MIDDLE EAST TECHNICAL UNIVERSITY

ELECTRICAL-ELECTRONICS ENGINEERING DEPARTMENT

EE300 - SUMMER PRACTICE REPORT

Student Name: Umut Güloğlu

Student ID: 2232007

SP Company Name: German-Turkish Advanced Research Centre for ICT

SP Date: 18.06.2019-06.09.2019

Supervisor Name: Doruk Şahinel

Supervisor E-mail: doruk.sahinel@gt-arc.com

Supervisor Phone Number: +49 30 31474221

Table of Contents

1. INTRODUCTION

I have performed my first summer practice in German-Turkish Advanced Research Centre for ICT (GT-ARC). GT-ARC is a company that makes research and development on technology, specifically smart cities, telecommunication& future internet, logistic& transportation, and security& cybersecurity. My internship lasted 12 weeks, started from 18.06.2019 and ended on 06.09.2019. I have worked on a project called ‘CHARIoT’. The aim of this project is designing and developing an IoT middleware solution for different types of IoT technologies. I have mainly focused on creating a secure communication between IoT nodes. I have used Ubuntu in all of my works. During my internship, my supervisor was Doruk Şahinel who has a B.Sc. degree in Electrical and Electronics Engineering, and a M.Sc. degree in Information and Communication Systems.

I have preferred GT-ARC for a couple of reasons. First of all, I have wanted to have an abroad experience. Secondly, I have wanted to work on a topic which is related with IoT or Computer Vision. Finally, I have wanted to work in a research center. During my internship period, I simply learned what Internet of Things, Message Queuing Telemetry Transport, and TLS/SSL are. Moreover, I have learned how to use certificates to authenticate, and I have created a secure communication between nodes or between the broker and a computer.

This report includes a detailed description of what I learned and what I did during my internship. The report begins with the description of the company. Then, it continues with the main part of the report including the details about IoT network prototype of GT-ARC, what I have learned and what I have done to create a secure communication between my computer and gateway broker. After that, there is a conclusion part to summarize my internship. And finally, the report finishes with the references part.

2. DESCRIPTION OF THE COMPANY

2.1. Company Name

German-Turkish Advanced Research Centre for ICT (GT-ARC)

2.2. Company Location

GT-ARC has two offices. I have performed my summer practice in Berlin.

Location 1: Berlin

Address: *Ernst-Reuter-Platz 7, 10587 Berlin, Germany*

Location 2: Istanbul

Address: *Informatics Institute, ITU Ayazaga Campus, 34469 Maslak-Istanbul, Turkey*

2.3. General Description of the Company

German-Turkish Advanced Research Centre for ICT (GT-ARC) was founded by Şahin Albayrak in 2012. The company is supported by the German Federal Ministry of Education and Research, and the Turkish Ministry of Transport, Maritime Affairs and Communications, and it is in cooperation with DAI-Labor and Berlin Technical University. The mission of GT-ARC is to take a leading role in creating ICT-based innovation through German-Turkish cooperation by bringing the bright minds of the two countries together to design and develop core technologies for future innovative solutions, providing a framework for joint projects, staff mobility, idea exchange, and value creation among relevant stakeholders. GT-ARC aims research and development of enabling basic ICT techniques and platforms for the contemporary ICT-based solutions which address the grand challenges and needs of our society such as transportation, energy, safety, and telecommunications.

GT-ARC mainly focuses on Multi Agent Systems, Data Analytics, Human Robot Interaction, and Autonomous Solutions. The completed projects of GT-ARC are Intermodal Mobility Assistance for Megacities (IMA), Autonomous Analysis and Verification of Distributed e-Government Systems (AuVeGoS), Development of an Innovative Healthcare Assistant for Migrants (GeM), and Intelligent Solutions for Protecting Interdependent Critical Infrastructures (ILIas). The ongoing projects of GT-ARC are An Intelligent Framework for Service Discovery and Composition (ISCO) , A Scalable Middleware Approach for Holistic Networking of IoT (CHARIoT), Cognitive Data Analytics Framework (CODA), 5G for Cooperative & Connected Automated Mobility (5GMOBIX) [1].

2.4. Number and Duties of the Engineers Employed

GT-ARC contains 1 Chief Executive Officer, 1 Executive Officer, 1 Research Director, 12 Researcher, and many students working in the company part-time. However, since the company uses joint laboratories with DAI-Labor, there are some people working in both of the companies.

2.5. Organizational Structure of the Company

The organizational structure of the company can be found in Figure 1.

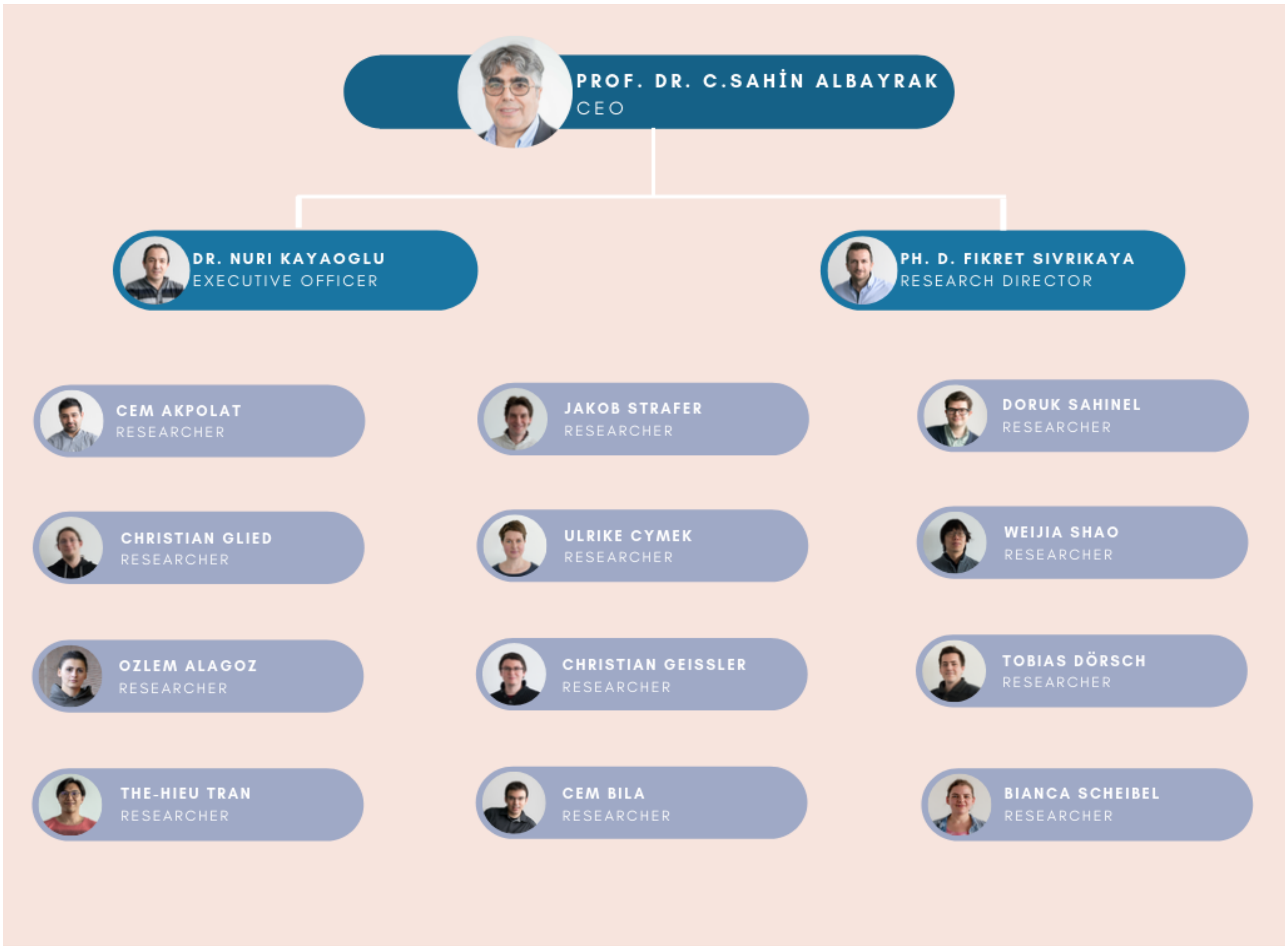


Figure 1. The Organizational Chart of GT-ARC

3. INTERNET OF THINGS

Internet of things is a set of words using to describe the internet where every object is connected via the Internet. As I said earlier, there exists an IoT mesh network prototype in the company.

3.1. Mesh Network Prototype of IoT

In the mesh network prototype in GT-Arc, there exist three sensor nodes, two edge nodes and one gateway node. In every node, there is Raspberry Pi with Ubuntu. Sensor nodes contain different types of sensors such as temperature sensors, humidity sensors, and gas sensors. These nodes send the data (or message) which is generated by using the sensors to edge nodes by using 6lowpan. Then edge nodes send the data (or message) to the gateway node by using Wi-Fi. In other words, edge nodes act as a bridge to transmit to message to the gateway in this network. After transmitting the data (or message) to the gateway node, the data (or message) is transmitted to the clients. The process can be seen in Figure 2. MQTT protocol is used in all of the process explained above.

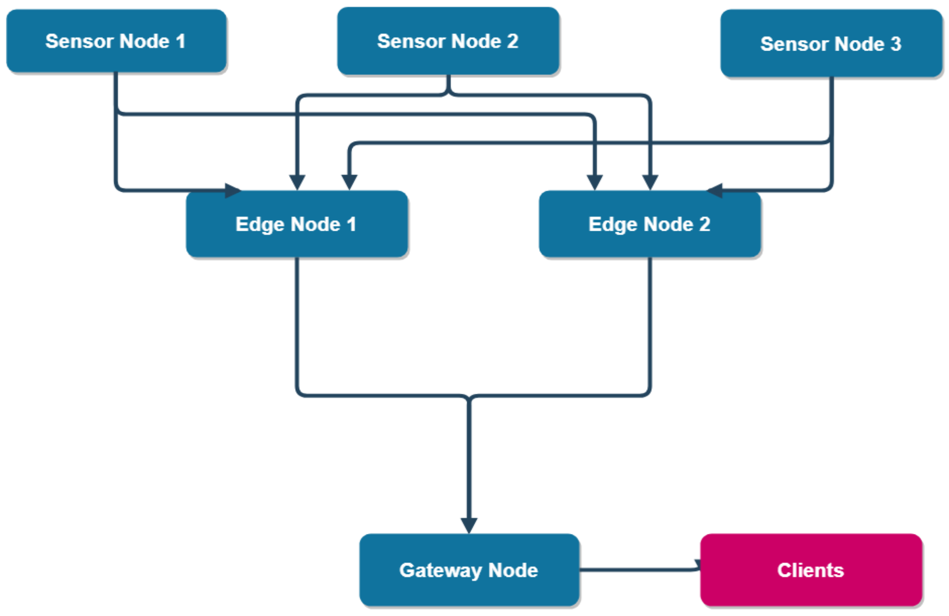


Figure 2.The Diagram of Mesh Network Prototype

3.2. Message Queuing Telemetry Transport (MQTT)

Message Queuing Telemetry Transport (MQTT) is a messaging protocol. It is used in our prototype to send or receive the message since MQTT is designed for low power devices and low bandwidth, which means that it is very useful for IoT devices.

