National College of Ireland

Configuration Manual

MSc Research Project
MSc. Cybersecurity

Keshav Singh
Student ID: 22101624

School of Computing
National College of Ireland

Supervisor: Prof. Imran Khan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Keshav Singh ……

Student ID: X22101624……..……

Programme: Msc Cybersecurity…… **Year:** …2023……..

Module: Research project…….……

Lecturer: Imran Khan…….……

Submission Due Date: …14-12-12…….……

Project Title: ……Quantum-safe IAM…….……

Word Count: ……2116…… **Page Count:** …13…….……

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……Keshav Singh……

Date: …14-12-12……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Keshav Singh
Student ID: 22101624

1 Introduction

The manual provides a step-by-step guide for identifying the IAM components that require post-quantum cryptographic protection, evaluating and selecting post-quantum cryptographic algorithms, ciphers, and digital signatures, installing the liboqs library, modifying the Apache configuration file, generating X.509 certificates, configuring the IAM components to use the chosen post-quantum cryptographic algorithms, and testing and monitoring the IAM infrastructure for potential issues or vulnerabilities. By following these steps, IAM administrators can successfully implement post-quantum cryptographic algorithms in their IAM infrastructure, providing a defence against potential quantum computer attacks and ensuring the security of sensitive data. As suggested in the main thesis report, we try to create a prototype for integrating post quantum cryptography. A TLSv1.3 architecture is created the details of which have been provided in the report. The steps to replicate the same has been provided below along with suitable links for the same. The subsequent parts include the hardware and software configurations used which is mostly not very demanding.

DISCLAIMER: The current algorithms used in this manual are approved by NIST but are still under development stage. It is not suggested to utilise the same in a production environment.

2 System configuration

The hardware configurations and software configuration used in the implementation of this prototype have been described in the part that follows. Hardware configurations were all easily available while it did require the download and installation of a wide variety of software programs. Detailed steps for installation and configuration have been made available along with required links for the same.

2.1 Hardware configuration

The below table gives out the hardware components used in the process of making this prototype along with specs for the same.

Item	Item Specification
CPU	AMD Ryzen 5 5600H with Radeon Graphics
RAM	16.0 GB (15.4 GB usable)
Disk space	475 GB
System type	64-bit operating system, x64-based processor

2.2 Software configuration

All software used and referred to in the prototype has been provided in the table below. The table also includes a brief description, version and name of the software utilized.

Item name	Brief Description	Software Version
OpenSSL OQS enabled fork	It is a fork of the original openssl library which is integrated with liboqs. This allows it support Post quantum cryptography.	OQS-OpenSSL_1_1_1-stable
Liboqs	Liboqs is an open source library created to aid in the development of Quantum resistant services. The library is written in C.	0.7.0
wolfSSL	lightweight SSL/TLS library	wolfSSL Release 5.6.4
Apache httpd server	open-source web server software	httpd-2.4.51
Apache Portable Runtime (APR)	APR provides a set of APIs that abstract operating system-specific functionality	apr-1.7.0
Apache Portable Runtime Utility (APR-util)	APR-util includes utility functions and abstractions for database connectivity	apr-util-1.6.1
libpcre3-dev package	development files and static library for compiling applications that use PCRE	2:8.39-12ubuntu0.1
Expat	Expat is an open source XML parsing library	expat-2.4.1
GNOME Web Browser(epiphany)	web browser developed for the GNOME desktop environment	Web 3.6.4
libpcre3	shared library that is needed at runtime by applications using PCRE for regular expression support	2:8.39-12ubuntu0.1

3 Prototype configuration and installation guideline

3.1 Hybrid key exchange prototype

We first create a prototype of hybrid Key exchange mechanism between a server and a client both on different terminals. We run first the server and wait for the TLS handshake after

which we run the client to complete the TLS hybrid handshake. But the key part here to note is that we run it using the post quantum cryptography algorithm P521_KYBER_LEVEL5. But to get this, we use WolfSSL as normal linux programs do not support post quantum cryptography yet. We start off by following the initial steps to install, build and configure the software to be required for the prototype. We are currently using a ubuntu 10.0.8 virtual box VM.

