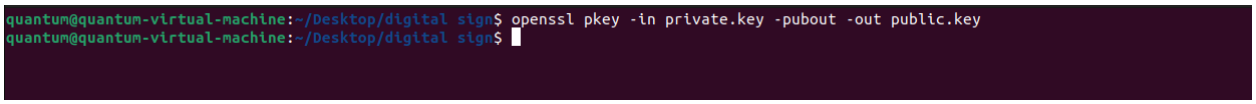
openssl genpkey -algorithm <dilithium3> -out private.key



Extract the Public Key from the Private key :

Once the private key is generated, we need to extract the public key for further use :

openssl pkey -in private.key -pubout -out public.key



Signing Data with the Private Key :

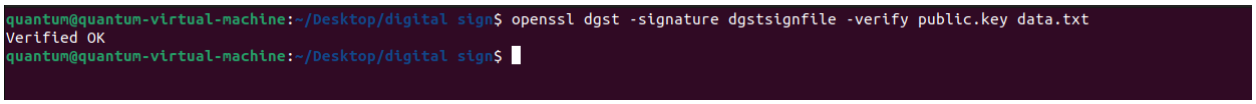
To sign data using the quantum-safe private key, execute the following command:

openssl dgst -sign private.key -out dgstsignfile data.txt

Verifying the Digital Signature :

To verify the file, run the following command:

```
openssl dgst -signature dgstsignfile -verify public.key data.txt
```



3. Generating Self Signed Certificates

3.1 Using CSR(Certificate Signing Request)

Creating a Private Key:

Generate the private key using the command

openssl genpkey -algorithm <dilithium5> -out key.key

```
a@a-virtual-machine:~/Desktop/self$ openssl genpkey -algorithm dilithium5 -out key.key
a@a-virtual-machine:~/Desktop/self$
```

Creating a Certificate Signing Request:

openssl req -key key.key -new -out domain.csr -extensions v3_req

```
a@a-virtual-machine:~/Desktop/self$ openssl req -key key.key -new -out domain.csr -extensions v3_req
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IN
State or Province Name (full name) [Some-State]:KA
Locality Name (eg, city) []:BLR
Organization Name (eg, company) [Internet Widgits Pty Ltd]:C-DAC
Organizational Unit Name (eg, section) []:Rise
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
a@a-virtual-machine:~/Desktop/self$
```

Signing Using CSR And Key:

Use the following command to sign the certificate :

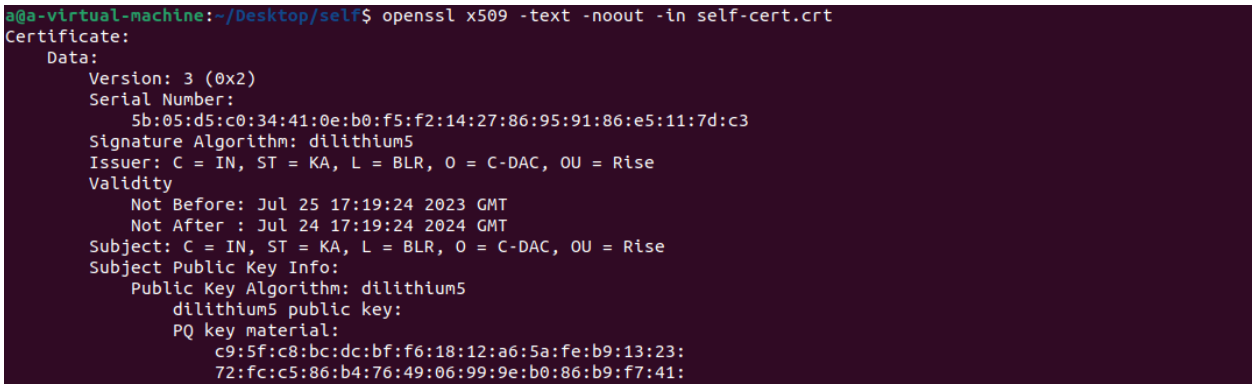
openssl x509 -signkey key.key -in domain.csr -req -days 365 -out self-cert.crt -extfile /usr/lib/ssl/openssl.conf -extensions v3_req

```
a@a-virtual-machine:~/Desktop/self$ openssl x509 -signkey key.key -in domain.csr -req -days 365 -out self-cert.crt  -extfile /usr/lib/ssl/openssl.cnf -extensions v3_req
Certificate request self-signature ok
subject=C = IN, ST = KA, L = BLR, O = C-DAC, OU = Rise
a@a-virtual-machine:~/Desktop/self$
```

