**Name:** Dang Khoa Le

**Student ID:** 103844421

**Desired Grade**: Up to D

**Lab Time:**  Tuesday 10.30 AM

**Portfolio Task – Project**

**1. Introduction and Motivation**

The IoT-Based Elderly Health Monitoring System (IEHMS) project provides a secure solution for monitoring health data, ensuring safety and well-being of elderly individuals. The project builds upon TNE20003’s IoT solution and integrates TNE30024’s requirements by using the Mosquitto broker secured via TLS, digital certificates, and a user-friendly GUI. This project builds upon the **TNE20003** IoT-based solution, expanding the system to meet the requirements of the **TNE30024** unit by incorporating enhanced security through TLS, certificate chain configuration, and a user-friendly GUI. The project simulates the use of MQTT communication between sensor devices, users, and clients, leveraging a secure Mosquitto broker for message handling. This system ensures that critical health data (wake-up times and heart rates) from User and health advices from Client (suggested medicine and exercise intake) are securely monitored and shared between authorized users on a daily basis.

**2. Project Overview**

IEHMS aims to ensure the comfort and safety of elderly individuals by monitoring environmental conditions (temperature, humidity, air quality) and personal health data (heart rate and wake-up time). The system allows caregivers to access this information remotely, providing real-time data that is crucial for the elderly's health management.

Key components of the system include:

* **MQTT Message Broker (Mosquitto)**: Facilitates communication between devices and users using secure TLS-based encryption.
* **Sensor Devices (Device 1, 2, and 3)**: These devices generate and publish environmental data such as temperature, humidity, and air quality.
* **User.py**: Acts as the elderly user’s interface, allowing them to post their wake-up time and heart rate and receive data from sensors and the caregiving client.
* **Client.py**: Allows caregivers to monitor the elderly’s health data, keep track on sensors’ data providing them with real-time notifications and alerts, alongside with posting health advices (suggested medicine and exercise intake).
* **GUI.py**: Utilises the graphical MQTT client, providing a user-friendly interface for monitoring and posting health data for the elderly user.

**3. Configuration and Setup**

**3.1. Mosquitto Server Configuration**

To ensure secure communication between devices, the Mosquitto broker was reconfigured to support **TLS encryption**. Each grade level also has its specific configurations in the mosquitto.conf file. Here is a breakdown:

For the **Pass level**, a self-signed certificate is sufficient for securing the Mosquitto server.

* cafile and certfile point to the self-signed server certificate.
* require\_certificate false: Not requiring client certificates as the self-signed certificate is used for basic encryption only.

For the **Credit level**, the Mosquitto server certificate is signed by a Root CA, enhancing security by using a CA-signed certificate.

* cafile and certfile are both set to the Root CA-signed server certificate.
* require\_certificate false: Not enforcing client certificates but using CA-signed server certificates for increased trust.

For the **Distinction level**, a full certificate chain is set up, ensuring that the entire chain of trust (Root CA -> Intermediate CA -> Server Certificate) is established.

* cafile is set to rootCA.crt, establishing the highest level of trust by requiring the Root CA in the certificate chain.
* certfile is set to fullchain.crt, which includes the server, intermediate, and root certificates, ensuring full chain validation.

A computer screen shot of a computer code

Description automatically generatedThe following configuration was applied in **mosquitto.conf**:

**Notice: Please make sure you comment-out the task-section not run (if you are running D task, then P and C task have to be commented out by a ‘#’ at the beginning).**

**A white screen with black text

Description automatically generated3.2. Access Control Configuration (aclfile)**

he **aclfile** sets permissions for users, controlling which MQTT topics they can access:

* **General Client Access**:
  + pattern readwrite %u/#: Each user can read and write to topics under their own username path (e.g., username/#).
  + pattern readwrite public/#: All users have read and write access to the public/# topic, allowing shared communication.
* **Admin Access (default)**:
  + user admin: The "admin" user has full read and write access to all topics, this feature is provided at default.
* **Specific User Access (User 'khoa')**:
  + user khoa: The user "khoa" has read and write access only to the 103844421/# topic, restricting access to relevant data.

This setup secures data by isolating user access, enabling public interactions, and providing private oversight.