Step 0: Set home directory as current working directory

```
cd ~
```

Step 1: Before installing liboqs and OQS OpenSSL, we Install pre-requisites for liboqs and OQS OpenSSL build.

```
sudo apt install cmake
sudo apt install gcc
sudo apt install libtool
sudo apt install make
sudo apt install ninja-build
sudo apt install git
sudo apt install libssl-dev
```

Step 2: Clone the Open Quantum Safe OpenSSL repository into our current directory.

```
git clone --branch OQS-OpenSSL_1_1_1-stable https://github.com/open-quantum-safe/openssl.git
```

Step 3: We also clone, configure, build, and install liboqs before OQS OpenSSL as liboqs is the C library used by OQS OpenSSL.

```
git clone --branch main https://github.com/open-quantum-safe/liboqs.git
cd liboqs
git checkout 0.8.0
mkdir build
cd build
cmake -DOQS_USE_OPENSSL=0 ..
make all
sudo make install
```

Step 4: After liboqs is completed, we build and install the OQS OpenSSL fork, configured to work all current Level 5 NIST Round 3 hybrid TLS handshake algorithms.

```
cd ~/openssl
./Configure no-shared linux-x86_64 -
DOQS_DEFAULT_GROUPS=\"p521_kyber1024:p521_kyber90s1024:p521_ntru_hps40961
229:p521_ntru_hps4096821:p521_ntru_hrss1373:p521_firesaber:secp521_r1\" -lm

make -j 1
sudo make install
```

Step 5: Confirm that installed OpenSSL version is the Open Quantum Safe build

cd ~
/usr/local/bin/openssl version

This should return "OpenSSL 1.1.1u , Open Quantum Safe as shown in the figure below.

```
Password:
root@ubuntu2:~#
root@ubuntu2:~# /usr/local/bin/openssl version
OpenSSL 1.1.1u  30 May 2023, Open Quantum Safe 2023-07
root@ubuntu2:~#
```

Figure 1: OQS OpenSSL version

To make it easier, a bash file has been created for the setup to run seamlessly and attached with the code.

```
#!/bin/bash

#Step 0: Set home directory as current working directory

cd ~

#Step 1: Install pre-requisites for liboqs and OQS OpenSSL build

sudo apt install cmake gcc libtool libssl-dev make ninja-build git -y

#Step 2: Clone the Open Quantum Safe (OQS) OpenSSL repository

git clone --branch OQS-OpenSSL_1_1_1-stable https://github.com/open-quantum-safe/openssl.git

#Step 3: Clone, configure, build, and install liboqs

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

cd liboqs

mkdir build && cd build

cmake -GNinja -DCMAKE_INSTALL_PREFIX=~/openssl/oqs ..

ninja

ninja install

#Step 4: Build and install the OQS OpenSSL fork, configured to announce all current Level 5 NIST Round 3 hybrid TLS handhsake algorithms

cd ~/openssl

./Configure no-shared linux-x86_64 -DOQS_DEFAULT_GROUPS=\"p521_kyber1024:p521_kyber90s1024:p521_ntru_hps40961229:p521_ntru_hps4096821:p521_ntru_hrss1373:p521_firesal -lm

make -j 1

sudo make install

#Step 5: Confirm that installed OpenSSL version is the Open Quantum Safe build

cd ~

/usr/local/bin/openssl version

# This should return "OpenSSL 1.1.1l 24 Aug 2021, Open Quantum Safe 2021-xx-dev snapshot"
```

Figure 2: Configuration Bash file

After installation of dependecies and required libraries, we move onto install wolfSSL which is the main library for our prototype. This is done via the following steps.

git clone --depth 1 https://github.com/wolfssl/wolfssl
cd wolfssl
./autogen.sh (Might not be necessary)
./configure --with-liboqs
make all

After successful installation of wolfssl, we create the TLS1.3 Hybrid handshake by creating a server and a client. After the successful Key exchange mechanism, we get the following confirmations.

