National College of Ireland

Configuration Manual

MSc Research Project
Msc Cyber Security

Sylesh Sathish Kumar
Student ID: x23408880

School of Computing
National College of Ireland

Supervisor: Eric Gyamfi

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Sylesh Sathish Kumar
Student ID:	x23408880
Programme:	Msc Cyber Security
Year:	2025
Module:	MSc Research Project
Supervisor:	Eric Gyamfi
Submission Due Date:	11/12/2025
Project Title:	Configuration Manual
Word Count:	715
Page Count:	7

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:	Sylesh Sathish Kumar
Date:	10th December 2025

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

Sylesh Sathish Kumar
x23408880

1 Introduction

This configuration manual explains the entire software, tools, Python libraries, and system dependencies to execute the Post-Quantum Cryptographic (PQC) IoT authentication framework, which was implemented in this study. The digital signature applied in the project is ML-dSA-44 with an IoT architecture comprising of Sensor Node, Gateway Relay and Verification Server. The given design guarantees the reproducibility of findings and the similarity of all experimental tests.

2 Python Version

This project was developed using: Python 3.12.3 (latest stable release)
This version is better in terms of performance and its ability to handle cryptographic libraries.
Official Release Notes: https://docs.python.org/3/whatsnew/3.12.html

3 Python Libraries Used

To facilitate the generation of data, compression of the same, networking, logging and to evaluate encrypted data, the following Python libraries were installed:

Library	**Purpose**
json	Managing IoT payloads and metadata
base64	Certificate encoding/decoding
hashlib	SHA-256 hashing for payloads
zlib	DEFLATE compression for data & certificate variants
time / datetime	Timestamping payloads and logging events
os / sys	Handling file operations and environment variables

Table 1: Python Libraries and Their Purposes

Library	**Purpose**
Flask	Gateway API and Verification Server
requests	Sending payloads between Sensor → Gateway → Server

Table 2: Libraries Used in Gateway and Communication

Library	Purpose
psutil	Measuring CPU usage, memory footprint, and energy cost
logging	Writing performance logs and experiment summaries
csv	Exporting experimental results to summary tables

Table 3: Libraries for Performance Measurement and Logging

Library	Purpose
matplotlib	Plotting experimental graphs
seaborn	Generating comparative and statistical plots

Table 4: Visualization Libraries Used in Experiments

4 Tools and Technologies

4.1 OpenSSL (for PQC and X.509 Certificates)

OpenSSL was used to generate ML-DSA-44 keys, sign payloads, validate certificates, and test X.509 functionalities. The primary commands used in the experiments are:

- `openssl genpkey` — Key generation
- `openssl pkeyutl -sign` — PQC signing
- `openssl pkeyutl -verify` — PQC verification
- `openssl req -new` — CSR generation
- `openssl x509` — Certificate creation and validation

4.2 Flask-based Microservices

The experimental setup uses three microservices implemented using Flask:

- **Sensor Node** — Sends payloads to the gateway.
- **Gateway** — Forwards data, logs metrics, and performs protocol-level checks.
- **Verification Server** — Validates signatures and verifies X.509 certificates.

4.3 Linux Environment

All experiments were executed in a Linux environment to ensure compatibility with OpenSSL, network utilities, and performance-measurement tools.

4.4 Visual Studio Code (Editor)

- Development of Python scripts
- Running gateway/server sensors
- Managing virtual environments
- Viewing logs and debugging

5 Implementation

This section describes the entire implementation process of the PQC-enabled IoT data pipeline.

5.1 Step 1: Certificate and Key Generation using OpenSSL

To enable post-quantum signatures (ML-DSA-44) and X.509 validation, all cryptographic material was generated using OpenSSL. The script `make_certs.sh` automates the process and produces:

- CA key and certificate
- Sensor ML-DSA-44 private key
- Sensor X.509 certificate signed by the CA

