Journal of Engineering Research and Education
Volume 12, 2020 [43-54]

Real Time and Secure Messaging Service for IoT Applications using MQTT

MD Jiabul Hoque[1*], MD Akibur Rahman[1] and Shihab Uddin[1]

[1]*Department of Computer and Communication Engineering (CCE), International Islamic University Chittagong (IIUC), Kumira-4318, Chattogram, Bangladesh.*

ABSTRACT

The Internet of Things (IoT) is a system or framework of things or objects that have internet connectivity as well as that are interrelated and are capable to collect and exchange information without human intervention over a wireless network. Most of the IoT applications such as medical applications require real time and secure exchange of information among connected devices. Message Queuing Telemetry Transport (MQTT) is a communication protocol used for real time communication between networks with limited bandwidth as it is significantly lightweight. Even though security of using MQTT is still a matter of concern for using it in IoT network, MQTT is still prime choice for IoT communication protocol as there is no alternative has yet been developed. Authors of this article have reviewed numerous published researches on secure MQTT protocol for IoT network and discover some alarming security loop holes of MQTT communication protocol on IoT application network that need to be addressed. The prime aim of this work is to propose a secure and real time MQTT protocol for IoT application without compromising any loss of data. The research has been implemented in Raspberry Pi 3B system using Python 3.4.10 development platform along with Numpy 1.11.1 and scipy 0.18.0 (for mathematical analysis), paho-mqtt 1.5.0 (for MQTT publication/subscription), and Chromium (for displaying research result). The output of this research shows that the proposed MQTT protocol has tighten security during exchanging information over IoT network without any loss of data.

Keywords: Internet-of-Things, MQTT, Real-time, Raspberry-Pi, Secure.

1. INTRODUCTION

IoT has been revealed a hot topic of research and innovation around the world right now. Internet of Things applications in various fields for instance medical, agriculture, food, production, horticulture, space, mining and many more are getting popular as they are making human being life easy, simple, smart, safe, relaxing and secure (Ahmed, Rahman & Hoque, 2020). Therefore, numerous entities incorporating research scholars, public and private sector entities are involving with IoT related research in order to generate state of the art applications of it to grab billions (Ahmed *et al*., 2018). As people are using IoT applications such as home automation, biomedical devices, automatic greenhouse for agriculture or smart farming etc., security as well as loss of information is a growing matter of concern right now (Hakim, Uddin & Hoque, 2020). Researchers around the world are working on how to make this essential IoT mechanism safe and secure as majority of the IoT applications require real time and secure communication for instance medical applications (Hoque, Ahmed & Hannan, 2020).

MQTT has been used as a de facto standard for coordinating communication of IoT network applications for long time. Even though it has some security issues, no other alternative has yet been developed (Hoque, Kabir & Hossain, 2018). So, researchers are concentrating on how to

*Corresponding Author: *jiabul.hoque@iiuc.ac.bd*

make MQTT communication more secure than ever to ensure trustworthiness to IoT applications user. It has been appeared that security can be compromised not only on client side but also on broker of MQTT infrastructure and hence issues of security must be considered in both ways (Kabir *et al*., 2019).

Internet of Things and MQTT both are took off now. Mosquitto is the first MQTT broker which is open source. It was created around 2008. And in 2014 it became the name of Eclipse Mosquitto project (Hoque *et al*., 2020). The open source MQTT client libraries were published in 2012 as Paho project C, Python, JavaScript and Java. And since then it is growing day by day. The most remarkable things are the broker version 3.1.1 becomes OASIS standard in late 2014 and MQTT turn into an ISO standard in 2016 (Sharma *et al.*, 2018).

The main objective of this article is to propose a secure and real time MQTT protocol for IoT application without compromising any loss of data. In this regards, a clear and step by step procedure for securing MQTT has been presented in this paper. The procedure has been begun by tightening usage control in MQTT communication protocol. This step has been done by continuously observing mutable attributes related to data, the environment or the subscriber itself for the purpose of imposing the constraint on subscriber's rights to access information. The research has been implemented in Raspberry Pi 3B system using Python 3.4.10 development platform along with Numpy 1.11.1 and scipy 0.18.0 (for mathematical analysis), paho-mqtt 1.5.0 (for MQTT publication/subscription), and Chromium (for displaying research result). The output of this research shows that the proposed MQTT protocol has tighten security during exchanging information over IoT network without any loss of data.

After the introductory section, the rest of article is organized as follows: Literature review sections begin right after this section that reviews existing works relevant to home automation system. After rigorous reviews of existing pertinent works, authors identified some short coming on existing literatures that need to be resolved for better home automation system. Section 3 provides the solutions of the problems identified in literature review section by designing a state of the art home automation system. Section 4 illustrates the implementation process and subsequent section presents conclusion and future direction of this work.

2. LITERATURE REVIEW

Author has reviewed substantial amount of literature related to MQTT based IoT communication. Among them few notable literatures that cannot be missed out are presented in this section.
According to Bansal & Garg (2019), MQTT is a lightweight real time transmission protocol that is fully adaptable to emergency services like vehicle accidental notification system. They used vibration sensors, Node MCU, Adafruit cloud, IFTTT applet to send notification regarding vehicle accident via SMS in their research on MQTT. This research has been used as a baseline for our research which is a real time and emergency communication through all kind of IoT application without any data loss.

