How to run the code guidelines using the online MQTT broker:

1. Install Docker Desktop.
2. Build the hub image and run it:
   1. Open the command line.
   2. Navigate to the hub subfolder within the Docker folder.
   3. Run the command **docker build -t mqtt-hub-test .**
   4. Run the hub container **docker run -i -t mqtt-hub-test**
3. Build the lock image and run it:
   1. Open the command line.
   2. Navigate to the lock subfolder within the Docker folder.
   3. Run the command **docker build -t mqtt-lock-test .**
   4. Run the lock container **docker run -i -t mqtt-lock-test**
4. When the hub container is run for the first time, the user must set up their custom wake up word.
5. Upon the successful custom word setup, the user must provide the word they have set up.
6. When the wake up word is entered, the user can lock the door, unlock it and exit the controller hub.
7. When the command to lock/unlock the door is entered, the command prompt window for the lock will be updated to show the lock has received the message from the controller hub.  
   Upon the successful request execution, the lock will respond to the controller hub and a message will be presented in the controller hub advising the user of the current door status.  
   The user will then be presented with the option to lock, unlock the door again or to exit the hub.

[Optional] How to install and configure the Mosquitto MQTT server locally on a Linux system.

This demo requires the use of an MQTT broker server. We have deployed a test server hosted by one of our team members. If for any reason this server is not available or accessible, you can use a local version instead by following these instructions:

1. Install mosquitto server: sudo apt-get install mosquitto
2. Install mosquitto clients: sudo apt install mosquitto-clients
3. Create a new mosquitto server config file: sudo nano /etc/mosquitto/conf.d/my.conf
   1. Paste in the following:

allow\_anonymous false

password\_file /etc/mosquitto/passwd

* 1. Save and close the file

1. Edit the password file: sudo nano /etc/mosquitto/passwd
   1. Paste in the following:

hub:j6i3dd009153ef76

lock:k3dd651mniofd90q

* 1. Save and close the file

1. Encrypt the passwords: sudo mosquitto\_passwd -U /etc/mosquitto/passwd
2. Restart mosquitto: sudo systemctl restart mosquitto
3. Modify lock.py to point to localhost by replacing “mqtt.aliahmed.app” with “0.0.0.0” on line 79.
4. Modify hub.py to point to localhost by replacing “mqtt.aliahmed.app” with “0.0.0.0” on line 143.

Implementation discussion

* The code for the controller hub and the smart lock has been written using Python and Object-Oriented programming principles.
* The connection between the devices uses a lightweight network layer protocol MQTT, designed specifically with the IoT and its restrictions in mind (Sardeshmukh & Ambawade, 2017).
* The online MQTT broker used by our prototype is based on the open source EPL/EDL licensed Eclipse Mosquitto Message Broker (Mosquitto, 2022). It is hosted online using ISO 27001:2013 standard compliant company Scaleway (2022). Root access to the server is secured via a pair of SSH keys.
* The MQTT broker contains a list of ‘topics’. The consumer subscribes to the topic and reads messages from it whilst the producer delivers messages to a chosen topic (Yassein et. al., 2018). Our implementation uses two topics:
  + Lock - messages sent to the lock by the controller hub.
  + Lock\_status - current lock status messages sent by the smart lock.
* Only authorised devices are permitted to connect to the MQTT broker; the authorised users have been added to the broker’s config file. When connecting, the hub and the lock pass the credentials to the broker who then checks it and authorises the access.
* To address common problems of distributed systems (i.e., lost messages, reliability), a multi-threaded producer-consumer model for both the smart lock and the hub have been implemented (Brownlee, 2022).

Each device uses a producer thread to listen for messages from distributed systems (i.e., the MQTT broker) and places the messages into a queue. The consumer thread then reads the messages from the queue and takes appropriate actions.

* Python’s encryption library Fernet was used to encrypt the custom wake word set up by the user; Fernet uses symmetric cryptography and AES and CBC 128-bit key for encryption (Cryptography, 2022).
* The MQTT Quality of Service (QoS) has been set to 2: the message will always be delivered at least once (IBM, 2022). Whilst it is the slowest delivery method, due to the sensitive nature of the smart locks and the necessity of reliability, we believe the highest QoS level should be used.
* Whilst there are other more lightweight protocols such as COAP, the MQTT provides a better reliability and power consumption balance in case of smart locks (Safaei et al, 2017).
* Both hub and lock have been containerised using Docker to address common challenges such as minimal hardware resources and lack of power/memory resources. Containers require minimalistic configuration (Tozzi, 2018), allow deployment of new software by running a single command line (Mason, 2017) and it provides scalability by allowing several containers within a single host (Arora, 2022).

