Åbo Akademi University

System Architecture of IoT

Lab Report

Luis Araújo(2004624)

May 5, 2021

Contents

Chapter 1

Introduction

The following pages serve the purpose of demonstrating the knowledge obtain throughout the classes and the most important the lab section. First, in each section of the report, it's explained the purpose and the goal of each lab, following by how it was executed. From a brief overview of the document we can find out three main sections:

1. Lab 1: Get a working understanding of the MQTT protocol
2. Lab 2: Establish a secure MQTT connection and understand various parts of the setup
3. Lab 3: Establish an MQTT connection that is safe from (unintended) snooping and is private using a gateway

Finalizing this work, on the last page of the report, a conclusion can be found, where it is described what have I learned, what has surprised me and what challenges have I discovered from each lab.

Chapter 2

Lab 1: Get a working understanding of the MQTT protocol

2.1 Purpose and Goal

For this first lab, we had to get familiar with the workings of the MQTT protocol. So, after setting up the computer, we began by creating a device to measure the room temperature and transmit it to a remote server using MQTT protocol, using Arduino MKR WiFi1010 board.

2.2 Hardware and Software setup

We started by testing if the hardware and software were well configured, by making the yellow LED on the board blink.

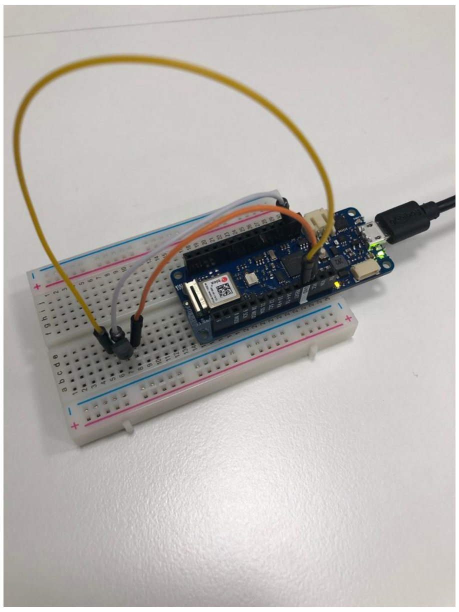


Figure 2.1: Arduino Board On

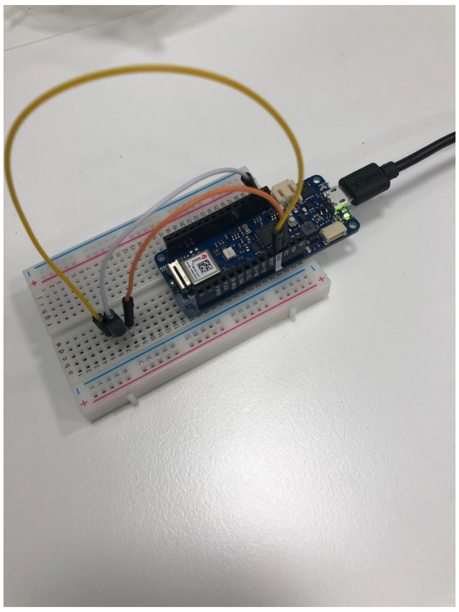
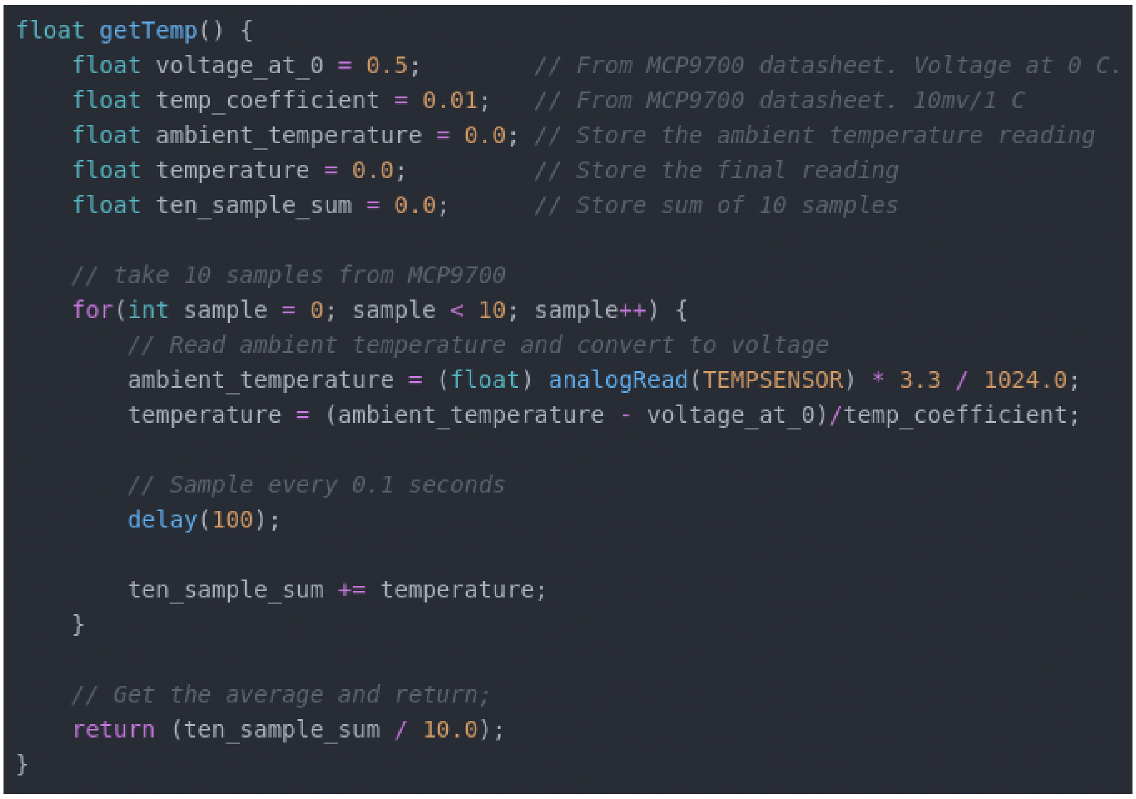