Singh *et al.* (2015) presented a secure version of MQTT communication named SMQTT-SN in their research on secure MQTT for IoT applications. In their work they have incorporated some added security features such as lightweight Elliptic Curve Cryptography based policy attribute encryption on MQTT communication that enhanced security. However, some security flaws such as key revocation during group subscription and publication are big concern for SMQTT-SN that has been identified during implementation stage of their research.

Lee *et al.* (2019) analyzed the relationship between delay and associated loss of data according to QoS measure in their recent work on MQTT. Rigorous analysis has been done by the researchers on MQTT communication in this paper that incorporate subscribe client, publish

client (both wired and wireless) and broker server etc. The outcome of this research depicted that message loss under varying payload has been significant impacted by end-to-end delay in communication. However, the behaviour under numerous Quality of Service measure is remain in doubt.

In the scenario of Wireless Sensors Network (WSN) based IoT infrastructure where a middleware is required between sensors and server for flawless MQTT communication. A recent study on performance analysis between MQTT and Constrained Application Protocol (CoAP) using a common middleware in the case of IoT application shows that MQTT communication has significantly low data loss and negligible delay in communication compare to its counterpart (Upadhyay, Borole & Dileepan, 2016). A similar trend has been reveal in another research on health information sharing using IoT where MQTT come out as a clear winner in case of WSN based IoT applications network (Katsikeas, 2017).

3. MATERIAL AND METHODS

In this section, the author has described the components required to test the proposed MQTT protocol as well as the author presented methodology that has been followed for successful research outcome.

3.1 Hardware and Software Components

The research has been implemented in Raspberry Pi 3B incorporating 1.2GHz 64-bit quad-core Arm Cortex-A53 CPU with 1GB RAM and Raspbian operating system has been used to run the system.

Besides, Python 3.4.10 has been used as premier development platform along with Numpy 1.11.1 and scipy 0.18.0 for mathematical analysis as well as paho-mqtt 1.5.0 has been used for MQTT publication/subscription. An MQTT mosquitto broker has also been installed within raspberry pi. The research outcome has been displayed graphically by the use of Chromium.

3.2 Proposed Architecture of MQTT

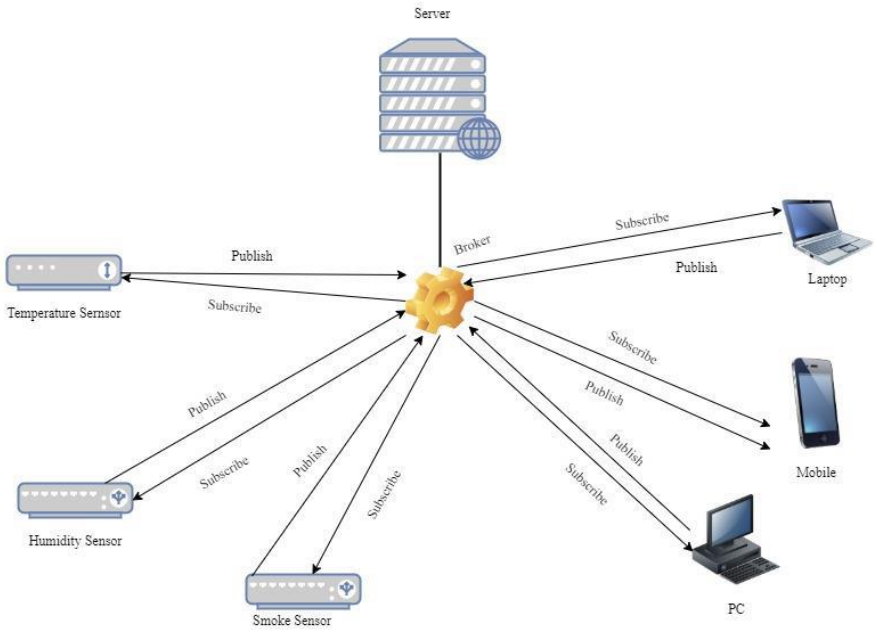

Figure 1. The proposed architecture of MQTT.

Above diagram (Figure 1) illustrates the proposed MQTT architecture where a client is regarded as any electronic medium that is able to communicate with broker. In the proposed system few (PC, Mobile, Laptop, numerous sensors and even a server) of many such devices has been projected. The responsibilities of a broker in MQTT architecture are immense for instance it needs to be active all the time to coordinate the communication with its connected clients. The coordination begins by identifying clients and then authorizing those clients by using secure authentication method. Afterwards, broker coordinates messages that have been received from various clients and publish those messages that have been received from subscribe clients. There is only one constraint in the MQTT architecture which is a broker only communicates one client at a given time while maintaining connection with all.