**3.3. Password File (pwfile):**

Create a new file **pwfile** containing the username and password pair in the following format: username:password.

echo "khoa:103844421" > /usr/local/etc/mosquitto/pwfile

Use the mosquitto\_passwd command to hash the passwords in the file.

mosquitto\_passwd -U /usr/local/etc/mosquitto/pwfile

**3.4. Mosquitto Server (rc.conf):** The following line was added to the end of the /etc/rc.conf file, to enable the mosquitto server and allowing the server to start/stop:

mosquitto\_enable=”YES”

**3.5. Certificate and Configuration Files Location:**

This setup serves two main purposes:

1. **Centralized Certificate Management on Rule Host**: By storing all certificates and configuration files in /usr/local/etc/mosquitto/, with tasks devided to ‘**pass’**, ‘**credit’**, and ‘**distinction’** directories. the Rule Host has a central repository for managing security configurations and certificates. This organization simplifies maintaining, updating, and accessing these files, ensuring they are available to the Mosquitto broker for secure communications.
2. **Local Access for Python Scripts**: Copying the certificates to the /crt/ directory on the local machine with tasks divided into ‘**pass’**, ‘**credit’**, and ‘**distinction’** sub-folders, allows the Python scripts to reference and use these certificates directly for TLS authentication. This setup enables the scripts to establish secure, encrypted connections with the Mosquitto server by verifying the server’s identity through the certificate chain. It provides seamless access to necessary security resources for the Python MQTT client applications.
3. **Centralized Configuration Files Management on Rule Host: mosquito.conf**, **acflfile** and **pwfile** locate on usr/local/etc/mosquitto/ directory according to the task requirement. **rc.conf** locates on /etc/ directory.
4. **Local Access for Configuration Files: mosquito.conf**, **acflfile,** **pwfile**, and **rc.conf** are copied to /config/ local directory and sent within this submission (in the zip file).

**3.6. MQTT TLS Setup in Python:**

All Python programs were updated to use **TLS** for secure communication with the Mosquitto server. Each client program includes the following configuration:

In this project, all Python programs were updated to use TLS for secure communication with the Mosquitto server. Each grade level (Pass, Credit, Distinction) requires a slightly different configuration based on the TLS setup requirements:

**a. Pass Level**

For Pass level, the setup uses a self-signed certificate for basic TLS communication.

* **TLS Parameters**

`ca\_certs="crt/pass/server.crt"`: Specifies the path to the self-signed certificate for server verification.

`tls\_version=ssl.PROTOCOL\_TLSv1\_2`: Ensures that the connection uses TLS v1.2 for secure encryption.

* **Certificate Validation**

tls\_insecure\_set(True)`: Server is using a **self-signed** certificate, which the client does not have a trusted root authority to verify it against, which means the client cannot verify the authenticity of the certificate. Setting client.tls\_insecure\_set(True) here allows the client to accept the self-signed certificate without (mutual) verification.

**b. Credit Level**

For Credit level, the setup includes a Root CA, with the Mosquitto server certificate signed by this CA.

* **TLS Parameters**

`ca\_certs="crt/credit/rootCA.crt"`: Specifies the Root CA certificate path, allowing the client to verify the server’s CA-signed certificate.

`tls\_version=ssl.PROTOCOL\_TLSv1\_2`: Enforces the use of TLS v1.2 for encryption.

* **Certificate Validation**

tls\_insecure\_set(True)`: **Root Certificate Authority (CA)** is introduced, and the Mosquitto server certificate is signed by this CA. However, there is still no full certificate chain (with an intermediate authority). Setting client.tls\_insecure\_set(True) here allows the client to accept the self-signed certificate without (mutual) verification.

**c. Distinction Level**

For Distinction level, a full certificate chain is used, including Root CA, Intermediate CA, and a server certificate signed by the Intermediate CA.

* **TLS Parameters**

`ca\_certs="crt/distinction/rootCA.crt"`: Specifies the Root CA certificate, which is part of a full certificate chain (Root CA -> Intermediate CA -> Server Certificate). The client validates the chain to ensure trustworthiness.

`tls\_version=ssl.PROTOCOL\_TLSv1\_2`: Ensures that TLS v1.2 is used for secure communication.

* **Certificate Validation**

tls\_insecure\_set(False)`: Enforces strict certificate validation through the full certificate chain.

**Notice: Please make sure you comment-out the task-section not run (if you are running D task, then P and C task have to be commented out by a ‘#’ at the beginning).**

**3.7. Certificate Chain and Requirements for IEHMS**

This section outlines the certificates required at each level—Pass, Credit, and Distinction—to secure the Mosquitto server and MQTT communication.

**a. Pass**

For the Pass level, a self-signed server certificate is used. This provides basic TLS encryption without requiring a CA chain.

* **server.crt**
  + Description: Self-signed certificate for the Mosquitto server.
  + Location: /usr/local/etc/mosquitto/pass/server.crt
* **server.key**
  + Description: The private key associated with the self-signed server certificate.
  + Location: /usr/local/etc/mosquitto/pass/server.key

**b. Credit**

For the Credit level, a Root CA is used to sign the Mosquitto server certificate, adding an additional level of trust compared to the self-signed certificate.