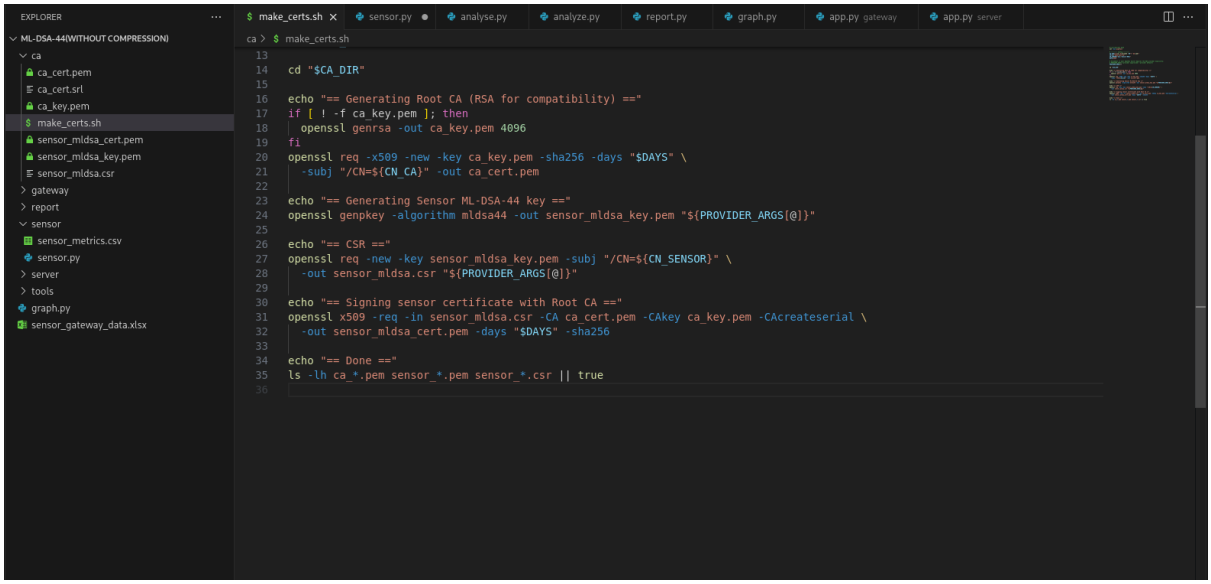


Figure 1: Certificate and Key Generation

5.2 Step 2: Sensor Node Implementation

The workflow in the system starts with the reading of simulated telemetry data, which consists of the temperature and humidity, and the formation of a canonical payload string based on the readings. The system then creates size-controlled messages in the form of JSON with a payload size ranging from 200 KB to 5000 KB, followed by hashing of the payload using a hash-based scheme with the 256-bit SHA-256 algorithm. The resulting hash is signed with the ML-DSA-44 algorithm using OpenSSL to ensure post-quantum secure authentication. After this, the system calculates the amount of energy consumed during hashing, signing, and radio transmission. All generated metrics are appended to a file named `sensor_metrics.csv`, and finally, the signed JSON message is sent to the Gateway for further processing.

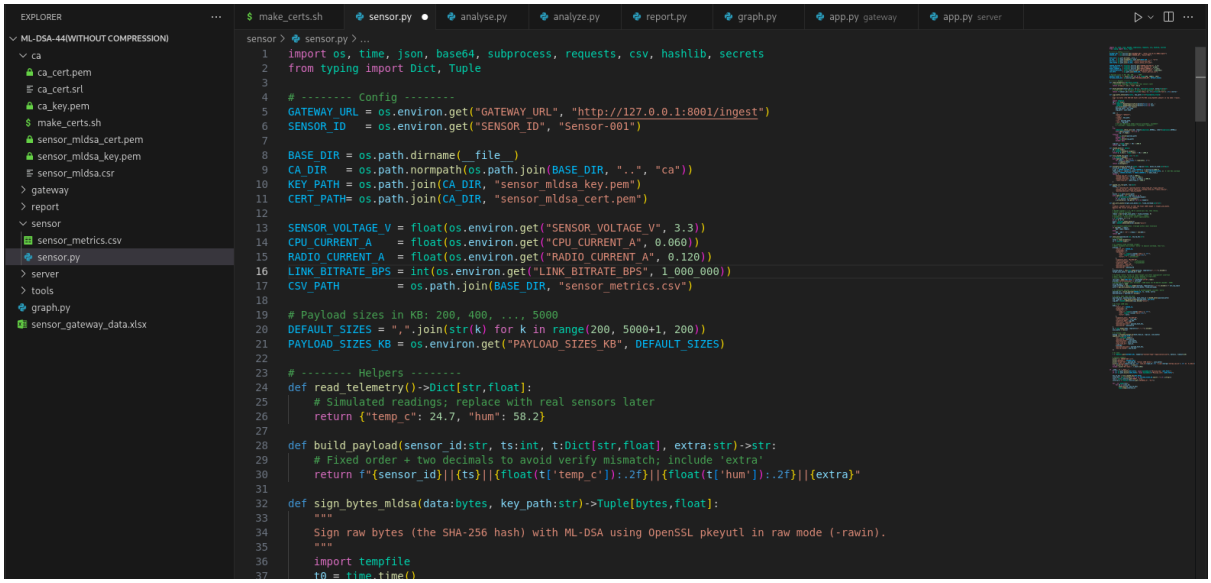


Figure 2: Sensor Node

5.3 Step 3: Gateway Relay Service

The implementation of the Gateway (app.py in the Gateway folder) carries out some of the most important tasks such as acceptance of incoming signed JSON messages by the Sensor and passing them to the Verification Server. It also measures the round trip time (RTT), gathers CPU and memory statistics by use of the psutil library and stores the entire information in gatewaymetrics.csv file.