3.3 Basic MQTT Configuration

The method of this proposed MQTT secure communication system in IoT applications start with configuring Mosquitto server as Mosquitto MQTT broker has been used as out-broker. First of all, status of the broker need to check as broker must be up and running to test the proposed system. Figure 2 shows the status of the Mosquitto MQTT broker used in this system:

```
pi@raspberrypi:~ $ systemctl status mosquitto.service
● mosquitto.service - Mosquitto MQTT v3.1/v3.1.1 Broker
   Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset
   Active: active (running) since Sat 2020-01-18 18:00:00 +06; 1 weeks 0 days ag
     Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
 Main PID: 520 (mosquitto)
    Tasks: 1 (limit: 2200)
   Memory: 2.2M
   CGroup: /system.slice/mosquitto.service
           └─520 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf
```

Figure 2. Mosquitto MQTT broker status.

After confirming broker status as active (running), the status of WebSocket connection is needed to checked. WebSocket retrieves the values from mqtt.js file as such connect() function inside mqtt.js file is required to be configured. Figure 3 depicts the configuration variables of connect() function:

```
function connect() {
    console.log("Connecting to server")
    var hostname = "192.168.0.111";
    var port = "9001";
    clientId = "attendanceWeb_" + makeid() + String(Date.now());
    var keepAlive = Number(10);
    var timeout = Number(3);
    client = new Paho.MQTT.Client(hostname, Number(port), clientId);
    client.onConnectionLost = onConnectionLost;
    client.onMessageArrived = onMessageArrived;
    var options = {
        invocationContext: {host: hostname, port: port, path: client.path, clientId:
        clientId},
        timeout: timeout,
        keepAliveInterval: keepAlive,
        useSSL: true,
        cleanSession: true,
        reconnect: true,
        onSuccess: onConnect,
        onFailure: onFail
    };

    client.connect(options);
```

Figure 3. Connect() function configuration.

Connect() function in the above Figure 3 has number of variables such as hostname, port number, session and connection etc that are responsible for successful connection. After successful configuration of mqtt.js file, the status of WebSocket is checked this is shown in the figure 4 below:

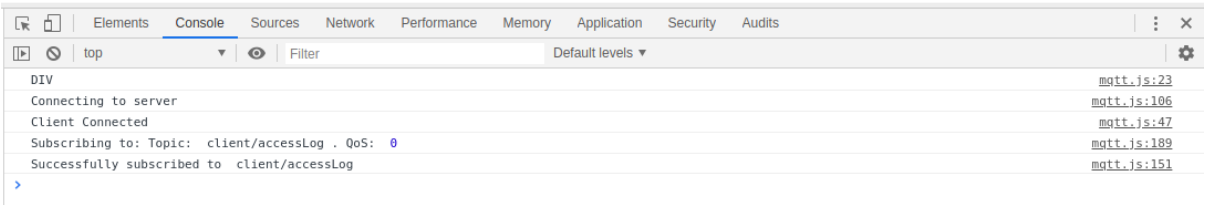

Figure 4. Status of WebSocket.

The status of WebSocket in the above figure 4 shows that the clients are connected to the server and are ready to communicate.

3.4 Secure MQTT Configuration

There are mainly three security mechanisms such as authentication, identity and authorization should be kept in any researchers mind when it relates to sensitive communication in real time using IoT infrastructure. There are two ways that can be used to make IoT network secure such as Username and password approach and SSL approach. In order to make secure MQTT communication throughout IoT network, username and password authentication can be added initially and on top of that SSL certificate can be implemented.

3.4.1 Username and password approach

Using authentic username and password, the mosquitto MQTT broker can be configured as to require client authentication before broker connection established or permitted. Even though, a clear text username and password combination is not secure without transport encryption like ssl, it is good first step to restrict access to a broker easily using username and password authentication. The restrictions forms are client id, topic, QoS, username/password etc which all are implemented in this MQTT broker. After implementing all the credentials to broker, it is now task for client to add all these credentials to connect, publish and subscribe with maintaining all the restrictions.

For configuring the Mosquitto broker, there are some steps need to be done such as a file for storing password need to be created as well as mosquito.conf file need to be reconfigured to force to use password from a password file that has been created earlier.

3.4.1.1 Creating and using a password file

To create the password file, mosquitto_passwd utility file which has been install during the installation of mosquito broker need to be used. There are numerous ways to create password file however the following way is the best procedure (terminal command) that is used for this system.

mosquitto_passwd -b passwordfile secure secureconnection

The following figure 5 depicts the creation of a password file.

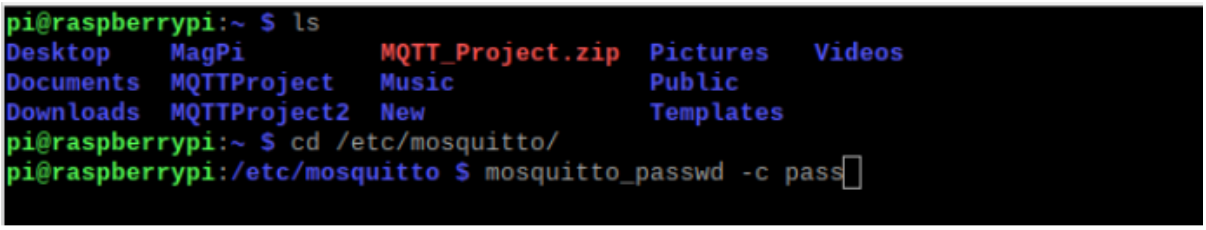

Figure 5. Creation of password file.

