Secure Smart Lock System Utilizing the MQTT Protocol

Adnan Shaout
The Electrical and Computer Engineering Department
The University of Michigan-Dearborn
shaout@umich.edu

Nidhi Shah
The Electrical and Computer Engineering Department
The University of Michigan-Dearborn
nidhii@umich.edu

Sneka Dhandapani
The Electrical and Computer Engineering Department
The University of Michigan-Dearborn
sneka@umich.edu

2022 13th International Conference on Information and Communication Systems (ICICS) | 978-1-6654-8097-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICICS55353.2022.9811136

***Abstract* — Interconnected communication system facilitating home automation is becoming increasingly popular Internet of Things' (IoT) domain. One such application is a smart lock. Today, physical locks are being replaced by electro-mechanical ones, with limited network and power capabilities. A physical key is replaced with software that can be operated remotely from a mobile device. Since, these systems widely rely on the wireless network, they are more vulnerable to a security breach. Therefore, development of a network of connected system that ensures reliability and security is fundamental to the field of IoT. Often, securing mechanism requires using complex decision-making algorithms. However, the foundation of a robust system is laid during the design, and development of such a critical embedded system. Thus, it is important to identify, and prioritize the curial system requirements and the steps to engineer them. This paper presents our work that identified and addressed this issue. As part of the research, a proof of concept, of a network of connected devices serving as a secure smart lock, is implemented. It utilizes Message Queuing Telemetry Transportation (MQTT) protocol to enable message transmission between the users and lock. It is light weight, uses less memory on devices and network usage is at par with quality of services that are already available in the market. This paper describes the requirements and design, the technical details of the implementation and the performance aspects of this system.**

Keywords — IoT system, smart Lock, safe home, home automation inter-connected devices, raspberry pi4 broker, MQTT protocol

I. Introduction

Smart devices have gained an increased attention over the recent years. They are mostly operated and controlled through a mobile device or remote software. Home automation is one such popular advancement where network of connected devices has revolutionized the living standard of the people. Smart locks are among many home automation products. These locks have replaced the traditional locks with physical keys. They are usually connected through wireless network, Bluetooth and are often controlled by the user via smartphone. However, since these locks are connected to the internet, they are highly vulnerable [1]. In case of a breach, the entire home security can be compromised. Thus, it is important to offer a secure, reliable locking systems assure user's safety, security, and reliability of the device.

A smart lock's primary requirement is user-access management. Securing the communication channel between the lock and the user is of utmost importance. System access must be governed by user authentication and authorization. This essentially means that only personnel with the correct access rights should be able to use and/or manipulate the system. Access control policies discussed by Kim et all [2] include 1. Full Control, 2. Restricted Control, 3. Partial Control, 4. Minimal Control. These provide a basic pattern describing the hierarchy of the access rights and control over the system. Which forms a base for developing custom role-based access for our system.

Network security algorithms requires high bandwidth, power consumption and have high latency [3]. Securing a low-powered system with limited network recourses requires well planned and tested engineering of its functional requirements. Therefore, A careful evaluation of each development step contributing to the safety of the system is important. Grant ho et al [4] in their research highlight the critical aspect of securing smart devices by designing appropriate system. System architecture plays a crucial role in building a robust foundation, that can be extended depending on individual system's requirement.

Applications of critical nature require continuous quality engineering, Verification and Validation (V-model) is better suited for the development of systems requiring high reliability [5]. We developed our system using the V-model Software Development Lifecycle Model. Using this model, each deliverable was thoroughly tested before it was deployed. Thereby assuring high quality and reliability.

A. Literature Review

Home automation is a vigorously researched area. There are several products that are available in the market today which offers multiple varieties of smart lock system. In [7], a home security system called "HDSL" has been designed to operate as a locking system that is controlled using Bluetooth Handshake. Home security is guaranteed through two functional requirements; 1. By analyzing who is at the door, and 2. Locking the door via mobile device.

Many home automation apps are available in the market. One such popular app is the Home Assistant [8]. It facilitates users to configure the smart door lock along with the other smart devices like light bulb, television, switches, etc. using wireless network. Home Assistant uses MQTT enabled communication between multiple devices. It also provides real time tracking to monitor who has accessed the lock and when.

