

DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

SECURITY ENHANCED LIGHTWEIGHT
MESSAGING PROTOCOL



by
Özlem YERLİKAYA

July, 2018
İZMİR

SECURITY ENHANCED LIGHTWEIGHT MESSAGING PROTOCOL

**A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Master of
Science in Computer Engineering**

**by
Özlem YERLİKAYA**

**July, 2018
İZMİR**

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled “SECURITY ENHANCED LIGHTWEIGHT MESSAGING PROTOCOL” completed by ÖZLEM YERLİKAYA under supervision of ASST. PROF. DR. GÖKHAN DALKILIÇ and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



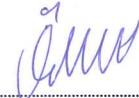
Asst. Prof. Dr. Gökhan DALKILIÇ

Supervisor



Asst. Prof. Dr. Enis KARAARSLAN

(Jury Member)



Asst. Prof. Dr. Özlem AKTAŞ

(Jury Member)



Prof. Dr. Latif SALUM
Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor Asst. Prof. Dr. Gökhan DALKILIÇ for sharing valuable time and for his precious advice. On the coding part, I want to thank to Marcin Bachry for sharing his knowledge and time with me.

Finally, I like to thank my family, my colleague Göksu Tüysüzoğlu, and my friends Yunus Şeker, Halime Kardaş and Gülsüm Uzut for their contribution and constant support.

Özlem YERLİKAYA

SECURITY ENHANCED LIGHTWEIGHT MESSAGING PROTOCOL

ABSTRACT

Internet of things (IoT) allows devices especially with low process capability and power consumption, to transmit data to each other using various communication technologies such as wired, wireless network and radio frequency. When huge number of devices with limited resources are connected to IoT application, provided security gains significant importance to ensure the integrity, confidentiality, accessibility of these sensitive data. In addition, the availability of a variety of specialized devices and communication technologies demonstrate the hassle of providing a standard security mechanism.

When device features such as power and process capability are taken into account, message queue telemetry transport (MQTT) is the most appropriate lightweight communication protocol. In this study, the MQTT security is defined, and preliminary work related to MQTT on the basic security issues such as privacy, authentication and access control is examined.

This study provides authorization mechanism by using open authorization (OAuth 2.0) protocol, which is recommended to gain authorization. In addition to the authenticator token, authentication is performed in two steps using a HMAC-based one-time password (HOTP) due to its short life span. Since MQTT protocol does not have bidirectional authentication other than using transport layer security (TLS), mutual authentication is provided by using OTP with hash chain. Advanced encryption standard (AES) is used for providing confidentiality to prevent against potential security vulnerabilities. Security analysis of the implementation has been discussed by giving an alternative solution by using these methods against selected attacks. It has been observed that various security vulnerabilities and difficulties have been successfully resolved.

Keywords: MQTT, mutual-authentication, authorization, HOTP, OAuth 2.0, AES

GÜVENLİĞİ GELİŞTİRİLMİŞ HAFİF MESAJLAŞMA PROTOKOLÜ

ÖZ

Nesnelerin interneti, işlem yeteneği düşük, kısıtlı güç tüketimine sahip cihazların kablolu, kablosuz ağ, radyo frekansı gibi çeşitli haberleşme teknolojileri kullanarak birbirleriyle veri iletimi gerçekleştirmesine olanak sağlar. Nesnelerin interneti uygulamalarında çok sayıda kısıtlı kaynağa sahip cihazın bağlı bulunduğu düşünüldüğünde, bu hassas verilerin gizliliğini, bütünlüğünü, ulaşılabilirliğini sağlamak güvenliğin önemini artırmaktadır. Buna ek olarak, çeşitli özellikte cihazın bulunması ve çeşitli haberleşme teknolojileri ile bağlantı kurulabilmesi standart bir güvenlik mekanizması sağlamanın güçlüğüne göstermektedir.

Cihazın güç ve işlem yeteneği gibi özellikleri göz önünde bulundurulduğunda mesaj kuyruk telemetri taşıma (MQTT) en uygun hafif haberleşme protokolü olarak belirlenmiştir. MQTT güvenlik mekanizması tanımlanıp, verinin gizliliği, kimlik doğrulama, erişim kontrolü gibi temel güvenlik konularına yönelik yapılmış çalışmalar incelenmiştir.

Bu çalışma, yetkilendirme için önerilen açık yetkilendirme (OAuth 2.0) protokolünü kullanarak yetkilendirme mekanizması sağlamaktadır. Kimlik doğrulama belirtecine ek olarak, kısa ömürlü olmasıyla HMAC tabanlı tek kullanımlık şifre (HOTP) kullanılarak iki adımda kimlik doğrulama uygulanmıştır. Taşıma katmanı güvenliği (TLS) kullanımı dışında çift yönlü doğrulamaya sahip olmayan MQTT protokolü için, OTP'yi karma zincir ile kullanarak karşılıklı kimlik doğrulama sağlanmıştır. Olası güvenlik açıklarına karşı gizlilik, gelişmiş şifreleme standardı (AES) ile sağlanmıştır. Uygulamanın güvenlik analizi, belirlenen saldırılara karşı bu yöntemleri kullanarak alternatif bir çözüm sunarak tartışılmıştır. Çeşitli güvenlik açıklarının ve zorluklarının başarılı bir şekilde çözüldüğü gözlenmiştir.

Anahtar kelimeler: MQTT, karşılıklı kimlik doğrulama, yetkilendirme, HOTP, OAuth 2.0, AES

CONTENTS

	Page
M.Sc THESIS EXAMINATION RESULT FORM.....	ii
ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
ÖZ	v
LIST OF FIGURES	ii
LIST OF TABLES	ii
CHAPTER ONE - INTRODUCTION	1
1.1 Motivation	1
1.2 Contribution.....	3
1.3 Outline of the Thesis.....	4
CHAPTER TWO - RELATED WORK.....	6
2.1 Literature Review	6
CHAPTER THREE - LIGHTWEIGHT COMMUNICATION PROTOCOLS 12	
3.1 Constrained Application Protocol	12
3.2 Extensible Messaging and Presence Protocol	13
3.3 Advance Message Queuing Protocol	13
3.4 Data Distribution Service	13
3.5 Message Queue Telemetry Transport Protocol	14
3.5.1 Security of Message Queuing Telemetry Transport Protocol	16

CHAPTER FOUR – THE PROPOSED METHODS OF SECURITY MECHANISM.....	18
4.1 The OAuth 2.0 Authorization Framework	18
4.2 HMAC-Based One-Time Password	25
4.3 Advanced Encryption Standard Algorithm	26
CHAPTER FIVE - IMPLEMENTATION	29
5.1 Experimental Setup	29
5.1.1 MQTT Client	30
5.1.2 MQTT Broker.....	30
5.1.3 Authorization Server (WSO2is)	31
5.1.4 MongoDB	31
5.2 Authentication and Authorization Mechanism.....	32
CHAPTER SIX - SECURITY ANALYSIS OF THE IMPLEMENTATION ...	44
6.1 Prevent Replay Attacks	44
6.2 Prevent Password Guessing.....	44
6.3 Prevent Man-in-the-Middle Attack	44
6.4 Protect from Eavesdropping	45
6.5 Impersonation Attack Protection.....	45
6.6 Provide Mutual Authentication	45
6.7 Restricted Device Access	46
6.8 Manage De-synchronization.....	46
6.9 Prevent physical attack	46
6.10 Provide Confidentiality	47

6.11 Prevent Denial of Service Attack	47
CHAPTER SEVEN - CONCLUSION AND FUTURE WORK	48
REFERENCES	50



LIST OF FIGURES

	Page
Figure 3.1 MQTT message format.....	14
Figure 3.2 MQTT message types	15
Figure 4.1 The OAuth 2.0 protocol flow	19
Figure 4.2 Obtained access token step by step.....	22
Figure 4.3 Obtained access token step by step.....	23
Figure 4.4 Flow of obtained access token.....	24
Figure 4.5 AES structure.....	27
Figure 4.6 CFB mode encryption.....	28
Figure 4.7 CFB mode decryption.....	28
Figure 5.1 The flow of authentication and authorization mechanism.....	32
Figure 5.2 The pub/sub clients connect to the Mosquitto	43

LIST OF TABLES

	Page
Table 3.1 Comparison of lightweight communication protocol	12
Table 5.1 The selected software and hardware information	29
Table 5.2 Client information	33
Table 5.3 Access token information.....	34
Table 5.4 HOTP count is based on client secret	35
Table 5.5 Registered three clients' information and their 6th HOTP counts.....	37
Table 5.6 Connect message type.....	38
Table 5.7 Returned client information	39
Table 5.8 Parse scope of access token	42
Table 5.9 The information of client's access	43

CHAPTER ONE

INTRODUCTION

The terms of the Internet of things (IoT) is defined as so many devices that can be sensor, radio frequency identification (RFID) tag, smart card with their low capacity of central processing units (CPU) and power, communication each other without human interaction or user interface by using in diverse communication technologies such as wireless, Ethernet, Bluetooth, ZigBee etc. Obtained data is recorded to the cloud or storage and managed over the network by accessing regardless of physical location. It is difficult to provide security due to these heterogeneous communication technologies and protocols. Furthermore, IoT architecture is not stated as certain layer. Literature is examined to indicate security hole for each layer and to supply essential security for the IoT application. The message queue telemetry transport (MQTT) protocol is chosen in order to design authentication and authorization mechanism and a security plugin is written by considering the resource-restricted device to be added to the protocol. The following page in chapter refers to the contribution of security on the MQTT protocol.

1.1 Motivation

Internet of things (IoT) concept was firstly mentioned in 1999 by Kevin Asthon's radio frequency identification (RFID) presentation (Sheng et. al., 2013). A huge number of devices are connected to the Internet in order to share their data via lightweight communication protocols and to monitor these data from anywhere. When a serious number of devices with limited storage and scarce energy source are taken into account, providing the security requires too much cost. There is no highly accepted standard security mechanism for Internet of things (IoT) applications as seen from previous studies. Both studies in (Jing, Vasilakos, Wan, Lu & Qiu 2014; Oh & Kim, 2017) defined the IoT architecture as 3 layers that are perception, network, application; analyzed the security threats of each layers in consideration of heterogeneity, device features, etc. Whereas, (Kraijak & Tuwanut, 2015) explained

IoT architecture with its security vulnerabilities as five layers that are perception, network, middleware, application, business layer in detail.

So as to provide the essential main security requirements that are data confidentiality, integrity, availability for IoT application, firstly which application layer protocol will be determined for restricted-resource device such as smart card, RFID, sensor etc. To make right choice for application protocol, existing protocols that are constrained application (CoAP), message queue telemetry transport (MQTT), advanced message queuing (AMQP), extensible messaging and presence (XMPP), data distribution service (DDS), are compared with each other's in terms of their CPU utilization, network bandwidth capacity (Asim, 2017), storage capacity, computational availability, limited power (Karagiannis, Chatzimisios, Vazquez-Gallego & Alonso-Zarate, 2015) and low header size (Salman & Jain, 2013).

MQTT protocol is one of the most appropriate communication protocols especially for real time applications compared to aforementioned lightweight protocols with regard to having quality of service (QoS), low header size, bandwidth usage and efficient power consumption (Manohar & Reuban Gnana Asir, 2018) for constrained resource devices. For this reason, in this study MQTT is preferred. Data is sent by Eclipse-paho client to Mosquitto that is MQTT broker, and the data is received by different clients such as Android mobile application, Linux machine from broker over the Internet via MQTT communication protocol. MQTT supports secure end-to-end data transmission by using the transport layer security (TLS) as a secure channel. Certificates provides mutual authentication between client and broker (Banks & Gupta, 2014). However, TLS is inappropriate for IoT device due to the complex computation requirements. The initial issue to be intended in this study is to prevent the access on the IoT application by the unidentified devices. The next step is determining the access policies such as read and write, of the verified device. Therefore, a security mechanism is developed and added to the existing security mechanism of MQTT and Mosquitto.

OAuth 2.0 protocol is preferred to delegate authority without sharing username and password in order to prevent user credentials from being re-used without the awareness of the user by third party applications (Windley, 2016). Since MQTT protocol has no security mechanism for access principle, the token that includes the scope of the client is obtained via OAuth2.0 protocol to restrict the access area (Fremantle & Aziz, 2016).

In an event where TLS/SSL is not used as the secure channel, one-time password (OTP) with its short life span may be a good choice to provide mutual authentication. It can be used to prevent the impersonation attack if the token is eavesdropped during sending plain text in token based authentication (Haller, 1994). Shivraj, Rajan, Singh & Balamuralidhar (2015) compared their suggested OTP scheme with existing OTP schemas that are HMAC based one-time password (HOTP), Time based one-time password (TOTP) and Lamport's OTP algorithm over the lightweight identity based elliptic curve cryptography for IoT to provide authentication and observe their performances. Esfahani et. al. (2017) proposed authentication mechanism for IoT environment that only includes computations of XOR and hash function. Their authentication mechanism prevents replay attack by using nonce whereas to ensure that the message is not altered by using one-way hash function to avoid the modification attack. Our study is implemented and contributed one more step including authentication with hash chain based OTP to confirm both client and broker by increasing security against token stealing in the insecure channel.

The proposed the MQTT security mechanism is referred authentication and authorization that are provided by using token over insecure channel with HOTP-hash chain, advanced encryption standard (AES) encryption. Security analysis is performed, then.

1.2 Contribution

The aim of the thesis is to introduce a number of lightweight protocols used transmit data which are obtained from IoT applications such as health, industry, automotive and

financial, over the network. The proposed lightweight protocols are examined under five main headings: MQTT, CoAP, XMPP, AMQP, DDS.

Another remarkable point for this thesis is to suggest the utilization of MQTT protocol as a proposed lightweight communication protocols on the IoT scenarios. Especially in real time applications, efficient power consumption, minimized network traffic with low header size, message loss and latency reduction that depend on QoS will be advantageous for devices that have weak computational ability. However, the existing security mechanism of MQTT may be exposed to various security attacks due to the security weaknesses of MQTT.

Other purpose on the following sections is to introduce the OAuth 2.0 protocol framework, HOTP, AES-256 cipher feedback (CFB) mode symmetric encryption. They are explained to show how to perform authentication and authorization security mechanism so as to provide the access control and securely data transmission by identifying both client and broker in cases where fake server, attacker and huge number of client are available.