Simply, in a system using MQTT, there exist several clients and a broker. It is very similar to the working principle of Instagram or Facebook. There are two options for the clients, which are ‘publish’ and ‘subscribe’. The client can subscribe to a specific topic, or it can send a message on a specific topic. The client who subscribed takes the messages which are sent by any other client on that topic. The aim of the broker is sending the message on a specific topic to the clients which are subscribed to that topic. In other words, the broker acts as a bridge to transmit the message from one client, which publishes the message in a specific topic, to another client, which subscribes to a specific topic. The representation of MQTT can be seen in Figure 3.

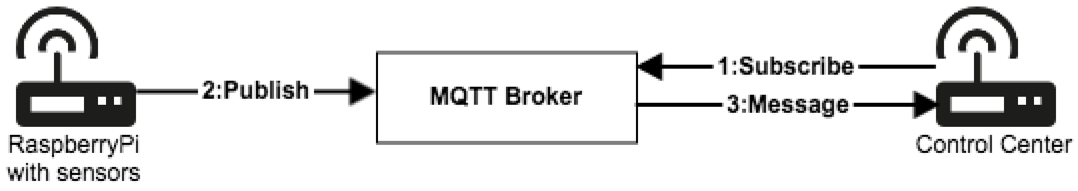


Figure 3. The Representation of MQTT [2]

In the mesh network we use, there are three brokers, one in the gateway node, and one in each of the edge nodes. When a sensor node sends a message to an edge node on a specific topic, let’s say ‘temperature’, it publishes to the broker in the edge node, let’s say Edge Node 1. Moreover, Edge Node 1 is subscribed to the topic ‘temperature’ to the same broker inside Edge Node 1, so that when a sensor node publishes a message in the topic ‘temperature’, Edge Node 1 can receive the message. Then Edge Node 1 publishes the message to the broker in Gateway Node in the topic ‘temperature’, Gateway Node can receive the message since it subscribed to the topic ‘temperature’ to the broker inside Gateway Node. As a result, a message in the topic ‘temperature’ is transmitted to the gateway by using Message Queuing Telemetry Transport (MQTT).

4. NETWORK THEORY

4.1. Transport Control Protocol (TCP) & User Datagram Protocol (UDP)

Both Transport Control Protocol and User Datagram Protocol are the protocols used for sending or receiving data. In both of the protocols, the data is sent by using packets. Packets contain bits. Every packet has to have a header that contains IP addresses, and port numbers of the source and the destination, and information about communication and packets. The size of a header in TCP is bigger than the size of a header in UDP since it contains more information.

There are several main differences between TCP and UDP. First of all, in Transport Control Protocol, there exists a three-way handshake. Before sending or receiving the information, the client and the server have to complete the three-way handshake to start the communication. In User Datagram Protocol, there is not any handshake. It is called as three-way handshake because it contains three parts. Firstly, the client sends a SYN message to the server (or another client). This message simply means 'I want to connect you'. When the server receives this message, it sends SYN-ACK message to the client. It simply means 'I get your message, and I want to connect you as well'. Finally, when the client receives SYN-ACK message, it sends ACK message, meaning 'I get your message', and starts to send or receive the data. Secondly, in Transport Control Protocol, the recipient is guaranteed to receive the data in correct order, whereas it is not guaranteed in User Data Program. In TCP, the sender gives a number for every packet. If the packet could not receive to the recipient, the recipient sends a message, about the piece which is missing, to the sender. Then, the sender sends the message again until the process is completed. However, in UDP, the sender just sends the data. It does not important to be received or not by the recipient. As a result, TCP is more reliable than UDP, whereas UDP is faster than TCP. Therefore, TCP is usually used in the cases that every data is important, such as file transferring, World Wide Web, email, etc. On the other hand, UDP is usually used in the cases that the speed is much more important than losing data, such as streaming a video, playing online computer games that every millisecond is very important [3].

4.2. Transport Layer Security (TLS)/ Secure Socket Layer (SSL)

Transport Layer Security (TLS) / Secure Socket Layer (SSL) are protocols which are used in order to provide a secure communication over a computer network by using cryptology. Before TLS, SSL is the first name of this protocol. After SSL version 3.0, the developed version of SSL has been called as TLS 1.0. However, SSL is still used in some circumstances today. HTTPS can be given as an example of TLS/SSL communication. HTTP stands for “Hyper-Text Transfer Protocol”, whereas HTTPS stands for “Hyper-Text Transfer Protocol Secure”. TLS/SSL is used to provide security in HTPPS. When one visits a website, one can see a green padlock before the website. It means that this website uses TLS/SSL in order to protect the user from Man-in-the-Middle attacks. Primarily, Transport Layer Security is used with the protocols which are reliable such as TCP, but it rarely can be used with UDP because before starting to send or receive the data, there is a handshake which is more complex than a three-way handshake. Therefore, it does not very suitable for the protocols where speed is important.

In a secure communication, X.509 certificates are used; therefore, asymmetric cryptology is used for TLS/SSL. Certificates are used both authentication and encryption. There are two different secure communication types. The first one is one side authentication. In this type, only the server has a certificate. It can be guaranteed that this server is the server that the client wants to establish a communication. However, the server cannot be sure whether the client is the client that sends a connection request. Besides, the messages send from the client to the server will be encrypted, whereas the messages send from the server will not be encrypted, meaning everyone catching the data packets send from the server can see the data. HTTPS can be considered as an example of one side authentication. When user asks Wikipedia to give information about a topic, one can reach the information about that topic; however, one cannot reach user’s messages send to Wikipedia since it is encrypted. The second type is two side authentications. In this type, both the server and the client have to have certificates. Therefore, both the server and the client can be sure whether the opposite side is the correct one, and all messages will be encrypted. The difference between one side authentication TLS/SSL handshake and two side authentication TLS/SSL handshake is that two side handshake has two more steps, which are certificate request of the client and sending client’s certificate.

As it is mentioned earlier, asymmetric cryptology is used in TLS/SSL. In asymmetric cryptology, there are key pairs used for encryption and decryption. Each pair has one public key, shared by everyone, and one private key. Both the client and the server have to have a key pair. The aim of TLS/SSL handshake is both authenticating the other side, and giving the public keys to the other side by using certificates. After handshake, the sender encrypts the message by using other side’s public key and sends the message. By doing that, sender guarantees the message can be understood by the recipient who has the private key which is the duo of the public key which is used to encrypt the message.

5. IMPLEMENTATIONS OF TLS/SSL OVER MQTT

5.1. Creating Certificates

As I stated earlier, my aim was creating a secure channel between my computer and a MQTT broker which is on the gateway node by using TLS/SSL. In order to create secure communication, the first thing which I need was certificates signed by a certificate authority. Certificate authority is an entity that can be trusted. It is responsible for managing and issuing public keys and security certificates. Because IoT Mesh Network will contain a lot of nodes in the future, it will be very expensive to buy a certificate for every node. Therefore, I have created the certificates in gateway node myself by using OpenSSL. In this case, I was the certificate authority. The steps that I did to create certificates have been explained below. These steps can be done by using Ubuntu after connecting Pi by using SSH.

- *First, I have installed OpenSSL on Raspberry Pi (Gateway Node).*

```
sudo apt-get install openssl
```

- *I have opened a new directory.*

```
mkdir /path/to/file/
```

- *I have created a certificate authority key so that I could sign the certificates.*

```
openssl genrsa -des3 -out CerAut.key 2048
```

CerAut.key is the name of the key of the certificate authority. It is generated as 2048-bit RSA key pair. I wanted the key to have a password. Therefore, I have used '–des3' command. After creating, I have entered the password that I want to use.

- *I have created CA certificate which is signed by CA, which is CerAut.key in our case. The password was asked. I have entered the password which I chose in the previous step.*

```
openssl req -new -x509 -days 1826 -key CerAut.key -out caforDAI.crt -subj
"/C=DE/ST=Berlin/L=Berlin/O=GT-ARC/OU=CA-DAI/CN=130.149.232.227"
```

caforDAI.crt is the name of the certificate which is signed by CerAut.key. The validity will expire in 1826 days. '–subj' part is the information about the CA. Common name is '130.149.232.227' because it is the IP address of the gateway node. The certificate of certificate authority can be seen in Figure 4.

- *I have created a key for the broker. I did not use -des3 command as in the CA since the broker cannot decode the password. broker.key is the name of the key.*

```
openssl genrsa -out broker.key 2048
```

- *I have created a certificate request for the certificate of the broker.*

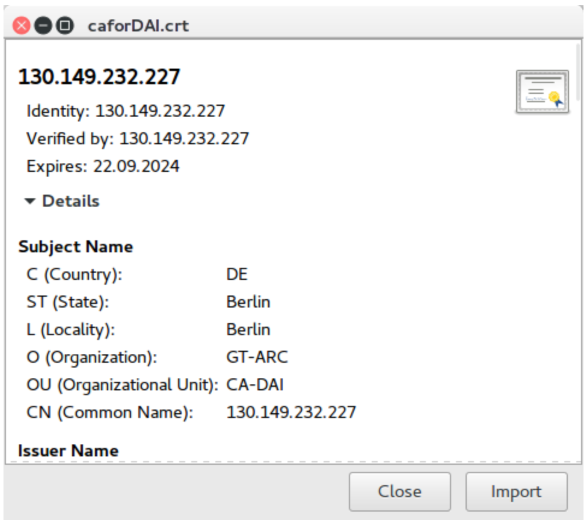
```
openssl req -new -out brokerforDAI.csr -key broker.key -subj
"/C=DE/ST=Berlin/L=Berlin/O=GT-ARC/OU=Gateway-DAI/CN=130.149.232.227"
```

brokerforDAI.csr is the name of the certificate request. We put broker.key to this request so that we can transmit the public key of the broker to other side. '-subj' part is the information about the broker. Common name must be the IP address of the broker, which was '130.149.232.227' in the broker which I have worked on. Now, we all need is signing this request.