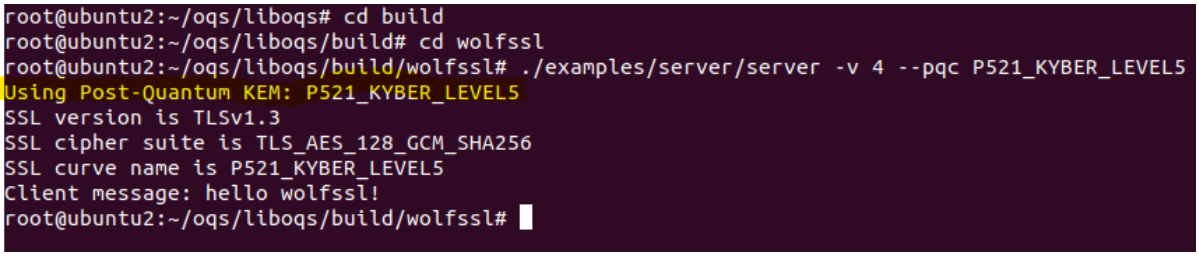

Figure 3: TLSv1.3 Server Hybrid key exchange

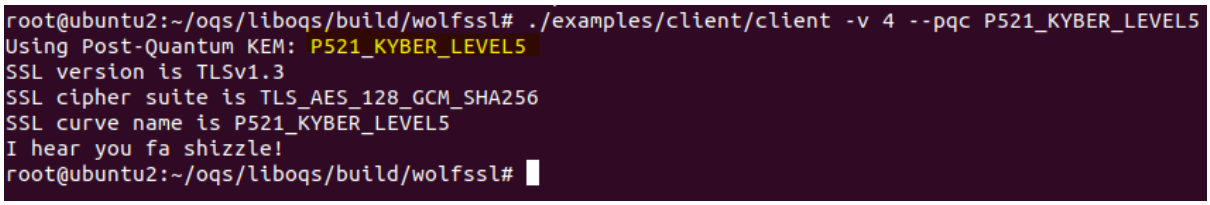

Figure 4: TLSv1.3 Client Hybrid key exchange

After successfully testing out the prototype, we utilise the same concept to create our own web application which supports TLS1.3 web server architecture. As well as X.509 certificates and TLS hybrid key exchange mechanism. In order to do this, we make use of the liboqs library and configure the Apache web server to support post quantum cryptography.

Since we already have both liboqs and OpenSSL OQS installed, we only need to install other dependencies for apache server.

Step 0: Download, configure, build required dependencies of httpd, apr and apr-util along with other requirements. Here we subsequently, use wget command to download followed by tar to extract.

wget https://archive.apache.org/dist/httpd/httpd-2.4.51.tar.bz2

tar -xvf httpd-2.4.51.tar.gz

wget https://archive.apache.org/dist/apr/apr-1.7.0.tar.gz

tar -xvf apr-1.7.0.tar.gz

wget https://archive.apache.org/dist/apr/apr-util-1.6.1.tar.gz

tar -xvf apr-util-1.6.1.tar.gz

Step 1: we move necessary files around to apache directory.

mv apr-1.7.0 httpd-2.4.51/srclib/apr

mv apr-util-1.6.1 httpd-2.4.51/srclib/apr-util

Step 2: we install libpcre3-dev and libpcre3 as they are required dependencies followed by expat install.

sudo apt install libpcre3-dev libpcre3 -y

wget https://src.fedoraproject.org/repo/pkgs/expat/expat-2.4.1.tar.gz/sha512/1f08861e9b766fdbbc40159404a3fe1a86451d635ef81874fa3492845eda83ac2dc6a0272525891d396b70c9a9254c2f6c907fe4abb2f8a533ccd3f52dae9d5a/expat-2.4.1.tar.gz

tar -xvf expat-2.4.1.tar.gz

Step 3: We configure and install required files.

cd expat-2.4.1
./configure
make
sudo make install
cd ~/httpd-2.4.51
./configure --with-ssl=/usr/local/bin/openssl --with-expat=/usr/local/include
make
sudo make install
cd ~

Step 4: We generate a Certificate Authority key and certificate. We use the p521_falcon1024 hybrid signature scheme. To serve as a trusted authority in the system.