Verifying the Certificate

To verify the certificate's details, run the following command:

```
openssl x509 -text -noout -in self-cert.crt
```

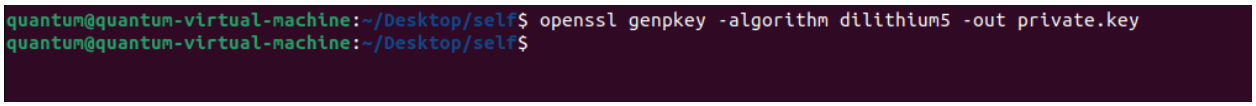


3.2 Using Private Key

Creating a Private Key:

Generate Private Key using the command :

openssl genpkey -algorithm <dilithium5> -out private.key



Signing Using Private Key:

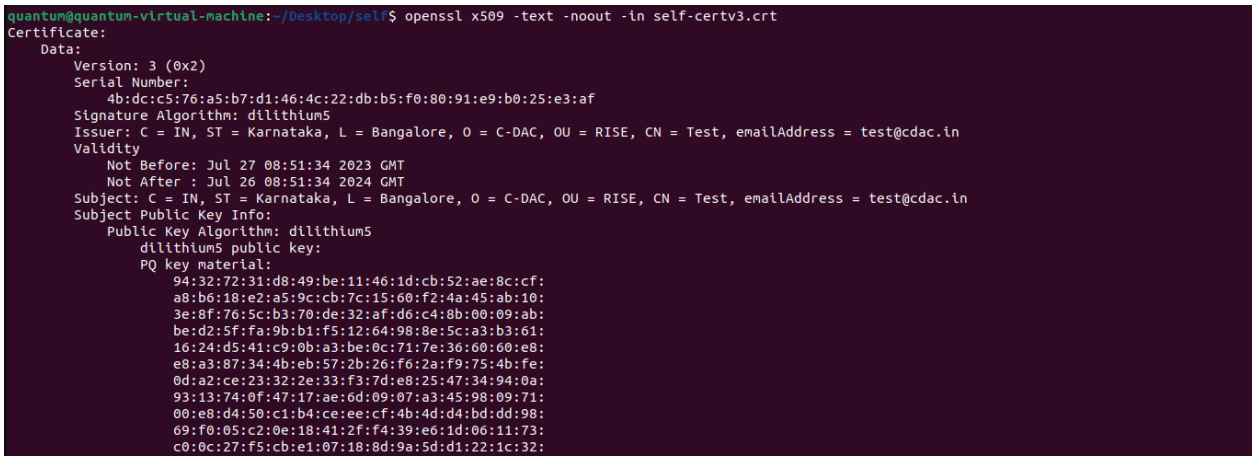
Use the following command to sign the certificate :

openssl req -key private.key -new -x509 -days 365 -out self-certv3.crt -extensions usr_cert

Verifying the Certificate

To verify the certificate's details, run the following command:

```
openssl x509 -text -noout -in self-cert.crt
```



4. Generating Hybrid Certificates

Generating the Root Certificate (Certificate Authority)

Step 1: Generate a private key for the Certificate Authority (CA) with 2048-bit key length :

```
openssl genrsa -out CA-pvt.key 2048
```

```
quantum@quantum-virtual-machine:~/Desktop/Hybrid$ openssl genrsa -out CA-pvt.key 2048
quantum@quantum-virtual-machine:~/Desktop/Hybrid$
```

Step 2: Once the private key is generated, we need to extract the public key for further use :

```
openssl rsa -in CA-pvt.key -pubout -out CA-pub.pem
```

```
quantum@quantum-virtual-machine:~/Desktop/Hybrid$ openssl rsa -in CA-pvt.key -pubout -out CA-pub.pem
writing RSA key
quantum@quantum-virtual-machine:~/Desktop/Hybrid$
```

Step 3: Create the self-signed Root Certificate (CA Certificate):