As the final contribution of this study is to classify several significant security attacks in IoT scenarios. For this purpose, firstly, the previous study (Fremantle et. al., 2014) which suggests providing access control with OAuth 2.0 protocol, has been taken as the model and implemented with a view to obtain token in order to validate the client and client's access. Possible challenges and security vulnerabilities based on the study have been determined and an alternative solution was generated that includes HOTP with hash-chain to provide mutual authentication without using static password and to avoid authentication with plain text implementing AES encryption without the need of extra computation for the key generation and distribution.

1.3 Outline of the Thesis

The rest of the thesis is organized as follows: The next chapter introduces the examined literature study. In the first step, IoT architecture and security hole of each

layer representation, lightweight communication protocols and MQTT security are explained. Chapter 2 includes definition of the lightweight communication protocols by comparing with MQTT protocol and explanation of MQTT protocol and its security in detail. In the next chapter, the OAuth 2.0 authorization framework is explained in general and detailed knowledge on the implementation is provided. Furthermore, HOTP is stated by expanding different IoT scenarios used in the literature. It is shown that AES encryption and HOTP are needed against the security attacks. Implementation of study by the methods described in the previous chapter, is shown in chapter 5. Moreover, Chapter 6 discusses possible security vulnerabilities on the MQTT communication, and presents a solution against these attacks by the applied written plugin while Chapter 7 concludes the thesis with a summary of the learned key lessons.

CHAPTER TWO

RELATED WORK

Nowadays, data privacy gains significant importance in large scale IoT application areas especially in health, industry, automotive and financial areas that include sensitive personal data, when it is thought that huge number of devices are connected. One needs to have a good determination of how unauthorized devices is restricted in order to ensure privacy for being able to design a successfully authentication and authorization mechanism, whether paper based or digitalized. For obtaining knowledge of how to make a good providing identity and authority of the device by chosen most suitable communication protocol for resource-restricted devices, some studies were examined. The studies show that possible attacks on IoT layers are mentioned and how with the cryptographic studies are taken precautions. The lightweight communication protocols on application layer are compared with each other in terms of devices with constrained resources. The most appropriate MQTT protocol is determined and the knowledge of security MQTT protocol is indicated. Some of the key concepts of these studies are summarized below.

2.1 Literature Review

Ali, Sabir & Ullah (2016) present IoT structure with four parts such as perceptual layer, network layer, support layer and application layer. They also indicate possible security attack of each layer. Perceptual layer has heterogeneous devices like sensor, RFID tag, Bluetooth, Zigbee device that are restricted with having low battery and limited storage, and these IoT devices spread over a large area. Huge amount of data is obtained from these devices via communication protocol in network layer and it is processed or storage in cloud in support layer. Application layer provides different application such as smart healthcare, smart home, smart agricultures to user according to their areas of interest. Each layers have specific security attacks. Perceptual layer, for instance, has attacks such as node tampering, fake node, physical damage, mass node authentication, whereas network layer has network congestion, node jamming in wireless sensor network (WSN), eavesdropping attack, denial of services attacks. In

addition, data security, interoperability and portability, cloud audit can be occurred in support layer while data access and authentication, phishing attack, malwares attack is happened in application layer. The content of this study refers some precautions to come up with these attacks. These precautions are lightweight encryption algorithm for data privacy and authentication and access control mechanism for device in perceptual layer, secure communication protocol for replay, routing, jamming attack in network layer, going on cloud audits for confidentiality, integrity and storage encryption for user's data in support layer, accessing control mechanism with secure application code in application layer. In conventional security, some cryptographic methods that using AES as symmetric encryption for data privacy, preferring Rivest-Shamir-Adleman (RSA) as asymmetric encryption algorithm for digital signature, secure hash algorithm (SHA) for data accuracy and Diffie-Hellman for key generation, are strongly used to provide security. However, these algorithms call for much more CPU power and battery due to including complex computation. Moreover, key management is necessary for cryptographic algorithms due to the number of device excessing in IoT scenarios. Also, authorization and authentication mechanism are forced out to identify devices and data confidentiality because of huge number of IoT devices. Denial of services attack is another security problem that led to consume much more power in IoT scenarios. Consequently, the study informs possible attack in each IoT architecture layer, explains these attacks in detail, suggests solution against the attacks, and gives information about difficulties of applying traditional secure solution way in IoT.

IoT architecture is defined in detail as three layers; perception, network and application layers. Security vulnerabilities for each layer are emphasized with their challenges (Mendez, Papapanagiotou, & Yang, 2017). Whereas, IoT architecture is described in detail with five layers, which are perception, network, middleware, application, business layer. The data is collected by varied sensor in perception layer and is forwarded to network layer via insecure way. In this phase, it can be exposed attacks such as eavesdropping, denial of service attack, on account of sensor device placement on physical environment. Attacker can modify program in sensor device and then access its information or using data adversely. Secure communication is

necessary between perception layer and network layer due to data privacy. It can be provided by encryption connection. Network layer contains huge number of connected devices and their data exchange lead to high traffic. Therefore, denial of service attack can occur. Sensing data is operated and stored in middleware layer, and the IoT applications such as smart health depending on the process data is managed in application layer. Business layer consists of whole IoT system that gives a result as a variety graph or report based on processing data. Analysis is made by taking into consideration this result. So data confidentiality, integrity, availability is so important during all communication. In addition, untrusted users should not access data or device, authentication and control mechanism must be provided. In this study, real time IoT application is implemented by using Arduino, publish/subscribe library, Mosquitto as a broker and Ponte framework. Sending/receiving data is provided over the Internet via MQTT communication protocol. A problem is identified that publish sends message with a topic to broker and subscribe receives this message from broker by using same topic. If attacker knows this topic, he publishes message that include irrelevant information, with same topic. Subscriber gets data under this topic, and obtains wrong data. In the scope of paper, currently there is no implemented solution for this problem except for suggested payload encryption (Kraijak & Tuwanut, 2015).

Yassein, Shatnawi & Al-zoubi (2016) gives information about lightweight protocols that are suitable for IoT scenarios in terms of constrained device with low batteries, low bandwidth, limited storage, weakness process, untrusted network. These protocols are CoAP, MQTT, XMPP, DDS, AMQP and their architecture, communication and message transmission reliability are examined in detail. In the event of different applications and devices, it is determined which protocol is suitable for IoT application by taking in consideration properties of these devices.

In the study (Katsikeas et. al., 2017), some of open source IoT protocols that are devices profile for web services (DPWS), XMPP, CoAP, MQTT, compare with each other according to their specific features and strong performance on restricted device in terms of low battery and computation as both literature and real application. The strengths and weaknesses of these protocols have been demonstrated relative to each

other. The characteristic of MQTT protocol is explained in detail and its security is informed in terms of the authentication, authorization and data confidentiality. A pair of username and password as a plaintext are used to identify client, when client connects to broker. Access control list (ACL) can be used in MQTT protocol in order to provide authorization mechanism that contains topic besides the username and password, to manage client access. Data confidentiality is provided by using encryption algorithm between client and broker or publisher and subscriber. In this study, analysis performance and power consumption results are obtained by applying secure on MQTT protocol to Contiki OS and Cooja simulator as a theoretical and also using MQTT protocol, Mosquitto, zolertia Z1 motes, and Raspberry Pi as a real platform. AES with a 128-bit key as symmetric key algorithm is used to send message securely from publisher to subscriber. Sending message with AES payload encryption is applied to only one block size length whereas AES cipher block chaining (AES-CBC) mode is up to four-block sizes. Moreover, AES offset codebook (AES-OCB) mode that is for both authentication encryption and payload encryption with message is sent to subscriber, then subscriber verifies this encrypted message and this allows the complete transmission of data. Unlike payload encryption, AES cipher block chaining - message authentication code (AES-CCM) mode is used as link layer encryption to provide node-to-node encryption. Consequently, the analysis power consumption, average round ripe time, random access memory (RAM) and read only memory (ROM) utilization of comparative with each encryption mechanism are shown in detail with different payload of sizes in byte.

OAuth 2.0 protocols is integrated to provide access control in MQTT communication protocol due to huge number of devices in IoT applications. Conventional authorization and authentication system is not suitable for IoT scenarios because it is strongly difficult to manage each device by using username, password and role credentials. In addition, there is no user interface or interaction between machine and developer so it is not easy to manage access authority. Access token and scope are obtained by using OAuth 2.0 protocol to connect MQTT broker that is Mosquitto. Whether device is publisher or subscriber is determined by using the scope information. MQTT protocol runs over TCP so it can use TLS as the secure channel

but IoT devices is limited in terms of battery and storage. The connection between client and MQTT broker is provided without TLS due to using 8bit-Arduino as a client. Refresh token is difficult due to limited storage. Only using same token leads to security vulnerabilities such as spoofing, denial of services. In addition to this, exchange scope is not possible because each scope belongs to an access token (Fremantle, Aziz, Kopecky & Scott, 2014).

Security with key/cipher text policy attribute base encryption (KP-CP ABE) using lightweight elliptic curve cryptography is provided due to broadcast encryption in MQTT communication protocol. Elliptic curve is used for optimizing complexity. Firstly, publisher prepares access list to send to broker and demands the key from broker to encrypt data with this key. It also adds information about user access to decrypt message and sends to broker. If there is access permission of subscriber, it receives this data and key from broker and decrypts data with the private key. However, there is no information about how the key is generated. Also publisher both sends access tree to the broker and takes the key from the broker with no secure channel so attacker can reach this data (Singh, Rajan, Shivraj & Balamuralidhar, 2015).

Zamfir, Balan, Iliescu & Sandu (2016) analyzed security based on the approach of pre-shared key, raw public key and certificate on both CoAP and MQTT protocol. In order to share securely videos data in real time, based on Hierarchical inner product encryption (HIPE) was proposed as end to end encryption on publish/subscribe architecture (Rajan et.al., 2016).

Kinikar & Terdal (2016) developed a fire alarm IoT application by using OAuth protocol for giving access without shared username and password to Gmail, twitter. Niruntasukrat et. al. (2016) implement to provide access control mechanism with adapted OAuth 1.0 protocol on IoT systems that IoT devices communicate with each other over MQTT protocol. The aim of this study is to avoid too heavy encryption algorithm in order to authenticate device to broker and ensure data privacy between client and broker by using OAuth 1.0 protocol. Unlike OAuth 2.0 protocol, client sends its password by digital signing to provide confidentiality in OAuth 1.0 protocol. The

scope of this study, their security design is analyzed in terms of security hole. Real IoT application is implemented by using three different IoT devices and their message latency are compared with in terms of devices resource after applying authorization mechanism.

Lightweight asymmetric encryption AA-Beta is preferred for constrained device in IoT application and is analyzed encryption and decryption based on CPU clock cycle for each process (Syed Farid, Mohd Anuar & Habibah, 2016).

Mosquitto auth plugin is added to provide access control with ACL by storing user name and password as encrypted on the database (Upadhyay, Borole & Dileepan, 2016).

Gantait, Patra & Mukherjee (2016) developed security mechanism that includes web, mobile, IoT client applications on the three layers of IoT application. Device authentication was provided with using a token as the username and application id on IBM Watson IoT platform in consideration of without using TLS due to constrained resources. OTP was added as an extra security level with mobile application in order to protect device from physical attack. Payload encryption was implemented to prevent against eavesdropping. After the secure data transmission, how to securely store the data on the cloud was explained by IBM Bluemix cloud IoT platform. Device access was managed by using OAuth 2.0 protocol which HiveMQ broker supports.

Suggested authorization security mechanism related to OAuth 2.0 on MQTT protocol is implemented. In addition to this existent and suggested security mechanism on MQTT protocol, HOTP with hash chain is used for mutual authentication and AES is used for data confidentiality.

CHAPTER THREE

LIGHTWEIGHT COMMUNICATION PROTOCOLS

In order to give a better making decision of which lightweight communication protocol is used in IoT scenario firstly, the details of communication protocols by considering to their capacity of storage and energy consumption must be compared. In particular, used protocols in IoT scenarios are mentioned in terms of their usage purpose and features. Especially for constrained resource device, MQTT is much more suitable communication protocol than other aforementioned lightweight protocols that are shown in Table 3.1 (Yerlikaya & Dalkılıç, 2017).

Table 3.1 Comparison of lightweight communication protocol

PROTOCOL	XMPP	CoAP	AMQP	DDS	MQTT
Rest	✗	✓	✗	✗	✗
Publish/subscribe	✓	✗	✓	✓	✓
Request/response	✓	✓	✗	✗	✗
Transport	TCP	UDP	TCP	TCP	TCP
security	TLS	DTLS	TLS	TLS/DTLS	TLS
Head size	-	4	8	-	2
QoS	-	-	✓	✓	✓

3.1 Constrained Application Protocol

CoAP was designed as a web protocol by working group of internet engineering task force (IETF) for restricted resource device and unreliable network (Frigieri, Mazzer & Parreira, 2015). Both CoAP and MQTT are suitable than HTTP for sensor network environment in terms of efficient energy consumption and low bandwidth thanks to their low header size respectively 4, 2 bytes (Bandyopadhyay & Bhattacharyya, 2013). In comparison of MQTT, CoAP is based on RESTFUL architecture and provide security end-to-end encryption over the datagram transport layer security (DTLS) instead of TLS since it runs over datagram transport layer (UDP) (Elgazzar, 2015).

3.2 Extensible Messaging and Presence Protocol

XMPP is based on both publish/subscribe and response/request schemes. It provides messaging in real time communication. The transmission and storage of the data takes place in extensible markup language (XML) format (Shelat, Patel & Bhatt, 2016). Therefore, much more power is required for resource-restricted device. In addition, XML structure needs high bandwidth so MQTT communication protocol with low header size and power efficiency is more preferable than XMPP communication protocol (Kirsche & Klauck, 2012).

3.3 Advance Message Queuing Protocol

AMQP is based on publish/subscribe scheme and generally most suitable for financial application as provide server-to-server communication than device-to-device communication (Patierno, 2016). As different from MQTT, AMQP contains exchange and queue features that are used to save data and forward the data to a related queue depending on specific rules. AMQP runs over TCP, so secure communication is provided by using TLS. AMQP provides more reliable communication in insecure network environments, as it stores incoming data. The AMQP header size is 6 bytes more than the MQTT header size. Therefore, MQTT contains more capacity of message. The communication is more secure and reliable with AMQP, while energy efficiency is ensured with MQTT (Luzuriaga, 2015).