Many researchers have proposed home security system that incorporated PIR motion detection sensor. In [9], authors have proposed a system that detects the intruders based on the data from PIR sensor and IP camera. In this system, raspberry pi which is connected to multiple PIR sensors that are installed around the home premise. Whenever a motion is detected, the raspberry pi sends out an email notification to the user.

In [10], authors have designed a similar security monitoring system based on the IoT technology using ESP32 and a PIR motion sensor. PIR sensor is used to detect any unwanted motion at the door. Whenever any unknown motion is detected, the owner receives an alert. This system also maintains an access log of the lock and provides log management to the owner of the system. The communication between the lock and their application is established using the MQTT protocol.

The primary goal of this paper is to implement a proof of concept of a secure smart lock that utilizes the MQTT (Message Queuing Telemetry Transportation) protocol to enable message transmission between the users and the lock. MQTT uses a publisher-subscriber architecture where a publisher will post messages on a topic, which will be routed to all the subscribers via a broker/server. The state-of-the-art functionality in this paper is 1. To introduce two levels of user authorization and authentication, one at the broker level and other at client level and 2. Using distributed architecture, separating the lock from any direct packet exchange, all communication to be handled by a server installed in the sub-network secured by the firewall and custom authentication configuration.

This system can be more reliable since the IoT components are set to operate within the same secured home network. We will also design our system to allow the authorized user outside the home network to be able to connect to system's server as per their access privileges.

The rest of the paper is organized as follows. The project requirements in this paper are stated in section 2. In section 3 the high-level design and detailed system functionalities are given. The implementation details are discussed in section 4. In section 5, testing results are stated and explained. Section 6 is used to conclude the paper.

II. PRODUCT REQUIREMENTS

The Smart-Lock system is designed to be a distributed system built using microcontrollers with limited on-board resources. At the least, system facilitates users the mechanism to control a electrically powered, mechanical bolt that locks and unlocks doors. In our work, we are assuming the end use of the product to be to lock and unlock a home's main door.

System requirements are as follows:

- User-Access Management: The system can be accessed conditionaly by registerd users. The roles defined are based on the designation of the user.
    - Owner – has admin rights:
        1. Can lock and unlock the door for undefined time period
        2. Check lock's access log
        3. Have rights to lock management (health check, resetting the device etc.)
        4. Receive emergency alerts
    - Resident – has unlimited access to the lock, but not the management of the lock:
        1. Can lock and unlock the door for undefined time period
    - Guest – limited access to the lock (only for defined period of the time) and no access to the management:
        1. Can lock and unlock the door for a defined period/number of time
- Intrusion detection:
    - If an attempt to access the lock by an unregisterd system is detected, an alert is sent to the registered "owner" and/or "resident/s", activity is logged and the lock is temproraily disabled.
    - 3 unsuccessful attempts to access the lock – specially to unlock the door, will disable the door for a short period of time and alert the registered "Owner" via chosen form of communication
- *Record logs:*
    - Record every successful and unsuccessful communication with the lock in chronological order
    - Record the system health – health is defined to be the connection status between the broker and the lock

III. DESIGN

The system architecture is shown in Fig. 1 which has six main actors, broker, lock, owner, guest, resident, and an authorized user outside the home network. Within the same home network, A broker is set up on a Raspberry Pi4 Model B. It has 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE with 4 GB memory and 16 GB of storage, due to knowledge constraints in the mechanical area, a windows-based machine is mocked to perform as a lock, depicted in figure-1 as a microcontroller. The memory usage by the software for the lock is constrained to be under 256 KB. Another Linux machine is used to SSH into Raspberry PI to program the broker, an android phone (One plus 3) and an iPhone 11 are used to set up the roles of an authorized and unauthorized clients with different access rights - owner, resident, and guest. The phones use general MQTT app available on the google play store and app store for validation and verification. Another constraint, for the first iteration of this system is to limit the access to the broker from outside the network to the users with designated role of an "owner" and "resident".

The system uses MQTT protocol to exchange data packets between the different actors [11]. No device can communicate with the lock directly. There is a broker (a server) that relays the data packets to and from the devices. There are two roles each device internally maintains – a publisher and a subscriber. Python 3.9 is used for development, Paho library provides the needed APIs to create the web-clients.