Figure 2.2: Arduino Board Off

2.3 Arduino + Temperature Sensor

After making the verification of the hardware and software, we began to try to obtain the temperature of the room through a sensor with the Arduino board.
The first step was to understand the code behind, especially the **getTemp** function:



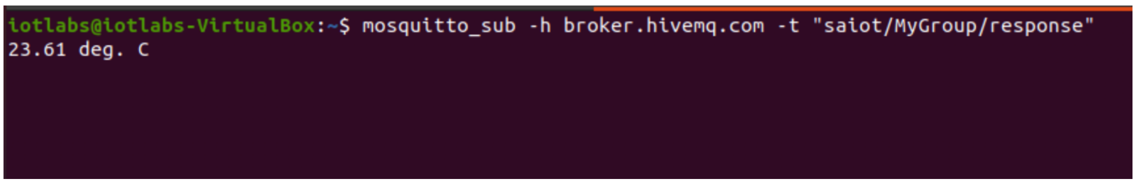
```
float getTemp() {
    float voltage_at_0 = 0.5;        // From MCP9700 datasheet. Voltage at 0 C.
    float temp_coefficient = 0.01;   // From MCP9700 datasheet. 10mv/1 C
    float ambient_temperature = 0.0; // Store the ambient temperature reading
    float temperature = 0.0;         // Store the final reading
    float ten_sample_sum = 0.0;      // Store sum of 10 samples

    // take 10 samples from MCP9700
    for(int sample = 0; sample < 10; sample++) {
        // Read ambient temperature and convert to voltage
        ambient_temperature = (float) analogRead(TEMPSENSOR) * 3.3 / 1024.0;
        temperature = (ambient_temperature - voltage_at_0)/temp_coefficient;

        // Sample every 0.1 seconds
        delay(100);

        ten_sample_sum += temperature;
    }

    // Get the average and return;
    return (ten_sample_sum / 10.0);
}
```

Figure 2.3: getTemp function

Question 1.1) Read and understand the code. Specifically understand the getTemp function. Explain its working in your report.

From the **getTemp** function we find out the average of the sum of 10 different samples of the temperature of the room, this average is done to get a more reliably measure. Each measurement is taken from the **analogRead** function that gives the ambient temperature, and after, it's converted to a voltage. Before storing this converted value, the system applies a delay of 0.1 seconds so that the next measure is not the same as the one just taken.

Once understood the **getTemp** function we continued by compiling and uploading it to the board. After we can verify the state of the Arduino board by connecting to the Monitor, we can inspect this message by subscribing on "broker.hivemq.com" with the code mosquitto_sub -h broker.hivemq.com -t "saiot/MyGroup/response" we acquire the following measurement of the temperature.



```
iotlabs@iotlabs-VirtualBox:~$ mosquitto_sub -h broker.hivemq.com -t "saiot/MyGroup/response"
23.61 deg. C
```

Figure 2.4: First measurement of the temperature

2.4 Command and Reponse

The code also has a subscriber component built-in, which can be called when the device receives a message on a specific topic, with the following command mosquitto_pub -h broker.hivemq.com -t "commandTopic" -m "command".
This call uses the function **onMqttMessage**, which we are going to take a closer look at.

Question 1.2) What is the value returned by mqttClient.messageQoS()? What does it means?

The Quality of Service (QoS) level is an agreement between the sender of a message and the receiver of a message that defines the guarantee of delivery for a specific message.[1]
The return value of the function **mqttClient.messageQoS()** is either:

- 0 - There is no guarantee of delivery. The recipient does not acknowledge receipt of the message and the message is not stored and re-transmitted by the sender
- 1 - The sender stores the message until it gets a PUBACK packet from the receiver that acknowledges receipt of the message.
- 2 - This level guarantees that each message is received only once by the intended recipients.