* **rootCA.crt**
  + Description: Root CA certificate used to sign the Mosquitto server certificate.
  + Location: /usr/local/etc/mosquitto/credit/rootCA.crt
* **rootCA.key**
  + Description: Private key for the Root CA.
  + Location: /usr/local/etc/mosquitto/credit/rootCA.key
* **rootCA.srl**
  + Description: Serial number file for the Root CA, used to track certificates issued by the CA.
  + Location: /usr/local/etc/mosquitto/credit/rootCA.srl
* **server.crt**
  + Description: The Mosquitto server certificate signed by the Root CA.
  + Location: /usr/local/etc/mosquitto/credit/server.crt
* **server.key**
  + Description: The private key for the Mosquitto server.
  + Location: /usr/local/etc/mosquitto/credit/server.key
* **server.csr**
  + Description: Certificate signing request for the Mosquitto server, used to obtain the CA-signed server certificate.
  + Location: /usr/local/etc/mosquitto/credit/server.csr

**c. Distinction**

For the Distinction level, a complete certificate chain is required, involving a Root CA, Intermediate CA, and a server certificate signed by the Intermediate CA.

* **rootCA.crt**
  + Description: The root certificate used to sign the intermediate CA certificate.
  + Location: /usr/local/etc/mosquitto/distinction/rootCA.crt
* **rootCA.key**
  + Description: The private key for the root certificate authority.
  + Location: /usr/local/etc/mosquitto/distinction/rootCA.key
* **rootCA.srl**
  + Description: Serial number file for the root CA, used to keep track of certificates issued by the root CA.
  + Location: /usr/local/etc/mosquitto/distinction/rootCA.srl
* **intermediateCA.crt**
  + Description: Intermediate certificate used to sign the Mosquitto server certificate.
  + Location: /usr/local/etc/mosquitto/distinction/intermediateCA.crt
* **intermediateCA.key**
  + Description: Private key for the intermediate CA.
  + Location: /usr/local/etc/mosquitto/distinction/intermediateCA.key
* **intermediateCA.srl**
  + Description: Serial number file for the intermediate CA, used to track certificates issued by the intermediate CA.
  + Location: /usr/local/etc/mosquitto/distinction/intermediateCA.srl
* **server.crt**
  + Description: The Mosquitto server certificate signed by the intermediate CA.
  + Location: /usr/local/etc/mosquitto/distinction/server.crt
* **server.key**
  + Description: The private key for the Mosquitto server.
  + Location: /usr/local/etc/mosquitto/distinction/server.key
* **server.csr**
  + Description: Certificate signing request for the Mosquitto server, used to obtain the signed server certificate.
  + Location: /usr/local/etc/mosquitto/distinction/server.csr
* **fullchain.crt**
  + Description: A concatenated certificate chain containing the server, intermediate, and root certificates, used by Mosquitto for full trust chain validation.
  + Location: /usr/local/etc/mosquitto/distinction/fullchain.crt

**3.8. Summary - List of Modified Files:**

1. **mosquitto.conf:**
   * Configured the listener for port 8883 for secure TLS communications.
   * Specified the certificate locations and TLS version.
   * Set up access control rules using the ACL file and password file.
   * Enabled logging for debugging purposes.
2. **aclfile:**
   * Configured access rules allowing specific users to publish and subscribe to certain topics.
   * Rules enforce privacy for private topics and grant access to public topics.
3. **pwfile:**
   * Create username and hashed password.
4. **rc.conf:**
   * Enable mosquitto server.
5. All **Python Programs** from TNE20003 (Client.py, User.py, Device1.py, Device2.py, Device3.py, GUI.py):
   * Configured each program to establish a secure connection with the Mosquitto broker using the TLS certificates.
   * Implemented the tls\_set function with the correct paths to the certificate chain (rootCA.crt).
   * Enforced TLS version 1.2 and certificate validation.
   * Added user authentication using the password file (khoa, 103844421).

**4. Python Program Descriptions**

**4.1. Client.py (Caregiver Interface)**

This program acts as the caregiver interface, allowing the user to either subscribe to private topics (monitoring) or publish messages to specific health topics:

* **Topics**: "103844421/medicine", "103844421/exercise", "public/temperature".
* **Random Message Generation**: The client publishes randomized medicine and exercise advice to the elderly.
* **TLS-based Communication**: Ensures secure transmission of data between the client and the Mosquitto server.

**4.2. User.py (Elderly Interface)**

The User.py program allows the elderly to publish their health status (wake-up time, heart rate) and subscribe to messages from the client or devices:

* **Health Data**: Publishes wake-up time and heart rate to private topics such as "103844421/wake\_up\_time", "103844421/heart\_rate".
* **TLS Security**: Secures all communications with the broker using TLS and certificates.