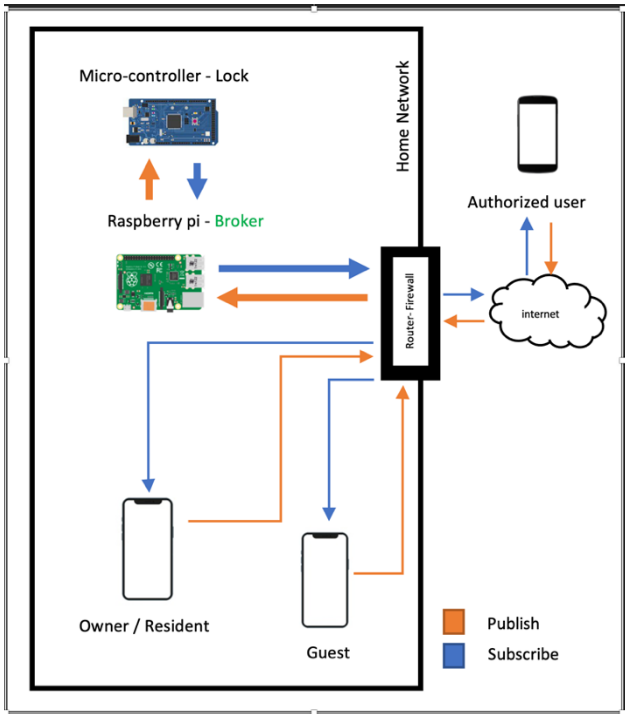


Fig. 1. High-level System Architecture

The system's basic operation is depicted in the data flow diagram displayed in Fig. 2. The following subsections outline the system's most important features:

On the right Fig. 2. Data Flow diagram that depicts high level state change conditions of the proposed system, the hypothetical condition for the guest based role in this path is that they cannot access the system more than 3 times.

A. *Basic system setup:*

We use MQTT publisher-subscriber architecture. Raspberry pi4 is setup as the broker. It maintains the code scripts for each topic and the configuration for publishers and subscribers. All the devices are classified as "clients" including the broker. These clients are subscribers to "topics" that they are listening to and publishers to "topics" they are requesting an action from [11].

Lock is the client that is set to listen and respond to requests on locking and unlocking topics exclusively from the broker's machine. Broker listens, responds, and manages the requests. The user-clients (Owner, Resident and Guest), described in the previous section have custom privileges to request actions on

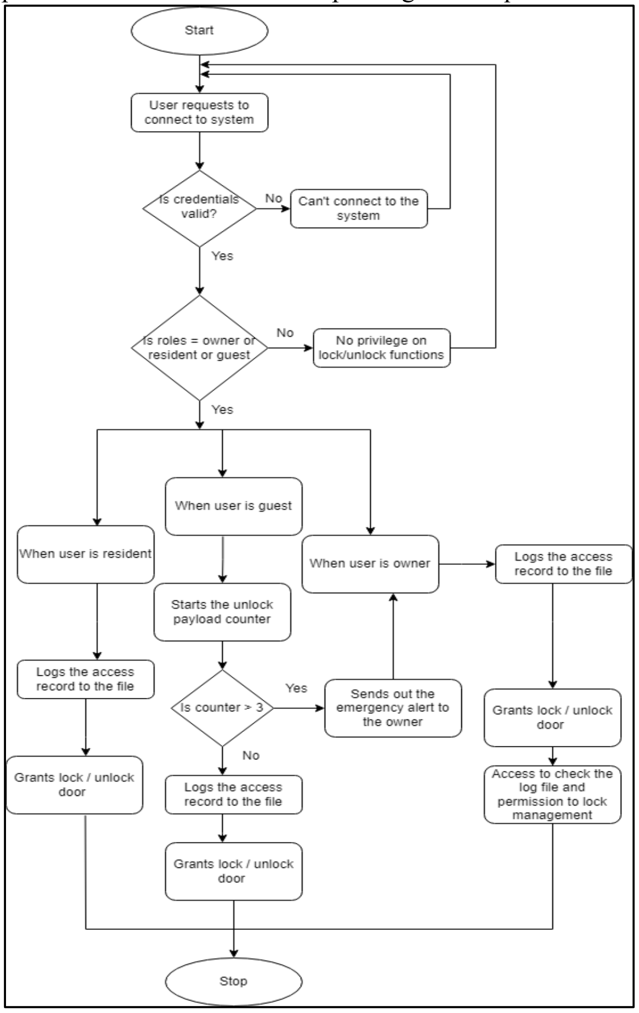

Start
User requests to connect to system
Is credentials valid?
No
Can't connect to the system
Yes
Is roles = owner or resident or guest
No
No privilege on lock/unlock functions
Yes
When user is guest
When user is resident
When user is owner
Logs the access record to the file
Starts the unlock payload counter
Logs the access record to the file
Is counter > 3
Yes
Sends out the emergency alert to the owner
Grants lock / unlock door
Grants lock / unlock door
No
Logs the access record to the file
Access to check the log file and permission to lock management
Grants lock / unlock door
Stop