Due to time and resource constraints, some vulnerabilities identified in the design document could not be addressed:

* The smart home prototype delivered has been limited to the features necessary to demonstrate the main functionality. As such, this build does not include the removal of the user data partition during the factory reset, CAPTCHA and the hardcoded DNS server.
* Some encryption has been included in this build (encryption of the wake word), however, additional encryption of data in transit should be implemented within the future releases.
* Machine Learning to identify attacks and sandboxing for zero-day vulnerability detection is out of scope of this project due to its complexity.

The work completed could be further improved in terms of security during the future iterations as follows:

* MQTT transfers username and password in clear text (Sardeshmukh & Ambawade, 2017). TLS should be implemented on the MQTT server to authenticate and encrypt data (Lesjak et. al., 2015).
* A new, more secure, MQTT version SMQTT is being developed (Singh et. al., 2015). The viability of it for this project should be investigated in future.

**References**

Arora, S. (2022) What Is and What Are the Benefits of Docker Container? Available from: [https://www.simplilearn.com/tutorials/docker-tutorial/what-is-docker-container](https://www.simplilearn.com/tutorials/docker-tutorial/what-is-docker-container#the_benefits_of_docker_wrapping_it_all_up) [Accessed 8 September 2022].

Brownlee, J. (2022) Thread Producer-Consumer Pattern in Python. Available at: <https://superfastpython.com/thread-producer-consumer-pattern-in-python/> [Accessed 7 September 2022].

Cryptography. (2022) Fernet (symmetric encryption). Available from: <https://cryptography.io/en/latest/fernet/> [Accessed 8 September 2022].

IBM. (2022) Qualities of service provided by an MQTT client. Available from: <https://www.ibm.com/docs/en/ibm-mq/8.0?topic=concepts-qualities-service-provided-by-mqtt-client> [Accessed 8 September 2022].

Lesjak, C. et. al. (2015) ‘Securing smart maintenance services: Hardware-security and TLS for MQTT’, 2015 IEEE 13th International Conference on Industrial Informatics (INDIN). Cambridge, UK, 22-24 July. New York: IEEE. 1243-1250.

Mason, D. (2017) Docker and why you should run containers on your home automation hub. Available from: <https://danielbkr.net/docker-and-its-place-in-your-home-autmation-setup/> [Accessed 8 September 2022].

Mosquitto (2022) Eclipse Mosquitto: An open source MQTT broker. Available from: <https://mosquitto.org/> [Accessed 10 September 2022].

Sardeshmukh, H. & Ambawade, D. (2017) ‘Internet of Things: Existing Protocols and Technological Challenges in Security’, 2017 *International Conference on Intelligent Computing and Control.* Coimbatore, India, 23-24 June. New York: IEEE. 1-7.

Safaei, B., Monazzah, A.M.H., Bafroei, M.B. and Ejlali, A. (2017) Reliability side-effects in Internet of Things application layer protocols.

Scaleway (2022) Security and resilience. Available from: <https://www.scaleway.com/en/security-and-resilience/> [Accessed on 10 September 2022].

Singh, M., Rajan, M. A., Shivraj, V. L., Balamuralidhar, P. (2015) ‘Secure MQTT for Internet of Things (IoT)’, 2015 Fifth International Conference on Communication Systems and Network Technologies. Gwalior, India, 4-6 April. New York: IEEE. 746-751.

Tozzi, C. (2018) Why the Internet of Things Needs Docker Containers. Available from: <https://containerjournal.com/features/why-the-internet-of-things-needs-docker-containers/> [Accessed 8 September 2022].

Yassein M. B., Shatnawi, M. Q. Aljwarneh, S. & Al-Hatmi, R. (2017) ‘Internet of Things: Survey and open issues of MQTT protocol’, 2017 International Conference on Engineering & MIS (ICEMIS). Monastir, Tunisia, 8-10 May. New York: IEEE. 1-6.