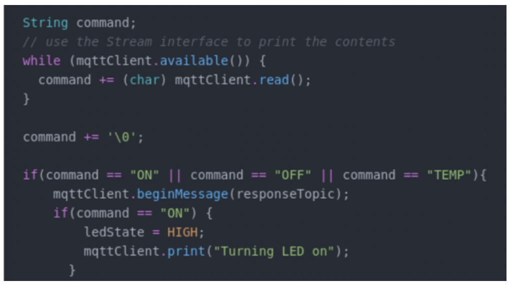
Question 1.3) What is the value returned by mqttClient.messageRetain()? What does it means?

The returned value by the **mqttClient.messageReatain()** is either a true or a false. To understand what is a retained value, first, we have to know how these messages are managed. Let's take this temperature measurement as an example, when a temperature is measured, a message is sent to the MQTT broker which will store it until a new value appears. This massage retain value is set to true so the client gets the message immediately even if they are offline.

2.5 To do

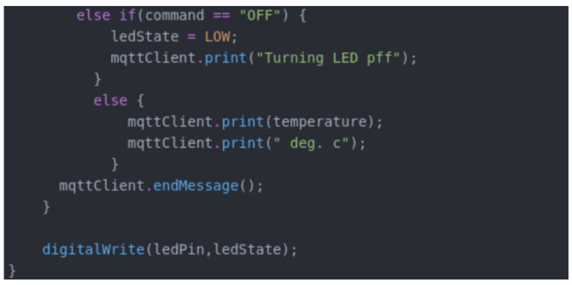
Question 1.4) Modify your subscriber to implement these two commands

1. **The ON command will turn on the onboard LED.**
2. **Similarly, OFF command will turn off the onboard LED.**
3. **The TEMP command will send the temperature on the appropriate response topic, once.**
4. **Any other command will not generate a response.**



```
String command;
// use the Stream interface to print the contents
while (mqttClient.available()) {
  command += (char) mqttClient.read();
}

command += '\0';

if(command == "ON" || command == "OFF" || command == "TEMP"){
    mqttClient.beginMessage(responseTopic);
    if(command == "ON") {
        ledState = HIGH;
        mqttClient.print("Turning LED on");
      }
```

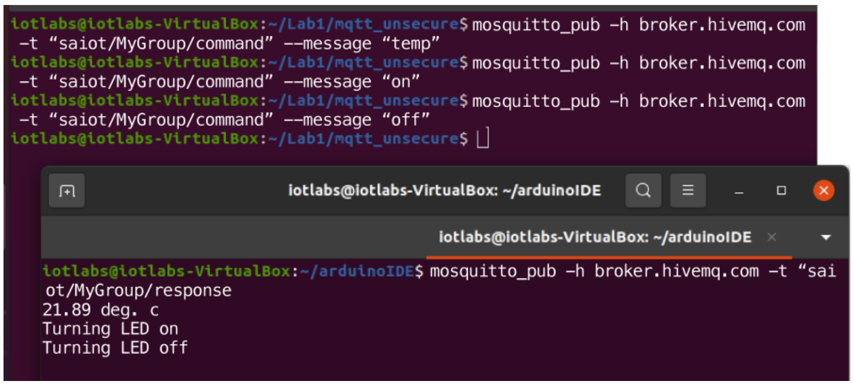
Figure 2.5: onMqttMessage modification



```
        else if(command == "OFF") {
            ledState = LOW;
            mqttClient.print("Turning LED pff");
          }
          else {
              mqttClient.print(temperature);
              mqttClient.print(" deg. c");
            }
       mqttClient.endMessage();
    }

    digitalWrite(ledPin,ledState);
}
```

Figure 2.6: onMqttMessage modification

Question 1.6) Test the implementation by sending the ON, OFF and TEMP commands on the relevant topic from your desktop machine (your ubuntu VM machine).



```
iotlabs@iotlabs-VirtualBox:~/Lab1/mqtt_unsecure$ mosquitto_pub -h broker.hivemq.com
 -t “saiot/MyGroup/command” —-message “temp”
iotlabs@iotlabs-VirtualBox:~/Lab1/mqtt_unsecure$ mosquitto_pub -h broker.hivemq.com
 -t “saiot/MyGroup/command” —-message “on”
iotlabs@iotlabs-VirtualBox:~/Lab1/mqtt_unsecure$ mosquitto_pub -h broker.hivemq.com
 -t “saiot/MyGroup/command” —-message “off”
iotlabs@iotlabs-VirtualBox:~/Lab1/mqtt_unsecure$
```

iotlabs@iotlabs-VirtualBox: ~/arduinoIDE
iotlabs@iotlabs-VirtualBox: ~/arduinoIDE

```
iotlabs@iotlabs-VirtualBox:~/arduinoIDE$ mosquitto_pub -h broker.hivemq.com -t “sai
ot/MyGroup/response
21.89 deg. c
Turning LED on
Turning LED off
```

Figure 2.7: Confirmation message of the commands ”ON”, ”OFF” and ”TEMP”

Question 1.7) In your report provide a block diagram of the implemented system and a screenshot of the implemented functions.

Screenshots of the implemented functions already inserted in the previous questions.