Fig. 2. Data Flow diagram

topics. Their authentication and authorization mechanism are distributed in client's request algorithm and broker's configurations.

For validation and verification steps there are test clients set up on machines with the permutation and combination of different access-rights and number of clients. The list of primary topics defined include "lockReq", "lockRes", "lock", "lockStatus", "health" and "securityAlert".

B. *User and Broker interaction:*

When a user makes a request for locking/unlocking to our broker, this would be a publish action on topic "lockReq". Since our broker will already be subscribed to this topic, it can receive the payload sent by users. As a first step, broker validates the user credentials using defined steps in config file. If client is an authorized and authenticated user, broker collects timestamp of the request and checks for the user role. If the authentication is successful, then broker will publish on topic "lock". Broker relays this valid request to the lock client. Once the lock receives this payload, it sends back a message response to the broker by publishing on topic "lockStatus". Broker then sends out this payload response to the user based on "lockRes". The sequence diagram that shows the interaction between clients and broker is shown in Fig. 3.

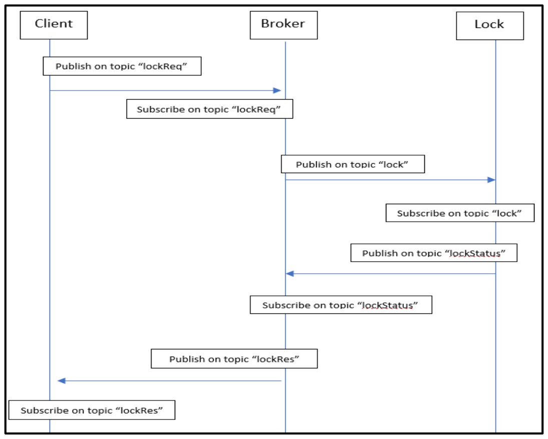

Client
Broker
Lock
Publish on topic "lockReq"
Subscribe on topic "lockReq"
Publish on topic "lock"
Subscribe on topic "lock"
Publish on topic "lockStatus"
Subscribe on topic "lockStatus"
Publish on topic "lockRes"
Subscribe on topic "lockRes"

Fig. 3. Sequence diagram showing clients and broker interaction

C. *Intrusion Detection:*

To handle any intrusion detection, The system tracks the lock requests. The payload is observed for data pertaining to request-origin, request-destination, timestamp, and most importantly the user-role. This information is fed to the authorization algorithm to access the authenticity of the request.

Intrusion detection is dependent on the primary use case of the lock. As per the criticality, the tolerance value can be changed. For this first iteration, the equation to identify intrusion is dependent on the number of times the unlocking request is made with respect to the time elapsed since the last request and weather the following requests fall in the time range. For testing purposes, the value of suspicious[1] attempts are kept to three. If this is greater than three, broker fails the request and will publish failure status on topic "lockRes". The requesting client will receive this response. At the same time, broker will also publish on topic "emergencyAlert" which is subscribed by the owner and/or resident client/s. Hence, owner will be notified with a security alert message. All access record details will also be logged in log file.

D. *Health check of Lock client:*

For every interval of 60 seconds, the lock will send heartbeat signal to record its functioning status. The primary aim of this health check is to identify if the lock is running and there are no malfunctions or failure of the lock. To achieve this functionality, the lock client will publish on topic "health". Broker who is also subscribed to the same topic will receive the payload. It will record the timestamp and access the log file. Once the log file has been opened, it then writes the timestamp data to the log file. Fig. 4 shows the sequence diagram of the health check function.