Figure 6 shows the status of secure password.

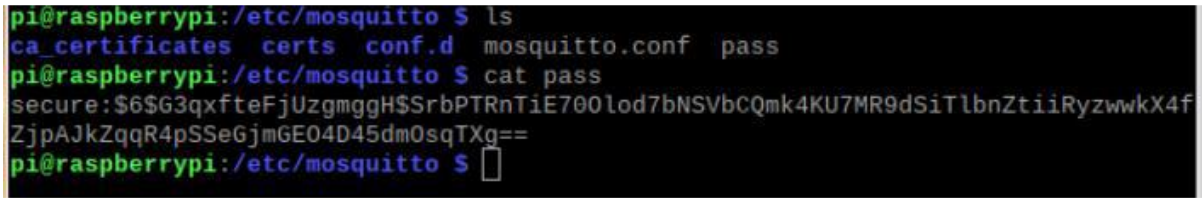

Figure 6. Mosquitto secure password.

Now it is the time to use the password file through mosquito.conf file. At first, in the Raspberry Pi the password file that has been created earlier needs to be copied to the directory etc/mosquitto. Afterwards, mosquitto.conf file needs to be reconfigured in such a way that MQTT communication use password file to make the communication secure. There are two changes have been made. The changes made to mosquitto.conf file are setting the password file path and also setting allow anonymous to false cause by default it remains true.
Settings of mosquitto.conf:

password_file /etc/mosquitto/password.txt
allow_anonymous false

After creating a password file and reconfiguring mosquitto.conf file, mosquitto broker needs to be restarted in order to make the effect of such changes. However, in Raspberry Pi, without restarting broker the file can be reloaded using the following command:

kill-HUP 519

After executing the above command, the terminal shows that moquitto.conf file is reloaded. Figure 7 depicts the console log status:

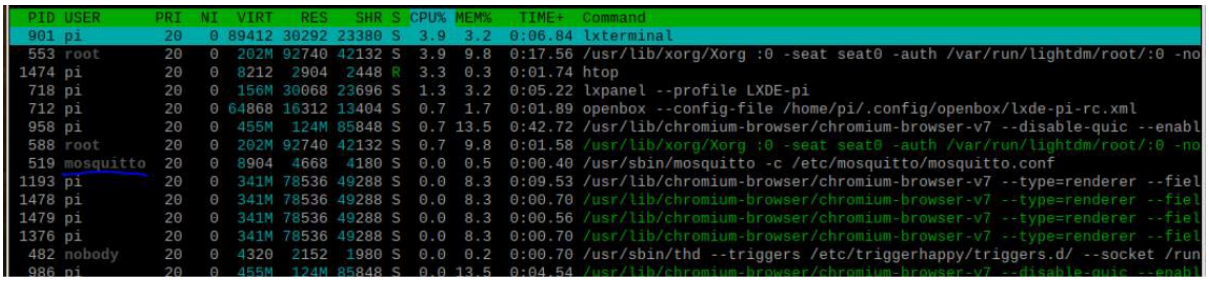

Figure 7. Console log status.

After successfully setting the username and password to MQTT broker, username and password is added to mqtt.js file for secure and authenticated connection of WebSocket. Figure 8 shows secure WebSocket connection configuration.

```
function connect() {
    console.log("Connecting to server")
    var hostname = "192.168.0.111";
    var port = "9001";
    clientId = "attendanceWeb_" + makeid() + String(Date.now());
    var user = "secure";
    var passw = "secureconnection";
    var keepAlive = Number(10);
    var timeout = Number(3);
    client = new Paho.MQTT.Client(hostname, Number(port), clientId);
    client.onConnectionLost = onConnectionLost;
    client.onMessageArrived = onMessageArrived;
```

Figure 8. Secure WebSocket connection configuration.

3.4.2 SSL Approach

In this approach, OpenSSL must be installed to generate all certificate files using Raspberry Pi. To create SSL certificates, first step is to connect Raspberry Pi using ssh. And then create a private key using following command:

openssl genrsa -out mosq-ca.key 2048

The command above will create a 2048-bit key called mosq-ca.key. And the result of this command is showing the figure 9 below:

```
pi@raspberrypi:~/certs2 $  openssl genrsa -out mosq-ca.key 2048
Generating RSA private key, 2048 bit long modulus
.................................+++
.....................+++
e is 65537 (0x10001)
```

Figure 9. Creation of 2048 bit private key.