/usr/local/bin/openssl req -x509 -new -newkey p521_falcon1024 -keyout p521_falcon1024_CA.key -out p521_falcon1024_CA.crt -nodes -subj "/CN=oqstest CA" -days 365 -config /usr/local/ssl/openssl.cnf

```
root@ubuntu2:~# cd ..
root@ubuntu2:/# /usr/local/bin/openssl req -x509 -new -newkey p521_falcon1024 -keyout p521_falcon1024_CA.key -out p521_falcon1024_CA
.crt -nodes -subj "/CN=oqstest CA" -days 365 -config /usr/local/ssl/openssl.cnf
Generating a p521_falcon1024 private key
writing new private key to 'p521_falcon1024_CA.key'
```

Figure 5: Generating p521_falcon1024 private key

Step 5: We place the certificate in the directory where it is accessible and we add it as a trust anchor for the web browser. This ensures that the system trusts the certificate which is signed by the CA.

sudo cp p521_falcon1024_CA.crt /etc/ssl/certs
sudo trust anchor /etc/ssl/certs/p521_falcon1024_CA.crt

Step 6: we generate a Certificate Signing Request and Server's Hybrid Private Key. A hybrid private key and Certificate Signing Request (CSR) for the server are generated using the p521_falcon1024 hybrid signature scheme. The CSR is a request sent to the CA to obtain a digital certificate which includes the public key.

/usr/local/bin/openssl req -new -newkey p521_falcon1024 -keyout p521_falcon1024_srv.key -out p521_falcon1024_srv.csr -nodes -subj "/CN=localhost" -config /usr/local/ssl/openssl.cnf

Step 7: Generate Server's Hybrid Certificate and Sign it using the CA Hybrid Key. The server's CSR is used to generate a hybrid certificate for the server. This certificate is then signed by the CA's hybrid key, creating a trusted hybrid certificate for the server. The -days 365 parameter specifies the validity period of the certificate which has been set for 365 days.

/usr/local/bin/openssl x509 -req -in p521_falcon1024_srv.csr -out p521_falcon1024_srv.crt -CA p521_falcon1024_CA.crt -CAkey p521_falcon1024_CA.key -CAcreateserial -days 365

Step 8: We customise Apache configuration files (httpd.conf and httpd-ssl.conf) - these configure Apache to use only TLS 1.3 with 2 high-security cipher suites with forward secrecy and authenticated encryption with associated data (ECDHE with AES-256 in Galois Counter Mode and SHA-384 hashing, or ECDHE with ChaCha20 and Poly1305). These files also set Apache to announce the NIST Round 3 Level 5 hybrid handhake algorithms to clients.

cd Downloads
sudo mv httpd.conf /usr/local/apache2/conf/httpd.conf
sudo mv httpd-ssl.conf /usr/local/apache2/conf/extra/httpd-ssl.conf

Step 9: we move the generated server certificate and key to appropriate Apache folder from home working folder.

cd ~
sudo mv p521_falcon1024_srv.crt /usr/local/apache2/conf/p521_falcon1024_srv.crt
sudo mv p521_falcon1024_srv.key /usr/local/apache2/conf/p521_falcon1024_srv.key

Step 10: We create a simple flask application and write a code for app.py and index.html. We place the files in the apache directory to view the same.