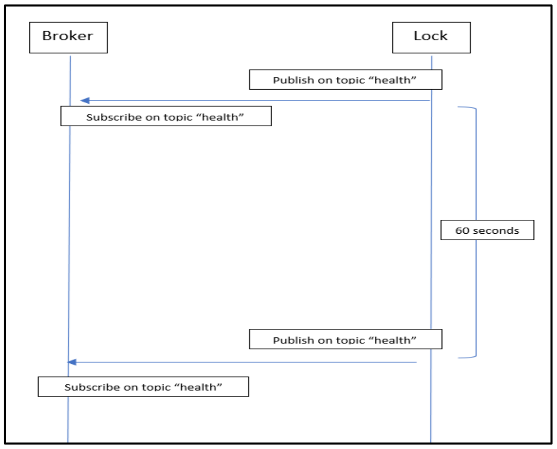

Broker
Lock
Publish on topic "health"
Subscribe on topic "health"
60 seconds
Publish on topic "health"
Subscribe on topic "health"

Fig. 4. Sequence diagram showing the health check function

IV. IMPLEMENTATION

The publisher, subscriber scripts in this system are written using the python programming language based on the Eclipse Paho MQTT Python Client. The clients connect to the MQTT broker for publishing messages, subscribing to the relevant topics and get the published message responses. The below table of functions explains the detailed message transmission logic between the clients and broker.

TABLE I. TABLE OF FUNCTIONS

Module	***Description***	***Subscriber***	***Output***
Client publish on the topic "lockReq"	1. Username and password authentication using config file. 2. Time stamp of when the request was received. 3. Counter to keep how many times a payload of "unlock" was received from the same user in the time period of 5 mins (time of 1st req received - timeNow). If counter>3 fail the request and publish failure status on "lockRes" and publish on topic "emergencyAlert" 4. If authentication is successful, save the data in the a log file and then publish on the topic "lock".	Lock is subscribed to the topic "lock"	After receiving a payload on the topic "lock", record the payload, wait for 30 seconds and publishes on the topic "lockStatus" to the broker
Lock publish on	Reroutes lockStatus by	Client is subscribed to	Lock response status will be

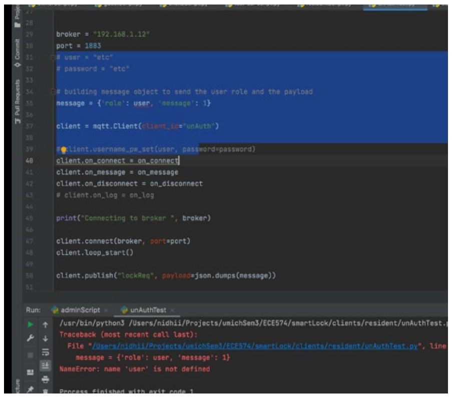
Module	***Description***	***Subscriber***	***Output***
the topic "lockStatus"	publishing it the topic "lockRes-owner" / "lockRes-guest" / "lockRes-resident"	corresponding topic "lockRes"	received by the client
After 3 unsuccessful attempts in unlocking	Process after broker encounters >3 attempts to unlock. 1. Log user info and time 2. Run the publisher script on the topic security alert.	Lock is subscribed to the topic "securityAlert"	An alert message will be logged
Lock to send heartbeat to record its functioning status. Lock publishes on the topic "health" every 60 seconds	Process on the broker after receiving payload on "health" and 1. Get the payload 2. Get the time stamp 3. Open the log file 4. In a given format, add write to the log file.	Lock is subscribed to the topic "health"	Functioning status can be viewed in the log file

V. TESTING AND VALIDATION

Developing this system using verification and validation required each defined requirement to be tested as it was developed. Once the individual modules are developed and tested, system and acceptance test are carried out. The complete system was setup and tested from the same home network and an authorized request was tested from outside the home network. The results are validated against each of the described requirements.

A. *Case 1:*

Only the users who are authorized will be successfully connected to our broker / server. If a user who is not authorized i.e., if the user without valid credentials tries to connect to the broker, then the system will reject the unauthorized access from the user. An error message should be displayed indicating that the user is not defined. The output looks as shown in Fig. 6.



```
broker = "192.168.1.12"
port = 1883
# user = "etc"
# password = "etc"

# building message object to send the user role and the payload
message = {'role': user, 'message': 1}

client = mqtt.Client(client_id="unAuth")

#client.username_pw_set(user, password=password)
client.on_connect = on_connect
client.on_message = on_message
client.on_disconnect = on_disconnect
# client.on_log = on_log

print("Connecting to broker ", broker)

client.connect(broker, port=port)
client.loop_start()

client.publish("lockReq", payload=json.dumps(message))
```