- *I have signed this request since I was Certificate Authority. We have used caforDAI.crt, CerAut.key, broker.key, and brokerforDAI.csr.*

```
openssl req -new -out brokerforDAI.csr -key broker.key -subj
"/C=DE/ST=Berlin/L=Berlin/O=GT-ARC/OU=Gateway-DAI/CN=130.149.232.227"
```

As a result, I could create a certificate for broker, which can be seen in Figure 4. For one side authentication, it is enough. However, for two side authentication, client certificate must be created as well.


caforDAI.crt

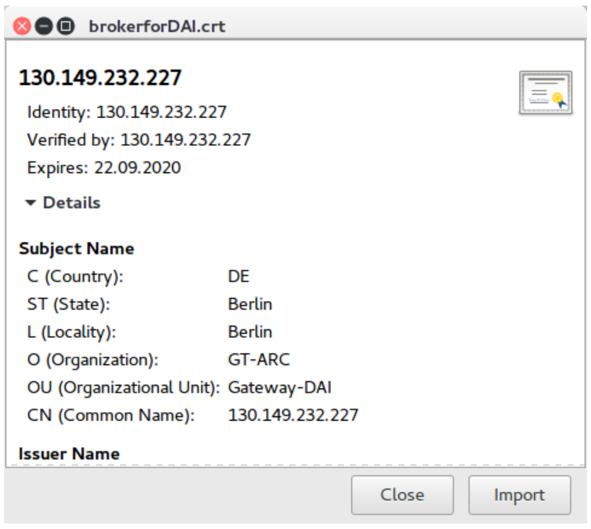
130.149.232.227
Identity: 130.149.232.227
Verified by: 130.149.232.227
Expires: 22.09.2024

▾ Details

Subject Name

C (Country):	DE
ST (State):	Berlin
L (Locality):	Berlin
O (Organization):	GT-ARC
OU (Organizational Unit):	CA-DAI
CN (Common Name):	130.149.232.227

Issuer Name


brokerforDAI.crt

130.149.232.227
Identity: 130.149.232.227
Verified by: 130.149.232.227
Expires: 22.09.2020

▾ Details

Subject Name

C (Country):	DE
ST (State):	Berlin
L (Locality):	Berlin
O (Organization):	GT-ARC
OU (Organizational Unit):	Gateway-DAI
CN (Common Name):	130.149.232.227

Issuer Name

Figure 4. The certificates of the certificate authority (left) and the broker (right)

Up to now, I have created the certificates for secure communication by using only broker's certificate, meaning it is one side authentication. To create a secure communication with two side authentication, the client has to have the certificate as well. Therefore, I have done the steps below to create client certificate in Gateway Node.

- *First, I have created the client key pair.*

```
openssl genrsa -out client.key 2048
```

- *I have created certificate request for the client, which is client1.csr.*

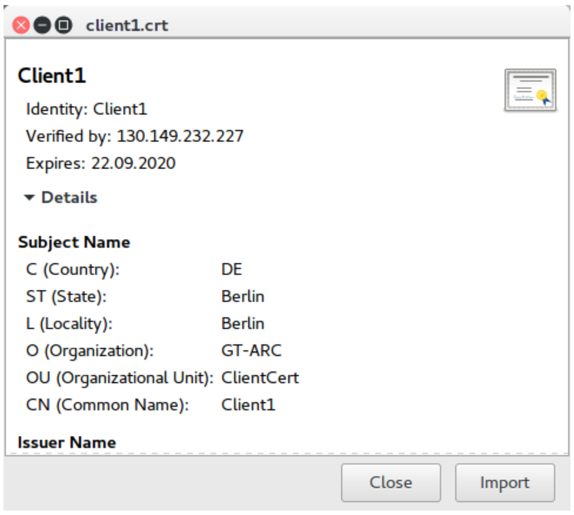
```
openssl req -new -out client1.csr -key client.key -subj
"/C=DE/ST=Berlin/L=Berlin/O=GT-ARC/OU=ClientCert/CN=Client1"
```

While creating client certificate, Common Name does not have to be IP Address of the client. For example, I have given Client1 as common name of the certificate.

- *Finally, I have signed the request by using CerAut.key and caforDAI.crt.*

```
openssl x509 -req -in client1.csr -CA caforDAI.crt -CAkey CerAut.key -CAcreateserial -
out client1.crt -days 365
```

After this step, we have created the certificate of the client which will expire in 365 days. Client certificate can be seen in Figure 5. The problem is that the certificate and the key pair of the client have to be in the client side. However, it was in the broker side now. Therefore, we needed to transfer these file to the client side, which is my computer in my case. Since it contains more than one step and we will use a lot of nodes in the future, I have created bash script, which can be seen in Figure 6, to copy the necessary files to the client side.



Client1

Identity: Client1
Verified by: 130.149.232.227
Expires: 22.09.2020

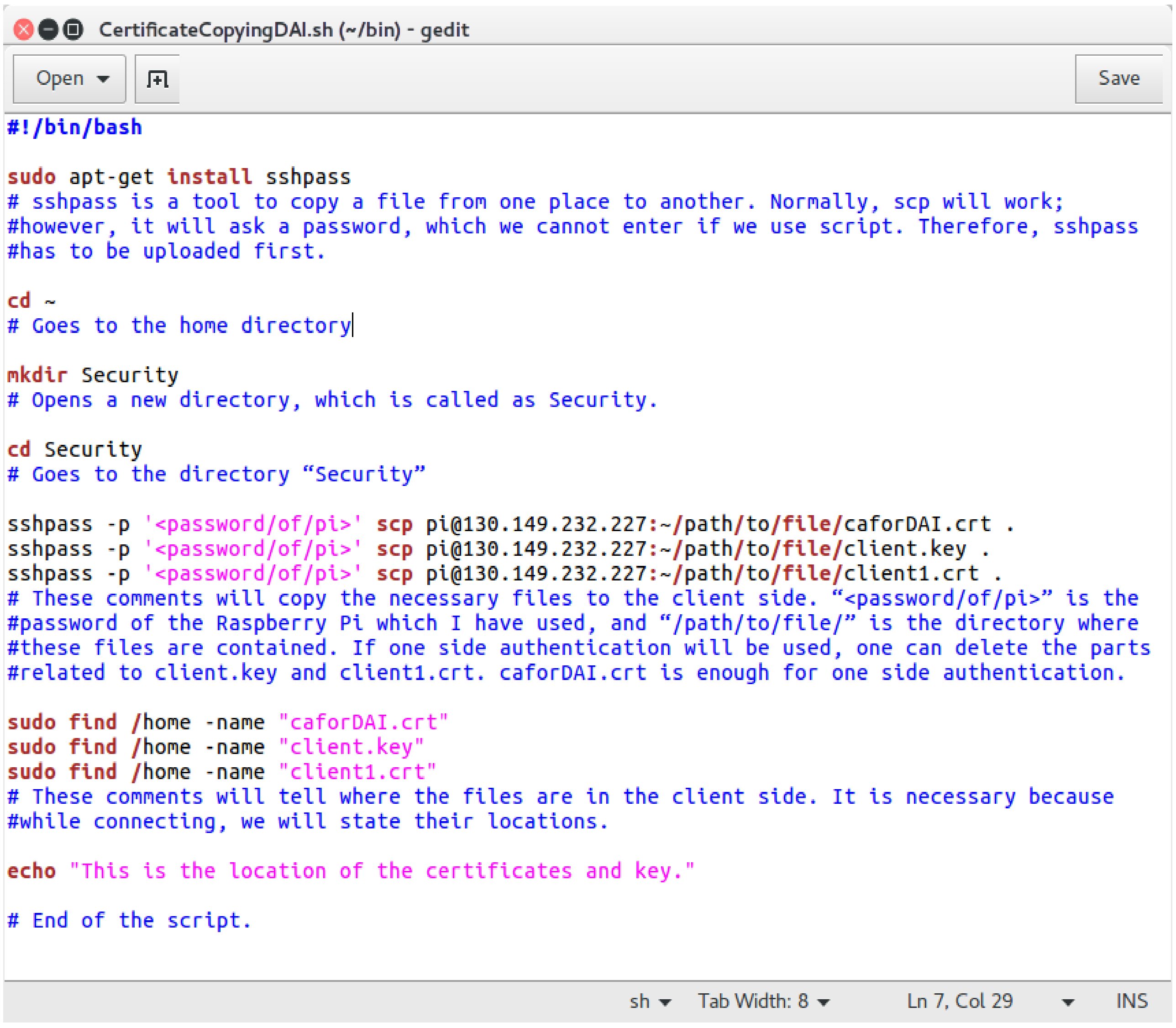
▾ Details

Subject Name

C (Country):	DE
ST (State):	Berlin
L (Locality):	Berlin
O (Organization):	GT-ARC
OU (Organizational Unit):	ClientCert
CN (Common Name):	Client1

Issuer Name

Figure 5. The certificate of the client



```
#!/bin/bash

sudo apt-get install sshpass
# sshpass is a tool to copy a file from one place to another. Normally, scp will work;
#however, it will ask a password, which we cannot enter if we use script. Therefore, sshpass
#has to be uploaded first.

cd ~
# Goes to the home directory

mkdir Security
# Opens a new directory, which is called as Security.

cd Security
# Goes to the directory “Security”

sshpass -p '<password/of/pi>' scp pi@130.149.232.227:~/path/to/file/caforDAI.crt .
sshpass -p '<password/of/pi>' scp pi@130.149.232.227:~/path/to/file/client.key .
sshpass -p '<password/of/pi>' scp pi@130.149.232.227:~/path/to/file/client1.crt .
# These comments will copy the necessary files to the client side. “<password/of/pi>” is the
#password of the Raspberry Pi which I have used, and “/path/to/file/” is the directory where
#these files are contained. If one side authentication will be used, one can delete the parts
#related to client.key and client1.crt. caforDAI.crt is enough for one side authentication.

sudo find /home -name "caforDAI.crt"
sudo find /home -name "client.key"
sudo find /home -name "client1.crt"
# These comments will tell where the files are in the client side. It is necessary because
#while connecting, we will state their locations.

echo "This is the location of the certificates and key."

# End of the script.
```

Figure 6. The script to copy the files from gateway node to the client

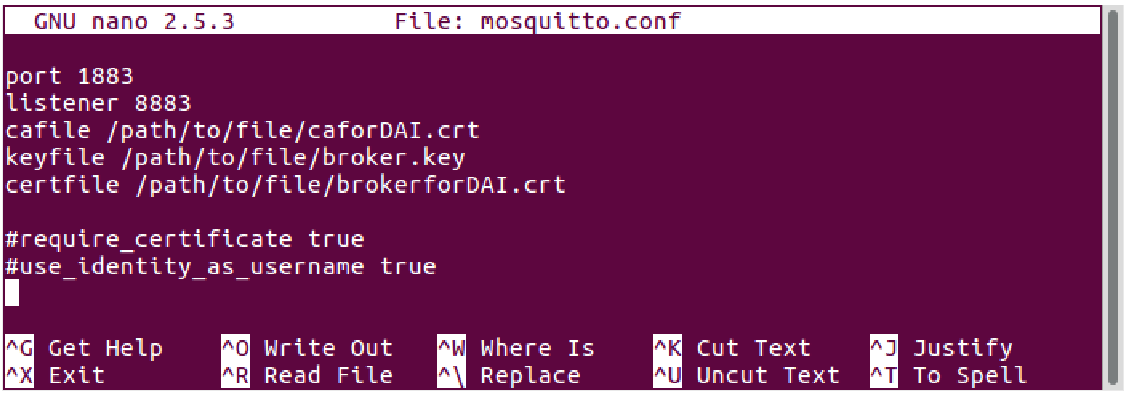
5.2. Mosquitto

Eclipse Mosquitto is an open source (EPL/EDL licensed) message broker that implements the MQTT protocol versions 5.0, 3.1.1 and 3.1. Mosquitto is lightweight and is suitable for use on all devices from low power single board computers to full servers [4]. In other words, it is a useful broker to use in Internet of Things devices. In our company this was already installed. When it is installed, default port is 1883. 1883 is a port which you cannot imply anything which is related with encryption and decryption. On the other hand, one can crate secure communication by using port 8883. Therefore, we needed to change this port. I had to use configuration files. Configuration file is used change the port number and to state the locations of the necessary files.