After creating the private key, the next task is to create an X509 certificate which will use the private key that has been created before. Now after opening another terminal (directory must be the same), the private key can be kept by writing the following code:

openssl req -new -x509 -days365 -key mosq-ca.key -out mosq-ca.crt

The following figure 10 shows the terminal after submitting the command:

```
pi@raspberrypi:~/certs2 $ openssl req -x509 -new -key mosq-ca.key -days 365 -out mosq-ca.crt
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IT
State or Province Name (full name) [Some-State]:Perugia
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:mosquitto_server
Email Address []:
```

Figure 10. Key certificated

Now, it is the time to move on to create MQTT server certificate by creating a CSR (Certificate Signing Request). Usually, before using the newly created certificate it is required to send certification authority for verification. Here, self-signed certificate has been used to avoid such complexity. The following commands are used to create the certificate and required credentials to use in our server:

```
openssl req -new -key mosq-serv.key -out mosq-serv.csr
openssl x509 -req -in mosq-serv.csr -CA mosq-ca.crt -CAkey mosq-ca.key -CAcreateserial
-out mosq-serv.crt -days 365 -sha256
```

3.4.2.1 Securing MQTT Mosquitto server

After creating secure private certificates, it is time to secure MQTT mosquitto server for secure communication throughout IoT network. In our system, three secure certificates listed below have been used:

1. mosq-ca.crt
2. mosq-serv.crt
3. mosq-serv.key

In order to ensure SSL certificate to the mosquitto broker, few lines of code from mosquitto.conf file are needed to be edited. The lines are given below:

```
listener 1883
cafile /home/pi/ssl-cert-mosq/mosq-ca.crt
certfile /home/pi/ssl-cert-mosq/mosq-serv.crt
keyfile /home/pi/ssl-cert-mosq/mosq-serv.key
```

Here, default MQTT mosquitto port has been changed to 1883 and private certificates have been kept to the path /home/pi/ssl. Now, it's time to restart the mosquitto service by using following two commands:

```
sudo service mosquitto stop
sudo service mosquitto start
```

After successfully setting up SSL certificate to MQTT broker, some security options of mqtt.js file must be changed for secure WebSocket connection with web page to broker. Following figure 11 shows the secure WebSocket connection.

```
var options = {
    invocationContext: {host: hostname, port: port, path: client.path, clientId:
    clientId},
    timeout: timeout,
    keepAliveInterval: keepAlive,
    useSSL: true,
    cleanSession: true,
    reconnect: true,
    onSuccess: onConnect,
    onFailure: onFail
};
```

Figure 11. Secure WebSocket Connection.

Finally, the procedure to make secure and encrypted MQTT protocol for securing communication throughout IoT network has been completed with expected result.

4. RESULTS AND ANALYSIS

At first, it is essential to check whether files required to test our secure MQTT protocol is loaded or not with minimal delay. Following figure 12 depicts files along with their size and loading time requires testing the proposed system. Here, the loading time might vary depending on the speed and performance of the testing server.

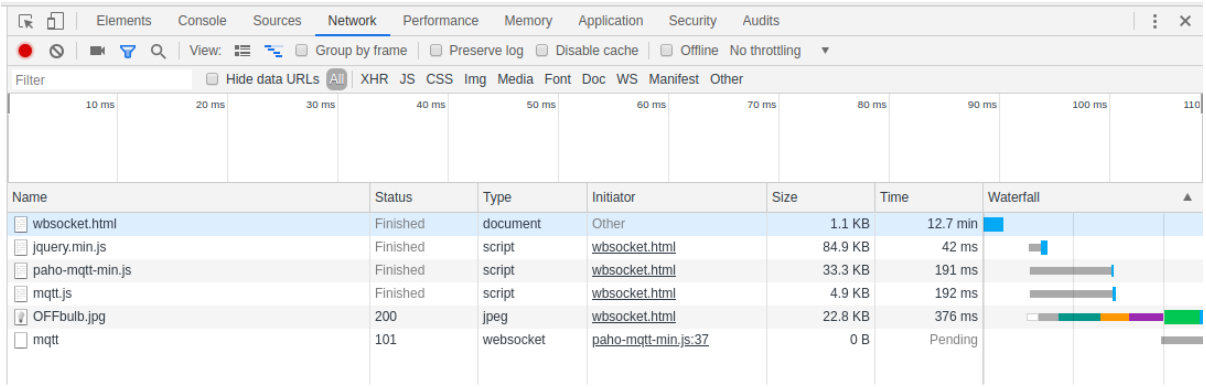

Figure 12. Loading dependency files.

To test the MQTT server, it is required to have a client from which secure communication can be made with MQTT server. As such, a Java base MQTT client (MQTT.fx) has been install in our windows operating system based client computer. After that the settings and connection parameters of client have been changed as well as MQTT mosquitto has been configured to secure MQTTprotocol with necessary information as shown in the figure 13 below:

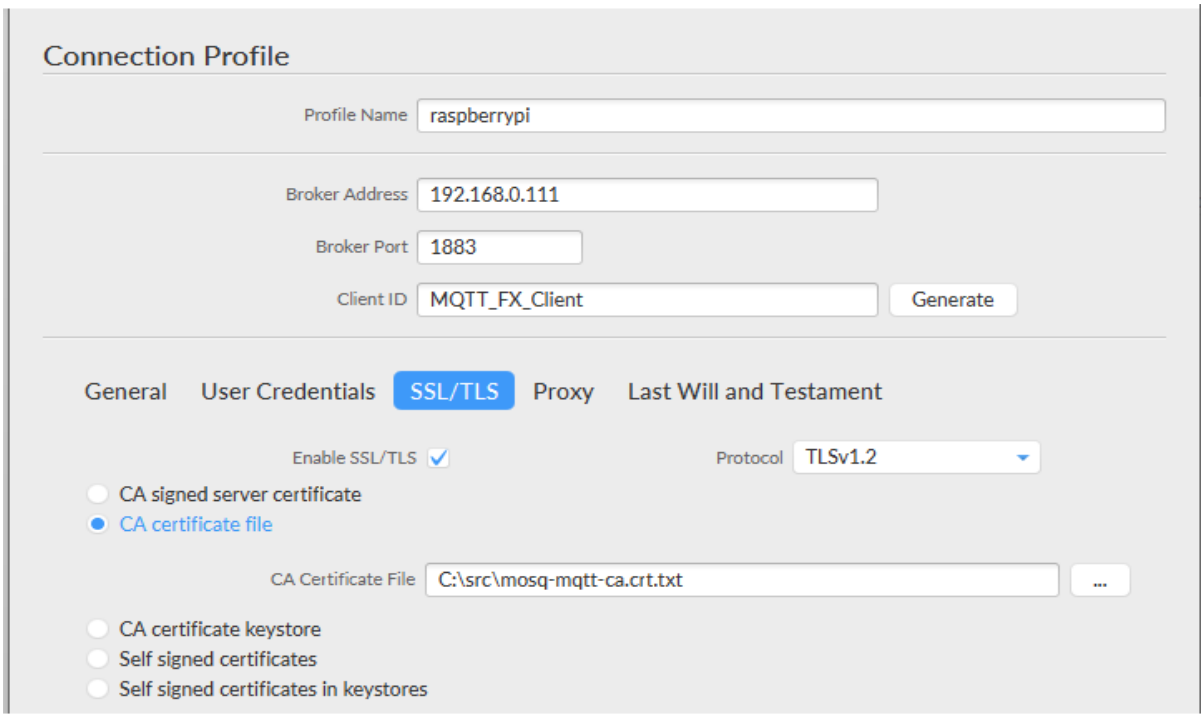

Figure 13. Configure MQTT Mosquitto Server to Secure MQTT.

It can be seen from the figure 13 that the information about profile name, broker address, broker port number and client ID has been provided. Besides, SSL/TSL configuration has been enabled by providing the certificate files like mosq-ca.crt in previous steps.

Now secure connection can be made between MQTT server and client by clicking simply connect button as shown in figure 14:

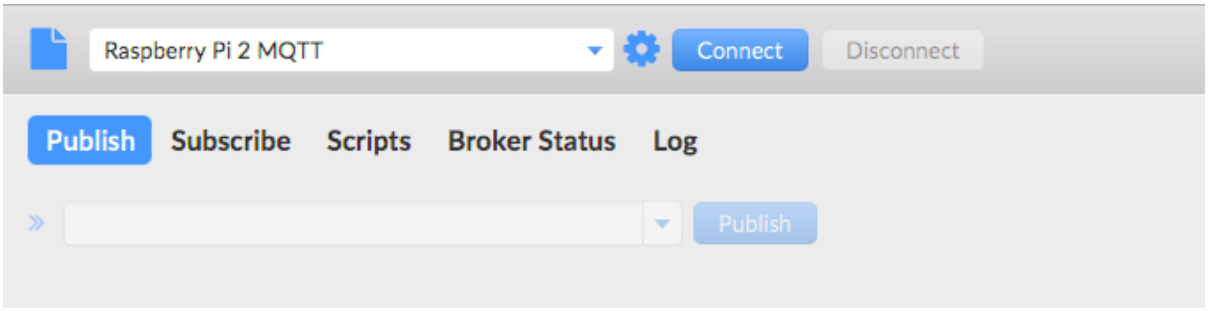

Figure 14. Secure MQTT connection.

The status of the connection between client and broker can be seen by clicking the Log tab shown in figure 14. The following figure 15 depicts the connection status along with the size of sending packets.

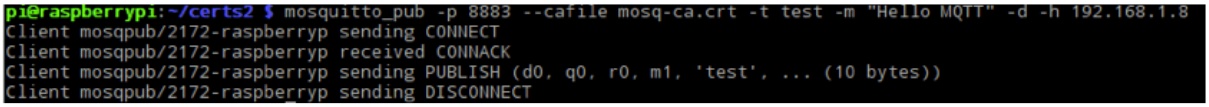

Figure 15. Connection status.