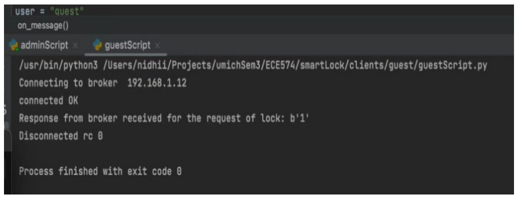
```
/usr/bin/python3 /Users/nidhii/Projects/umichSem3/ECE574/smartLock/clients/resident/unAuthTest.
Traceback (most recent call last):
  File "/Users/nidhii/Projects/umichSem3/ECE574/smartLock/clients/resident/unAuthTest.py", line
    message = {'role': user, 'message': 1}
NameError: name 'user' is not defined

Process finished with exit code 1
```

Fig. 6. An error message is displayed successfully with the user is not defined message

B. *Case 2:*

If an authorized guest requests lock access to the broker, then the broker validates the request and sends a response with "1" indicating the success of the request. The output response received by the guest user from the broker looks as shown in Fig. 7. The letter 'b' stands for byte, this is the console output.



```
user = "guest"
on_message()
```

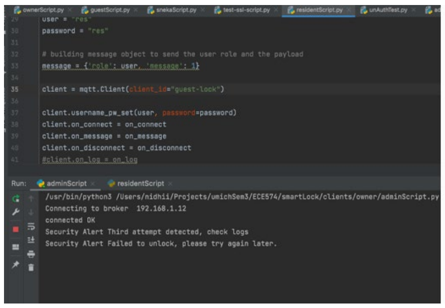
```
/usr/bin/python3 /Users/nidhii/Projects/umichSem3/ECE574/smartLock/clients/guest/guestScript.py
Connecting to broker  192.168.1.12
connected OK
Response from broker received for the request of lock: b'1'
Disconnected rc 0

Process finished with exit code 0
```

Fig. 7. Log file showing the system access record

C. *Case 4:*

If any user who is connected to the broker but doesn't have authorized credentials attempt to access the lock more than 3 times in a row, then this action should be notified as "Security Alert" to the admin. Along this with, the user will not be allowed another attempt of access and can try again after sometime. An output of this function is shown in Fig. 8.



ownerScript.py guestScript.py snekaScript.py test-ssl-script.py residentScript.py unAuthTest.py

```
user = "res"
password = "res"

# building message object to send the user role and the payload
message = {'role': user, 'message': 1}

client = mqtt.Client(client_id="guest-lock")

client.username_pw_set(user, password=password)
client.on_connect = on_connect
client.on_message = on_message
client.on_disconnect = on_disconnect
#client.on_log = on_log
```

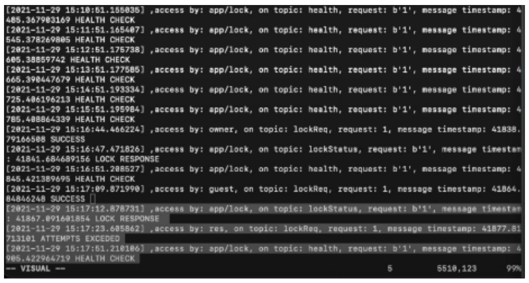
Run: adminScript residentScript

```
/usr/bin/python3 /Users/nidhii/Projects/umichSem3/ECE574/smartLock/clients/owner/adminScript.py
Connecting to broker  192.168.1.12
connected OK
Security Alert Third attempt detected, check logs
Security Alert Failed to unlock, please try again later.
```

Fig. 8. A security alert message is displayed to the owner

D. *Case 3:*

Whenever an attempt to access lock has been made by the clients, it should be logged on to log file that has been created. This log file should contain the role of the user, what action was performed, success or failure status and attempt number along with the timestamp. All lock access record information is maintained to enable safety and easy maintenance. The log file with the access result is shown in Fig. 9.



