Efficient MQTT Platform Facilitating Secure Group Communication

Hung-Yu Chien[1], Pei-Chih Lin[2], Mao-Lun Chiang[2]

[1] Department of Information Management, National Chi Nan University, Taiwan
[2] Department of Information and Communication Engineering, Chaoyang University of Technology, Taiwan
hychien@ncnu.edu.tw, s10730621@gm.cyut.edu.tw, mlchiang@cyut.edu.tw

Abstract

Message Queue Telemetry Transport (MQTT) has become one of the most popular Internet-of-Things (IoT) communication protocols, owing to its high efficiency and simplicity. However, it does not support the desirable security functions; instead, it assumes the use of Secure Sockets Layer (SSL)/Transport Layer Security (TLS) in the lower layer. Unfortunately, it is too costly to support SSL/TLS for many low-end IoT devices, and SSL/TLS does not support secure group communications. Group communications are popular in many IoT application scenarios [1-2, 28] and in many MQTT scenarios; for example, a message generator needs to share its data to many interested receivers. In this paper, we propose and implement a secure MQTT group communications, based on our previous secure MQTT key agreement scheme [26]. Comprehensive experiments have been performed and evaluated. The results confirm that the proposed scheme and implementation greatly improves the communication latencies of MQTT applications.

Keywords: Internet of Things, Security, Authentication, MQTT, SSL

1 Introduction

The technologies of Internet-of-Things (IoT) boom many new applications by facilitating easy data and control message transmissions among various devices. These applications include smart cities, smart agricultures, smart transportations, industrial IoTs, and so on. Among these applications, group communication (or multicasting) is quite common and popular: the data need to being sent to many potential receivers. It is desirable that the adopted IoT communication protocols support group communications.

IoT communication protocols like Message Queue Telemetry Transport (MQTT) [8], Advance Message Queuing Protocol (AMQP) [10], Constrained Application Protocol (CoAP) [11], Extensible Messaging and Presence Protocol (XMPP) [13], and Data Distribution Service (DDS) [13] facilitate the data transmissions. Among them, MQTT is one of the most popular ones, owing to its high efficiency and easiness to use. MQTT gains its efficiency at the cost of not supporting security capacities by itself. Instead, it assumes the systems should enable SSL/TLS in the lower layer to secure the transmissions. Unfortunately, supporting SSL/TLS is costly in terms of computation, communication, and energy for many low-end IoT devices. Furthermore, SSL/TLS does not support group communications by itself, and it is clumsy to extend SSL/TLS to support group communications. One possible extension of SSL to support group communication is described here. In Mektoubi et al.'s design [18], each topic is assigned a certificate and all the members of the group have to be distributed the private key of the certificate; when a sender multicasting a public-key-based encryption, all the members can decrypt the encryption using the private key; this approach is very costly in terms of the distribution of the private key and the management of the topic-level certificates.

The mechanisms of MQTT message encryptions can be classified as two approaches, according to how and where the messages are encrypted and decrypted. The first approach is the session-based encryption: a publisher encrypts its messages using the session keys shared with the broker, the broker decrypts the messages, and then the broker re-encrypts the messages several times, using the individual session keys with each subscriber. This approach would incur lots of computational efforts and communication delay, when there are large number of subscribers for a message. The second approach is the end-to-end approach: a publisher encrypts the messages, the broker forwards the messages without decrypting them, and then the subscribers decrypt the encryptions. In this approach, the broker does not decrypt then re-encrypt; it, therefore, saves lots of computational efforts and shortens the communication latency accordingly. The merits of the performance improvement from the second approach would be greatly amplified, when the number of subscribers

*Corresponding Author: Hung-Yu Chien; E-mail: hychien@ncnu.edu.tw
DOI: 10.3966/160792642020122107007

become very large. MQTT group communication with group key distribution is one potential mechanism to facilitate the second approach. Figure 1 shows the scenarios of the two approaches.