3.4 Data Distribution Service

DDS is based on publish / subscribe schemes like MQTT and provides real time communication with running over TCP. In difference with MQTT, in order to minimize message loss and delay, it has much more QoS than MQTT has (Ungurean & Gaitan, 2015).

3.5 Message Queue Telemetry Transport Protocol

MQTT communication protocol was developed by IBM in 1999 and was standardized to provide lightweight machine-to-machine (M2M) communication by OASIS in 2013. MQTT is a suitable communication protocol for IoT scenarios based on publish/subscribe architecture. The protocol is more preferred for device with limited processing, limited energy, memory, lower bandwidth and better performing data size than CoAP in insecure network environment (Soni & Makwana, 2017). In Figure 3.1, the MQTT message format is displayed.

MQTT consists of client that can be publisher or subscriber, and broker components. The client (publisher) transmits the sensor data to the broker under a specific topic. The broker will forward the data to the subscriber by filtering out the access right and the client (subscriber) of interest. Publisher and subscriber do not have the knowledge of each other, such as IP address. All communication is realized through over the broker. Publisher and subscriber do not have to be connected to the same network at the same time, since the protocol performs asynchronous transmission (Shinde, Nimkar, Singh, Salpe & Jadhav, 2016).

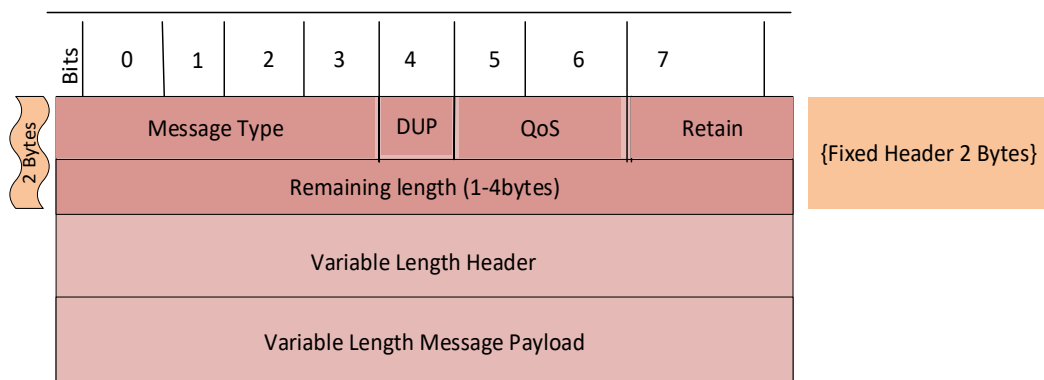


Figure 3.1 MQTT message format

The fields of MQTT message format are:

- **Message Type:** MQTT protocol contains 14 MQTT message types that are CONNECT, CONNACK, PUBLISH, SUBSCRIBE, SUBACK,

UNSUBSCRIBE, UNSUBACK, PUBACK, PUBREC, PUBREL, PUBCOMP, PINGREQ, PINGRESP and DISCONNECT. As an example the message types used for message transmission between client and broker are shown in Figure 3.2.

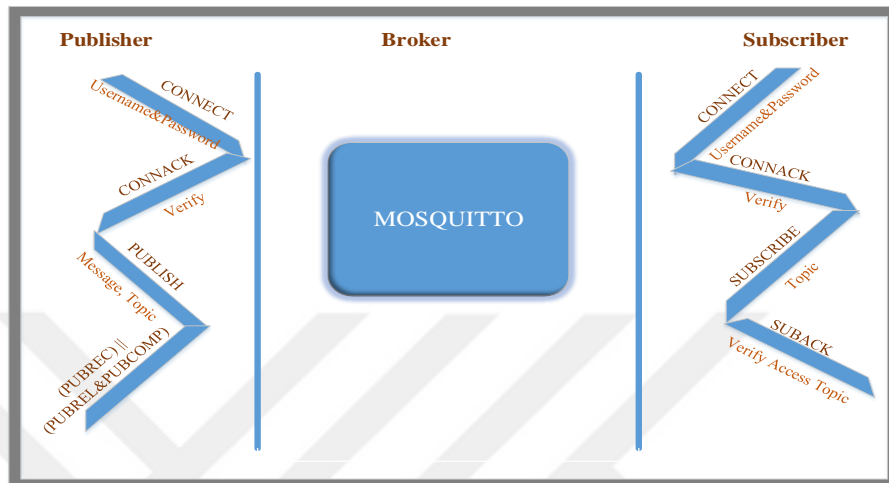


Figure 3.2 MQTT message types

- **DUP:** It refers as a flag that either the message is duplicate or not.
- **QoS Level:** MQTT includes three different messaging quality of service levels such as at most once, at least once and exactly once in case of unsafe network environments or connection failures that may occur on the network. Data is transmitted with the appropriate one of these services by depending on the constrained bandwidth. The most reliable level requires much more bandwidth. Data may be lost at the lowest level and network traffic is low in this level (Lee, Kim, Hong & Ju, 2013). The levels are:
 - **At most once:** Message is only sent once and this level does not guarantee the transmission of the message.
 - **At least once:** Message is sent at least once. The message can be transmitted repeatedly. Network traffic may occur increasingly.

- **Exactly once:** This level is preferred to be sure that the message has been delivered. When the message is received, response redirects PUBREC, PUBREL, PUBCOMP messages as acknowledgement. High network traffic occurs in this level.

- **Retain:** It informs to the broker by pointing out last published message and the message is transmitted as the first message to the new subscriber.

- **Remaining Length:** It indicates the remaining size of message.

- **Variable header:** It includes information about protocol name and version, keep alive timer, connect flags such as QoS, retain, clean session.

- **Payload:** It includes information about username, password, topic name and message. topic is a UTF-8 string and can consist one more topic with slash.

3.5.1 Security of Message Queuing Telemetry Transport Protocol

IoT scenarios are getting increased day by day and many devices connect to the Internet for transmitting data with each other by using communication protocols. Eavesdropping and accessing transmitted data between client and server, modifying accessed data or re-routing to an untrusted server, denial of service attack, timing attack, etc. are some of the vulnerabilities of the MQTT protocol. Furthermore, MQTT protocol has no specific security mechanism. Credential sets such as OAuth token and the pair of username and password can be added to the connection message of the MQTT protocol. MQTT server can verify a client by using access control list (ACL), database or lightweight directory access protocol (LDAP). However, this identity information sent in plain text can cause man in the middle and replay attacks (Banks & Gupta, 2014).

Since MQTT protocol runs over TCP/IP, an alternative solution against the attacks is the TLS. However, the TLS requires heavy computation that is not suitable for IoT devices having limited resources (Mahmood, 2016). Whenever using the TLS is possible in implementation, it provides the secure channel between client and MQTT server that means all communication is encrypted between client and broker. So, even if the attacker eavesdrops the message, he cannot alter this data and cannot interrupt the communication.

Some brokers use extra security mechanisms such as payload encryption, payload signing, complex authentication protocol, authorization/topic permission, etc. by using plugin mechanisms (MQTT security fundamentals, n. d). Publisher encrypts the payload that includes the user name and the password with the message and then sends this data to the broker. In an alternative payload signing mechanism provides, if the payload is accessed by the broker, it is signed with a private key by the broker and then sent to the subscriber. Therefore, the subscriber can be sure that the message comes unaltered from the correct source. Authorization/topic permission provides restriction to access all topics by all subscribers. Broker filters the received message from the publisher in terms of topic and then sends this filtered message to the subscriber. The subscriber receives the message from the topic that he is interested in (Obarmaier, 2015).

CHAPTER FOUR

PROPOSED METHODS OF SECURITY MECHANISM

In chapter three, MQTT security is explained from start to end without getting into excessive details. In this chapter, combining chapter three's information with contribution facilities with OAuth 2.0 protocol, AES encryption algorithm and HOTP with hash chain application's development process and authentication and authorization security mechanism will be put forth.

4.1 The OAuth 2.0 Authorization Framework

The OAuth 2.0 is an open authorization and authentication protocol that was standardized in 2012 by IETF RFC 6749 (Hardt, 2012). Third party application can store and use user credentials such as password and username as a without user information until the user changes his/her credential. OAuth 2.0 protocol is preferred to manage client's own authority without sharing its credential in order to provide communication with secure IoT device (S.W. Jung & S. Jung, 2017). OAuth 2.0 protocol has four roles (Chae, Kim & Cho, 2017) as stated below;

- **Resource owner:** This is a person who has a resource that is gained authority.
- **Resource server:** A server contains protected resource owner that is owned by user and it manages access to resource owner by using access token.
- **Client:** A client is an application that runs on the server and makes a request on behalf of the resource owner.
- **Authorization server (AS):** Resource owner is authenticated and gained authorization by server and then, access token is sent to client.

These four roles can be explained in social network like this example: Resource owner can be you that shared your images, videos and tweets are your special data in your tweet application. Client is an application that wants access resource owner's data, that is, your shared data from your tweet application. Client can be a mobile application, web application etc. Authorization server is the identity system to give an authority to the client application to get resource owner's tweet, so your tweet server needs to know client application. In this case, firstly you need to save the address information of client application on twitter servers. Client application will then be authorized when a request is made to your twitter servers from the client application. The twitter servers that authenticate and authorize, are the authorization server in the OAuth 2.0 protocol. Resource server is the location where the protected data to be accessed is located. For example, twitter API endpoints is called resource server which the tweets client application want to gain access.

The flow of OAuth 2.0 protocol is indicated interaction between four roles (Li, Mitchell & Chen, 2018) as shown in Figure 4.1.



Figure 4.1 The OAuth 2.0 protocol flow

A- Client wants to gain authority from resource owner.

B- The client receives an authorization grant, that is an identity acting the resource owner's authorization, phrased using one of four grant types defined as authorization code, implicit, password credentials, client credentials used by client to request authorization by the authorization server.

1. Authorization code grant: It is running on server-based application. Authorization server generates code and sends to client after user is authenticated or gain access. Client sends its credential via this code to demand access token.

2. Implicit: Access token is obtained directly without using authorization code. It is preferable for client with implemented in the browser using a script language.

3. Resource owner password credentials: In case trust between client and resource owner, access token is obtained directly with username and password.

4. Client credentials: Client can be access its own resource by using client credentials as an authorization grants.

C- The client is authenticated by authorization server and request access token via authorization grant.

D- Client and authorization grant is validated by authorization server. If they are valid, authorization server gives access token.

E- Client demands protected resource from resource server and is authenticated via access token.

F- Client is validated via access token from resource server and if it is valid, resource server gives response to request.

The MQTT protocol has no an access principle designed. The OAuth 2.0 protocol is preferred to provide authorization mechanism in IoT scenario because it is difficult to manage setting up credentials and fine-gained access of huge number of devices. In addition to this, it is difficult to manage their role-based access control due to a great number of devices. Furthermore, there is no user interface or interaction between human and device to control.

The MQTT protocol provides authentication with username and password. CONNECT message includes client id that is generated randomly by broker or generated as statically and embedded on device by developer. In this study, clients are prevented to have same credentials by developed authorization and authentication mechanism based on OAuth 2.0 identity and access management protocol that gives client id, client secret, scopes and access token in order to indicate unique client and client's access.

The WSO2 identity server (WSO2is) acts as an authorization server and running playground2 OAuth 2.0 web application on it to obtain access token and scope in order to identify device and restrict device authority. The WSO2is consists identity and access management protocols such as OAuth 2.0 and OpenID Connect is an open-source implementation (WSO2is, 2017). It is indicated to show respectively step by step in Figure 4.2 and Figure 4.3 how the access token is obtained via the OAuth 2.0 with playground2 web application on WSO2is.

Curl command line tool is used to obtain access token on Linux machine. Client id, client secret, username and password are required to request a token.

```
Curl -- user mfsEQkwtfKINK0npCSWRb66l6jIa: 1xIzWKPWOHIGTY g1bw4IM0QfQwa -k -d
“grant_type=password&username=admin&scope=
W3sienCiOijydyIsInRvcGljIjoiL3RlbXB1cmF0dXJILyJ9XQ==” -H “Content-Type:application/x-
www-form-urlencoded” https://localhost:9443/oauth2/token
```