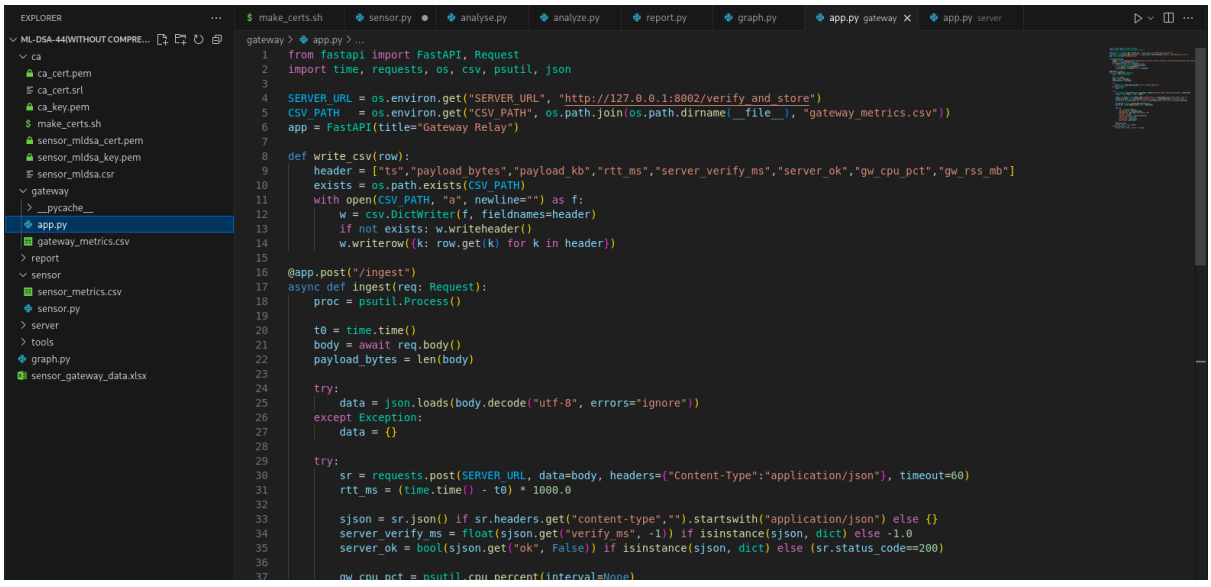


Figure 3: Gateway

5.4 Step 4: Verification Server

All the cryptographic verification processes are done by the Verification Server (app.py in the server directory). It authenticates the certificate of the Sensor first with the Certificate

Authority and derives the public key of the certificate. Then it re-calculates the canonical payload string it re-computes the SHA-256 hash and verifies the ML-DSA-44 signature with OpenSSL.