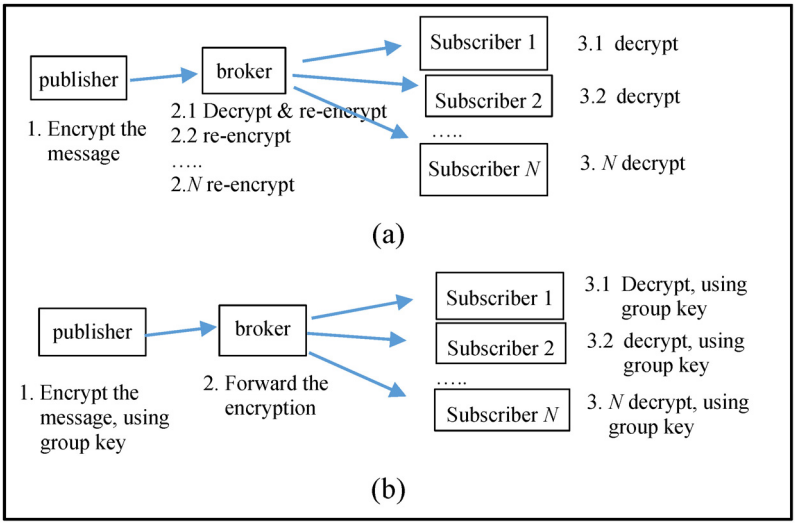

Figure 1. The end-to-end encryption

There are many researches and implementations aiming at enhancing the security support of MQTT systems. Unfortunately, none of them provide secure and efficient group communications. Here, we extend a secure MQTT platform [26] to support secure group communication framework; [26] proposed a two-phase key authentication approach for MQTT and implemented it using the classic Challenge-Response key agreement mechanism; but it did not support secure group communications. The implementation and experiments confirm that the proposed MQTT group communication framework greatly improves the communication performance.

This paper is organized as follows. In the rest of this section, we discuss the related works. Section 2 introduces our secure MQTT group communication system. Section 3 analyzes the security properties. Section 4 introduces our implementation and evaluates the performance. Finally, Section 5 states our conclusions.

1.1 Related Work

A MQTT system consists of a set of clients and a broker who acts as an intermediary among the clients, where clients play the role of publishers, the role of subscribers, or both. A publisher generates data to be subscribed by subscribers. The data sharing from publishers to subscribers is via a broker. The message exchange among clients is based on the concept of "topic". A client publishes messages for a specified topic, and a client can receive the messages of that topic by subscribing the topic. MQTT itself does not provide security protections like encryption, authentication, and integrity on the transmissions; instead, it assumes SSL/TLS being supported in the underlying layer. However, several works like [4, 6-7] have shown that SSL/TLS demands lots of computations and communications, which is too costly to many low-end IoT devices. Andy et al. [14] and Firdous et al. [3] respectively demonstrated several attacks scenarios on the MQTT platforms and evaluated the vulnerabilities of MQTT systems. The security vulnerabilities of several Arduino products acting as MQTT clients were evaluated in Chien-Chen's study [13].

There are several research works aiming at enhancing the security of MQTT systems [3, 15-27]. These works can be classified into three categories, according to which level of authentication and secure transmission they focus on. The first category focuses on the topic-level authentication [9-12], where those clients accessing the same topic all share the same keys. The second category like [4-5, 8, 14-15, 17-24, 26-27] focuses on both topic-level authentication and device-level authentication: the broker verifies whether the client is authorized to access a topic and authenticate the device individually. The third category, in addition to individual device authentication and topic-level authentication, provides additional secure group key distribution to enhance the security and performance of group communications. Chien et al. [25] proposed a group communication design for MQTT systems, but only few functions have been implemented and only very few experiments have been considered.

For the first category, there are several open source/commercial MQTT platforms [9-12] providing the basic security support, where only a topic-level secret is created for each specific topic so that all the devices authorized to access that topic would share the same topic-level secret. It is obvious that a single device compromise would endanger the security of all other devices that share the same topic key. Amazon's IoT security solution [8] supports TLS/SSL client authentication, but it does not support MQTT-level

encryption, multicast, and dynamic multicast. It supports customized authentication, but there is no enough information to tell whether it is compatible with MQTT Application Interfaces (APIs). Chien et al. [26] proposed an efficient two-phase authentication mechanism to be compatible with existent MQTT APIs, and designed a function-and-security-enhanced MQTT platform. In their scheme, both topic-level and device-level authentication are considered; the experiments confirmed that it greatly improves the computational performance and communication, compared to the SSL/TLS approach. But, for a simple multicast from a publisher to many subscribers, it requires a broker perform one decryption and many re-encryptions for those subscribers. It would seriously downgrade the overall performance, when there are millions or even billions of devices accessing the system.