**4.3. Device1.py, Device2.py, Device3.py**

Each device program generates fake data and publishes environmental data (temperature, humidity, air quality) to respective public and private topics:

* **Device1.py**: Publishes temperature data.
* **Device2.py**: Publishes humidity data.
* **Device3.py**: Publishes air quality and pollution rate.

These device programs are designed to simulate sensor data in an IoT environment.

**4.4. GUI.py (Graphical Client Interface)**

The GUI.py replaces the graphical MQTT client with a Python-based user interface that allows the elderly users to:

* **Monitor Data**: Subscribe to any topic and display the incoming data in a text box.
* **Post Data**: Publish wake-up time and heart rate to the broker, mimicking the user functionality. This is a new feature from TNE20003 project, which was previously only allow user monitoring data.

The GUI serves as an intuitive interface for both elderly users and caregivers to interact with the MQTT system securely.

**5. Achievements and Verification**

The project successfully meets the following requirements for the **Distinction** grade:

1. **Self-signed Certificates**: The Mosquitto server is secured using a certificate chain.
2. **TLS Communication**: All Python programs securely connect to the broker using TLS.
3. **Multiple Topic Subscriptions**: Both Client.py and User.py support subscribing to multiple topics.
4. **Message Generation**: The Client program generates multiple messages (medicine and exercise) and publishes them securely.
5. **Graphical Interface**: A GUI interface was developed to replace the Graphical MQTT Client.

**6. Instructions**

To run the programs:

1. Ensure the Mosquitto server is running with TLS enabled.

/usr/local/etc/rc.d/mosquitto start # Start Mosquitto server

/usr/local/etc/rc.d/mosquitto stop # Stop Mosquitto server

/usr/local/etc/rc.d/mosquitto restart # Restart Mosquitto server

1. Install the required (third-party) libraries: paho-mqtt, tkinter (for GUI).

pip install paho-mqtt tkinter

1. Run the programs as follows:
   * **Client.py**:

python3 Client.py <topic> # User to subscribe topics

python3 Client.py # User acting as a device posting health status

* + **User.py**:

python3 User.py <topic> # User to subscribe topics

python3 User.py # User acting as a device posting health status

* + **Devices**:

python3 Device1.py # Device 1 (Temperature)

python3 Device2.py # Device 2 (Humidity)

python3 Device3.py # Device 3 (Air Quality)

* + **GUI.py**:

**A screenshot of a phone

Description automatically generated**A screenshot of a phone

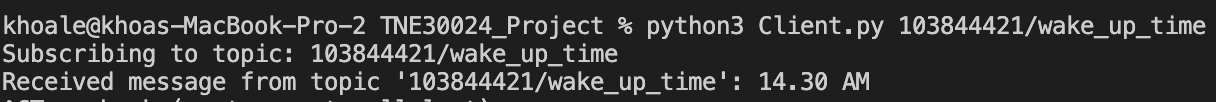
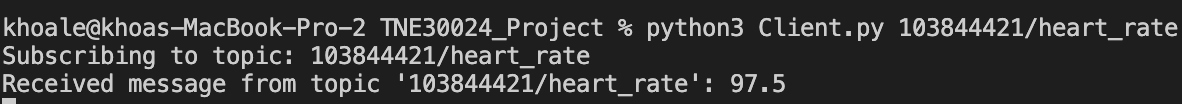
Description automatically generated****python3 GUI.py # Launch the graphical interface

Figure. Image shows caregiver client subscribe and receives wake\_up\_time and heart\_rate data from the elderly user.

Figure. Image show GUI monitoring topics.

Figure. GUI shows elderly user post their health status.

**7. List of topics**

This is the list of public (as public base-topic) and private (as 103844421 base-topic) topics that can be processed between, user, client and the 3 devices:

**Publish temperature data (Device 1):**

public/temperature/inside

public/temperature/outside

103844421/temperature/inside

103844421/temperature/outside

**Publish humidity data (Device 2):**

public/humid\_rate

103844421/humid\_rate

**Publish air quality and pollution data (Device 3):**

public/air\_quality

public/polution\_rate

103844421/air\_quality

103844421/polution\_rate

**Publish patient (user) health status data (User):**

103844421/wake\_up\_time

103844421/heart\_rate

**Publish doctor/caregiver notice (client) data (Client):**

103844421/medicine

103844421/exercise

**8. Conclusion**

This project has successfully implemented a **secure IoT-based Elderly Health Monitoring System** using MQTT over TLS, certificate chains, and multiple client and device interactions. It satisfies up to the **Distinction** grade requirements by supporting secure communication, certificate chains, multiple topic subscriptions, and a custom GUI for monitoring and posting health data.