Chapter 3

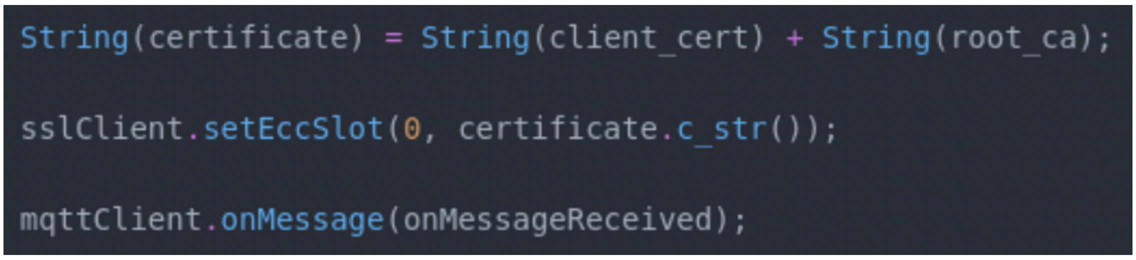
Lab 2: Establish a secure MQTT connection and understand various parts of the setup

3.1 Purpose and Goal

This lab it's demonstrated how to implement levels of security in an MQTT connection, implementing a password-based authentication, and mutual authentication using Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocol that sits on top of the TCP layer.

Question 2.1) Understand and explain the code of the sketch AWS_IoT. Take screenshot of received messages on the AWS dasboard.

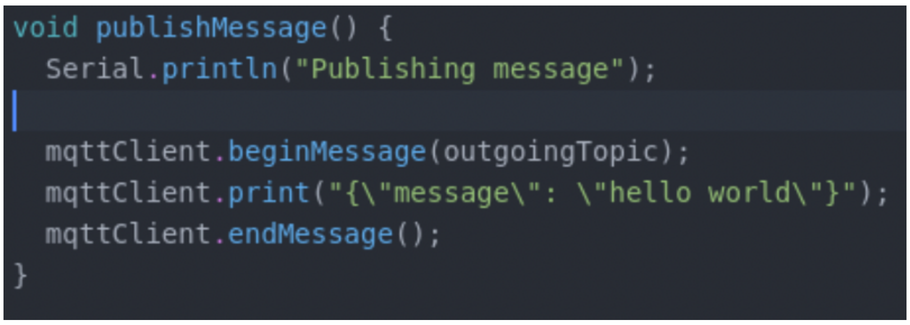
Firstly, on the **setup** function we can observe the concatenation of the client certificate and the root CA, which will form the certificate. This certificate will serve as a password-based system. Right after, the ECCX08 slot is set for the private key and the accompanying public certificate. Finalizing the function, the callback function is set as **onMessageRecevied**.



```
String(certificate) = String(client_cert) + String(root_ca);

sslClient.setEccSlot(0, certificate.c_str());

mqttClient.onMessage(onMessageReceived);
```

Figure 3.1: Certificate & ECCX08 slot & Callback function

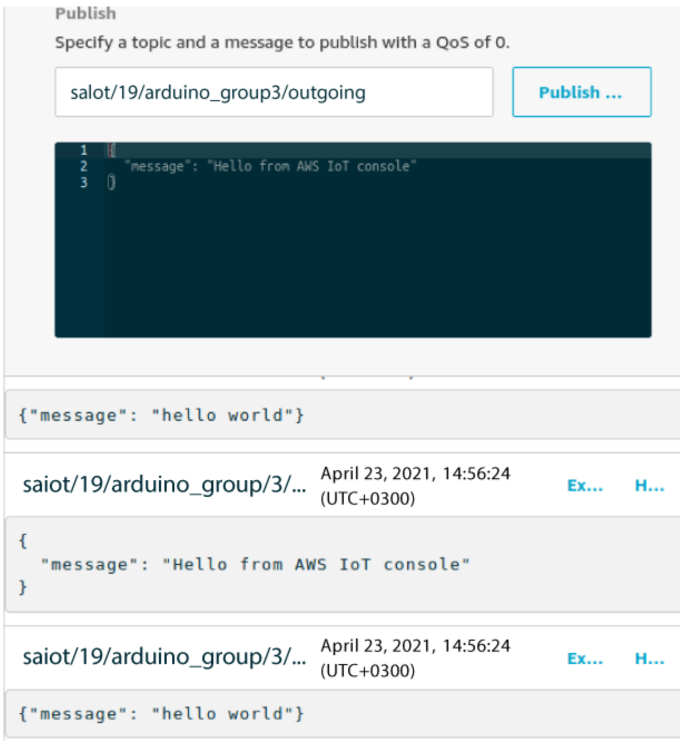
Next, we can examine the function called when publishing a message (**publishMessage**). This function returns the following string "{"message": "Hello word".}"



```
void publishMessage() {
  Serial.println("Publishing message");

  mqttClient.beginMessage(outgoingTopic);
  mqttClient.print("{\"message\": \"hello world\"}");
  mqttClient.endMessage();
}
```

Figure 3.2: Publish Message

And, as we can see the received message on the AWS dashboard was the one expected, from the **publishMessage** function.