For the second category, some works aim at designing special hardware to facilitate individual secure authentication and transmission. Espinosa-Aranda et al. [19] and Lesjak et al. [21] respectively designed a specialized hardware to help IoT devices facilitate their SSL/TLS connections. This extra hardware solution imposes high cost for many IoT deployments. Bhawiyuga et al. [17] proposed a token-based authentication solution for MQTT device authentication; unfortunately, the token is not properly protected (an attacker can easily derive the passwords), and no session key generation is specified for the connections. Shin et al. [15], based on their previous AugPAKE protocol [16], designed the AugMQTT platform to provides device authentication and to establish session keys. Neisse et al. [23] and Rizzardi et al. [20] respectively designed function-enhanced MQTT platforms that facilitate flexible policy management and enforcement mechanisms. Niruntasukrat et al. [24] applied OAuth1.0a [31] as the authorization mechanism for MQTT systems to facilitate device authentication; but their scheme requires the devices to get an authorization permission from the users during the authentication process. This requirement of interactive user involvement not only increases the communication delay but also significantly increases the inconvenience of IoT applications. All the above works only focus on secure unicast communications in IoT applications and do not consider secure group key distribution.

Some existent works related to the third category seem all applied some kinds of public-key cryptosystems. Mektoubi et al. [18], based on the Public-Key-Infrastructure (PKI) system and the symmetric key encryption, implemented the client authentication and the topic-related message encryptions; each topic has one corresponding certificate, and the messages for that topic are encrypted using the public key of the certificate; only those authorized clients having the corresponding private key can decrypt the encryptions. This design enables a broker simply forwards the public-key-based encryptions to all subscribers without decrypting the encryptions; however, it incurs several weaknesses and drawbacks: (1) it demands costly public-key computations; (2) the management of topic-level certificates and the distribution of the private keys incur lots of overhead. Singh et al. [22] augmented existent MQTT protocols with Key/Ciphertext Policy-Attribute Based Encryption (KP/CP-ABE) [29-30] for securing MQTT applications in sensor network environments. This attribute-based-encryption-based approach has the potential advantage of providing group communication in IoT applications. However, one of the key weaknesses is that the computation cost is too high to be applied in the current IoT practical scenarios; one attribute-based decryption could demand 3~6 ms, even if the number of attributes is only three in their experiments. In a normal IoT application scenario, the number of attributes would be much larger than three, and that would significantly increase the encryption/decryption time and the communication time. That is, these schemes demand high cost for implementing their group communications in MQTT systems.

Regarding various improvements on the authentication and the secure communication support of MQTT systems, we sort out a simplified conclusion here. The first category supports only the topic-level authentication and secure pairwise communication [9-12]; this category does not authenticate each device individually, and a single device compromise would endanger the security of the whole system of the same topic. The second category like [4-5, 8, 14-15, 17-24, 26-27] authenticates each device individually, and can resist the single-device–compromise attack; however, it cannot efficiently support secure group communication or multicasting. The third category additionally supports secure group communication and multicasting; however, both Mektoubi et al.'s extended-certificate-based scheme [18] and Singh et al.'s KP/CP-ABE-based scheme [22] are very computationally expensive; Chien et al.'s preliminary MQTT group communication [25] only supports few functions and only very few experiments have been considered. In a short summary, to the best of our knowledge, there is no efficient secure MQTT group communication has been proposed, been implemented, and been evaluated.

2 The Proposed MQTT Group Communication Framework

We extend Chien et al.'s secure Challenge-Response (CR) MQTT platform [26] into our MQTT group communication framework as Figure 2. In [26], they designed and implemented a secure MQTT platform that provides user/device/policy management web

portal, a broker, and a CR-based key agreement scheme. In the system, each device should be individually authenticated and it shares one session key with the broker; the MQTT messages between the device and the broker are encrypted, using the session keys. In the following, we will introduce our MQTT group communication framework, and Table 1 lists the notations used in this paper.