- *To create configuration file, first I needed to create a file which will be a configuration file.*

```
sudo nano mosquitto.conf
```

- *After that, I have written the necessary things in the configuration file. The example of the configuration can be seen in Figure 7.*



```
 GNU nano 2.5.3             File: mosquitto.conf

port 1883
listener 8883
cafile /path/to/file/caforDAI.crt
keyfile /path/to/file/broker.key
certfile /path/to/file/brokerforDAI.crt

#require_certificate true
#use_identity_as_username true

^G Get Help     ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify
^X Exit         ^R Read File    ^\ Replace      ^U Uncut Text   ^T To Spell
```

Figure 7. mosquitto.conf File

'port 1883' means mosquitto will use port 1883 as main port. We have used this because we want the client to be able to send by not using TLS in addition to secure communication. 'listener 1883' means mosquitto will use port 8883 as well. 'cafile …' have been used to specify the location of the certificate of certificate authority. 'keyfile …' have been used to specify the location of the key of the broker, and 'certfile …' have been used to specify the location of the certificate of the broker. By specifying these, we could send broker's certificate and key to the other side before the communication established. After these, one side communication can be established. However, to create two side authentication, we needed extra two commands. First, I have written 'require_certificate true', meaning that the client certificate is needed for connection. Also, I have written 'use_identity_as_username true'. It is not necessary but it is used not to give random names to the clients [5].

The problem was how to start this mosquitto by using this configuration. Normally, mosquitto is started by just using the command '**mosquitto**'. To use configuration file, the command '**mosquitto –c /path/to/file/mosquitto.conf**' has been used. Here –c states the location of the configuration file. If one uses –d, mosquitto will start in daemon mode, meaning that it will continue to work even if we exited from the gateway.

5.3. Sending & Receiving Message by using Secure MQTT

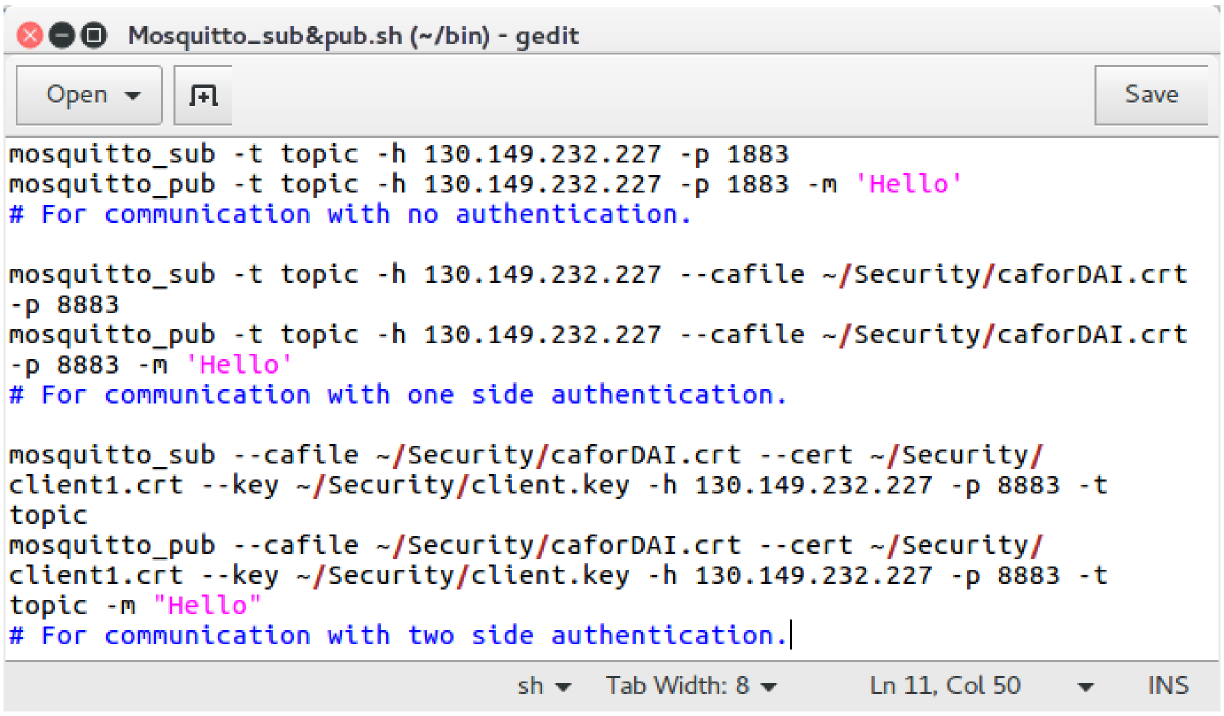
5.3.1. By using mosquitto_pub/mosquitto_sub

After opening the mosquitto, the clients can communicate by using MQTT. One of the ways to communicate is using mosquitto_pub & mosquitto_sub. It is a tool which the client can use to publish or subscribe just by using terminal in Ubuntu. The steps are given below.

- *To use that tool, first it must be downloaded.*

```
sudo apt-get install mosquitto-clients
```

- *After downloading, mosquitto_pub (publishing) & mosquitto_sub (subscribing) can be used. If one wants to use secure communication, one must specify the port number, 8883 in our case, and the location of the necessary certificates or/and keys. The examples of subscribe commands (mosquitto_sub) and publish commands (mosquitto_pub) can be seen in Figure 8.*



```
mosquitto_sub -t topic -h 130.149.232.227 -p 1883
mosquitto_pub -t topic -h 130.149.232.227 -p 1883 -m 'Hello'
# For communication with no authentication.

mosquitto_sub -t topic -h 130.149.232.227 --cafile ~/Security/caforDAI.crt
-p 8883
mosquitto_pub -t topic -h 130.149.232.227 --cafile ~/Security/caforDAI.crt
-p 8883 -m 'Hello'
# For communication with one side authentication.

mosquitto_sub --cafile ~/Security/caforDAI.crt --cert ~/Security/
client1.crt --key ~/Security/client.key -h 130.149.232.227 -p 8883 -t
topic
mosquitto_pub --cafile ~/Security/caforDAI.crt --cert ~/Security/
client1.crt --key ~/Security/client.key -h 130.149.232.227 -p 8883 -t
topic -m "Hello"
# For communication with two side authentication.
```

Figure 8. The examples of mosquitto_sub & mosquitto_pub

The expressions inside the commands in mosquitto_pub & mosquitto_sub are:

--cafile: Location of caforDAI.crt

--cert: Location of the certificate of the client

--key: Location of the key of the client

-h: Hostname or IP of the broker

-p: Port number (8883 for TLS/SSL)

-t: Topic

-m: The message which we want to send

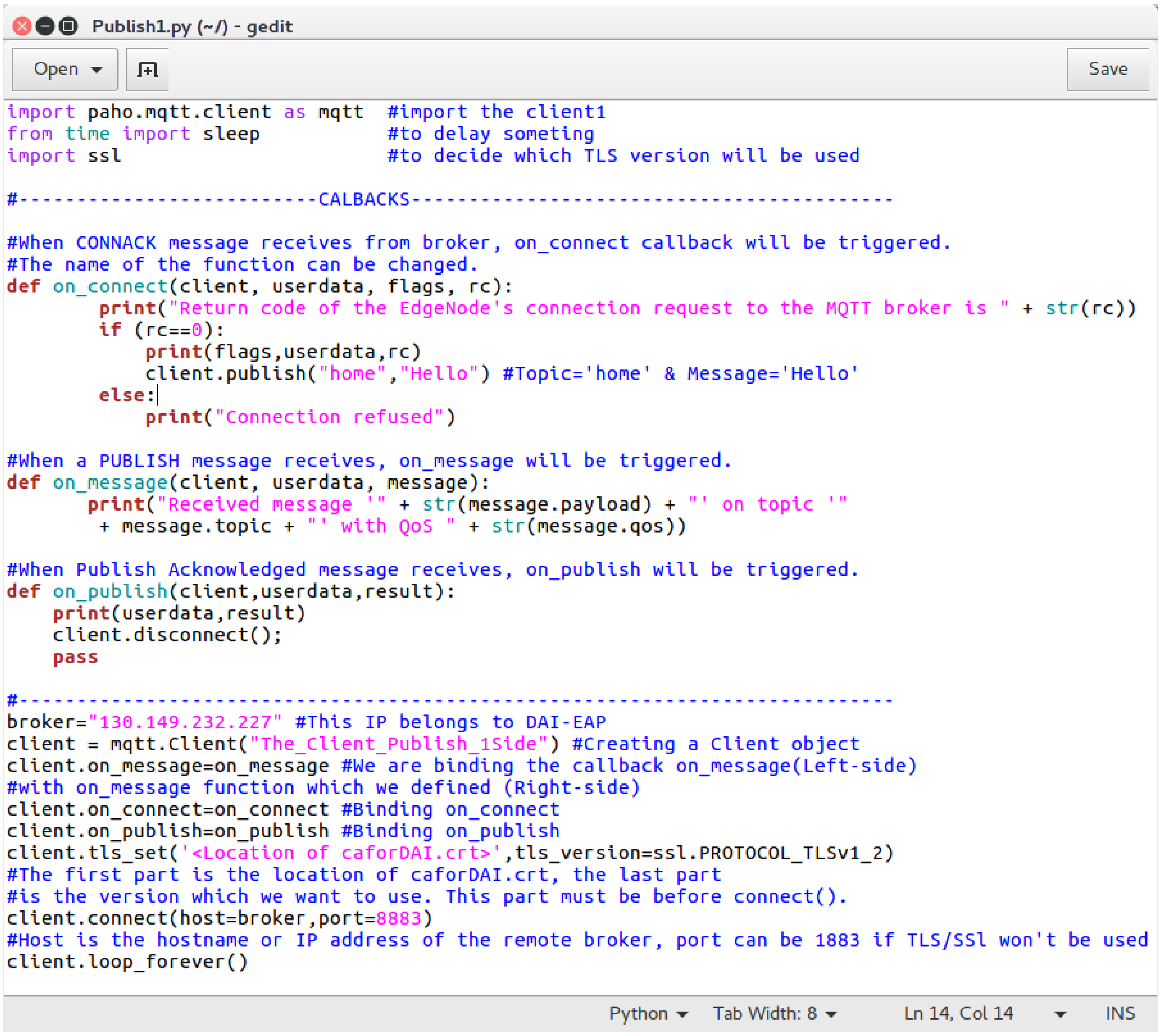
5.3.2. By using Python

The other way to communicate by using MQTT is using Python. In the code, the library of Eclipse Paho has been used to send or receive message.