Publish

Specify a topic and a message to publish with a QoS of 0.

salot/19/arduino_group3/outgoing

Publish ...

```
{
  "message": "Hello from AWS IoT console"
}
```

```
{"message": "hello world"}
```

saiot/19/arduino_group/3/... April 23, 2021, 14:56:24 (UTC+0300) Ex... H...

```
{
  "message": "Hello from AWS IoT console"
}
```

saiot/19/arduino_group/3/... April 23, 2021, 14:56:24 (UTC+0300) Ex... H...

```
{"message": "hello world"}
```

Figure 3.3: AWS Dashboard Message

Question 2.2) Explain your understanding of TLS/SSL mutual authentication mechanism with respect to the Arduino and AWS IoT core. Make use of diagrams. Check the references for some useful references on this topic.

First, a message is sent by the client to the server proposing the TLS/SSL options. Then, the server responds with a message selecting the TLS/SSL options and also sends a Certificate message, which contains the server's certificate. After, the server requests the client's certificate, so that the connection can be mutually authenticated. Once the request is made, the servers conclude their part of the negotiation. Next, the client sends a response to the message with its certificate and sends the session key information (the encrypted public key) for the server to verify it. Furthermore, and still, the client, sends the last message to activate the negotiated options and to let the server check the newly activated options. Finally, the server a message to activate the negotiated options for all future messages and finishes its job by letting the client check the newly activated options.

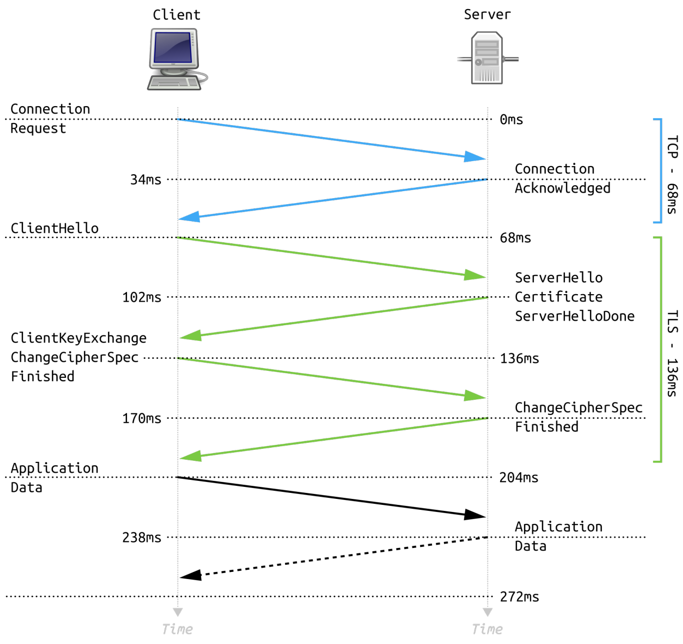


Figure 3.4: TLS/SSL

Question 2.3) What are the pros and cons of using TLS/SSL mutual authentication?

Here are some disadvantages of the TLS/SLL mutual authentication:

- Cost of Certificate: A certificate can be free, but isn't recommended for a lot of reasons;
- Mixed Modes: When the implementation isn't correctly set up and some files are still being served, clients are going to get a warning message, letting them know that some data isn't protected;
- Proxy Caching: Having a complex proxy caching system will make the encrypted content not "cachable";
- Mobile: It can be painful to set up and might require changes to in-house software or buying additional modules from application vendors.

From the advantages side:

- Trust: Gives the client a sense of trust;
- Verification: Important to guarantee visitors and server safety;
- Integrity of Data: Guarantees integrity of data;

Question 2.4) Assuming that the symmetric encryption was carried out by the WiFi module, what was the role of the crypto processor during the TLS/SSL handshake.

The role of the crypto processor during the TLS/SSL handshake was to verify each other, through encrypted messages sent by both parts.

Chapter 4

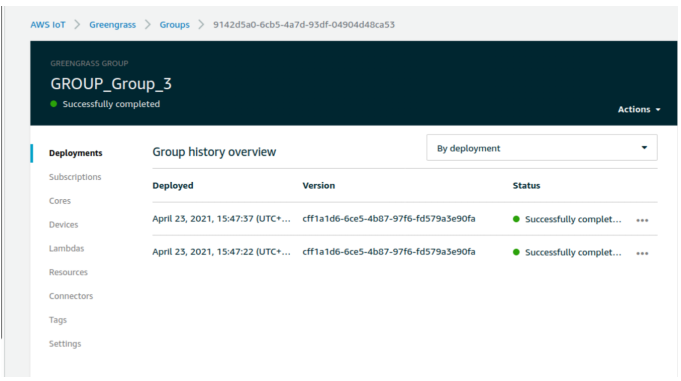
Lab 3: Establish a MQTT connection that is safe from (unintended) snooping and is private using a gateway

4.1 Purpose and Goal

Even tho we secured the connection between the MQTT broker and the clients, there is still an unintended inter-device cross-talk. This lab will focus on demonstrating a secure gateway using AWS Greengrass. AWS Greengrass is a software that extends cloud capabilities to local devices, negating many of the concerns associated with the gateway pattern expressed above.