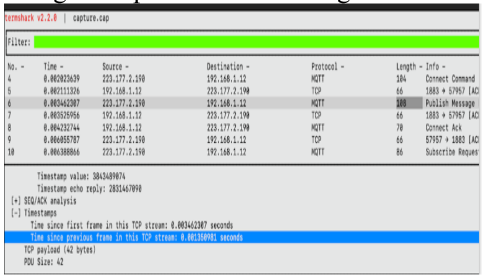
```
[2021-11-29 15:10:51.155035] ,access by: app/lock, on topic: health, request: b'1', message timestamp: 4
485.367903169 HEALTH CHECK
[2021-11-29 15:11:51.165407] ,access by: app/lock, on topic: health, request: b'1', message timestamp: 4
545.378269805 HEALTH CHECK
[2021-11-29 15:12:51.175738] ,access by: app/lock, on topic: health, request: b'1', message timestamp: 4
605.38859742 HEALTH CHECK
[2021-11-29 15:13:51.177585] ,access by: app/lock, on topic: health, request: b'1', message timestamp: 4
665.390447679 HEALTH CHECK
[2021-11-29 15:14:51.193334] ,access by: app/lock, on topic: health, request: b'1', message timestamp: 4
725.406196213 HEALTH CHECK
[2021-11-29 15:15:51.195984] ,access by: app/lock, on topic: health, request: b'1', message timestamp: 4
785.408864339 HEALTH CHECK
[2021-11-29 15:16:44.466224] ,access by: owner, on topic: lockReq, request: 1, message timestamp: 41838.
79166508 SUCCESS
[2021-11-29 15:16:47.471826] ,access by: app/lock, on topic: lockStatus, request: b'1', message timestan
: 41841.684689156 LOCK RESPONSE
[2021-11-29 15:16:51.208527] ,access by: app/lock, on topic: health, request: b'1', message timestamp: 4
845.421389695 HEALTH CHECK
[2021-11-29 15:17:09.871990] ,access by: guest, on topic: lockReq, request: 1, message timestamp: 41864.
84846248 SUCCESS
[2021-11-29 15:17:12.878731] ,access by: app/lock, on topic: lockStatus, request: b'1', message timestan
: 41867.891601854 LOCK RESPONSE
[2021-11-29 15:17:23.605862] ,access by: res, on topic: lockReq, request: 1, message timestamp: 41877.81
713101 ATTEMPTS EXCEDED
[2021-11-29 15:17:51.210186] ,access by: app/lock, on topic: health, request: b'1', message timestamp: 4
905.422964719 HEALTH CHECK
-- VISUAL --                                              5          5510,123       99%
```

Fig. 9. Log file showing the system access record

E. *Packet Analysis using termshark:*

Using termshark, we have analyzed the packet size of the payload that has been exchanged over the network. The highest size recorded during our testing phase is 108 bytes. The one-way latency is 00135098 seconds. Fig. 10 shows this result.

MQTT provides three quality of service (QoS) levels. The messages with the QoS value set to 2 are considered more reliable but the tradeoff is on the packet size. These were tested against HTTP packets for payload size and were found to be equitable and hence not suitable when working with controllers with limited resources. Due to the critical nature of our application, we did require the benefits of the QoS level 2 but not the latency incurred when using it. Therefore, after much of testing, we found the balance by using QoS 1 and placing the check for duplicate request at the broker before firing the final lock/unlock request to the client lock. This reduced and defined the size of the data packet to be dependent on the payload[2]. This brought the packets to an average size of 112 bytes.



```
termshark v2.2.0  |  capture.cap

Filter:

No. -   Time -        Source -        Destination -    Protocol -   Length - Info -
4       0.002023639   223.177.2.190   192.168.1.12     MQTT         104      Connect Command
5       0.002111326   192.168.1.12    223.177.2.190    TCP          66       1883 → 57957 [AC
6       0.003462307   223.177.2.190   192.168.1.12     MQTT         108      Publish Message
7       0.003525956   192.168.1.12    223.177.2.190    TCP          66       1883 → 57957 [AC
8       0.004232744   192.168.1.12    223.177.2.190    MQTT         70       Connect Ack
9       0.006055787   223.177.2.190   192.168.1.12     TCP          66       57957 → 1883 [AC
10      0.006388866   223.177.2.190   192.168.1.12     MQTT         86       Subscribe Reques

        Timestamp value: 3843489074
        Timestamp echo reply: 2831467090
  [+] SEQ/ACK analysis
  [-] Timestamps
       Time since first frame in this TCP stream: 0.003462307 seconds
       Time since previous frame in this TCP stream: 0.001350981 seconds
     TCP payload (42 bytes)
     PDU Size: 42
```