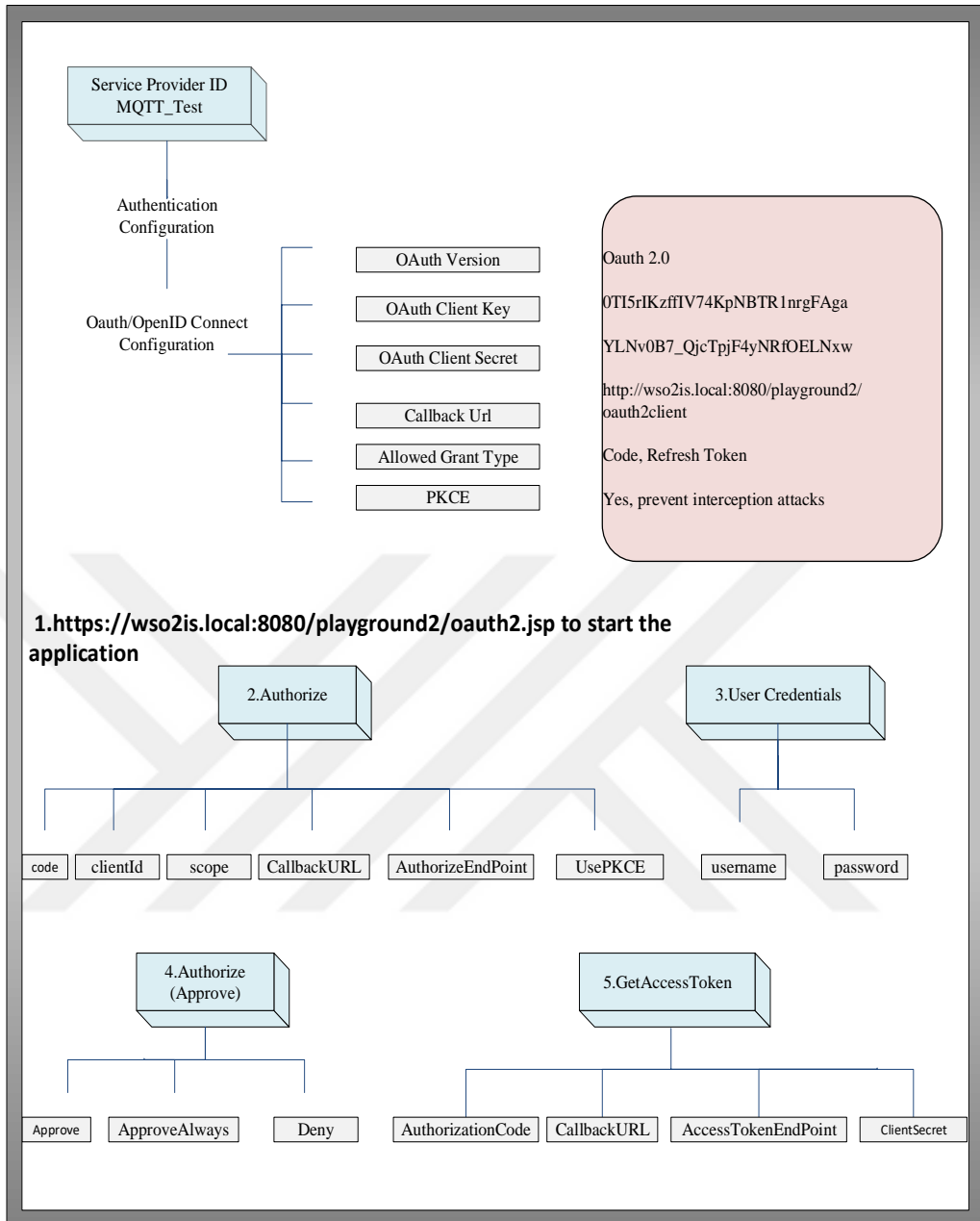



Figure 4.2 Obtained access token step by step

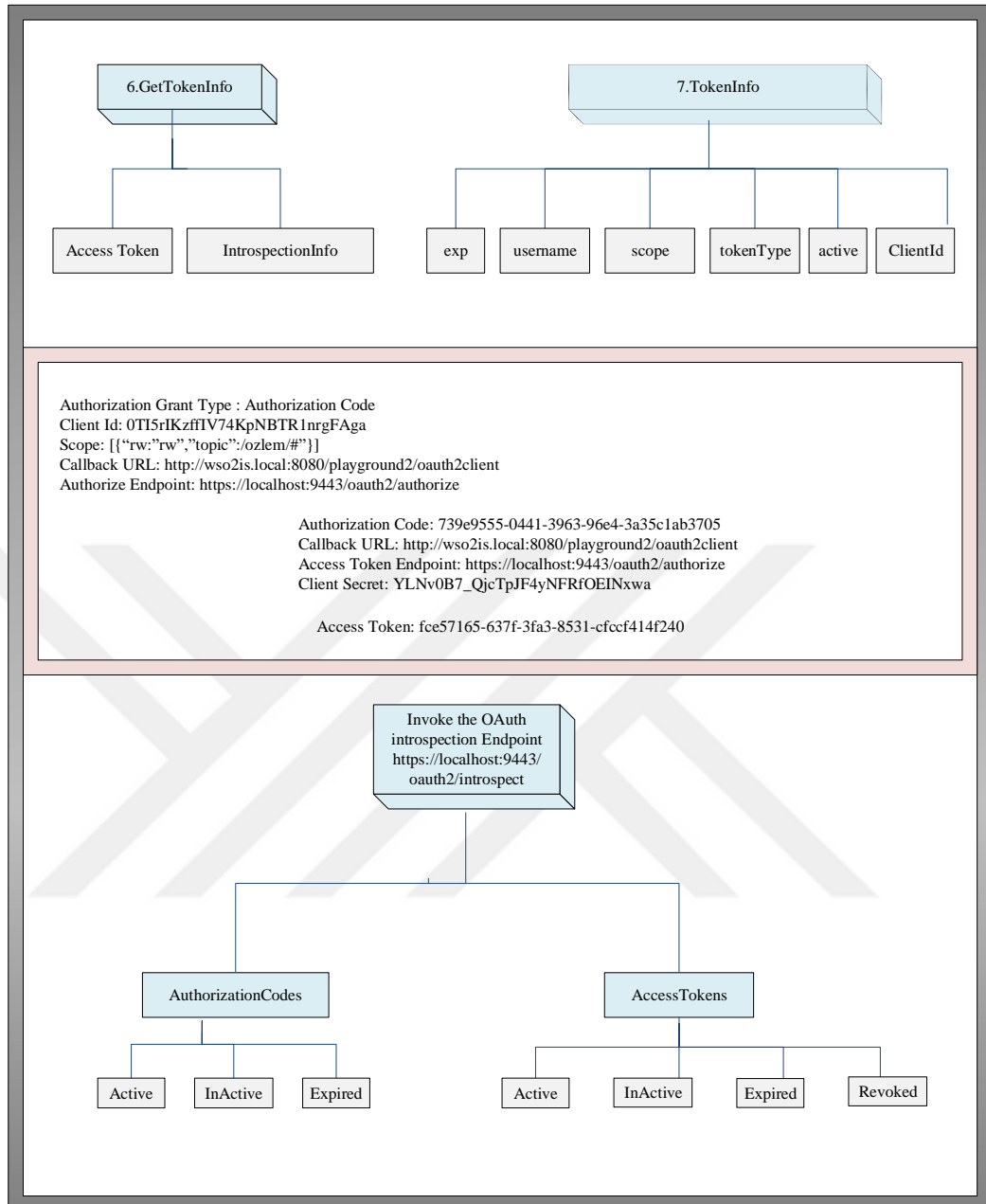


Figure 4.3 Obtained access token step by step

MQTT protocol consists client that can be publisher or subscriber. In order to indicate interested topic and whether permission of read or write, scope is created as JSON structure. Scope is [{"rw": "rw", "Topic": "/temperature/"}] in the curl example. Scope is defined in Base64-encode converted value in curl command. Obtained access token is embedded manually to the device that has scarce processing and energy resources. Each access token has static scope but client can be interest different scope more than existed scope, so exchange scope is major problem that is mentioned in this

implementation (Fremantle et. al., 2014). The flow of obtained access token from WSO2is is shown in Figure 4.4.

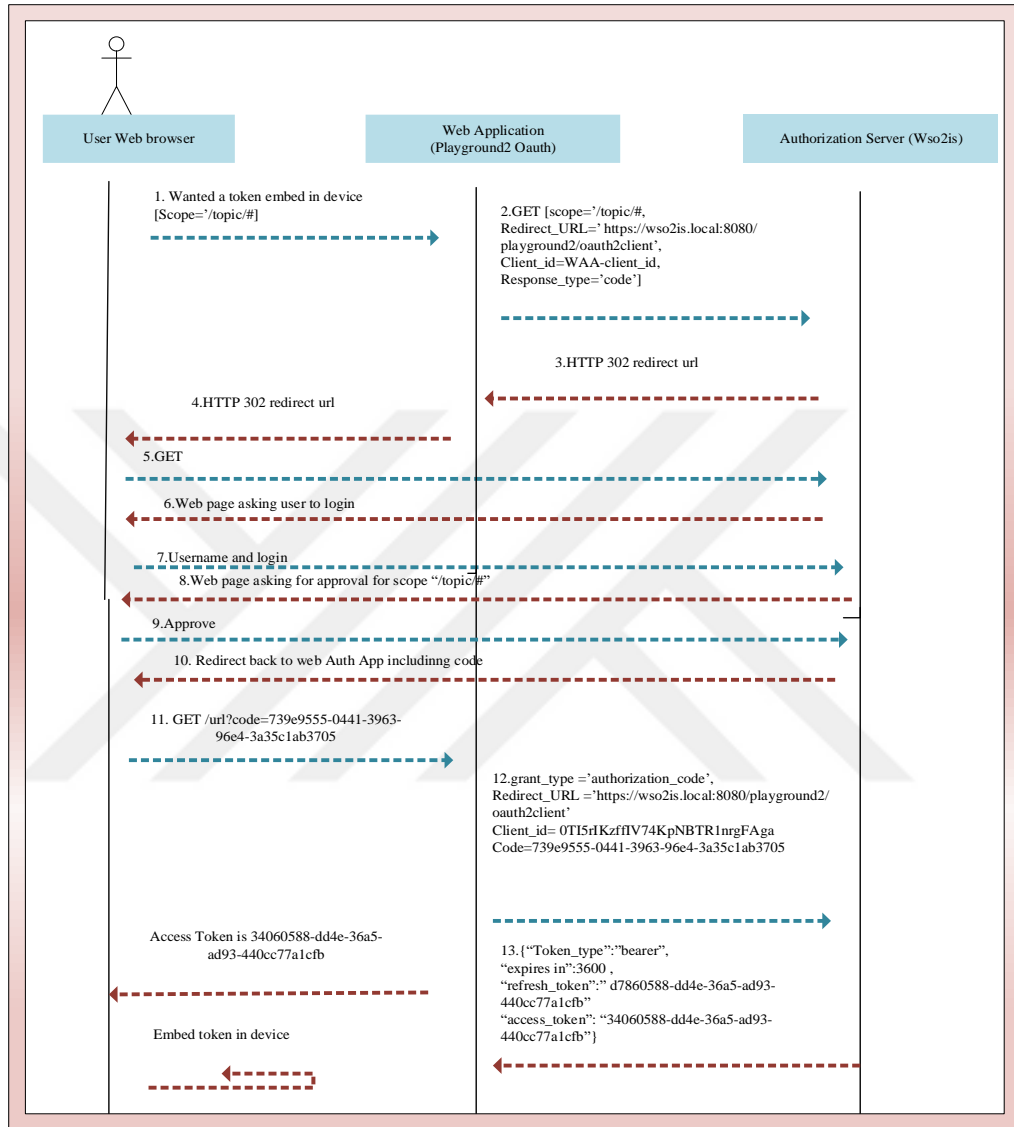


Figure 4.4 Flow of obtained access token

This implementation is based on previous done study (Fremantle et. al, 2014) that indicates in case of a time expiration of the access token, re-generating access token with refresh token in client code may not be possible because of the restricted resources of the device. In addition to this, access token is sent to broker as a plaintext. However, using the same token with insecure channel can give a rise to the security vulnerabilities such as spoofing, eavesdropping, man-in-the-middle, and replay. Therefore, in order to prevent these attacks, contribution on previous done study with

AES encryption and HOTP with hash chain are implemented to increase security level. They are detailed on the following pages.

4.2 HMAC-Based One-Time Password

There has been no method or plugin aside from using certificate to create a mutual authentication mechanism in MQTT protocol yet (Banks & Gupta, 2014). Client can be authenticated by the broker. However, there is no study to authenticate the broker by the clients aside from TLS. It is significantly critical since all communication is performed over the broker that receives data from the publisher and filters these data according to the topic where subscriber is interested in and then, sends it to the subscriber. Unless the broker is authenticated, client cannot be sure that it is communicating with the correct broker or has been forwarded to a third party broker. Considering the devices having restricted resource, hash chain without secret key to confirm the broker by the client is designed to avoid heavy computation on the client's side besides the authentication and access of the client.

HOTP was standardized as IETF RFC 4226 in December 2005. HOTP is preferred in order to provide not only two-way authentication but also to prevent replay attacks without time synchronization (H. J. Kim & H. S. Kim, 2015). The HOTP algorithm is based on HMAC-SHA-1, a shared unique secret (K) between a client and the broker, and a counter value (C) that must be synchronized with the HOTP generator and the HOTP validator. OTP is generated from a HMAC-SHA-1 value by truncate (M'Raihi, Bellare, Hoornaert, Naccache & Ranen, 2005; Saxena, 2007). The HOTP is expressed as in (4.1).

$$HOTP(K, C) = Truncate(HMAC-SHA-1(K, C)) \quad (4.1)$$

In this study, obtained client id from WSO2is is used as unique secret to generate HOTP value since every access token is owned unique client secret and client id. Furthermore, the client id is kept as a secret without sharing with broker because client id will be obtained if access token is valid after client is authenticated with access token from authorization server by broker. Therefore, Attacker does not easily guess HOTP value by not stealing or eavesdropping client id over insecure channel.

Broker acts OTP generator based on HOTP with Lamport's backward hash chain scheme (Park, 2018). As hash function exponentiation requires high computation aside from client, it is critical problem in the Lamport's scheme. However, N-times hash chain is generated by broker whereas client acts as a validator and only stores $N + 1^{th}$ hash chain value. Client validates broker until N equals to zero. Client has $(N + 1)^{th}$ (HOTP (client id, 5)) value. Broker generates N-times HOTP-hash chain value (4.2).

$$h^N(p)=h(h(\dots h(\text{HOTP}(\text{client id}, 5)))) \quad (4.2)$$

The other significant problem is re-initialization of hash value in case the value of N equals to zero. In this study, HOTP value and status are recorded on MongoDB, in order to update and re-initialize HOTP value by performing an algorithm with the indicated secret value.

In order to understand the advantages of using OTP to authenticate, it will be better to follow approach aware of occurring possible attacks between client and broker in case of insecure channel for implementation. First, MQTT connection between client and broker will be displayed without security mechanism, and then, details of each part will be explained as measure against security vulnerabilities in following sections.

4.3 Advanced Encryption Standard Algorithm

Advanced encryption standard (AES) is a symmetric encryption that encrypts and decrypts message by the same key, and it is supported by constrained devices. AES encryption is a block cipher that it includes 128-bit block length, key that has different length such as 128, 192, 256 bits. Each key has different process round respectively as 10, 12, 14. In encryption state, each round consists operations such as substitute bytes, shift rows, mix columns and add round key, whereas the operations of inverse shift rows, inverse substitute bytes, add round key and inverse mix columns takes place in decryption state (Stallings, 2006). AES structure is shown in Figure 4.5.