Question 3.1) Explain how the Thing connects to your gateway, including TLS mutual authentication. Use relevant diagrams.

A Thing connects to the gateway through Greengrass, which uses its device certificate, private key, and the AWS IoT Core root CA certificate to connect the AWS IoT Greengrass service, making a much more secured environment. The Greengrass core device downloads group membership information from the AWS IoT Greengrass service. When a deployment is made to the Greengrass core device, the Device Certificate Manager (DCM) handles local server certificate management for the Greengrass core device.[2]


AWS IoT > Greengrass > Groups > 9142d5a0-6cb5-4a7d-93df-04904d48ca53

GREENGRASS GROUP
GROUP_Group_3
● Successfully completed

Group history overview

Deployed	Version	Status
April 23, 2021, 15:47:37 (UTC+...	cff1a1d6-6ce5-4b87-97f6-fd579a3e90fa	● Successfully complet...
April 23, 2021, 15:47:22 (UTC+...	cff1a1d6-6ce5-4b87-97f6-fd579a3e90fa	● Successfully complet...

Figure 4.1: Deployment

A connected device connects to the AWS IoT Greengrass service using its device certificate, private key, and the AWS IoT Core root CA certificate. After making the connection, the AWS IoT Core device uses the Greengrass Discovery Service to find the IP address of its Greengrass core device. The device also downloads the group CA certificate, which is used for TLS mutual authentication with the Greengrass core device. A connected device attempts to connect to the Greengrass core device, passing its device certificate and client ID. If the client ID matches the thing name of the device and the certificate is valid (part of the Greengrass group), the connection is made. Otherwise, the connection is terminated.[2]

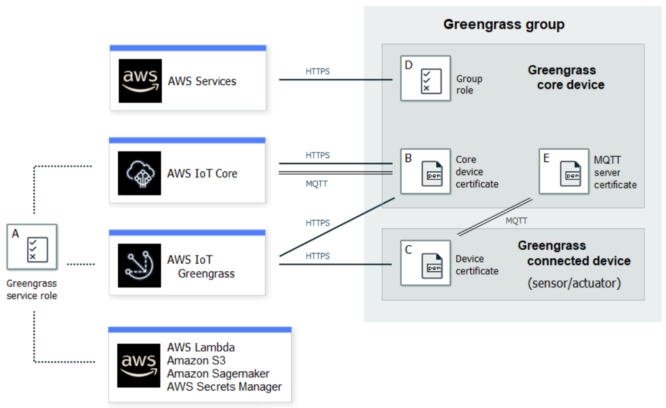
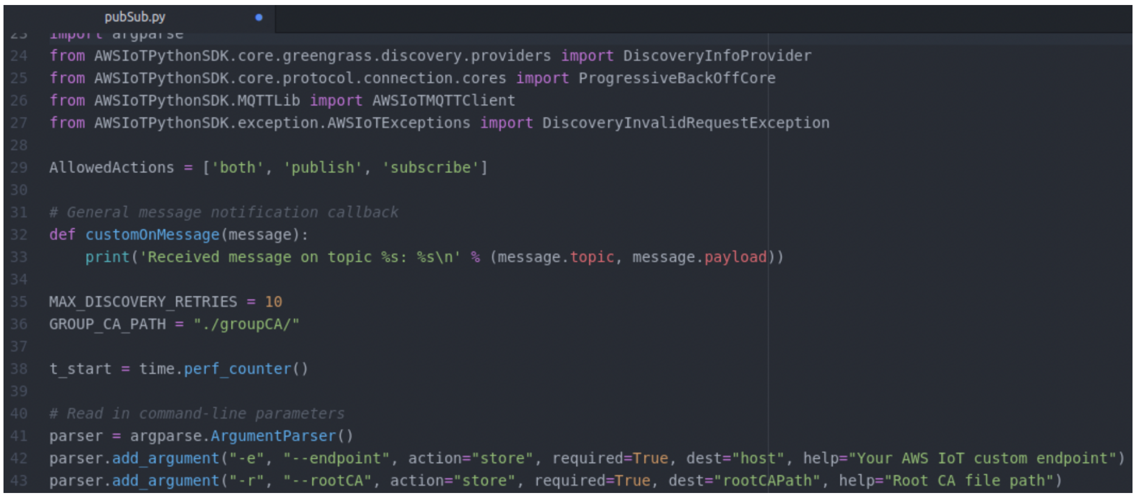