One can see the codes which I wrote to publish a message by using one side authentication, and publish a message by using two side authentication in Figure 9 and Figure 10, respectively. Besides, one can see the codes which I wrote to subscribe to a topic by using one side authentication, and subscribe to a topic by using two side authentication in Figure 11 and Figure 12, respectively.

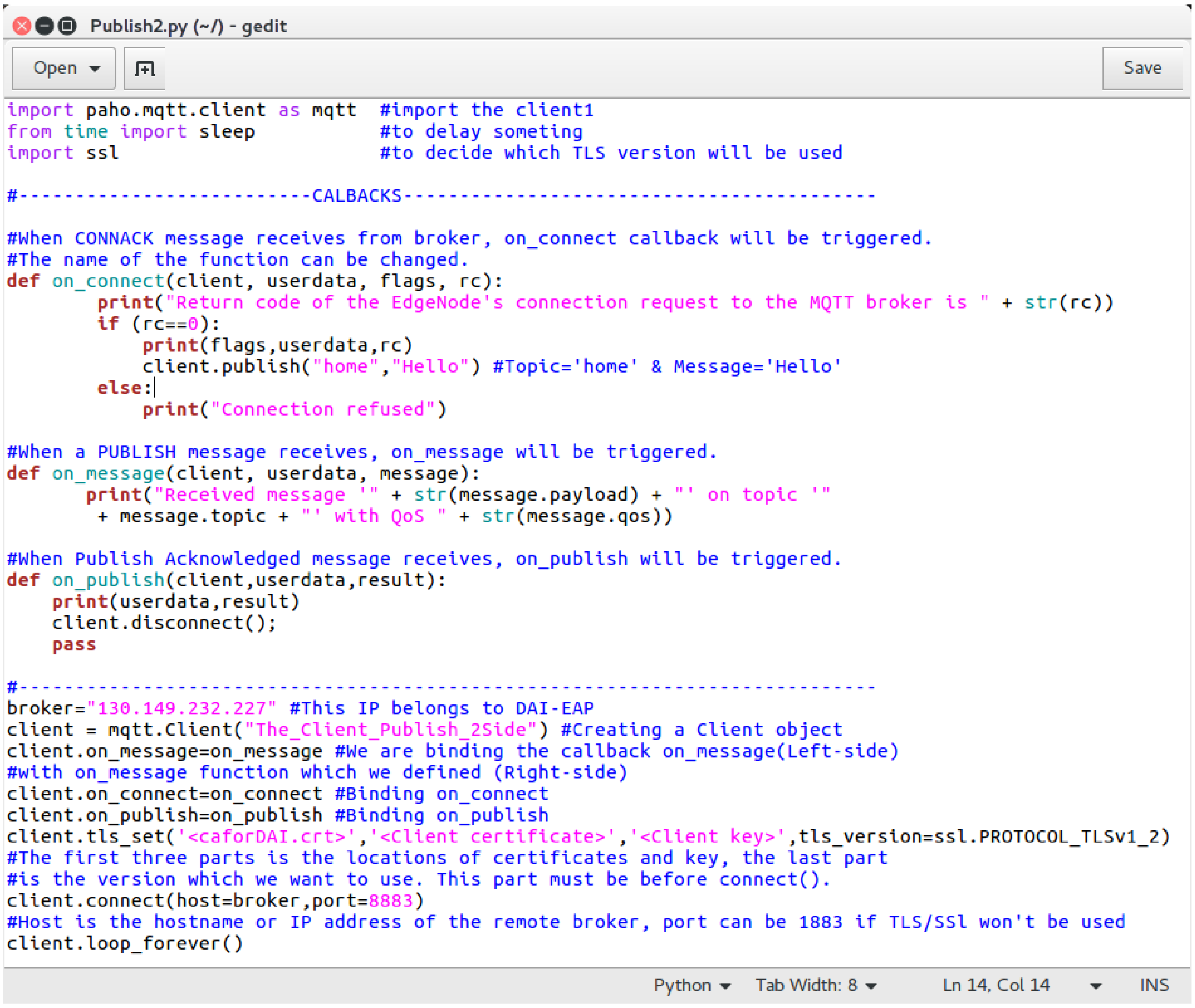
If one wants to communicate without using certificates, one have to omit the line **'client.tls_set(…)'** and use port 1883, by changing the port in **'client.connect(…)'**. **'client.tls_set(…)'** is used to implement TLS.

There are little differences between the codes. The codes written to publish have three parts, connection-publishing-disconnection. On the other hand, the codes written to subscribe have three parts, connection-subscription-disconnection (After receiving 3 messages). The only difference between Publish1.py (Python code written to publish by using one side authentication) and Publish2.py (Python code written to publish by using two side authentication) is specifying the locations of the client key and client certificate. In Publish1.py, only caforDAI.crt has been specified, whereas caforDAI.crt, client certificate and client key have been specified in Publish2.py. There is the same difference between Subscribe1.py (Python code written to subscribe by using one side authentication) and Subscribe2.py (Python code written to subscribe by using two side authentication).



```
import paho.mqtt.client as mqtt  #import the client1
from time import sleep           #to delay someting
import ssl                       #to decide which TLS version will be used

#--------------------------CALBACKS-----------------------------------------

#When CONNACK message receives from broker, on_connect callback will be triggered.
#The name of the function can be changed.
def on_connect(client, userdata, flags, rc):
        print("Return code of the EdgeNode's connection request to the MQTT broker is " + str(rc))
        if (rc==0):
            print(flags,userdata,rc)
            client.publish("home","Hello") #Topic='home' & Message='Hello'
        else:
            print("Connection refused")

#When a PUBLISH message receives, on_message will be triggered.
def on_message(client, userdata, message):
       print("Received message '" + str(message.payload) + "' on topic '"
        + message.topic + "' with QoS " + str(message.qos))

#When Publish Acknowledged message receives, on_publish will be triggered.
def on_publish(client,userdata,result):
    print(userdata,result)
    client.disconnect();
    pass

#---------------------------------------------------------------------------
broker="130.149.232.227" #This IP belongs to DAI-EAP
client = mqtt.Client("The_Client_Publish_1Side") #Creating a Client object
client.on_message=on_message #We are binding the callback on_message(Left-side)
#with on_message function which we defined (Right-side)
client.on_connect=on_connect #Binding on_connect
client.on_publish=on_publish #Binding on_publish
client.tls_set('<Location of caforDAI.crt>',tls_version=ssl.PROTOCOL_TLSv1_2)
#The first part is the location of caforDAI.crt, the last part
#is the version which we want to use. This part must be before connect().
client.connect(host=broker,port=8883)
#Host is the hostname or IP address of the remote broker, port can be 1883 if TLS/SSl won't be used
client.loop_forever()
```

Figure 9. The code to publish by using one side authentication (Publish1.py)