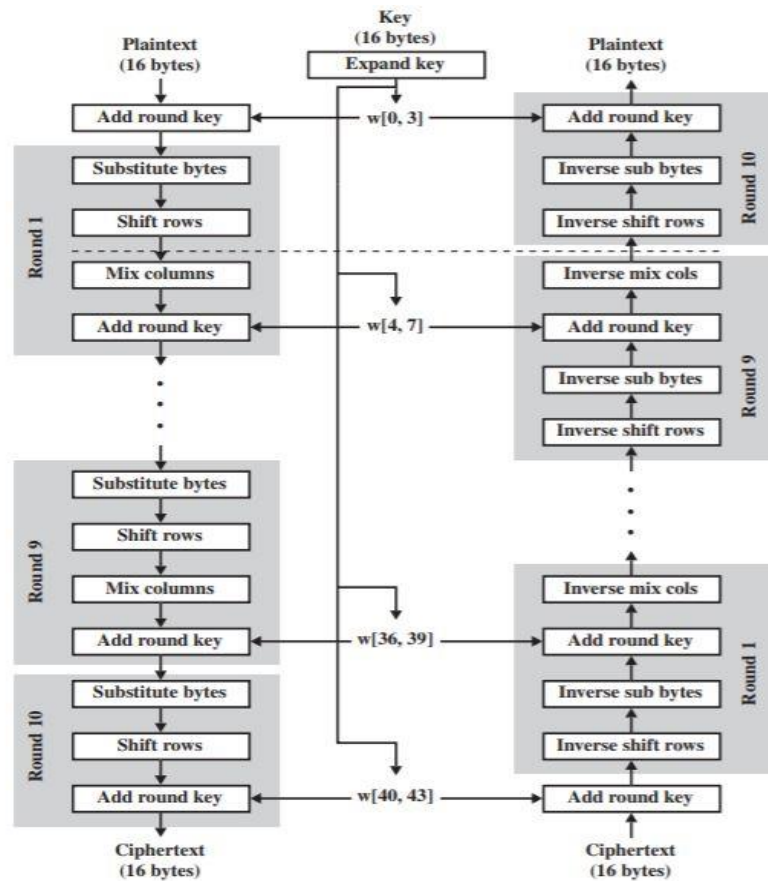


Figure 4.5 AES structure (Stallings, 2006)

Token is used for client's password-based authentication via insecure channel. In order to prevent eavesdropping, password is encrypted by cipher text feedback (CFB) mode of the AES. CFB mode requires that initially initialization vector (IV) is encrypted with encryption key (K). And then, the result of the encryption is XORed with plain text and operated result (C1) is used as IV for the following block cipher. The process performs for all blocks based on the previous operated result (C1, C2, C3 etc.) (Blazhevski, Bozhinovski, Stojchevska & Pachovski, 2013). Both encryption and decryption have the same process that it is respectively shown in Figure 4.6 and Figure 4.7 (Stallings, 2006).

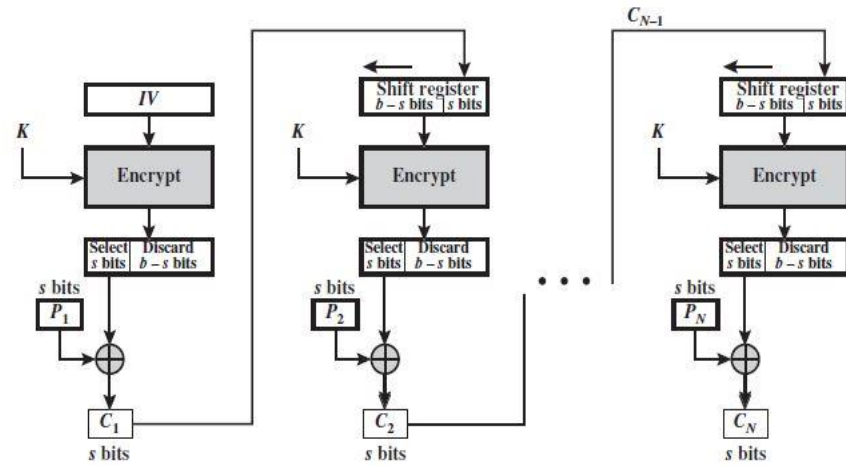


Figure 4.6 CFB mode encryption (Stallings, 2006)

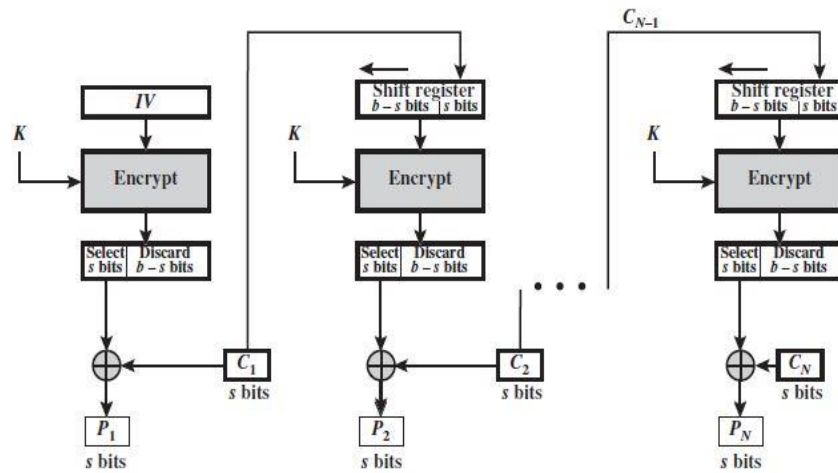


Figure 4.7 CFB mode decryption (Stallings, 2006)

Using AES encryption protects credentials against spoofing and eavesdropping whereas, it does not prevent replay and man in the middle attack. Password in every connection is changed by HOTP with hash-chain to protect against aforementioned attacks. AES symmetric encryption is supported on MQTT protocol, but key generation and distribution is a hassle when too many devices are considered. In this study, password decryption is implemented by broker in case of client-broker encryption. Client id which is used as public key that is owned by client, is obtained after the client is authenticated with the access token from the WSO2is by the broker. If token is validated, token information is requested that includes expires time, username, scope, token type, active and client id. Therefore, distribution key problem is solved as key is known by both client and broker.

CHAPTER FIVE IMPLEMENTATION

As mentioned in Chapter 4, the proposed methods are briefly summarized and the details of the implementation are given in this chapter.

MQTT protocol has two components that are client (publisher or subscriber) and the broker. Three main ideas exist in terms of security in this study. Firstly, the clients must be authenticated by the broker. There is no data transmission between an unauthorized client and the broker. The second, client must send or receive message from the correct broker to ensure that it takes data from the trusted source. The last one, every client can only access to its interested topic according to its read / writing permission.

5.1 Experimental Setup

The components of MQTT protocol and tools to provide security are explained in this section. The selected software and hardware information is given on the Table 5.1.

Table 5.1 Selected software and hardware information

WSO2is 5.3	Requirements	Version
	Oracle Java SE Development Kit	JDK1.8.0_144
	Apache ActiveMQ	5.15.3
	Apache Ant	1.10.1
	Apache Maven	3.5.0
OAuth 2.0 configure	Playground2 web application	
	Tomcat	7.0.82
	Sample of Identity Server	
MongoDB 3.6	Mongo-c-driver	1.10.0
	Libbson	
Mosquitto_pyauth app.	Python 2.7	
MQTT Broker	Mosquitto (Linux Machine)	1.4.8
MQTT 3.1.1 Client library	Eclipse_paho	
MQTT Client Tool	MQTT.fx	
	Mosquitto_pub/mosquitto_sub	

5.1.1 MQTT Client

Various MQTT client libraries are available based on in diverse programming languages. For example, M2MMQTT, FuseSource, machine head, MQTT.js, ruby-mqtt are implemented with programming languages respectively C#, Java, Clojure, JavaScript, Ruby. Eclipse-paho client is supported by many programming languages such as C, C++, Java, JavaScript, Python, Go, C# (Obermaier, 2015).

Eclipse-paho is an open source MQTT client and is implemented with Python 2.7 language in this study. The Eclipse-paho MQTT client is supported by MQTT 3.1 and MQTT 3.1.1 protocol (Paho project, 2015). It connects to MQTT broker and acts a publisher or subscriber.

MQTT 3.1.1 version is preferred since it allows the length of the identity of the client to be greater than 23 bytes. In order to evaluate the improved security mechanism, client tools are used. Mosquitto_sub / Mosquitto_pub is used on server without GUI whereas MQTT.fx shows broker statistics with GUI.

5.1.2 MQTT Broker

All communication takes place over the broker in the MQTT protocol. Eclipse Mosquitto is an open source message broker and is written with the C language. It is supported by MQTT 3.1 and 3.1.1 protocol. Mosquitto is especially suitable for IoT application (Light, 2013). To prevent stealing and modifying data between the client and the broker, Mosquitto is configured with TLS or authentication and authorization mechanism is added by using backend database, ACL file and certificates to increase secure communication. In this study, a plugin is written to develop security mechanism with Python language on Mosquitto-v1.4.8.

5.1.3 Authorization Server (WSO2is)

WSO2 identity server (WSO2is) is an open source identity and access management that includes many services such as authentication with Kerberos, X509 and OTP with SMS or mobile application in addition to access control with OAuth 2.0, OpenID etc. (WSO2is, 2017). WSO2is acts as an authorization server in this study. OAuth 2.0 authorization protocol requires callback URL so it is configured with playground 2.0 web application. Curl is used in command line to obtain access token via HTTPs Internet protocol on the authorization server. Client authority and identity are validated with the token and scope by introspection REST API on the plugin over the broker.

5.1.4 MongoDB

MongoDB is an open-source the one of the NoSQL database that stores data in JavaScript object notation (JSON) document format. It makes easier usage since complex hierarchical structures can be defined with embedded array or object as one record. Furthermore, it performs high scalability in case huge number of clients' connection to the IoT application. As MongoDB includes features such as indexing, sorting, filtering, aggregation unlike the relational database, instant queries are made faster (Chodorow, 2013). While many NoSQL databases allow access to the database only via keys, MongoDB also offers regular expressions, as well as querying for desired fields and range queries.

In this study, since the record can consist the embedded array of document, different size and type of the pair of field-value for each document, MongoDB is especially preferred to provide dynamically HOTP value synchronization between client and server than other database. Moreover, adding, removing and updating are operated directly by indicating certain value with index. MongoDB plugin is integrated on Mosquitto and client secret is used to provide uniqueness for each client on MongoDB. When the MQTT security mechanism is developed it is assumed that security is provided between the MQTT components and the MongoDB.

5.2 Proposed Authentication and Authorization Mechanism

Previous the done study (Fremantle et. al., 2014) related to OAuth 2.0 framework has been implemented with contribution of AES encryption and HOTP with hash-chain to identify securely both client and broker. The contribution is made to the previous work for the authentication and access authority is shown in the flow chart in Figure 5.1.

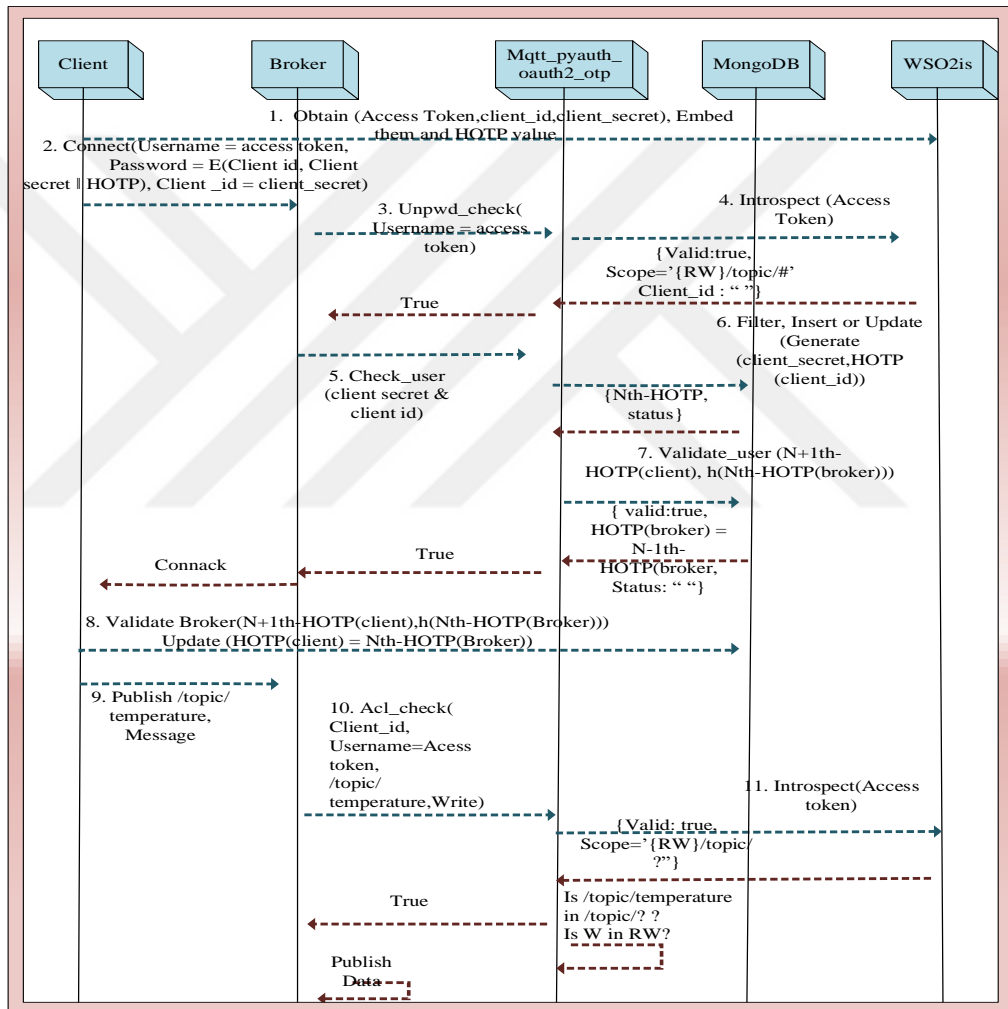


Figure 5.1 The flow of authentication and authorization mechanism

Initially, every one of the clients is registered to the identity server that is known as WSO2is in the study, and then configure OAuth 2.0 service provider with playground2 web application (Wso2is, 2017). Service provider generates unique client id and client secret for each registered client to obtain access token and determine client' access,

that is a scope. Access token is taken with using curl command line tool on Linux machine. Client's N^{th} (HOTP) value is generated by using client id as a secret key, taking C is a static number. Generation HOTP is defined in (5.1)

$$\text{HOTP}(\text{client id}, 5) = \text{Truncate}(\text{HMAC-SHA-1}(\text{client id}, 5)) \quad (5.1)$$

The HOTP value, client secret, client id and access token are manually embedded in client such as device, smart card, etc. by the developer. Each client's client secret is a unique value. The HOTP value depending on the client secret is inserted on MongoDB and is updated in every CONNACK message.