C: > Users > Keshav > Desktop > app.py > ...

```python
from flask import Flask, render_template
from liboqs import OQS

app = Flask(__name__)

# call and initialize liboqs
oqs = OQS()
oqs.init()

@app.route('/')
def index():
    # Quantum-resistant key exchange mechanism (KEM) using liboqs.
    algorithm_name = 'p521_kyber1024'
    client_public_key, shared_secret = oqs.key_exchange(algorithm_name)

    # Use template for index.html
    return render_template('index.html',
                           algorithm_name=algorithm_name,
                           client_public_key=client_public_key.hex(),
                           shared_secret=shared_secret.hex())

if __name__ == '__main__':
    app.run(debug=True)
```

Figure 6: App.py

C: > Users > Keshav > Desktop > index.html > html > body > h1

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Quantum-Resistant Web App</title>
</head>
<body>
    <h1>Quantum-Resistant Web Service architecture App</h1>
    <p>Key Exchange Algorithm: {{ algorithm_name }}</p>
    <p>Client Public Key: {{ client_public_key }}</p>
    <p>Shared Secret: {{ shared_secret }}</p>
</body>
</html>
```

Figure 7: index.html

The provided Python script, app.py, represents a simple Flask web application incorporating the liboqs library for post-quantum cryptography. In this script, the Flask framework is utilized to create a web server, and liboqs is employed for quantum-resistant key exchange.

The template, index.html, structures a basic webpage with placeholders for displaying the chosen key exchange algorithm, the client's public key, and the shared secret. When web

server is launched, we visit the root URL (http://localhost:4433/) in a browser triggers the display of a webpage showing details about the quantum-resistant key exchange, including the algorithm used, the client's public key, and the shared secret.

Step 11: Now since normal browsers are not able to support Post quantum cryptography, we install epiphany browser.

sudo apt install build-essential clang meson gnome-pkg-tools libglib2.0-dev libproxy-dev gsettings-desktop-schemas-dev ca-certificates epiphany-browser -y

Step 12: Clone GNOME's glib-networking back-end, and configure it to use OQS OpenSSL instead of its built-in GNU TLS library

git clone --branch 2.60.4 https://gitlab.gnome.org/GNOME/glib-networking.git

cd glib-networking
mkdir build
cd build

env PKG_CONFIG_PATH=$HOME/local/lib/pkgconfig CPATH=$HOME/local/include LIBRARY_PATH=$HOME/local/lib meson --prefix=$HOME/local -Dopenssl=enabled -Dgnutls=disabled ..

env CPATH=$HOME/local/include ninja
ninja install
cd ~

Step 13: Start the apache server in order to connect with the local host.

sudo /usr/local/apache2/bin/apachectl -k start

Step 14: Next we configure epiphany browser, which will force the Epiphany browser to use the glib-networking back-end we built, as well as the p521_firesaber hybrid TLS handshake. We do this by creating a client OpenSSL configuration file and environment variables.