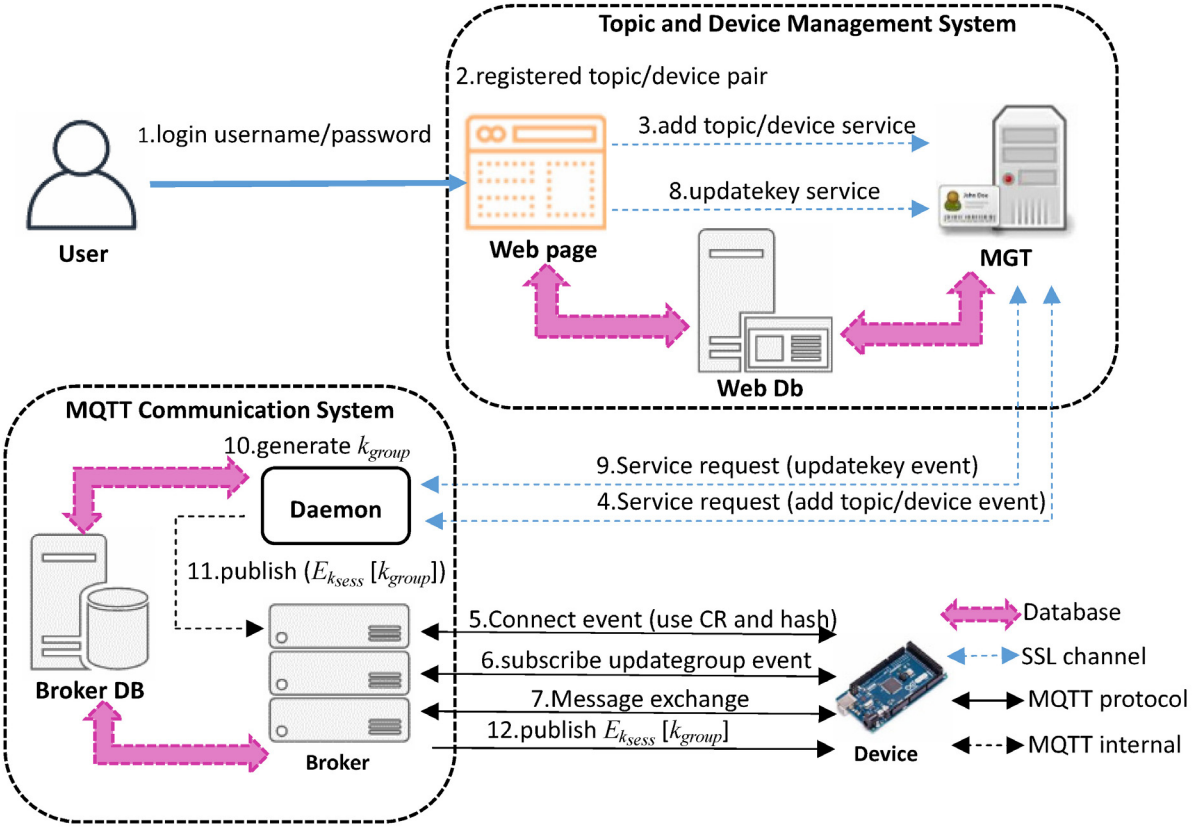

Figure 2. The MQTT group communication framework and flows

Table 1. The notation

TDMS; MGT	Topic and Device Management System (TDMS); ManaGemenT (MGT).
Daemon; DB	Daemon: a software thread takes care of interactions between broker and MGT; DB (DataBase).
K_{group}; K_{sess}	K_{group}: the group key; K_{sess}: the session key between a broker and a device.
$E_{Kgroup}()$, $E_{Ksess}()$	The encryption using the group key and the session key respectively.
ID_{topic}/ID_{device}	The identities of a topic/a device.
K_{topic}/K_{device}	The master key of a topic/a device.
$C1$, $C2$, $R1$, $R2$	Challenges and Responses.