Now it is time to check the status from subscribe side which is located in another computer. Figure 16 shows the status of the subscriber side:

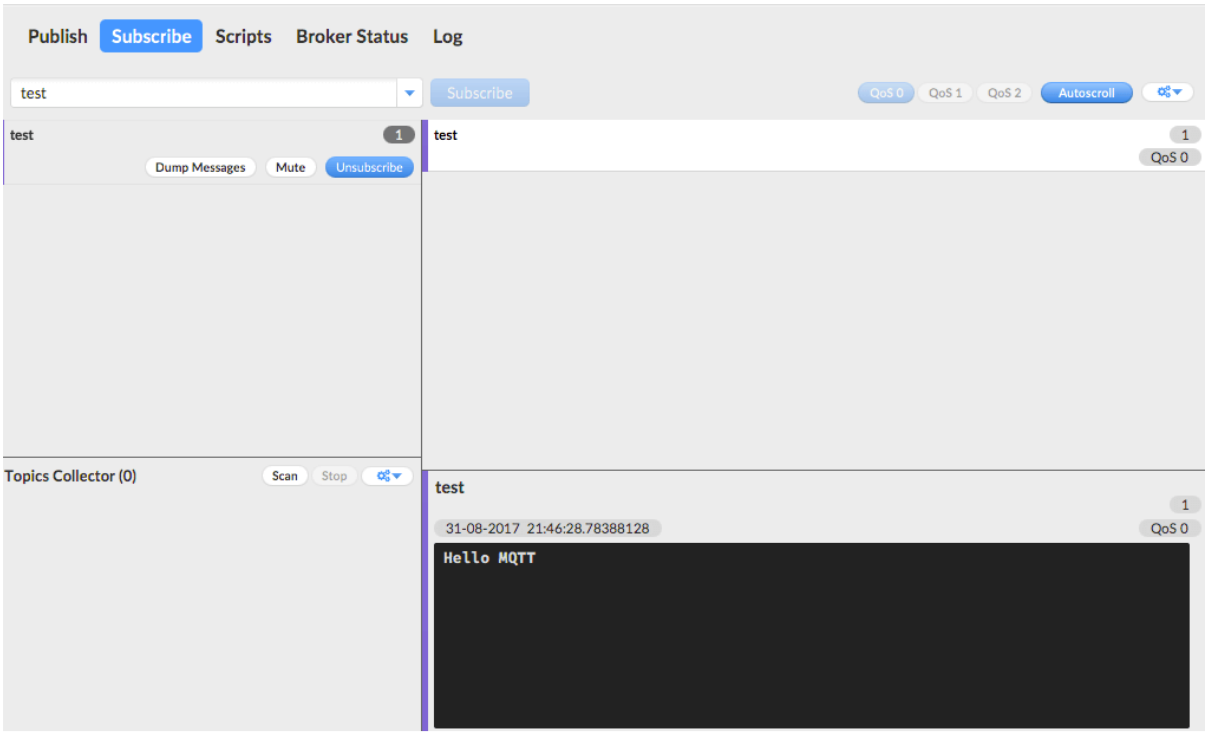

Figure 16. Subscriber status.

Next approach is testing MQTT connection over port 1883. When there is no authentication set, any user from any device can connect to this device. But after setting authentication and ssl

certificate, it becomes difficult to connect to MQTT broker without knowing the dependencies. After trying from other devices without proper dependencies, the message appears as shown in Figure 17.

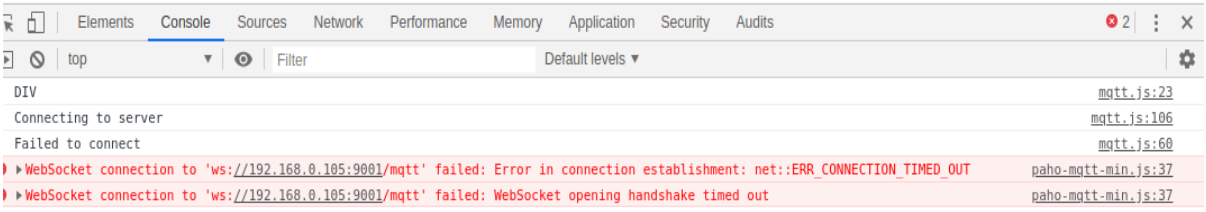

Figure 17. SSL WebSocket Status without dependencies.

After adding all the dependencies and ssl certificate, the console showing that the WebSocket successfully connected to topic: client/accessLog that is shown in figure 18.

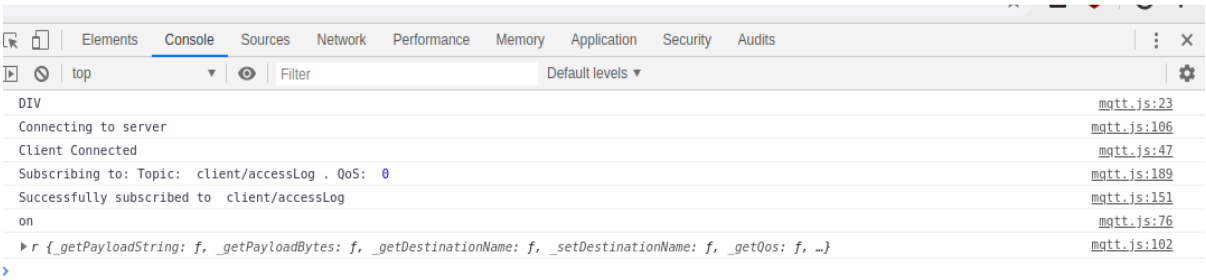

Figure 18. SSL WebSocket Status .