```
(
  echo 'openssl_conf = openssl_init'
  echo ''
  echo '[openssl_init]'
  echo 'ssl_conf = ssl_sect'
  echo ''
  echo '[ssl_sect]'
  echo 'system_default = system_default_sect'
  echo ''
  echo '[system_default_sect]'
  echo 'Groups = p521_firesaber'
) > openssl-client.cnf
```

Figure 8: OpenSSL client configuration file

export OPENSSL_CONF=$HOME/openssl-client.cnf

export LD_LIBRARY_PATH=$HOME/local/lib
export GIO_MODULE_DIR=$HOME/local/lib/x86_64-linux-gnu/gio/modules

We can check our browser to see if it is supporting post quantum cryptography, this can be done using the Test.openquantumsafe.org website. This site offers a range of algorithms to be tested. This gives us a positive reply and confirms that our epiphany browser is correctly set up.

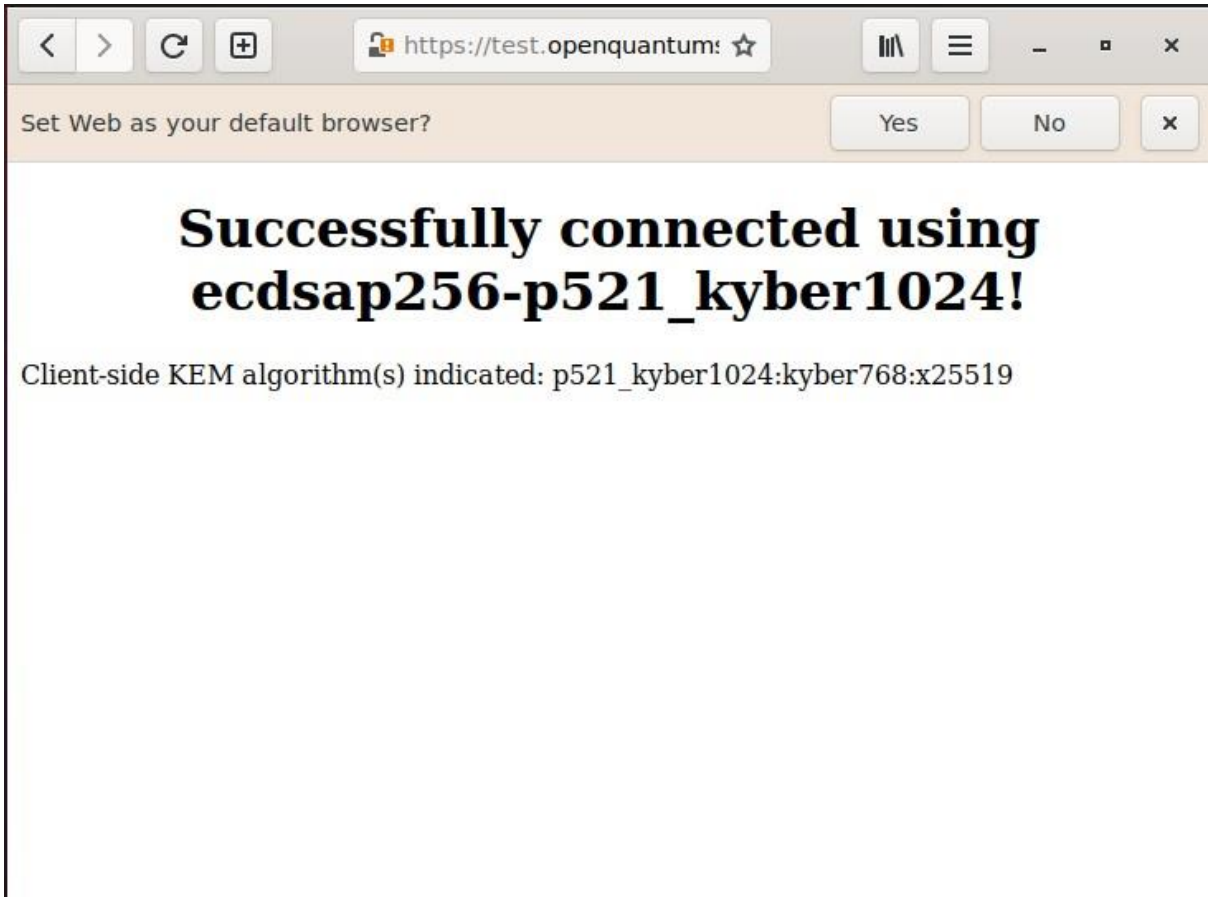

Figure 9: Test successful

Step 15: we start our browser and connect to local Apache server on port 4433

epiphany https://localhost:4433

This should deploy our epiphany browser where we can check if our website is working fine by https://localhost:4433.

References

https://github.com/open-quantum-safe/openssl/blob/OQS-OpenSSL_1_1_1-stable/README.md

https://github.com/open-quantum-safe/liboqs

https://src.fedoraproject.org/repo/pkgs/expat/expat-2.4.1.tar.gz/sha512/1f08861e9b766fdbbc40159404a3fe1a86451d635ef81874fa3492845eda83ac2dc6a0272525891d396b70c9a9254c2f6c907fe4abb2f8a533ccd3f52dae9d5a/expat-2.4.1.tar.gz

https://archive.apache.org/dist/httpd/httpd-2.4.51.tar.bz2

https://archive.apache.org/dist/apr/apr-1.7.0.tar.gz

https://archive.apache.org/dist/apr/apr-util-1.6.1.tar.gz

https://github.com/wolfssl/wolfssl

https://gitlab.gnome.org/GNOME/glib-networking.git