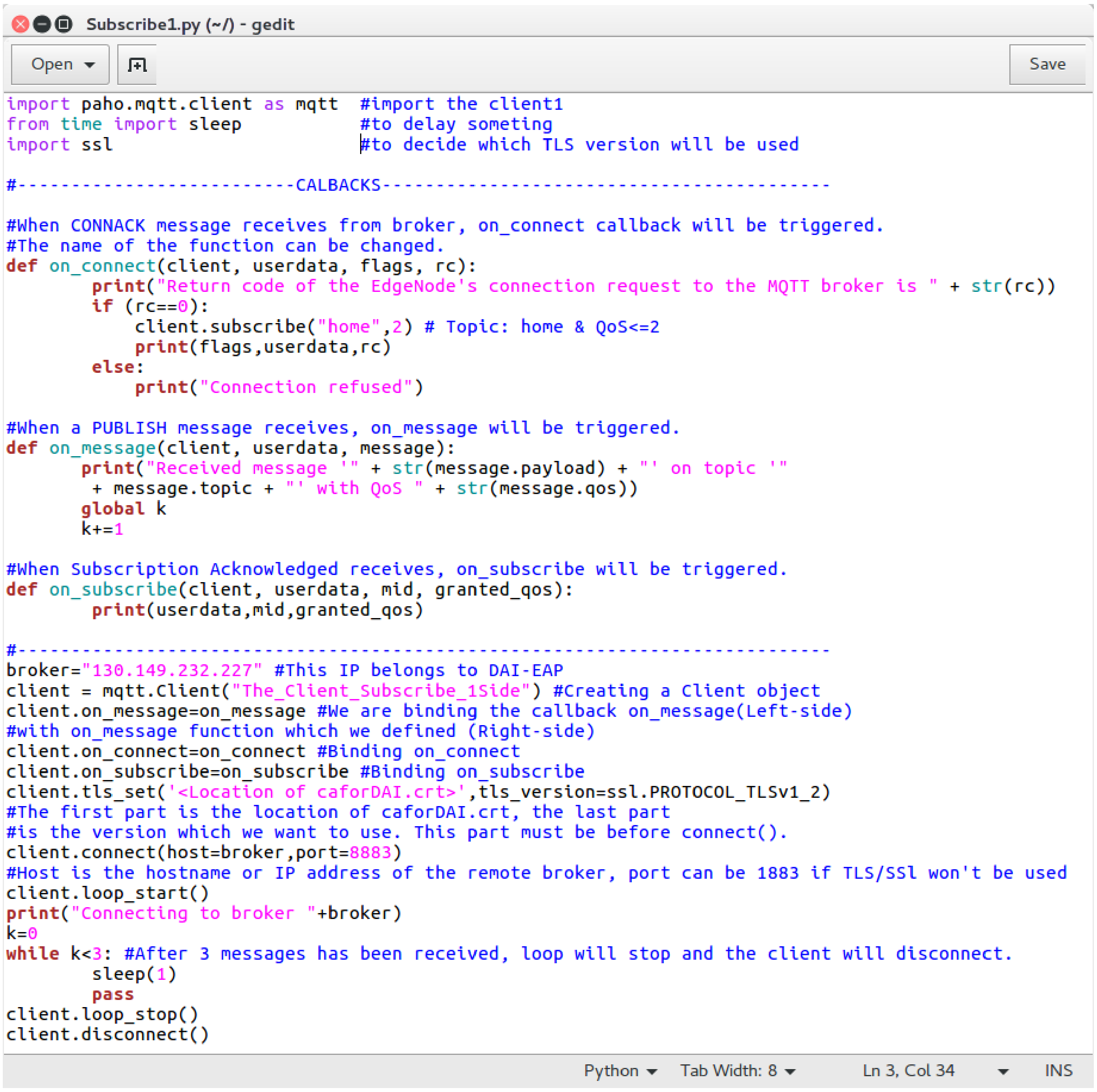

Publish2.py (~/) - gedit

Open | Save

```
import paho.mqtt.client as mqtt  #import the client1
from time import sleep           #to delay someting
import ssl                       #to decide which TLS version will be used

#-------------------------CALBACKS----------------------------------------

#When CONNACK message receives from broker, on_connect callback will be triggered.
#The name of the function can be changed.
def on_connect(client, userdata, flags, rc):
        print("Return code of the EdgeNode's connection request to the MQTT broker is " + str(rc))
        if (rc==0):
            print(flags,userdata,rc)
            client.publish("home","Hello") #Topic='home' & Message='Hello'
        else:
            print("Connection refused")

#When a PUBLISH message receives, on_message will be triggered.
def on_message(client, userdata, message):
       print("Received message '" + str(message.payload) + "' on topic '"
        + message.topic + "' with QoS " + str(message.qos))

#When Publish Acknowledged message receives, on_publish will be triggered.
def on_publish(client,userdata,result):
    print(userdata,result)
    client.disconnect();
    pass

#-------------------------------------------------------------------------------
broker="130.149.232.227" #This IP belongs to DAI-EAP
client = mqtt.Client("The_Client_Publish_2Side") #Creating a Client object
client.on_message=on_message #We are binding the callback on_message(Left-side)
#with on_message function which we defined (Right-side)
client.on_connect=on_connect #Binding on_connect
client.on_publish=on_publish #Binding on_publish
client.tls_set('<caforDAI.crt>','<Client certificate>','<Client key>',tls_version=ssl.PROTOCOL_TLSv1_2)
#The first three parts is the locations of certificates and key, the last part
#is the version which we want to use. This part must be before connect().
client.connect(host=broker,port=8883)
#Host is the hostname or IP address of the remote broker, port can be 1883 if TLS/SSl won't be used
client.loop_forever()
```

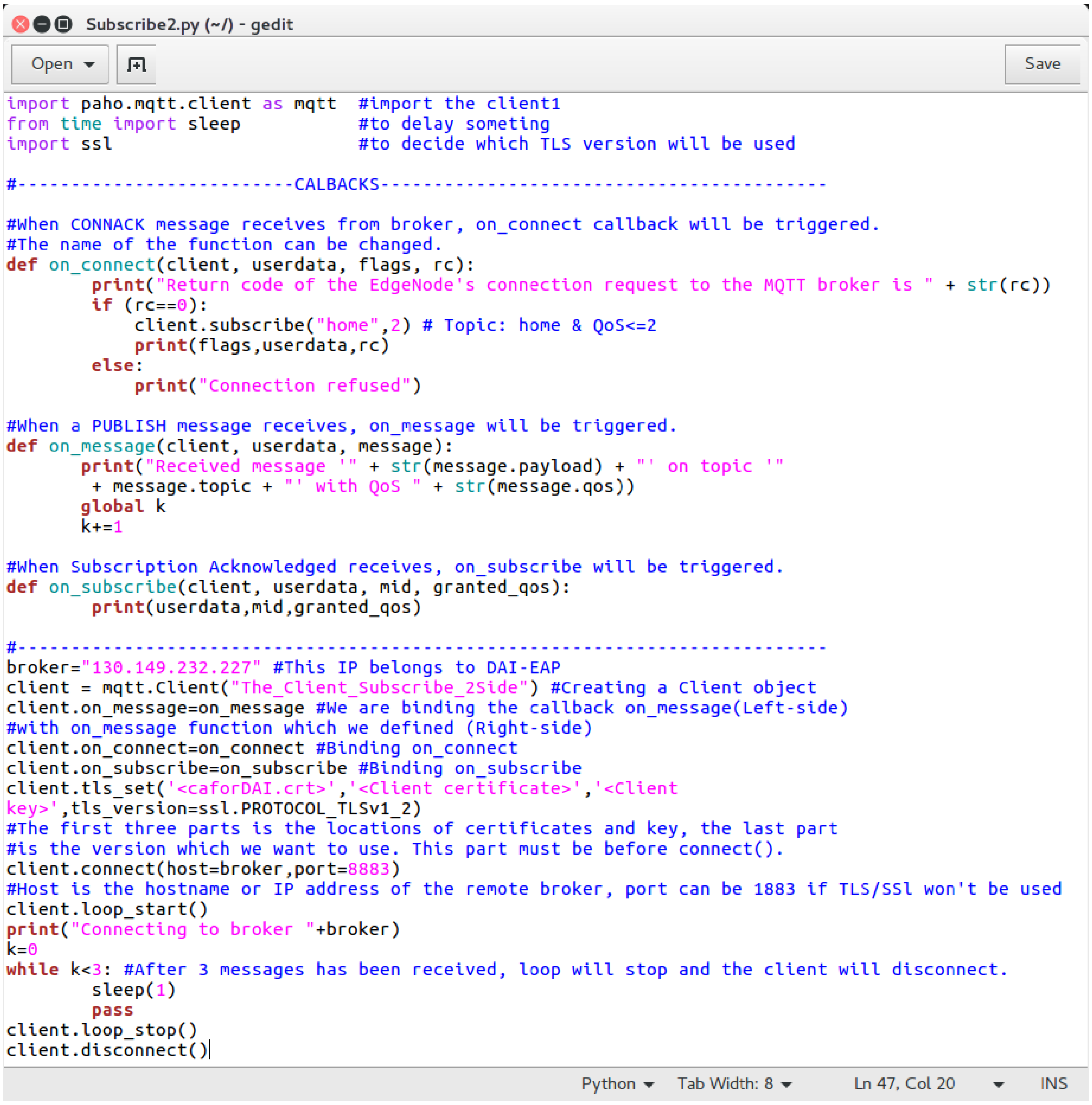
Python ▾ Tab Width: 8 ▾ Ln 14, Col 14 ▾ INS

Figure 10. The code to publish by using two side authentication (Publish2.py)



```python
import paho.mqtt.client as mqtt  #import the client1
from time import sleep           #to delay someting
import ssl                       #to decide which TLS version will be used

#-------------------------CALBACKS---------------------------------------

#When CONNACK message receives from broker, on_connect callback will be triggered.
#The name of the function can be changed.
def on_connect(client, userdata, flags, rc):
        print("Return code of the EdgeNode's connection request to the MQTT broker is " + str(rc))
        if (rc==0):
            client.subscribe("home",2) # Topic: home & QoS<=2
            print(flags,userdata,rc)
        else:
            print("Connection refused")

#When a PUBLISH message receives, on_message will be triggered.
def on_message(client, userdata, message):
       print("Received message '" + str(message.payload) + "' on topic '"
        + message.topic + "' with QoS " + str(message.qos))
       global k
       k+=1

#When Subscription Acknowledged receives, on_subscribe will be triggered.
def on_subscribe(client, userdata, mid, granted_qos):
        print(userdata,mid,granted_qos)

#-----------------------------------------------------------------------
broker="130.149.232.227" #This IP belongs to DAI-EAP
client = mqtt.Client("The_Client_Subscribe_1Side") #Creating a Client object
client.on_message=on_message #We are binding the callback on_message(Left-side)
#with on_message function which we defined (Right-side)
client.on_connect=on_connect #Binding on_connect
client.on_subscribe=on_subscribe #Binding on_subscribe
client.tls_set('<Location of caforDAI.crt>',tls_version=ssl.PROTOCOL_TLSv1_2)
#The first part is the location of caforDAI.crt, the last part
#is the version which we want to use. This part must be before connect().
client.connect(host=broker,port=8883)
#Host is the hostname or IP address of the remote broker, port can be 1883 if TLS/SSl won't be used
client.loop_start()
print("Connecting to broker "+broker)
k=0
while k<3: #After 3 messages has been received, loop will stop and the client will disconnect.
        sleep(1)
        pass
client.loop_stop()
client.disconnect()
```

Figure 11. The code to subscribe by using one side authentication (Subscribe1.py)



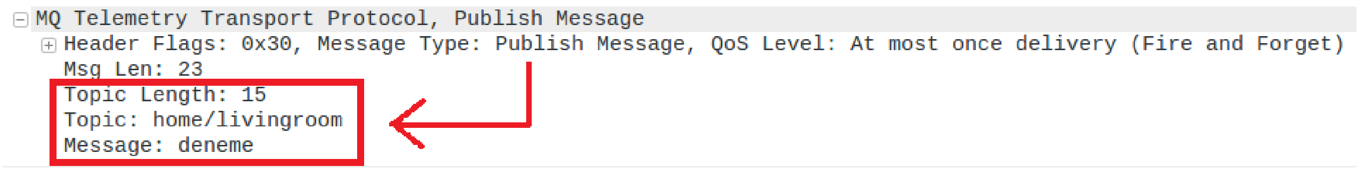
```
import paho.mqtt.client as mqtt  #import the client1
from time import sleep           #to delay someting
import ssl                       #to decide which TLS version will be used

#-------------------------CALBACKS-----------------------------------------

#When CONNACK message receives from broker, on_connect callback will be triggered.
#The name of the function can be changed.
def on_connect(client, userdata, flags, rc):
        print("Return code of the EdgeNode's connection request to the MQTT broker is " + str(rc))
        if (rc==0):
            client.subscribe("home",2) # Topic: home & QoS<=2
            print(flags,userdata,rc)
        else:
            print("Connection refused")

#When a PUBLISH message receives, on_message will be triggered.
def on_message(client, userdata, message):
       print("Received message '" + str(message.payload) + "' on topic '"
        + message.topic + "' with QoS " + str(message.qos))
       global k
       k+=1

#When Subscription Acknowledged receives, on_subscribe will be triggered.
def on_subscribe(client, userdata, mid, granted_qos):
        print(userdata,mid,granted_qos)

#-------------------------------------------------------------------------
broker="130.149.232.227" #This IP belongs to DAI-EAP
client = mqtt.Client("The_Client_Subscribe_2Side") #Creating a Client object
client.on_message=on_message #We are binding the callback on_message(Left-side)
#with on_message function which we defined (Right-side)
client.on_connect=on_connect #Binding on_connect
client.on_subscribe=on_subscribe #Binding on_subscribe
client.tls_set('<caforDAI.crt>','<Client certificate>','<Client
key>',tls_version=ssl.PROTOCOL_TLSv1_2)
#The first three parts is the locations of certificates and key, the last part
#is the version which we want to use. This part must be before connect().
client.connect(host=broker,port=8883)
#Host is the hostname or IP address of the remote broker, port can be 1883 if TLS/SSl won't be used
client.loop_start()
print("Connecting to broker "+broker)
k=0
while k<3: #After 3 messages has been received, loop will stop and the client will disconnect.
        sleep(1)
        pass
client.loop_stop()
client.disconnect()
```

Figure 12. The code to subscribe by using two side authentication (Subscribe2.py)