Eclipse-paho python client library is used as a publisher/subscriber client and Mosquitto which is an open source message broker written in C language, is used as the MQTT broker (Light, 2013). In order to generate HOTP in addition to the validation of the client and its authority on the broker, Mosquitto_pyauth github code (Bachry, 2013) is developed on Mosquitto broker. Client information and how to use this information with MQTT CONNECT message to connect to the Mosquitto broker is shown in Table 5.2.

Table 5.2 Client information

THE INFORMATION OF CONNECT MESSAGE	
CLIENT ID	"mfsEQkwtfKINK0npCSWRb66l6jIa"
CLIENT SECRET	"1xlZwKPW6OHIGTYg1bw4lM0QfQwa"
ACCESS TOKEN	"607c8f8b-9a16-3663-9bfe-0f05ddf34c33"
N^{th}HOTP	200289
USER NAME	Access Token
PASSWORD	AES-256(client id, Client secret HOTP)
PORT	1883
BROKER ADDRESS	193.x.x.x

Client id: Client id is used as both public key for AES encryption and secret key for generation HOTP. It does not share with broker. Client id is obtained as information of active token when the client is authenticated.

Client secret is used to create document for each one of the clients on MongoDB in order to provide a unique record Also for the re-initialization of HOTP value and the management of synchronization between client and broker are performed by filter the client secret with document on MongoDB. To prevent from eavesdropping, client secret is encrypted by AES-CFB mode with client id as a key that is 256-bit length. The length of the client id is not static, so its length is completed by padding with zero in case of being less than 32 bytes. The challenge of key distribution over the insecure channel is overcome by using the client id that is known to both the client and the broker. Only used AES encryption does not prevent replay attack or man in the middle attack. So client secret and HOTP value are concatenated to change password in every connection and then, are encrypted. The last 6 digit of password expresses HOTP count.

Access token is obtained by curl command line tool from authorization server to authenticate client by using as the username in CONNECT message. The information of the access token is informed in Table 5.3. So, as to avoid re-generating token in case of a time expiration of the access token, a long-lived expiration time is set up.

Table 5.3 Access token information

Access Token	607c8f8b-9a16-3663-9bfe-0f05ddf34c33
Refresh Token:	A05da6fb-ab14-35ea-8ab9-8a39086c2c1e
Scope:	W3sicnciOiJydyIsInRvcGljIjoiL3B6Zi8ifV0=
Token Type:	Bearer
Expires in:	1080000

In our study, undermentioned solution is recommended instead of using static same password and username in terms of transmitting without secure way. As well as using

a token, to put the security one more step ahead, hash chain is used to validate the client in case of a stolen token.

HOTP is used to prevent replay attack by generating different value for each one of the CONNECT message. Moreover, HOTP with hash-chain supplies broker authentication by the client side. Therefore, Publisher sends message to the trusted broker, and subscriber ensures to receive message from correct broker.

In respect of generation and verification of the HOTP, broker acts as generator whereas client stores only calculated last HOTP value and then, the HOTP count are updated depending on the verification processes on MongoDB.

Initially Generated HOTP Values:

$$\text{Client HOTP value} = N^{\text{th}}\text{HOTP}(\text{client id}, 5),$$

$$\text{Broker HOTP values} = (N - 1)^{\text{th}}\text{HOTP}\left(\text{HOTP}\left(\text{HOTP}\dots\left(1^{\text{th}}\text{HOTP}\left(0^{\text{th}}\text{HOTP}(\text{client id}, 5)\right)\right)\right)\right)$$

This equation is exemplified like below Table 5.4.

Table 5.4 HOTP count is based on client secret

HOTP Value / n^{th}	0	1	2	...	N-2	N-1	N
Client Secret: 1xlZwKPW6OHIGTYg1bw4lM0QfQwa							
Client HOTP	-	-	-	-	-	-	200289
Broker HOTP	354034	419503	763894	...	858123	543115	-
l's Connection Client	-	-	-	-	-	543115	-
l's Connection Broker	354034	419503	763894	...	858123	-	-
Client Secret: 5kK7XjmQf0RDOKgxVGQScJqT7rYa							
Client HOTP	-	-	-	-	-	-	224769
Broker HOTP	021964	211428	931713	...	273294	020723	-
l's Connection Client	-	-	-	-	-	020723	-
l'sConnection Broker	021964	211428	931713	...	273294	-	-

The first stage is that access token is validated from authentication server by broker, and then, if the client is valid, password is decrypted and obtained client's HOTP count

and client secret. Client secret is controlled on the MongoDB whether the HOTP was previously generated by the broker or not. If there is no match, firstly N-1 times HOTP values are created and inserted on MongoDB. The second stage, Client's N^{th} HOTP value is compared with taking hash $(N - 1)^{th}$ HOTP value that broker has. If the match is correct, client is validated by the broker. Broker updates its HOTP value with one step backward as $(N - 2)^{th}$ HOTP value. Broker responds as CONNACK message with result codes that indicates successfully connection, to client. Before the client receives/publishes message, Client's N^{th} HOTP value is compared with taking hash $(N - 1)^{th}$ HOTP value that broker has. If they match, broker is validated by the client. Client updates its HOTP value with $(N - 1)^{th}$ HOTP value that broker has, and then, publishes or subscribes message based on client authority. Synchronization between client and broker is managed by MongoDB operation.

Username is generally used for password-based on authentication. Username can be token, OTP etc. In this implementation, access token is used as username to provide uniqueness credential of huge number devices. The pair of username and password as credential is sent to Mosquitto broker with CONNECT message for first step authentication.

Password consists of concatenation of client secret and HOTP count. Before it is transmitted to the broker, password is encrypted with AES-256 CFB mode by using client id as a key. Password always changes in every connection thanks to HOTP count. Client id is not shared between client and server. In this manner, even if the password is captured by an attacker, the attacker cannot alter password without decryption via key and cannot reuse password due to HOTP's short life span. Password is defined like this;

$$\text{Password} = \text{AES-256}(\text{client id, Client secret} \parallel \text{HOTP})$$

Authentication mechanism is made up two stages. Each step is explained with indicated MQTT message type step by step.

Step 1. Registered client to WSO2is, whose information aforementioned client id, client secret and access token, are used to show how the authentication and

authorization in the implementation of local area is performed. Developer calculates default HOTP value, that is determined 6th HOTP to sample, and embeds this HOTP value and client's information in the MQTT client. Registered three clients' information is given in Table 5.5. One of the clients has two different access token due to base on its scopes.

Table 5.5 Registered three clients' information and their 6thHOTP counts

Client	Access Token	Client id	Client secret	6 th HOTP
Pub1	4e526038-00c5-37b-fbb55-149c67a9a7de	mfsEQkwtfKINk0npCSWRb66l6jla	1xlZwKPW6OHIGTYg1b w4lM0QfQwa	322924
Pub1	69c71be5-f987-3464-9da4-681387b6f0d1	mfsEQkwtfKINk0npCSWRb66l6jla	1xlZwKPW6OHIGTYg1b w4lM0QfQwa	322924
Sub1	33e578ee-3f69-3737-8f99-951edc5352e7	01F3krL2tXHtmt55cowbv93dlBoa	5kK7XjmQf0RDOKgxVG QScJqT7rYa	623106
Sub2	f9db5044-a9ba-3c44-84ce-be6c39774292	nSAy8NhBCPpiUwfXen5Yp0dXLqMa	40GiRwSpht3JiaTd5fDF0 U36oGAa	808139

Scope is defined as JSON format for publisher as: [{"rw": "rw", "Topic": "/temperature/"}]. Its base64encode converted value is written as the scope to request access token from WSO2is via curl command line tool and following this:

```
curl --user mfsEQkwtfKINk0npCSWRb66l6jla:1xlZwKPW6OHIGTYg1bw4lM0QfQwa -k -d
"grant_type=password&username=admin&password=admin&scope=W3sicnciOiJydyIsInRvcGljJoiL3RlbXBlc
mF0dXJlYj9XQ==" -H "Content-Type: application/x-www-form-urlencoded"
https://localhost:9443/oauth2/token
```

The response from WSO2is is shown as below. After the validation client, scope is parsed as JSON.

```
{"access_token": "4e526038-00c5-37b-fbb55-149c67a9a7de", "refresh_token": "fca938fc-4976-391e-94bc
51a492d47789", "scope": "W3sicnciOiJydyIsInRvcGljJoiL3RlbXBlc mF0dXJlYj9XQ==", "token_type": "Bearer"
, "expires_in": 10800}
```


Step 2. In order to connect to the Mosquitto message broker, firstly client sends CONNECT message that includes client id, port, broker address, clean session and keep alive. In addition to this, optional username and password are used to provide authentication. Clean session is preferred 'False' or '0' to store all message and client information by broker in case message is demanded as QoS 1 or 2. 'True' or '1' clean session specifies that broker clean previous all communication and client. Keep alive indicates time interval whether the connection is reachable or not via PINGREST or PINGREC. Client id which is known both the client and the Mosquitto broker, is used as the key to encrypt password that contains concatenation of client secret and 6th HOTP. One of the authentication stages takes places via access token that is used as username in clear text. The second authentication is validated encrypted password that is changed via HOTP in every connection to prevent replay attack. The CONNECT message is indicated in Table 5.6.

Table 5.6 Connect message type

Client	Client id	Username(Access token)	Password AES-256
Pub1	1xIZwKPW6OHIGTY g1bw4IM0QfQwa	4e526038-00c5-37bf-bb55- 149c67a9a7de	K5gpx1FXFuSjNOQJTyHtpe 5isY3uVWSSy8rcjCm2_isFl2 pGcIetMqo2P3Qo4gtvfQfQN kqmqQe46VhFhS7eSQ
Sub1	5kK7XjmQf0RDOKg xVGQScJqT7rYa	33e578ee-3f69-3737-8f99- 951edc5352e7	rSAc4cDbhcbCuFwxPOJ98m 86BSPMFSOPcjPwTuLoi45w xttJJPvqx0DP7g4fooY6d9myi LAffMTi7ARTsH4rGA
Sub2	40GiRwSpht3JiaTd5f DF0U36oGAa	f9db5044-a9ba-3c44-84ce- be6c39774292	inDrRo3kCMI1_qGkJr2BckF PkGe4_sH6WMXK1rZ0X9Pc LPwMIZ3sd4v1MGaoY9xvR V7ITii5bL6QT4tYmArSDQ
<pre>client=mqttClient.Client(client_id=client_secret, clean_session=False, userdata=None, transport="tcp") client.username_pw_set(user, password) client.connect(broker_address=193.x.x.x, port=1883, keepalive=60)</pre>			

Step 3. Mosquitto_pyauth (Bachry, 2013) github code is written to improve application with python language instead of C language on the Mosquitto by Martin.

In this study, the application with this github code is developed to implement security mechanism. This application runs over Mosquitto message broker. Unpwd_check function is called to validate access token from WSO2is to authenticate client when the client connects. if username or password is empty, regardless of calling introspection API, a response message will be returned as an unauthorized client with CONNACK message to the client.

Step 4. The developed application of mqtt-pyoauth2.0_otp calls introspection API to obtain client id that is used as the key for decryption of the cipher password, by validation access token from the authorization server via HTTPs protocol. Then, the authorization server returns client information as the response in Table 5.7.

Table 5.7 Returned client information

Username:	admin@carbon.super
Exp:	1523833511
Iat:	1523822711
Token_type:	Bearer
Client_id:	mfsEQkwtfKINk0npCSWRb66l6jIa
Active:	True
Scope:	W3sicnciOiJydyIsInRvcGljIjoiL3RlbXB1cmF0dXJlLyJ9XQ==

Unless the token is active, the client is unauthorized and its access will be denied. CONNACK message with result code 5 is returned to the client, that means connection refused, not authorized. In case of the active token, first authentication step is completed successfully. CONNACK message with result code 0 is returned to the client, that means connection accepted, an authorized client. Client secret and 6^{th} HOTP(client) are obtained by using the client id as the key.

Step 5. To authenticate both client and broker, MongoDB is implemented as a backend on Mosquitto with application. In order to perform second authentication, 6^{th} HOTP(client) and calculated HOTP(5^{th} HOTP(broker)) are compared with each other, if values are equal, client will be validated.

Step 6. For step 5, first thing to be checked on MongoDB is whether the broker generates HOTP depending on client secret or not. If not, broker generates HOTP values N-times and inserts on MongoDB. The sampling was done by assuming that N is 5. In order to provide synchronization between client and broker, status code is kept for one of the HOTP values depending on the client secret. HOTP values with status code '2' is noticed in order to indicate the ordered process for the broker and the client. As an example using exist user, a user document with a generation HOTP might look like:

```
{ObjectId("507f191e810c19729de860ea")
  "user_id": " mfsEQkwtfKINK0npCSWRb66l6jIa",
  "HOTP":      [{"status":0, "count": 0, "broker_HOTP:" 042439"},
                {"status":0, "count": 1, "broker_HOTP:" 459020"},
                {"status":0, "count": 2, "broker_HOTP:" 273755"},
                {"status":0, "count": 3, "broker_HOTP:" 237502"},
                {"status":0, "count": 4, "broker_HOTP:" 335460"},
                {"status":2, "count": 5, "broker_HOTP:" 991565"}],}}
{ObjectId ("507f191e810c19729de860eb")
  "user_id":5kK7XjmQf0RDOKgxVGQScJqT7rYa
  "HOTP":      [{"status":0, "count": 0, "broker_HOTP:" 543862"},
                {"status":0, "count": 1, "broker_HOTP:" 128892"},
                {"status":0, "count": 2, "broker_HOTP:" 608455"},
                {"status":0, "count": 3, "broker_HOTP:" 070065"},
                {"status":2, "count": 4, "broker_HOTP:" 711973"},
                {"status":1, "count": 5, "broker_HOTP:" 0633688"}],}}
```

If the client secret exists on MongoDB, the document is updated in case of validation client.