The system consists two main subsystems: one is the security-enhanced MQTT broker and the other is the Topic and Device Management System (TDMS) which is responsible for the user/device/policy management functions. In TDMS, the ManaGement (MGT) takes care of the interactions with the daemon thread, and a web portal provides the interfaces for accessing the system. A user can login the web portal to register his devices, to create MQTT topics, and to specify the policies for a specific topic in Step 1 and 2. Then the portal will notify, via the MGT, the enhanced MQTT broker the new created records or any updated records in Step 3 and 4. The user-system interactions are conducted on SSL connections. Our new system extends this subsystem an extra capacity to notify the broker a new request "Create group key update"; this event will trigger the broker create a "group key update" for each topic in Step 8 and 9. The broker will monitor the access status for that topic. As long as there is any device active for that topic, the broker will regularly update and distribute group keys for those active devices.

The enhanced MQTT broker authenticates each device and generates a session key, using the CR-based key agreement scheme [26]. Additionally, the broker periodically generates group keys for those active topic. The group keys are encrypted, using each active device's session key. The frequency of updating the group keys depends on two main factors: the security level of the topic and the overhead of updating the group keys. The more frequently the system updates the group keys, the more robust of the system; however, it incurs more overhead. These steps are specified as Step 5, 6, 7, 8, 9, 10, and 11 in Figure 2.

In our system design, once a client (either a publisher or a subscriber) registers in a topic, it is also automatically registered in a special topic of the form

"*updategroup/ID_{topic}/ID_{device}*"; this special topic is for automatically sending group key updates to this device. When a publisher or a subscriber being authenticated by the broker, it will receive one session key and periodically receives a series of group keys. The publisher encrypts its MQTT messages, using the active group key; the broker just forwards the encryptions without decrypting them; the subscribers then decrypt the encryptions, using the group keys.

Figure 3 shows the system sequence diagram of our security-enhanced MQTT platform performing the user/device/policy management functions, and the MQTT CONNECT/ publish/subscribe functions. The user-system interactions and the interactions between the system modules and the database are conducted via secure SSL channels ("⇧" in Figure 3). The interactions with the symbol "↑" in Figure 3 are for the MQTT connections. *Please note that the MQTT connections do not require the SSL support.* The system authenticates users before they can access the policy management module and register their devices. Each device is required to be registered. Once a device is registered, the administrator validates the data, and confirms the registration request by issuing a device identity and a device key to that device. The policy management module is responsible for the management of topics. Each topic is associated with an identity and a corresponding key. A user who creates a topic is responsible for authorizing other users' requests for that topic. Once a user has been authorized to access a topic and his advice registration has been authorized, he gets two sets of (identity, key) pairs: one is that for the device and the other is for the topic. Now a legitimate device, based on the two key pairs, can connect the MQTT broker and requests for accessing the topic. Before granting a device the access connection, the MQTT broker will initiate the device authenticator to authenticate the device, using the CR-based key agreement scheme. When this process is done, the authenticated device and the broker will share a session key.

Figure 4 shows the interactions when a client (a publisher or a subscriber) and the broker perform the CR-based key agreement scheme [26]. In Step 1, the client chooses a random nonce, and sends the identity of device (ID_{device}), the identity of the topic (ID_{topic}), and the nonce $C1$ to the broker. Then the broker chooses a random nonce $C2$, and responds with ($C2$, $R1$) in Step 2, where $R1 = h(K_{topic} \| K_{device}, C1)$. The client verifies the broker's response $R1$; if the verification succeeds, then the client computes $R2 = h(K_{topic}, \| K_{device}, C2)$ and responds with $R2$ in Step 3. Upon receiving $R2$, the broker verifies $R2$. If all the verifications succeed, then the client and the broker share the session key $K_{sess} = h(ID_{topic}, K_{topic}, ID_{device}, K_{device}, C1, C2)$. Please notice that, in Figure 4, the client, after the CR-based key agreement, further invokes the MQTT CONNECT API as $CONNECT$["ID_{device}, ID_{topic}", $h(K_{sess})$], where $h(K_{sess})$ is the hash of the session key. This arrangement of $h(K_{sess})$ is because (1) MQTT