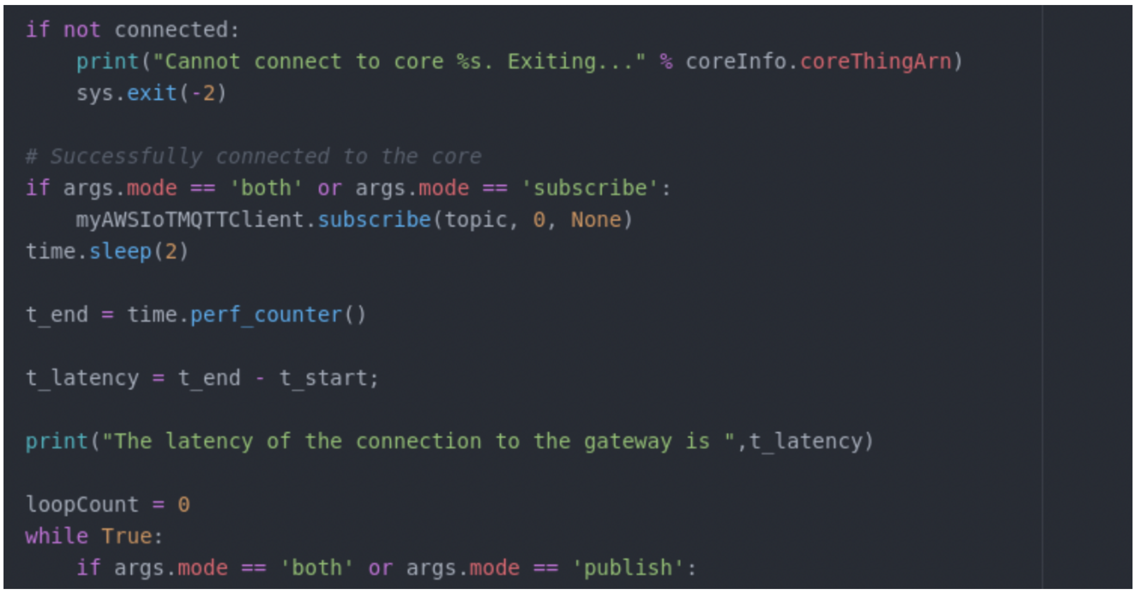
Figure 4.2: Overview of AWS IoT Greengrass security

Question 2.2) Modify pubSub.py to measure the latency of establishing the connection to the gateway. Use time module and use perf_counter() to measure the time.



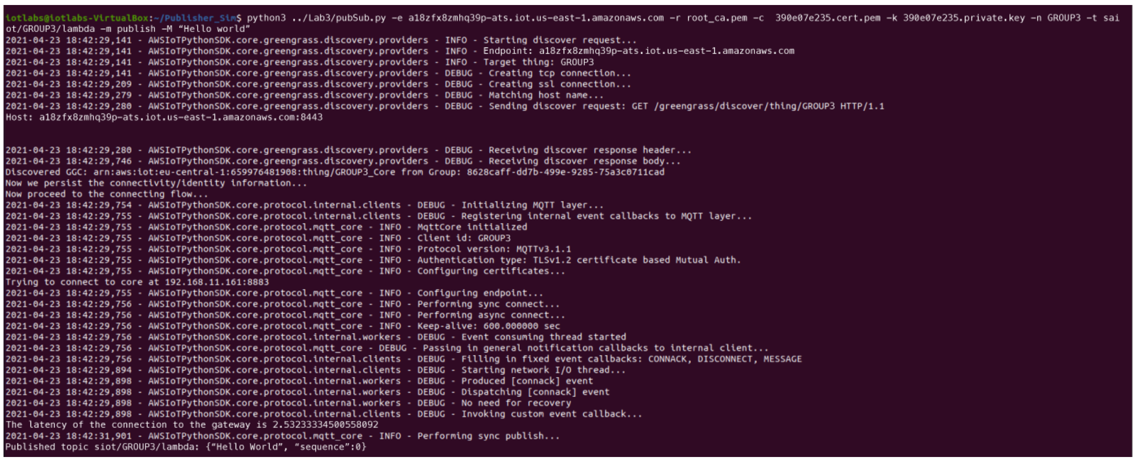
```python
import argparse
from AWSIoTPythonSDK.core.greengrass.discovery.providers import DiscoveryInfoProvider
from AWSIoTPythonSDK.core.protocol.connection.cores import ProgressiveBackOffCore
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient
from AWSIoTPythonSDK.exception.AWSIoTExceptions import DiscoveryInvalidRequestException

AllowedActions = ['both', 'publish', 'subscribe']

# General message notification callback
def customOnMessage(message):
    print('Received message on topic %s: %s\n' % (message.topic, message.payload))

MAX_DISCOVERY_RETRIES = 10
GROUP_CA_PATH = "./groupCA/"

t_start = time.perf_counter()

# Read in command-line parameters
parser = argparse.ArgumentParser()
parser.add_argument("-e", "--endpoint", action="store", required=True, dest="host", help="Your AWS IoT custom endpoint")
parser.add_argument("-r", "--rootCA", action="store", required=True, dest="rootCAPath", help="Root CA file path")
```

Figure 4.3: pubSub modification



```python
if not connected:
    print("Cannot connect to core %s. Exiting..." % coreInfo.coreThingArn)
    sys.exit(-2)

# Successfully connected to the core
if args.mode == 'both' or args.mode == 'subscribe':
    myAWSIoTMQTTClient.subscribe(topic, 0, None)
time.sleep(2)

t_end = time.perf_counter()

t_latency = t_end - t_start;

print("The latency of the connection to the gateway is ",t_latency)

loopCount = 0
while True:
    if args.mode == 'both' or args.mode == 'publish':
```