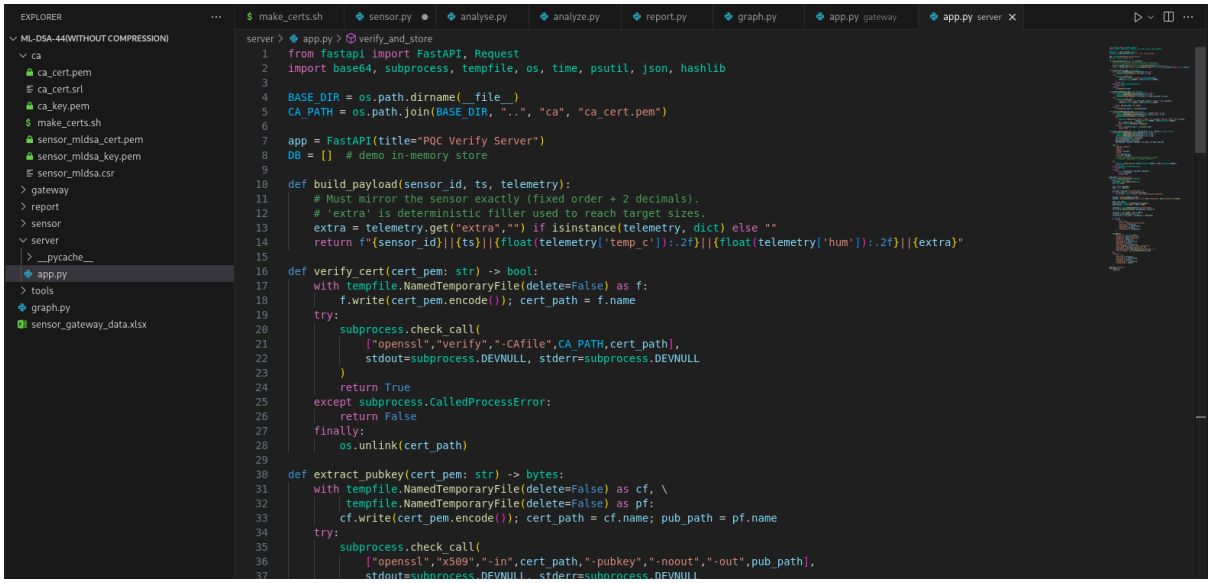


Figure 4: Verification Server

5.5 Step 5: Experimental Execution

The experiment is automated through environment variables:

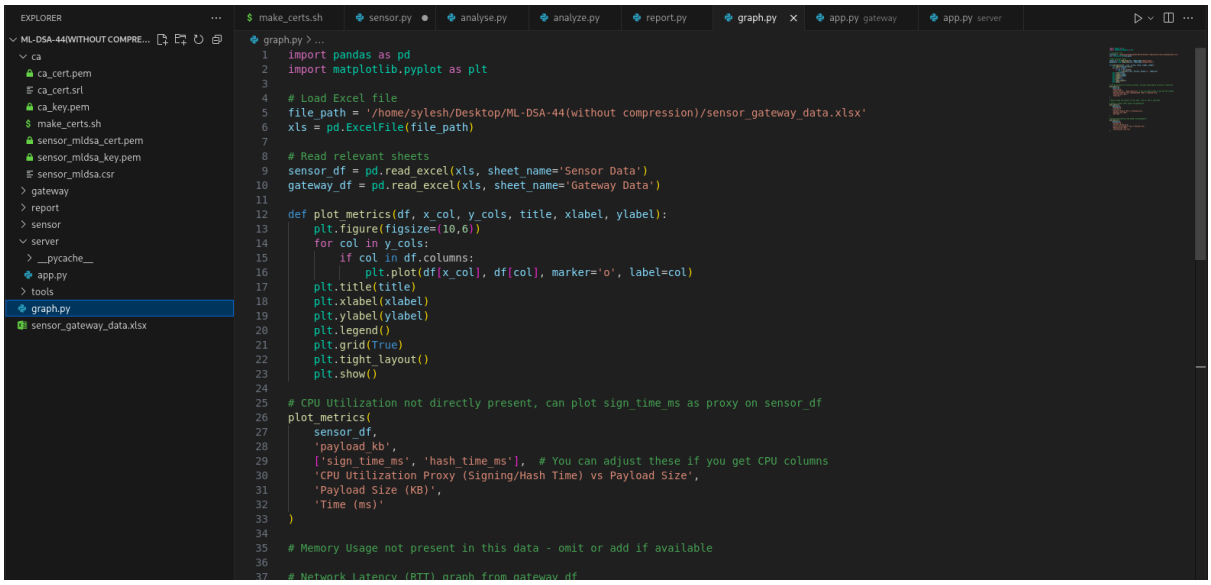
- payload sizes: 200 KB to 5000 KB,
- number of runs: 1–N,
- delay between transmissions,
- electrical model parameters (voltage, CPU/radio current, bitrate).

Each run records:

- message size,
- hashing and signing latency,
- energy consumption,
- network round-trip time,
- signature verification time,
- process memory usage.