6. THE RESULTS OF THE IMPLEMENTATIONS

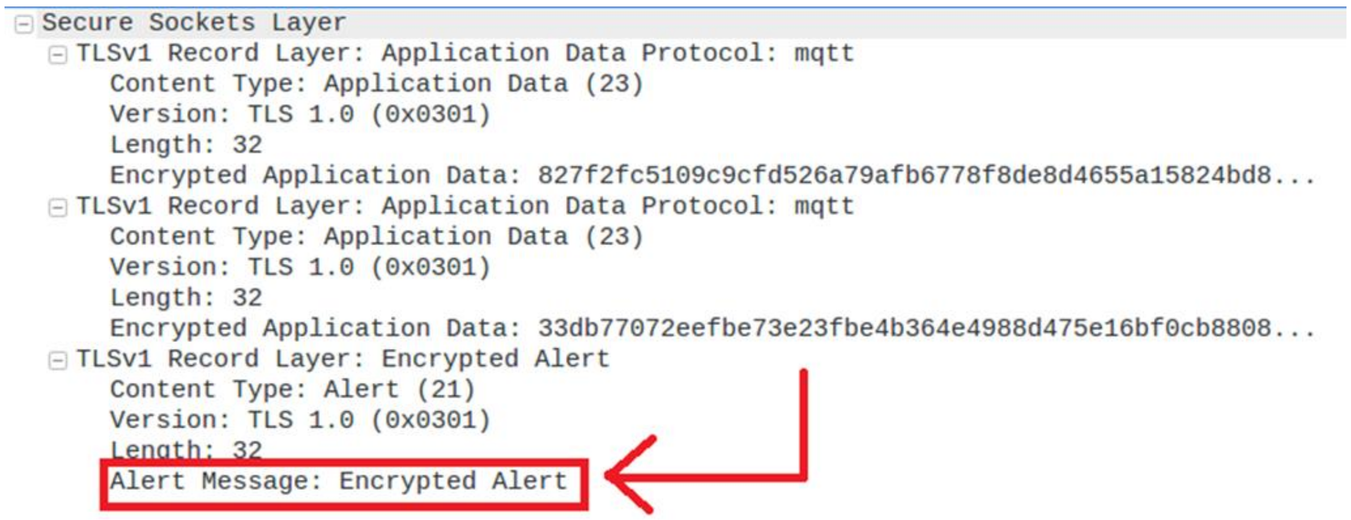
Up to now, I have written what I have learned to understand the fundamental concepts of IoT Mesh Network and TLS/SSL, and how I did the implementations of TLS/SSL over MQTT. After implementations, I have made some tests to see whether the implementations are successful or not. I have used Wireshark to see whether the message is encrypted, the steps of TLS/SSL, and the difference between the latencies of sending messages.

First of all, I wanted to test whether TLS/SSL is implemented successfully such that Wireshark cannot see the real message. First, I have sent a message 'deneme' in a specific topic 'home/livingroom' by using port 1883, i. e. , by not using authentication. Then, without changing the message and the topic, I have used port 8883 by using one side authentication. When I sent by not using authentication, MQTT was seen as the protocol which we use, and the message could be seen in Wireshark, as in Figure 13. However, when I used port 8883, TLS/SSL was seen as the protocol which we use, and the message was encrypted, as in Figure 14.



```
⊟ MQ Telemetry Transport Protocol, Publish Message
  ⊞ Header Flags: 0x30, Message Type: Publish Message, QoS Level: At most once delivery (Fire and Forget)
    Msg Len: 23
    Topic Length: 15
    Topic: home/livingroom
    Message: deneme
```

Figure 13. The characteristics of publishing message by not using authentication

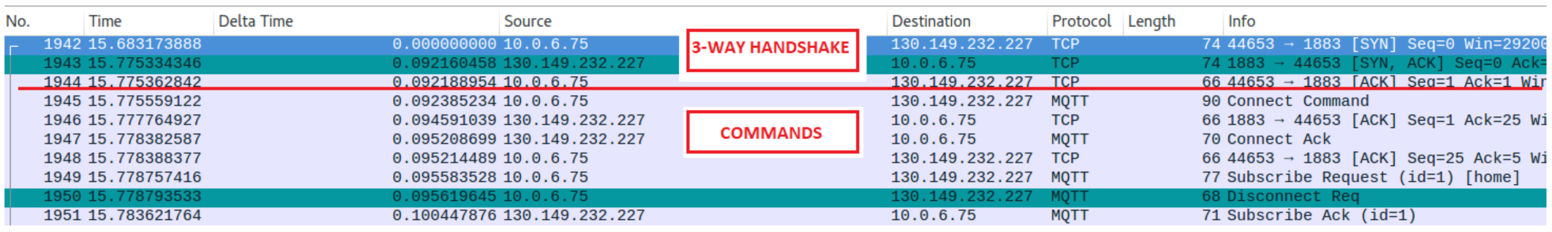


```
⊟ Secure Sockets Layer
  ⊟ TLSv1 Record Layer: Application Data Protocol: mqtt
      Content Type: Application Data (23)
      Version: TLS 1.0 (0x0301)
      Length: 32
      Encrypted Application Data: 827f2fc5109c9cfd526a79afb6778f8de8d4655a15824bd8...
  ⊟ TLSv1 Record Layer: Application Data Protocol: mqtt
      Content Type: Application Data (23)
      Version: TLS 1.0 (0x0301)
      Length: 32
      Encrypted Application Data: 33db77072eefbe73e23fbe4b364e4988d475e16bf0cb8808...
  ⊟ TLSv1 Record Layer: Encrypted Alert
      Content Type: Alert (21)
      Version: TLS 1.0 (0x0301)
      Length: 32
      Alert Message: Encrypted Alert
```

Figure 14. The characteristics of publishing message by using one side authentication

Then, I wanted to see whether the steps of TLS/SSL handshake are correct by using Wireshark. In this time, I have used Subscribe codes with a little difference. In these codes which I have written new, I set the codes such that when the client subscribed to a topic, it automatically disconnected from the broker.

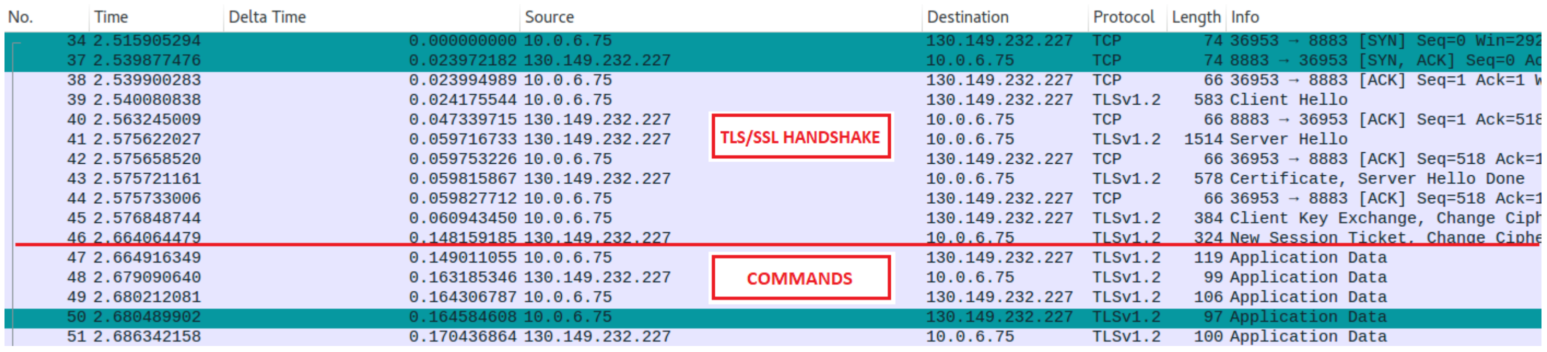
The steps of handshake and commands (Connect-Subscribe-Disconnect) for without authentication, one side authentication and two side authentication can be seen in Figure 15, Figure 16, and Figure 17, respectively.



No.	Time	Delta Time	Source	Destination	Protocol	Length	Info
1942	15.683173888	0.000000000	10.0.6.75	130.149.232.227	TCP	74	44653 → 1883 [SYN] Seq=0 Win=29200
1943	15.775334346	0.092160458	130.149.232.227	10.0.6.75	TCP	74	1883 → 44653 [SYN, ACK] Seq=0 Ack=
1944	15.775362842	0.092188954	10.0.6.75	130.149.232.227	TCP	66	44653 → 1883 [ACK] Seq=1 Ack=1 Wir
1945	15.775559122	0.092385234	10.0.6.75	130.149.232.227	MQTT	90	Connect Command
1946	15.777764927	0.094591039	130.149.232.227	10.0.6.75	TCP	66	1883 → 44653 [ACK] Seq=1 Ack=25 Wi
1947	15.778382587	0.095208699	130.149.232.227	10.0.6.75	MQTT	70	Connect Ack
1948	15.778388377	0.095214489	10.0.6.75	130.149.232.227	TCP	66	44653 → 1883 [ACK] Seq=25 Ack=5 Wi
1949	15.778757416	0.095583528	10.0.6.75	130.149.232.227	MQTT	77	Subscribe Request (id=1) [home]
1950	15.778793533	0.095619645	10.0.6.75	130.149.232.227	MQTT	68	Disconnect Req
1951	15.783621764	0.100447876	130.149.232.227	10.0.6.75	MQTT	71	Subscribe Ack (id=1)

Figure 15. The steps of subscribing by not using authentication

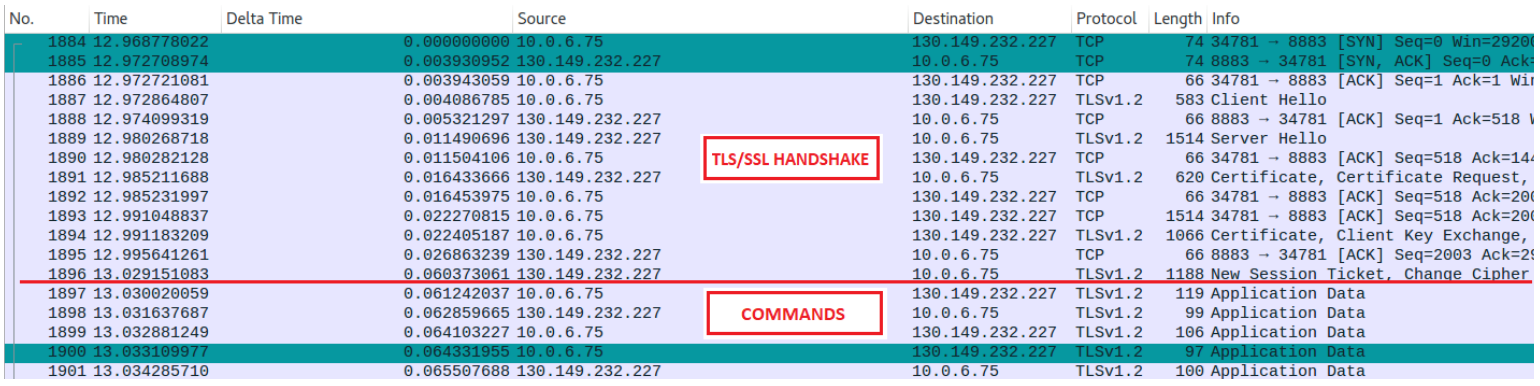
One can clearly see that there is 3-Way Handshake before the commands, above the red line. One can also see that the commands are visible, i.e., we can see the commands, what the aim of this code was, and the order of the commands.