5. CONCLUSION

In this paper, authors have proposed a secured and automated messaging system using MQTT protocol that takes message from other devices such as sensor, mobile, laptop and accepts only the ones with proper authenticity. This work specifically demonstrated an application of the system that can successfully communicate securely with IOT support devices without compromising any data loss. An experimental framework has been developed using a broker, some python and JavaScript scripts with associated supportive software tools to evaluate the performance of the proposed system. Besides, a hypothetical webpage were used as sample output device. The performance of the system was then evaluated in term of accuracy, time and security of communication system. The outcome of this paper will surely make IoT communication in real time and make more secure.

REFERENCES

[1] Ahmed, M., Rahman, M. & Hoque, M. Smart Home: An Empirical Analysis of Communication Technological Challenges. European Journal of Engineering Research and Science. **5**, 5 (2020) 571-575. doi: https://doi.org/10.24018/ejers.2020.5.5.1905.

[2] Ahmed, Z. U., Mortuza, M. G., Uddin, M. J., M. H. Kabir, & Hoque, M.J. Internet of Things Based Patient Health Monitoring System Using Wearable Biomedical Device. Paper presented at IEEE International Conference on Innovation in Engineering and Technology (ICIET), Bangladesh, (2018) 1-5. doi: 10.1109/CIET.2018.8660846.

[3] Bansal, B.N. & Garg, V. Development of Message Queuing Telemetry Transport (MQTT) based Vehicle Accident Notification System. International Journal of Engineering and Advanced Technology **9**, 2 (2019) 268-273. doi: 10.35940/ijeat.B2662.129219.

[4] Gomes, Y. F., Santos, D. F. S., Almeida, H. O. & Perkusich, A. Integrating MQTT and ISO/IEEE 11073 for health information sharing in the Internet of Things. Paper presented at IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, (2015) 200-201. doi: 10.1109/ICCE.2015.7066380.

[5] Hakim, M. L., Uddin, M. J. & Hoque, M. J. 28/38 GHz Dual-Band Microstrip Patch Antenna with DGS and Stub-Slot Configurations and Its 2×2 MIMO Antenna Design for 5G Wireless Communication. Paper presented at IEEE Region 10 Symposium (TENSYMP), Bangladesh, (2020) 56-59. doi: 10.1109/TENSYMP50017.2020.9230601.

[6] Hoque, M., Ahmed, M. & Hannan, S. An Automated Greenhouse Monitoring and Controlling System using Sensors and Solar Power. European Journal of Engineering Research and Science. **5**, 4 (2020) 510-515. doi: https://doi.org/10.24018/ejers.2020.5.4.1887.

[7] Hoque, M., Kabir, S., & Hossain, M. K. Electricity Crisis of Bangladesh and A New Low Cost Electricity Production System to Overcome this Crisis. International Journal of Scientific and Research Publications **8**, 7 (2018) 201-206. doi: http://dx.doi.org/10.29322/IJSRP.8.7.2018.p7933

[8] Hoque, M. J., Ahmed, M. R., Uddin, M. J. & Faisal, M. A. Automation of Traditional Exam Invigilation using CCTV and Bio-Metric. International Journal of Advanced Computer Science and Applications **11**, 6 (2020) 392-399. doi: http://dx.doi.org/10.14569/IJACSA.2020.0110651

[9] Kabir, M. H., Rashid, S. Z., Gafur, A., Islam, M. N. & Hoque, M. J. Maximum Energy Efficiency of Three Precoding Methods for Massive MIMO Technique in Wireless Communication System. Paper presented at IEEE International Conference on Electrical, Computer and Communication Engineering (ECCE), Bangladesh, (2019) 1-5. doi: 10.1109/ECACE.2019.8679238.

[10] Katsikeas, S. Lightweight & secure industrial IoT communications via the MQ telemetry transport protocol. Paper presented at IEEE Symposium on Computers and Communications (ISCC), Heraklion, (2017) 1193-1200. doi: 10.1109/ISCC.2017.8024687.

[11] Lee, S., Kim, H., Hong, D. & Ju, H. Correlation analysis of MQTT loss and delay according to QoS level. Paper presented at IEEE The International Conference on Information Networking (ICOIN), Bangkok, (2019) 714-717. doi: 10.1109/ICOIN.2013.6496715.

[12] Sharma, S., Hossen, M. K., Islam, M. S. & Hoque, M. J. Automatic Question and Answer Generation from Bengali and English Texts. GESJ: Computer Science and Telecommunications **2**, 54 (2018) ISSN 1512-1232.

[13] Singh, M., Rajan, M. A., Shivraj, V. L. & Balamuralidhar, P. Secure MQTT for Internet of Things (IoT). Paper presented at IEEE Fifth International Conference on Communication Systems and Network Technologies, Gwalior, (2015) 746-751. doi: 10.1109/CSNT.2015.16.

[14] Upadhyay, Y., Borole, A. & Dileepan, D. MQTT based secured home automation system. Paper presented at IEEE Symposium on Colossal Data Analysis and Networking (CDAN), Indore, (2016) 1-4. doi: 10.1109/CDAN.2016.7570945.