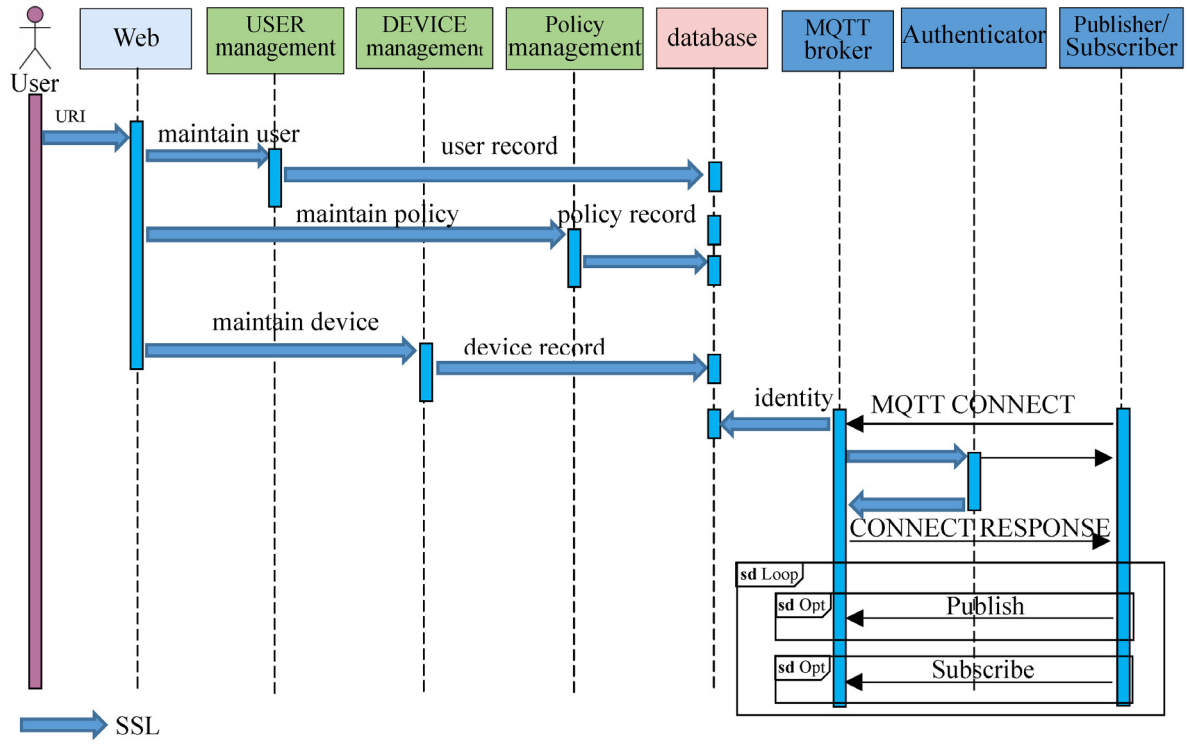

Figure 3. The system sequence diagram of the security-enhanced MQTT platform performing user/device/policy management and the MQTT connect/publish/subscribe functions

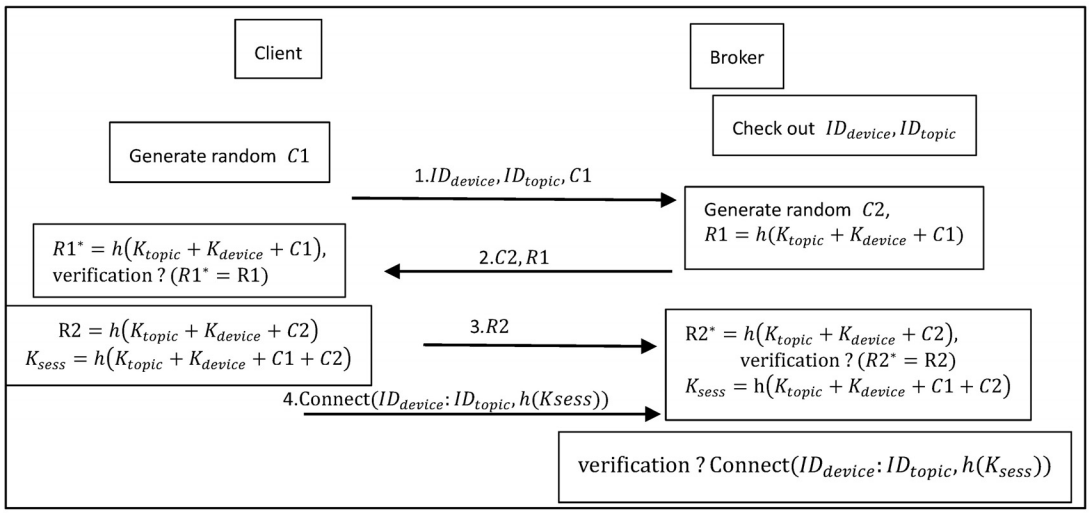

Figure 4. The CR-based key agreement of the enhanced MQTT platform

standard CONNECT API only offers only two parameters for specifying client's authentication parameters, and (2) using $h(K_{sess})$ can ensure the authentication while preserving the secrecy of the session key K_{sess}, even if SSL/TLS is not supported. The interested readers are referred to [26] for the details.