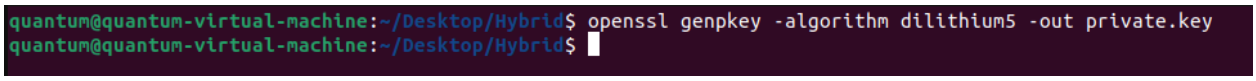
```
openssl req -new -x509 -key CA-pvt.key -days 3650 -out CA.crt
```

```
quantum@quantum-virtual-machine:~/Desktop/Hybrid$ openssl req -new -x509 -key CA-pvt.key -days 3650 -out CA
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IN
State or Province Name (full name) [Some-State]:KARNATAKA
Locality Name (eg, city) []:BANGALORE
Organization Name (eg, company) [Internet Widgits Pty Ltd]:CDAC
Organizational Unit Name (eg, section) []:RISE
Common Name (e.g. server FQDN or YOUR name) []:*.cdac.in
Email Address []:cdac@cdac.in
quantum@quantum-virtual-machine:~/Desktop/Hybrid$
```

Generating the Server Certificate

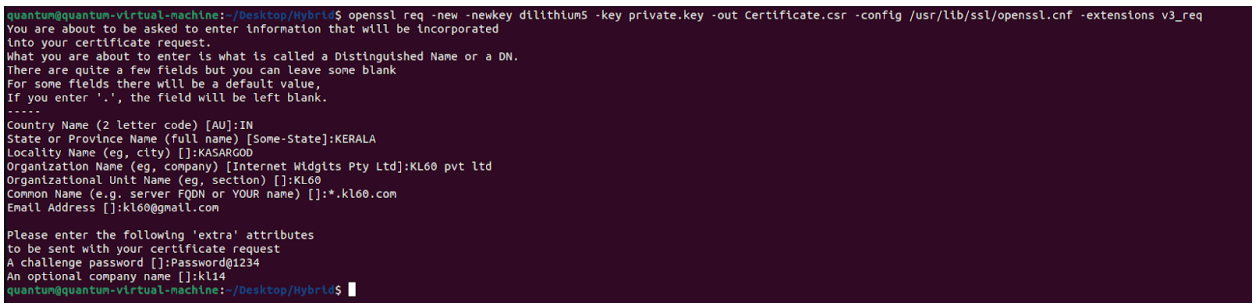
Step 1: Generate the private key for the server certificate:

```
openssl genpkey -algorithm <dilithium5> -out private.key
```



Step 2: Create a certificate signing request (CSR) for the server certificate:

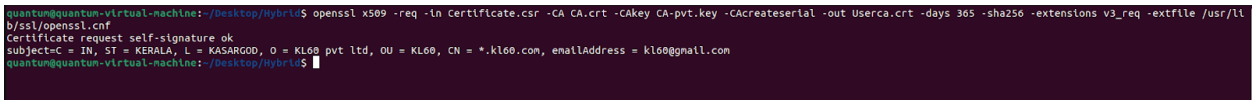
```
openssl req -new -newkey <dilithium5> -key private.key -out Certificate.csr
-config /usr/lib/ssl/openssl.cnf -extensions v3_req
```



Signing the Server Certificate

Step 1: Sign the server certificate using the root certificate and key:

```
openssl x509 -req -in Certificate.csr -CA CA.crt -CAkey CA-pvt.key -CAcreateserial
-out Userca.crt -days 365 -sha256 -extensions v3_req -extfile /usr/lib/ssl/openssl.cnf
```



REFERENCES

- [1] OpenSSL Foundation, Inc. (no date) OpenSSL, /docs/man3.0/man7/crypto.html. Available at: https://www.openssl.org/docs/man3.0/man7/crypto.html (Accessed: 17 July 2023).
- [2] “Home,” Open Quantum Safe, https://openquantumsafe.org/ (accessed Jul. 27, 2023).
- [3] Open-Quantum-Safe, “GitHub - open-quantum-safe/oqs-provider: OpenSSL 3 provider containing post-quantum algorithms,” GitHub. https://github.com/open-quantum-safe/oqs-provider (Accessed: 19 July 2023).
- [4] Open-Quantum-Safe, “GitHub - open-quantum-safe/liboqs: C library for prototyping and experimenting with quantum-resistant cryptography,” GitHub. https://github.com/open-quantum-safe/liboqs (Accessed: 19 July 2023).
- [5] “Creating a Self-Signed Certificate with OpenSSL | Baeldung,” *Baeldung*, Oct. 2022, [Online]. Available: https://www.baeldung.com/openssl-self-signed-cert (Accessed: 20 July 2023).
- [6] C. M. Jun , “Creating public key from private key | Baeldung on Linux,” *Baeldung on Linux*, Aug. 2022, [Online]. Available: https://www.baeldung.com/linux/public-key-from-private-key#:~:text=Getting%20the%20Public%20Key%20using%20openssl (Accessed: 26 July 2023).