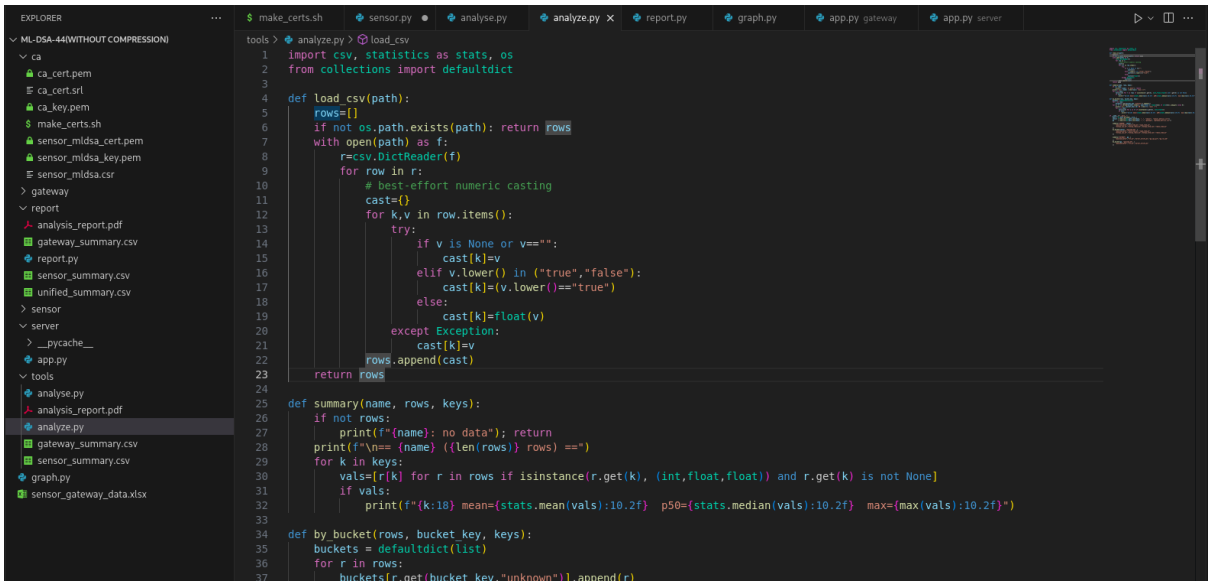
5.6 Step 6: Data Analysis and Aggregation

The scripts `analyze.py` and `report.py`, and `graph.py`



```python
import pandas as pd
import matplotlib.pyplot as plt

# Load Excel file
file_path = '/home/sylesh/Desktop/ML-DSA-44(without compression)/sensor_gateway_data.xlsx'
xls = pd.ExcelFile(file_path)

# Read relevant sheets
sensor_df = pd.read_excel(xls, sheet_name='Sensor Data')
gateway_df = pd.read_excel(xls, sheet_name='Gateway Data')

def plot_metrics(df, x_col, y_cols, title, xlabel, ylabel):
    plt.figure(figsize=(10,6))
    for col in y_cols:
        if col in df.columns:
            plt.plot(df[x_col], df[col], marker='o', label=col)
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()

# CPU Utilization not directly present, can plot sign_time_ms as proxy on sensor_df
plot_metrics(
    sensor_df,
    'payload_kb',
    ['sign_time_ms', 'hash_time_ms'],  # You can adjust these if you get CPU columns
    'CPU Utilization Proxy (Signing/Hash Time) vs Payload Size',
    'Payload Size (KB)',
    'Time (ms)'
)

# Memory Usage not present in this data - omit or add if available
```

Figure 5: Graph.py



```python
import csv, statistics as stats, os
from collections import defaultdict

def load_csv(path):
    rows=[]
    if not os.path.exists(path): return rows
    with open(path) as f:
        r=csv.DictReader(f)
        for row in r:
            # best-effort numeric casting
            cast={}
            for k,v in row.items():
                try:
                    if v is None or v=="":
                        cast[k]=v
                    elif v.lower() in ("true","false"):
                        cast[k]=(v.lower()=="true")
                    else:
                        cast[k]=float(v)
                except Exception:
                    cast[k]=v
            rows.append(cast)
    return rows

def summary(name, rows, keys):
    if not rows:
        print(f"{name}: no data"); return
    print(f"\n== {name} ({len(rows)} rows) ==")
    for k in keys:
        vals=[r[k] for r in rows if isinstance(r.get(k), (int,float,float)) and r.get(k) is not None]
        if vals:
            print(f"{k:18} mean={stats.mean(vals):10.2f}  p50={stats.median(vals):10.2f}  max={max(vals):10.2f}")

def by_bucket(rows, bucket_key, keys):
    buckets = defaultdict(list)
    for r in rows:
        buckets[r.get(bucket_key,"unknown")].append(r)
```

Figure 6: Analyse.py

6 Reference

References

[1] Python Software Foundation. (2024). *Python 3.12 Documentation*. Available at: `https://docs.python.org/3/`.

[2] OpenSSL Software Foundation. (2024). *OpenSSL Documentation*. Available at: https://www.openssl.org/docs/.

[3] Reitz, K. (2024). *Requests Documentation*. Available at: https://requests.readthedocs.io/.

[4] Rodola, G. (2024). *psutil Documentation*. Available at: https://psutil.readthedocs.io/.

[5] Hunter, J. et al. (2024). *Matplotlib Documentation*. Available at: https://matplotlib.org/stable/.

[6] Microsoft. (2024). *Visual Studio Code Documentation*. Available at: https://code.visualstudio.com/docs.

[7] Linux Project. (2024). *Linux Manual Pages*. Available at: https://man7.org/linux/man-pages/.