No.	Time	Delta Time	Source	Destination	Protocol	Length	Info
34	2.515905294	0.000000000	10.0.6.75	130.149.232.227	TCP	74	36953 → 8883 [SYN] Seq=0 Win=292
37	2.539877476	0.023972182	130.149.232.227	10.0.6.75	TCP	74	8883 → 36953 [SYN, ACK] Seq=0 Ac
38	2.539900283	0.023994989	10.0.6.75	130.149.232.227	TCP	66	36953 → 8883 [ACK] Seq=1 Ack=1 W
39	2.540080838	0.024175544	10.0.6.75	130.149.232.227	TLSv1.2	583	Client Hello
40	2.563245009	0.047339715	130.149.232.227	10.0.6.75	TCP	66	8883 → 36953 [ACK] Seq=1 Ack=518
41	2.575622027	0.059716733	130.149.232.227	10.0.6.75	TLSv1.2	1514	Server Hello
42	2.575658520	0.059753226	10.0.6.75	130.149.232.227	TCP	66	36953 → 8883 [ACK] Seq=518 Ack=1
43	2.575721161	0.059815867	130.149.232.227	10.0.6.75	TLSv1.2	578	Certificate, Server Hello Done
44	2.575733006	0.059827712	10.0.6.75	130.149.232.227	TCP	66	36953 → 8883 [ACK] Seq=518 Ack=1
45	2.576848744	0.060943450	10.0.6.75	130.149.232.227	TLSv1.2	384	Client Key Exchange, Change Ciph
46	2.664064479	0.148159185	130.149.232.227	10.0.6.75	TLSv1.2	324	New Session Ticket, Change Ciphe
47	2.664916349	0.149011055	10.0.6.75	130.149.232.227	TLSv1.2	119	Application Data
48	2.679090640	0.163185346	130.149.232.227	10.0.6.75	TLSv1.2	99	Application Data
49	2.680212081	0.164306787	10.0.6.75	130.149.232.227	TLSv1.2	106	Application Data
50	2.680489902	0.164584608	10.0.6.75	130.149.232.227	TLSv1.2	97	Application Data
51	2.686342158	0.170436864	130.149.232.227	10.0.6.75	TLSv1.2	100	Application Data

Figure 16. The steps of subscribing by using one side authentication

One can clearly see that there is TLS/SSL Handshake for one side authentication before the commands, above the red line. Besides, one can see that the commands are encrypted in this case. It is written as 'Application Data'; however, the important information about the signal is not seen. One cannot say that this is a subscription, and the topic which the client subscribed. Only the one who has the key can decode the signal, andsee the important information. Moreover, one can see that TLSv1.2 is the protocol for communication, and also one can see that when we implement TLS/SSL over MQTT, the length of the signal increased 29 bytes if we compare with the length of the signal in MQTT since of the header length.

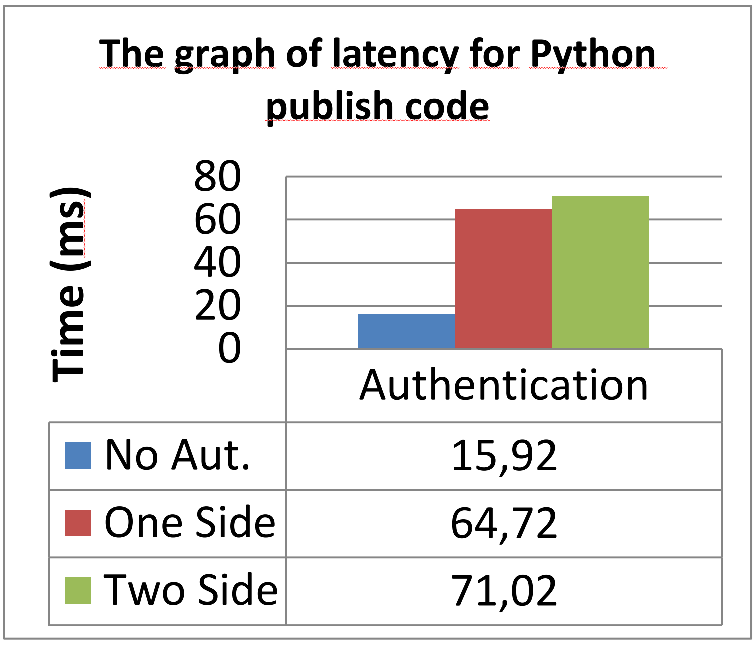


No.	Time	Delta Time	Source	Destination	Protocol	Length	Info
1884	12.968778022	0.000000000	10.0.6.75	130.149.232.227	TCP	74	34781 → 8883 [SYN] Seq=0 Win=2920
1885	12.972708974	0.003930952	130.149.232.227	10.0.6.75	TCP	74	8883 → 34781 [SYN, ACK] Seq=0 Ack
1886	12.972721081	0.003943059	10.0.6.75	130.149.232.227	TCP	66	34781 → 8883 [ACK] Seq=1 Ack=1 Wi
1887	12.972864807	0.004086785	10.0.6.75	130.149.232.227	TLSv1.2	583	Client Hello
1888	12.974099319	0.005321297	130.149.232.227	10.0.6.75	TCP	66	8883 → 34781 [ACK] Seq=1 Ack=518
1889	12.980268718	0.011490696	130.149.232.227	10.0.6.75	TLSv1.2	1514	Server Hello
1890	12.980282128	0.011504106	10.0.6.75	130.149.232.227	TCP	66	34781 → 8883 [ACK] Seq=518 Ack=14
1891	12.985211688	0.016433666	130.149.232.227	10.0.6.75	TLSv1.2	620	Certificate, Certificate Request,
1892	12.985231997	0.016453975	10.0.6.75	130.149.232.227	TCP	66	34781 → 8883 [ACK] Seq=518 Ack=20
1893	12.991048837	0.022270815	10.0.6.75	130.149.232.227	TCP	1514	34781 → 8883 [ACK] Seq=518 Ack=20
1894	12.991183209	0.022405187	10.0.6.75	130.149.232.227	TLSv1.2	1066	Certificate, Client Key Exchange,
1895	12.995641261	0.026863239	130.149.232.227	10.0.6.75	TCP	66	8883 → 34781 [ACK] Seq=2003 Ack=2
1896	13.029151083	0.060373061	130.149.232.227	10.0.6.75	TLSv1.2	1188	New Session Ticket, Change Cipher
1897	13.030020059	0.061242037	10.0.6.75	130.149.232.227	TLSv1.2	119	Application Data
1898	13.031637687	0.062859665	130.149.232.227	10.0.6.75	TLSv1.2	99	Application Data
1899	13.032881249	0.064103227	10.0.6.75	130.149.232.227	TLSv1.2	106	Application Data
1900	13.033109977	0.064331955	10.0.6.75	130.149.232.227	TLSv1.2	97	Application Data
1901	13.034285710	0.065507688	130.149.232.227	10.0.6.75	TLSv1.2	100	Application Data

Figure 17. The steps of subscribing by using two side authentication

One can clearly see that there is TLS/SSL Handshake for two side authentication before the commands, above the red line. As I stated earlier, the difference between TLS/SSL handshake for one side authentication and TLS/SSL handshake for two side authentication is that two side handshake has two more steps, which are certificate request of the client and sending client's certificate. These steps can be seen if one compares Figure 16 and Figure 17. As in one side authentication, the signals are encrypted and the length of the signal is bigger than the length of the signal when we do not use TLS/SSL.

Then, I wanted to see the difference between these 3 types of communication in terms of latency. I wrote a script which connects a broker, publishes a message and disconnects in every 3 seconds. My aim was seeing the average latency in 100 examples. While script is working, I have saved the latencies and I have calculated average latency for each type. The results can be seen in Figure 18.



	Authentication
🟦 No Aut.	15,92
🟥 One Side	64,72
🟩 Two Side	71,02

Figure 18. The graph of latencies for 3 types of authentication

One can see that the average latencies are 15,92 ms, 64,72 ms and 71,02 ms for without aut., one side aut. and two side aut., repectively. It is considerably difference between without authentication and with authentication latencies. However, the difference between one side aut. and two side aut. can be considered as small. As a result, if we want to use certificates, two side authentication is more preferrable than one side authentication.

7. CONCLUSION

I have completed my summer practice at German-Turkish Advanced Research Centre for ICT (GT-ARC) in Berlin, Germany. First of all, it was a wonderful experience to be in abroad. I have gone abroad for the first time. The good point is that the location was in Berlin where almost 300.000 Turkish people live,and there are a considerable amount of Turkish employees in the company. Therefore, we were not like in abroad. This helped me a lot to adopt the country and the city. Moreover, all of the employees were very helpful. They want us to improve ourselves and learn how a company works. My advisor even took us a match and taught us how to become a football scout.

Before I start my internship, my aim was gaining a knowledge on basic network theory and Internet of Things (IoT). My company asked me to learn fundemental concepts of network theory, TLS/SSL, and MQTT so that I can create secure communication. And I succeded in my goal. To reach my goal, I have learned Python programming language and Linux OS. I have faced with a lot of problems; however, it helps me a lot since I have learned how to solve a problem on my own. I can say that I have gained lots of experince during my summer practice.

After all, I recommend this summer SP location to other students, especially the ones who did not go abroad. It was a comperable long internship and wonderful experience. I have learned lots of technical and non-technical information. As a result, it was a nice 12-weeks internship.

8. REFERENCES

[1] Accessed on: Oct. 04, 2019. [Online]. http://www.gt-arc.com/?lang=en&data-slide=0

[2] Accessed on: Sept. 22, 2019. [Online]. Available: http://www.hivemq.com/blog/how-to-get-started-with-mqtt

[3] C. Hoffman, "What's the Difference Between TCP and UDP?," HowToGeek, 04-Jul-2017. [Online]. Available: https://www.howtogeek.com/190014/htg-explains-what-is-the-difference-between-tcp-and-udp/. [Accessed: 12-Oct-2019].

[4] Accessed on: Oct. 01, 2019. [Online]. Available: https://mosquitto.org

[5] "Creating and Using Client Certificates with MQTT and Mosquitto," Steves Internet Guide, 20-Sep-2019. [Online]. Available: http://www.steves-internet-guide.com/creating-and-using-client-certificates-with-mqtt-and-mosquitto/. [Accessed: 01-Oct-2019].