Figure 4.4: pubSub modification



```
iotlabs@iotlabs-VirtualBox:~/Publisher_Sim$ python3 ../Lab3/pubSub.py -e a18zfx8zmhq39p-ats.iot.us-east-1.amazonaws.com -r root_ca.pem -c  390e07e235.cert.pem -k 390e07e235.private.key -n GROUP3 -t sai
ot/GROUP3/lambda -m publish -M “Hello world”
2021-04-23 18:42:29,141 - AWSIoTPythonSDK.core.greengrass.discovery.providers - INFO - Starting discover request...
2021-04-23 18:42:29,141 - AWSIoTPythonSDK.core.greengrass.discovery.providers - INFO - Endpoint: a18zfx8zmhq39p-ats.iot.us-east-1.amazonaws.com
2021-04-23 18:42:29,141 - AWSIoTPythonSDK.core.greengrass.discovery.providers - INFO - Target thing: GROUP3
2021-04-23 18:42:29,141 - AWSIoTPythonSDK.core.greengrass.discovery.providers - DEBUG - Creating tcp connection...
2021-04-23 18:42:29,209 - AWSIoTPythonSDK.core.greengrass.discovery.providers - DEBUG - Creating ssl connection...
2021-04-23 18:42:29,279 - AWSIoTPythonSDK.core.greengrass.discovery.providers - DEBUG - Matching host name...
2021-04-23 18:42:29,280 - AWSIoTPythonSDK.core.greengrass.discovery.providers - DEBUG - Sending discover request: GET /greengrass/discover/thing/GROUP3 HTTP/1.1
Host: a18zfx8zmhq39p-ats.iot.us-east-1.amazonaws.com:8443


2021-04-23 18:42:29,280 - AWSIoTPythonSDK.core.greengrass.discovery.providers - DEBUG - Receiving discover response header...
2021-04-23 18:42:29,746 - AWSIoTPythonSDK.core.greengrass.discovery.providers - DEBUG - Receiving discover response body...
Discovered GGC: arn:aws:iot:eu-central-1:659976481908:thing/GROUP3_Core from Group: 8628caff-dd7b-499e-9285-75a3c0711cad
Now we persist the connectivity/identity information...
Now proceed to the connecting flow...
2021-04-23 18:42:29,754 - AWSIoTPythonSDK.core.protocol.internal.clients - DEBUG - Initializing MQTT layer...
2021-04-23 18:42:29,755 - AWSIoTPythonSDK.core.protocol.internal.clients - DEBUG - Registering internal event callbacks to MQTT layer...
2021-04-23 18:42:29,755 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - MqttCore initialized
2021-04-23 18:42:29,755 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Client id: GROUP3
2021-04-23 18:42:29,755 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Protocol version: MQTTv3.1.1
2021-04-23 18:42:29,755 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Authentication type: TLSv1.2 certificate based Mutual Auth.
2021-04-23 18:42:29,755 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Configuring certificates...
Trying to connect to core at 192.168.11.161:8883
2021-04-23 18:42:29,755 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Configuring endpoint...
2021-04-23 18:42:29,756 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync connect...
2021-04-23 18:42:29,756 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing async connect...
2021-04-23 18:42:29,756 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Keep-alive: 600.000000 sec
2021-04-23 18:42:29,756 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Event consuming thread started
2021-04-23 18:42:29,756 - AWSIoTPythonSDK.core.protocol.mqtt_core - DEBUG - Passing in general notification callbacks to internal client...
2021-04-23 18:42:29,756 - AWSIoTPythonSDK.core.protocol.internal.clients - DEBUG - Filling in fixed event callbacks: CONNACK, DISCONNECT, MESSAGE
2021-04-23 18:42:29,894 - AWSIoTPythonSDK.core.protocol.internal.clients - DEBUG - Starting network I/O thread...
2021-04-23 18:42:29,898 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [connack] event
2021-04-23 18:42:29,898 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [connack] event
2021-04-23 18:42:29,898 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - No need for recovery
2021-04-23 18:42:29,898 - AWSIoTPythonSDK.core.protocol.internal.clients - DEBUG - Invoking custom event callback...
The latency of the connection to the gateway is 2.53233334500558092
2021-04-23 18:42:31,901 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
Published topic siot/GROUP3/lambda: {“Hello World”, “sequence”:0}
```

Figure 4.5: Latency of the conection to the gateway

Chapter 5

Conclusion

Since I had never worked in this area, this lab was a completely new experience for me. What has surprised me the most was the fact that nowadays is not complicated to create an IoT device. Even though, in theory, is not so simple, there is a lot of information on the web, making it very easy to overcome the different challenges.

Bibliography

[1] Quality of Service 0,1 & 2 - MQTT Essentials: Part 6,
`https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/`

[2] Overview of AWS IoT Greengrass security
`https://docs.aws.amazon.com/greengrass/v1/developerguide/gg-sec.html`