Step 7. The HOTP value that depends on client secret and has status code as 2, is taken and its HOTP value is calculated. And then, it is compared with HOTP(client). If they matched, broker updates its HOTP value with the previous value of itself due to irreversible hash function and status of the new HOTP value is updated as 2. The status of the old HOTP value is changed as 1 to indicate ordered process for client. Pub1 is exemplified like this:

6-times HOTPs are generated by using client id and 5 respectively as secret, count.

Secret ← *Client id* 1xIZwKPW6OHIGTYg1bw4IM0QfQwa

$HOTP^0 = 042439$

$HOTP^2 = 273755$

$HOTP^4 = 335460$

$HOTP^1 = 459020$

$HOTP^3 = 237502$

$HOTP^5 = 991565$

$6^{th}HOTP$ ← *pub1 Client default HOTP* 322924

$5^{th}HOTP$ ← *status equals 2 based on client secret on mongodb* 991565

In case comparison is equivalent, the document based on client secret is updated.

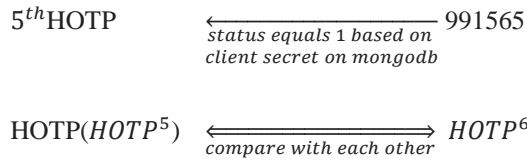
$HOTP(broker)$ ← *updated broker hotp as $HOTP^4$
 $HOTP^4$ status as 2 whereas $HOTP^5$ status is updated as 1* $HOTP^4$

Client receives CONNACK message with result code 0, that means client is authenticated and connection is successful. If the HOTP values do not matched, CONNACK message with result code as 5 is returned to the client, that means connection refused, not authorized.

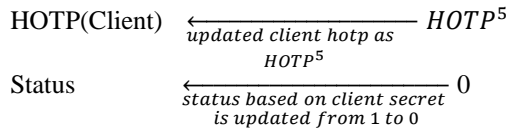
Step 8. After the client is authenticated with access token and HOTP value by the broker, broker is authenticated by client side. HOTP value is calculated with HOTP value that has status code as 1, and then, the calculated HOTP value is compared with client's own HOTP value. The aim of this step is to be ensured that publish or subscribe message is forwarded to the right broker. If the result of the comparison is true, client updates its HOTP value with the broker's HOTP value and also updates status codes from 1 to 0 to provide synchronization between the client and the broker to detect attack in case of interception of the MQTT message. Pub1 is exemplified with following like this:

In case CONNACK message with result code as 0 is received, broker is authenticated.

$6^{th}HOTP$ ← *pub1 Client HOTP* 322924



In case comparison is equivalent, client HOTP value and the document based on client secret are updated.



Step 9. After broker is authenticated by the client, the client acts as publisher or subscriber to send or receive message based on the client’s interested topic. In this step, client sends message with the related-topic on PUBLISH message to the broker.

Step 10. When the PUBLISH message is received by the broker, the broker calls `acl_check` function in `mqtt_pyauth2.0_otp` application.

Step 11. Application calls introspection API to give access control by comparing client access and scope that access token has. Authorization server replies the defined scope of access token and its validation to the application. Application parses scope as JSON format like in the Table 5.8.

Table 5.8 Parse scope of access token

Client/Access Token	Scope
Pub1/ 4e526038-00c5-37b-fbb55-149c67a9a7de	[{"rw":":rw", "Topic":"/temperature/"}]
Sub1/ 33e578ee-3f69-3737-8f99-951edc5352e7	[{"rw":":r", "Topic":"/temperature/"}]
Sub2/ f9db5044-a9ba-3c44-84ce-be6c39774292	[{"rw":":r", "Topic":"/humidity/"}]

Client read, write permission are shown as respectively r/ w. The topic of scope is checked with the topic of the message. If they match, client gains authority. For instance, four clients connect to broker that have different access authority in Figure 5.2. WEMOS has writing permission to publish message related to temperature topic, whereas the client tools with MQTT.FX on Linux machine, MQTT-lens on Chrome

web browser and MyMQTT on Android operating system have read permission with their interested topic in temperature to receive message.

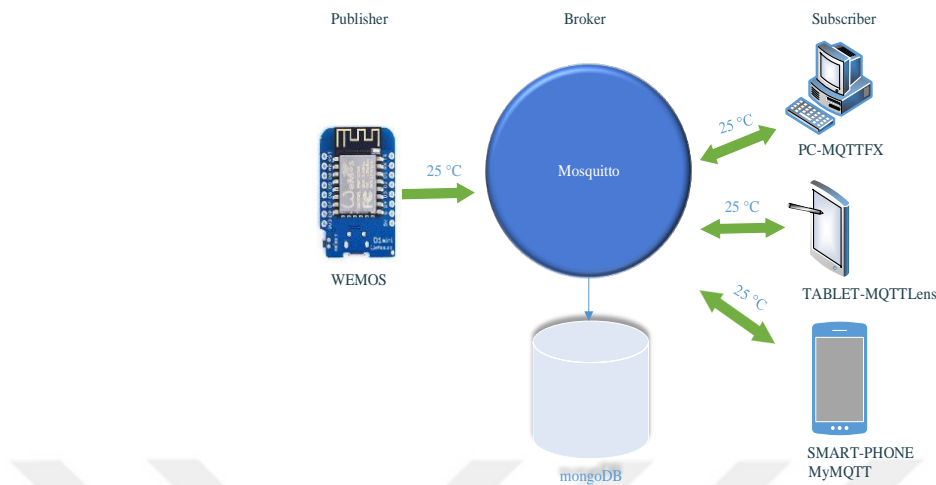


Figure 5.2 The pub/sub clients connect to the Mosquitto

When the publisher sends a message under a topic as temperature, the subscriber receives the message from broker according to the valid token and scope, that it is given in Table 5.9.

Table 5.9 The information of client's access

Client	Access token	Message	Topic	Scope Topic	Result Status
Pub1	Valid	25°	/temperature/	/temperature/	Authenticated and authorized
Sub1	Valid	25°	/temperature/	/temperature/	Authenticated and authorized
Sub2	Valid	denied	/humidity/	/humidity/	Authenticated but not authorized for temperature
Sub3	Invalid	denied	denied	denied	Not authenticated and authorized
Sub4	Valid	denied	/humidity/	/temperature/	Authenticated but not authorized due to not matching scope topic and message topic

Sub3 has not credential due to not registered to the WSO2is so its access is denied with response unauthorized. Whereas, sub1, sub2 and sub4 has valid access token and completed two authentication steps successfully. However, Sub2 can receive message only under a humidity topic so receiving message is restricted. Although sub4 can access the temperature topic it requests under the humidity topic and its access is denied. Sub1 gets the message since its authority and requested topic match.

CHAPTER SIX

SECURITY ANALYSIS OF THE IMPLEMENTATION

Chapter 6 describes how the vulnerabilities are prevented by using the developed MQTT security mechanism application.

6.1 Prevent Replay Attacks

Transmitted data is reused by an attacker on MQTT in order to interrupt the process by sending this data repeatedly. Generally, nonce and timestamp are suggested to prevent from replay attacks. In our implementation, in order to avoid time synchronization, HOTP has been preferred to change data continuously that means, obtained password has been invalid in a short-time.

6.2 Prevent Password Guessing

The necessary secret for HOTP has been kept confidential without transmitting it between client and broker to be not able to calculate HOTP. Therefore, an attacker cannot guess password easily without known secret.

6.3 Prevent Man-in-the-Middle Attack

The man-in-the-middle attack sniffs the data between the client and the server and can read, delete or modify this data. Even if data is encrypted, it can be altered due to pre-shared key. In order to prevent this attack, public key is not transmitted over insecure channel between the client and the Mosquitto broker. The key is obtained by verifying the client from the authorization server via HTTPs. In addition, password is modified by applying backward hash chain due to being non-invertible. Access token can be stolen because of sending it clear text. However, one more authentication step is implemented with hash-chain. Unless correct client secret key and HOTP value are obtained after decryption, there will be no user registration on MongoDB and client authority is denied.

6.4 Protect from Eavesdropping

Access token can be stolen but it is not enough to connect to the broker. Password is encrypted with AES-CFB mode and encryption key is kept secret without forwarding between the client and the broker. Unless the key is known by the attacker, plain text is not found.

6.5 Impersonation Attack Protection

If the attacker connects to the broker before the client, it is authenticated. At the same time, synchronization is degraded between the client and the broker. The MongoDB scheme is designed to understand whether there is any attacker or not. Client changes status attribute on MongoDB that includes three modes respectively 0, 1, 2. The mode 2, indicates the order the broker will process. The mode 1, specifies that the client updates the value of status attribute from 1 to 0 in case of receiving CONNACK message and then, publishes or subscribes message after the identified broker. The last mode 0, is to understand the re-initialization of the hash function in case all the values of status attribute are equal to zero. And then, the last count of HOTP is used as secret to generate HOTP again as many as the previously mentioned number and the client HOTP is updated with one more calculation hash of the last HOTP to provide synchronization. If synchronization cannot be achieved, the client is reset. If the impersonation attack is occurred, it can be determined by the situation that the client cannot alter the value of status attribute 1 to 0 since CONNACK message cannot be received by the client. The control mechanism is applied on before progressing the access control by the broker and impersonation attacker authority is denied. If the client connects to the broker before the attacker, impersonation attack cannot be possible due to the invalid password with HOTP.

6.6 Provide Mutual Authentication

There is no knowledge of client information such as IP address and port number between any client. All communication is occurred between the client and the broker.

Client should make sure of sending message to a trusted broker and receiving message from the right source. Therefore, hash chain is used to provide two-way authentication between the client and the broker. There is no mechanism to identify the broker aside from the certificates on MQTT protocol.

6.7 Restricted Device Access

In MQTT protocol, publisher and subscriber communicate with each other based on their topic. If there is no access control mechanism, each client accesses all messages regardless of their interested topic. Therefore, huge network traffic is occurred. In order to take advantages of MQTT protocol such as low bandwidth, the OAuth 2.0 authorization protocol that is also recommended by MQTT (Banks & Gupta, 2014), is used to restrict client access depending on its interested topic and permit reading or writing. OAuth 2.0 protocol implementation is explained in the previous section in detail. Even if the authentication information is impersonated by the attacker, no message is transmitted without known scopes.

6.8 Manage De-synchronization

MongoDB is used on written mqtt-oauth2-otp plugin to update and manage HOTP value in case of possible network disconnection and attacker.

6.9 Prevent physical attack

The scope of the study has no precautions against the physical attack. However, it is suggested that mobile application can be added to plugin in order to manage with the OTP by the user to revoke device or access.

6.10 Provide Confidentiality

Password and payload are encrypted by AES to provide data confidentiality. Attacker does not achieve the knowledge of message, topic and password without having encryption key that does not transmitted between client and broker.

6.11 Prevent Denial of Service Attack

Denial of service (DoS) is an attack which makes the service disabled. Due to the regular attacks on a system, the system cannot operate efficiently and becomes unserviceable. DoS attack causes system resources to run out quickly. Connect message is changed in every connection by using HOTP and updated regularly. Therefore, possible of synchronization problem is under the controlled between the client and the broker to maintain availability of communication. If the DoS attack is possible, client' credential is revoke and access is denied.

CHAPTER SEVEN

CONCLUSION AND FUTURE WORK

This study emphasizes the importance of device authentication and authorization in IoT scenarios due to the number of devices and their sensitive data. MQTT lightweight communication protocol is preferred to provide security as low battery power, storage, bandwidth of the IoT is taken into consideration. MQTT protocol has no specific security mechanism apart from authentication with username password, so a plugin is written over the broker. The main concern with uniqueness, key generation and delivery challenges of huge number of devices are coped with the OAuth 2.0 framework on WSO2is authorization server. Unique client id and secret is generated incase registration authorization server. Token and scope is created with the web application on the authorization server. Client id is used as key with AES encryption and kept secret to prevent spoofing and alteration of the message in case of the insecure channel whereas the client secret indicates client uniqueness. Broker obtains the key after the validation of the client with the introspection API on the authorization server. Client is authenticated in two steps with token and HOTP in order to prevent the replay attack and protected against the attacks due to using a static token. Therefore, CONNECT message is continuously changed by using short-lived HOTP in every connection to obtain much stronger security.

As different from the authentication of client, HOTP with hash chain is suggested to can be sure that communication takes place over the trusted broker. HOTP changes during every connection. Therefore, the connection will be attempts unsuccessful even if HOTP or token is obtained by the attacker. Time synchronization or nonce is suggested against the replay attack. However, time synchronization can be a problem between the client and the broker so protection from replay attack is provided with HOTP with hash chain as HOTP value is updated with the previous hash value using the invertible hash function. Moreover, if the attacker impersonates the client, HOTP will be invalid after its first connection. Hash chain synchronization is managed by updating status codes and the HOTP value on the MongoDB. Re-initialize HOTP value

is controlled with the status code and the count attribute on the MongoDB by a proposed algorithm.

The scope of paper focused on three main security issues for the MQTT protocol. Initially, authentication is achieved with token and HOTP in order to prevent connection with an unauthorized client by stronger security in two steps. Secondly, in order to transmit all communication over the trust broker, HOTP with hash chain is used to prevent forwarding message to an unknown third party. The last one, to decrease network traffic and message delay, OAuth 2.0 framework is used, which restricts unauthorized user access to all messages. Especially, HOTP is preferred to prevent replay attack and man-in-the middle attack as well as providing two-way authentication with hash chain. AES encryption without shared pre-key is used to prevent spoofing and eavesdropping in terms of using static token. The other security problems are discussed with the given solutions of aforementioned methods.

To sum up, In IoT applications, there is a difficulty in setting up a unique identifier for many devices with resource constraints. Furthermore, due to the availability of a large number of resource constraint devices, there is a key management challenge in implementing encryption algorithms to ensure the confidentiality, integrity and availability of the data in unsecured network environments. Our proposed authentication and authorization mechanism gives successful solution against the security vulnerabilities of IoT applications.