Fig. 10. Packet Analysis

VI. CONCLUSION

Our implementation of a smart lock implements a system of distributed network of devices with constrained resources. It is built to facilitate fast and safe home automation solution. For this, we have utilized a mature and light weight MQTT protocol. Since the MQTT works on TCP protocol, it already provides the message delivery guarantee even with the least QoS that we can choose from. With the level 0 QoS provided by the MQTT, light-weight packets are exchanged over the network for topics with relatively low security threat. The authorization and authentication algorithm are devised to work on a distributed architecture. With this feature, we have implemented our system that enables the message transmission between various clients in a more reliable and secure way. Our system also has health check running upon the lock client in an interval of every 60 seconds. This has enabled the owner to about its status. The lock's access and safety has been guaranteed by proving two levels of user authorization and authentication. Following, the V-model software methodology we were able to perform enhanced levels of testing. Whenever a client and the corresponding functionality has been implemented, we have performed the unit testing. Once all the functionalities were implemented, system testing were performed to check against the requirements. As part of future work, the packets can be encrypted and implemented it in SSL which could provide another layer of secure and anonymity.

We thank Dr Zheng Song, from Computer and Information Science Department at The University of Michigan-Dearborn for introducing us to the topic of a Smart Lock.

REFERENCES

[1] B. D. Davis, J. C. Mason and M. Anwar, "Vulnerability Studies and Security Postures of IoT Devices: A Smart Home Case Study," in IEEE Internet of Things Journal, vol. 7, no. 10, pp. 10102-10110, Oct. 2020, doi: 10.1109/JIOT.2020.2983983.

[2] Tiffany Hyun-Jin Kim, Lujo Bauer, James Newsome, Adrian Perrig, and Jesse Walker. 2010. Challenges in access right assignment for secure home networks. In Proceedings of the 5th USENIX conference on Hot topics in security (HotSec'10). USENIX Association, USA, 1.

[3] Amanlou, Sanaz & Abu Bakar, Khairul Azmi. (2020). Lightweight Security Mechanism over MQTT Protocol for IoT Devices. International Journal of Advanced Computer Science and Applications. 11. 202-207.

[4] Grant Ho, Derek Leung, Pratyush Mishra, Ashkan Hosseini, Dawn Song, and David Wagner. 2016. Smart Locks: Lessons for Securing Commodity Internet of Things Devices. In Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security (ASIA CCS '16). Association for Computing Machinery, New York, NY, USA, 461–472. https://doi.org/10.1145/2897845.2897886

[5] M. Rodriguez, M. Piattini and C. Ebert, "Software Verification and Validation Technologies and Tools," in IEEE Software, vol. 36, no. 2, pp. 13-24, March-April 2019, doi: 10.1109/MS.2018.2883354.

[6] B. Mishra and A. Kertesz, "The Use of MQTT in M2M and IoT Systems: A Survey," in IEEE Access, vol. 8, pp. 201071-201086, 2020, doi: 10.1109/ACCESS.2020.3035849.

[7] Agarwal A, Hada N, Virmani D, Gupta T. A Novel Design Approach for Smart Door Locking and Home Security using IoT. A High Impact Factor & UGC Approved Journal. 2017 August; 6(8): p. 1-5.

[8] "Home Assistant Documentation".HomeAssistant, https://www.home-assistant.io/blog/.htm.

[9] S. Tanwar, P. Patel, K. Patel, S. Tyagi, N. Kumar and M. S. Obaidat, "An advanced Internet of Thing based Security Alert System for Smart Home," 2017 International Conference on Computer, Information and Telecommunication Systems (CITS), 2017. https://0-ieeexplore-ieee-org.wizard.umd.umich.edu/document/8035326

[10] Andreas, Cornelio Revelivan Aldawira, Handhika Wiratama Putra, Novita Hanafiah, Surya Surjarwo, Aswin Wibisurya, "Door Security System for Home Monitoring Based on ESP32". Procedia Computer Science. 2019. https://0-doi-org.wizard.umd.umich.edu/10.1016/j.procs.2019.08.218

[11] "MQTT Publish and Subscribe Beginners Guide". Steve'sInternetGuide, http://www.steves-internet-guide.com/mqtt-publish-subscribe/

1. Suspicious attempts: A suspicious request is a function of, user role, request origin, and the number of attempts.

2. Payload is the section of the data packet where the actual message to be transmitted from the sender to receiver is placed