In the future work, the result of experimental studies using WEMOS will be evaluated according to security vulnerability and cost. Exchange scope has challenges due to using static token. In case the scope is changed, access token is modified. However, each client is defined with the manually embedded token. Mobile applications with cloud usage can be integrated to re-change the scope and prevent from physical attacks.

REFERENCES

- Ali, I., Sabir, S., & Ullah, Z. (2016). Internet of things security , device authentication and access Control : A Review. *International Journal of Computer Science and Information Security*, 14(8), 456.
- Asim, M. (2017). A Survey on application layer protocols for Internet of things (IoT). *International Journal*, 8(3).
- Bachry, M. (2013) *Mosquitto auth plugin* Retrieved May 05, 2018, from https://github.com/mbachry/mosquitto_pyauth.
- Bandyopadhyay, S., & Bhattacharyya, A. (2013). Lightweight Internet protocols for web enablement of sensors using constrained gateway devices. *International Conference on Computing, Networking and Communications, ICNC* , 334–340. <https://doi.org/10.1109/ICCNC.2013.6504105>.
- Banks, A., & Gupta, R. (2014). *MQTT Version 3.1.1. OASIS Standard*. Retrieved May 05, 2018, from <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>.
- Blazhevski, D., Bozhinovski, A., Stojchevska, B., & Pachovski, V. (2013). Modes of operation of the AES algorithm., 212–216.
- Chae, C., Kim, K., & Cho, H. (2017). A study on secure user authentication and authorization in OAuth protocol. *Cluster Computing*, 1-9, <https://doi.org/10.1007/s10586-017-1119-6>.
- Chodorow, K. (2013). *MongoDB: The Definitive guide: Powerful and scalable data storage* , (2nd ed.), O'Reilly Media, Inc.
- Elgazzar, M. H. (2015, December). Perspectives on M2M protocols. In *Intelligent*

Computing and Information Systems (ICICIS), 2015 IEEE Seventh International Conference on. 501-505. <https://doi.org/10.1109/IntelCIS.2015.7397267>.

Esfahani, A., Mantas, G., Maticsek, R., Saghezchi, F. B., Rodriguez, J., Bicaku, A., & Bastos, J. (2017). A lightweight authentication mechanism for M2M Communications in Industrial IoT Environment. *IEEE Internet of Things Journal*, 4662(c), 1–8. <https://doi.org/10.1109/JIOT.2017.2737630>.

Fremantle, P., & Aziz, B. (2016, November). OAuthing: Privacy-enhancing federation for the Internet of Things. In *Cloudification of the Internet of Things, CIoT*, 1–6. <https://doi.org/10.1109/CIOT.2016.7872911>.

Fremantle, P., Aziz, B., Kopecky, J., & Scott, P. (2014, September). Federated identity and access management for the Internet of things. In *International Workshop on Secure Internet of Things*, 10–17. <https://doi.org/10.1109/SIoT.2014.8>.

Frigieri, E. P., Mazzer, D., & Parreira, L. F. C. G. (2015). M2M protocols for constrained environments in the context of IoT : A comparison of approaches. In *Telecommunications Symposium (SBrT)*, 1–4. <https://doi.org/10.14209/sbrt.2015.2>.

Gantait, A., Patra, J., & Mukherjee, A. (2016, May). Design and build secure IoT solutions , Part 1 : Securing IoT devices and gateways. *IBM DeveloperWorks*, 1–20. Retrieved from <https://www.ibm.com/developerworks/library/iot-trs-secure-iot-solutions1/index.html>.

Haller, N. (1994). The S/KEY one-time password system, *Proceedings of the ISOC Symposium on Network and Distributed System Security*.

Hardt, D. (2012). The OAuth 2.0 Authorization Framework, *IETF, RFC 6749*. Retrieved 05 May, 2018, from <https://tools.ietf.org/html/rfc6749>.

- Jing, Q., Vasilakos, A. V., Wan, J., Lu, J., & Qiu, D. (2014). Security of the Internet of Things: perspectives and challenges. *Wireless Networks*, 20(8), 2481–2501. <https://doi.org/10.1007/s11276-014-0761-7>.
- Jung, S. W., & Jung, S. (2017). Personal OAuth authorization server and push OAuth for Internet of things. *International Journal of Distributed Sensor Networks*, 13(6), 1–11. <https://doi.org/10.1177/1550147717712627>.
- Karagiannis, V., Chatzimisios, P., Vazquez-Gallego, F., & Alonso-Zarate, J. (2015). A survey on application layer protocols for the Internet of Things. *Transaction on IoT and Cloud Computing*, 3(1), 11–17. <https://doi.org/10.5281/ZENODO.51613>.
- Katsikeas, S., Fysarakis, K., Miaoudakis, A., Van Bemten, A., Askoxylakis, I., Papaefstathiou, I., & Plemenos, A. (2017). Lightweight & secure industrial IoT communications via the MQ telemetry transport protocol. *Proceedings - IEEE Symposium on Computers and Communications*, 1193–1200. <https://doi.org/10.1109/ISCC.2017.8024687>.
- Kim, H. J., & Kim, H. S. (2011, June). AUTHHOTP- HOTP based authentication scheme over home network environment. *Proceedings of the International Conference on Computational Science and Its Applications - Volume Part III*, 622–637. Retrieved from <http://dl.acm.org/citation.cfm?id=2029312.2029360>.
- Kinikar, S. & Terdal, S. (2016, August). Implementation of open authentication protocol for IoT based application. *In Inventive Computation Technologies (ICICT), International Conference on*. (1), 1-4. <https://doi.org/10.1109/INVENTIVE.2016.7823267>.
- Kirsche, M., & Klauck, R. (2012, March). Unify to bridge gaps: Bringing XMPP into the Internet of Things. *IEEE International Conference on Pervasive Computing and Communications Workshops, PERCOM Workshops*, 455–458. <https://doi.org/10.1109/PerComW.2012.6197534>.

- Kraijak, S. & Tuwanut, P. (2015, October). A survey on Internet of things architecture, protocols, possible applications, security, privacy, real-world implementation and future trends, *In Communication Technology (ICCT)*, 26–31.
- Lee, S., Kim, H., Hong, D. K., & Ju, H. (2013). Correlation analysis of MQTT loss and delay according to QoS level. In *International Conference on Information Networking*, 714–717. <https://doi.org/10.1109/ICOIN.2013.6496715>.
- Li, W., Mitchell, C. J., & Chen, T. (2018). Mitigating CSRF attacks on OAuth 2.0 and OpenID connect, 1–18. Retrieved from <http://arxiv.org/abs/1801.07983>.
- Light, R. (2013). *Mosquitto-an open source mqtt v3. 1 broker*. Retrieved May 05, 2018, from <http://mosquitto.org>.
- Luzuriaga, J. E., Perez, M., Boronat, P., Cano, J. C., Calafate, C., & Manzoni, P. (2015, January). A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks. *12th Annual IEEE Consumer Communications and Networking Conference, CCNC*, 931–936. <https://doi.org/10.1109/CCNC.2015.7158101>.
- Mahmood, Z. (Ed.). (2016). *Connectivity Frameworks for Smart Devices: The Internet of Things from a Distributed Computing Perspective*. Springer.
- Manohar, H. L., & Reuban Gnana Asir, T. (2018). Data consumption pattern of MQTT protocol for IoT applications. *Communications in Computer and Information Science*, 808, 12–22. https://doi.org/10.1007/978-981-10-7635-0_2.
- Mendez, D. M., Papapanagiotou, I., & Yang, B. (2017). Internet of things: Survey on security and privacy, 1–16. Retrieved from <http://arxiv.org/abs/1707.01879>.
- M'Raihi, D., Bellare, M., Hoornaert, F., Naccache, D & Ranen, O. (2005). HOTP: An HMAC-based one-time password algorithm, *Internet Engineering Task Force*

- (*IEFT*), December 2005. Retrieved May 05, 2018, from <https://tools.ietf.org/pdf/rfc4226.pdf>.
- MQTT security fundamentals. (n. d) *HIVEMQ enterprise MQTT broker*. Retrieved May 05, 2018, from <https://www.hivemq.com/mqtt-security-fundamentals/>.
- Niruntasukrat, A., Issariyapat, C., Pongpaibool, P., Meesublak, K., Aiumsupucgul, P., & Panya, A. (2016, May). Authorization mechanism for MQTT-based Internet of Things. *IEEE International Conference on Communications Workshops, ICC*, 6, 290–295. <https://doi.org/10.1109/ICCW.2016.7503802>.
- Obermaier, B. D. (2015). *Getting started with MQTT - A Protocol for the Internet of Things*. Retrieved May 05, 2018, from <https://dzone.com/refcardz/getting-started-with-mqtt?chapter=1>.
- Oh, S.-R., & Kim, Y.-G. (2017, February). Security requirements analysis for the IoT. *International Conference on Platform Technology and Service (PlatCon)*, 1–6. <https://doi.org/10.1109/PlatCon.2017.7883727>.
- Paho, Eclipse. (2015) "*Paho project*." Retrieved May 05, 2018, from <http://www.eclipse.org/paho>.
- Park, C. S. (2018). One-time password based on hash chain without shared secret and re-registration, *Computers & Security* 75, 138-146. <https://doi.org/10.1016/j.cose.2018.02.010>.
- Patierno, P. (2016). *AMQP essentials*. Retrieved May 05, 2018, from <https://dzone.com/refcardz/amqp-essentials>.
- Rajan, M. A., Varghese, A., Narendra, N., Singh, M., Shivraj, V. L., Chandra, G., & Balamuralidhar, P. (2016). Security and privacy for real time video streaming using hierarchical inner product encryption based publish-subscribe architecture.

- Proceedings - IEEE 30th International Conference on Advanced Information Networking and Applications Workshops, WAINA*, 373–380. <https://doi.org/10.1109/WAINA.2016.101>.
- Salman, T., & Jain, R. (2013). Networking protocols for Internet of things, 1–28. Retrieved from http://www.cse.wustl.edu/~jain/cse570-15/ftp/iot_prot.pdf.
- Saxena, A. (2007). Dynamic authentication: Need than a choice. *3rd IEEE/Create-Net International Conference on Communication System Software and Middleware, COMSWARE*, 214–218. <https://doi.org/10.1109/COMSWA.2008.4554411>.
- Shelat, R., Patel, N., & Bhatt, C. (2016, March). A Survey of open source protocols XMPP and SIP for instant messaging system. *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies - ICTCS '16*, 1–4. <https://doi.org/10.1145/2905055.2905319>.
- Sheng, Z., Yang, S., Yu, Y., Vasilakos, A., McCann, J., & Leung, K. (2013). A survey on the ietf protocol suite for the internet of things: Standards, challenges, and opportunities. *IEEE Wireless Communications*, 20(6), 91–98. <https://doi.org/10.1109/MWC.2013.6704479>.
- Shinde, S. A., Nimkar, P. A., Singh, S. P., Salpe, V. D., & Jadhav, Y. R. (2016). MQTT-Message queuing telemetry transport protocol. *International Conference on Research and Recent Trends in Engineering and Technology. (ICRRTET)* 3(3), 240-244.
- Shivraj, V. L., Rajan, M. A., Singh, M., & Balamuralidhar, P. (2015). One time password authentication scheme based on elliptic curves for Internet of Things (IoT). *National Symposium on Information Technology: Towards New Smart World*, (c), 1–6. <https://doi.org/10.1109/NSITNSW.2015.7176384>.
- Singh, M., Rajan, M. A., Shivraj, V. L., & Balamuralidhar, P. (2015, April). Secure

- mqtt for internet of things (iot). In *Communication Systems and Network Technologies (CSNT), 2015 Fifth International Conference on*. 746-751.
- Soni, D., & Makwana, A. (2017, April). A survey on mqtt: a protocol of Internet of Things (IoT). *International Conference on Telecommunication, Power Analysis and Computing Techniques (Ictpac)*, 0–5. Retrieved from https://www.researchgate.net/publication/316018571_A_SURVEY_ON_MQTT_A_PROTOCOL_OF_INTERNET_OF_THINGS_IOT.
- Syed Farid, S. A., Mohd Anuar, M. I., & Habibah, H. (2016, May). Timing analysis of lightweight AA-Beta encryption scheme on embedded Linux for Internet of Things. *IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, 113–116.
- Stallings, W. (2006). *Cryptography and network security principles and practice* (6th ed.). India: Pearson Education.
- Ungurean, I., & Gaitan, N.-C. (2015). Data distribution service for real-time systems - a solution for the Internet of Things environments. *Annals of the University Dunarea de Jos of Galati: Fascicle II, Mathematics, Physics, Theoretical Mechanics*, 38(1), 72–77. Retrieved from <http://search.ebscohost.com/login.aspx?direct=true&site=eds-live&db=a9h&AN=113483718>.
- Upadhyay, Y., Borole, A., & Dileepan, D. (2016, March). MQTT based secured home automation system. In *Symposium on Colossal Data Analysis and Networking, CDAN*. 1-4. <https://doi.org/10.1109/CDAN.2016.7570945>.
- Windley, P. J. (2016). API access control with OAuth: Coordinating interactions with the Internet of Things, *IEEE Consumer Electronics Magazine*, vol. 4(3), 52–58, <https://doi.org/10.17485/ijst/2016/v9i9/86791>.
- WSO2. (2017). *WSO2 Identity Server Documentation*. Retrieved May 05,2018, from

<https://docs.wso2.com/display/IS530/WSO2+Identity+Server+Documentation>.

Yassein, M. B., Shatnawi, M. Q., & Al-zoubi, D. (2016, September). Application layer protocols for the Internet of things: A survey. In *Engineering & MIS (ICEMIS), International Conference on*, 1-4.

Yerlikaya, Ö., & Dalkılıç, G. (2017, May). Security of message queue telemetry transport protocol. In *Signal Processing and Communications Applications Conference (SIU), 25th*, 1-4.

Zamfir, S., Balan, T., Iliescu, I., & Sandu, F. (2016, October). A security analysis on standard IoT protocols. In *International Conference on Applied and Theoretical Electricity*, 1-6. *Proceedings*. <https://doi.org/10.1109/ICATE.2016.7754665>.