Once a device being authenticated, it will get the session key and the group keys. Then publishers and subscribers use the group keys to encrypt and to decrypt the MQTT messages respectively, as being depicted in Figure 1(b). To enhance the security robustness, our system regularly updates and distributes the group keys to those active clients. The sequence diagram of the group key updating and distribution process is shown in Figure 5. First, the MGT creates a group key updating timer for each topic, according to the policy. When a timer of a topic expires, the MGT notifies the daemon the event. Then the daemon inquires whether there is any active device for that topic; if so, then the MGT randomly generates a new group key and stores the key in the broker's database. The daemon looks up all the session keys of those active clients of that topic, and then generates the encrypted group key $E_{Ksess}[timestamp, group\ key]$ for each active client. It then sends, via the broker, the group key updating message $E_{Ksess}[timestamp, group\ key]$ to the devices.

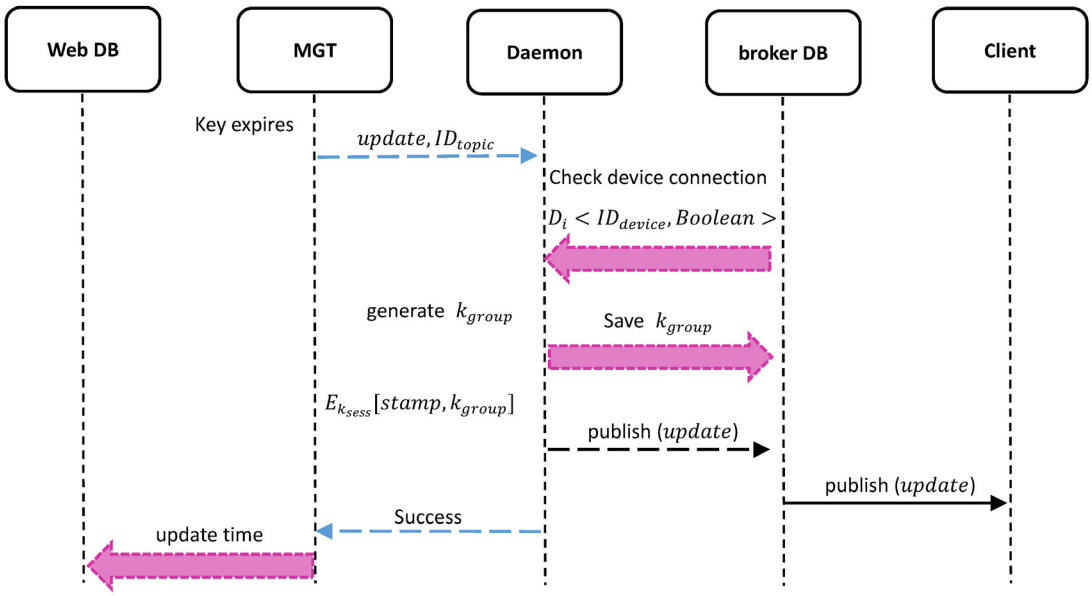

Figure 5. Group key updating process

3 Security Analysis

Now we examine the security of the proposed framework. The security functions of the framework mainly consist of four components: (1) the SSL protection for user accessing the web portal (Step1 in Figure 2), (2) the SSL protection of the interactions

among the system components, (3) the two-phase CR-based key agreement for establishing secure session keys, and (4) the group key distribution scheme.

3.1 The SSL Protection for User Accessing the Web Portal

Users use computers, notebooks, or smart phones to access the web portal. Because these devices have abundant resources to run the SSL protocol, we adopt SSL to protect the access and the transmissions when users access the web portal.

3.2 The SSL Protection of System Component Interactions

For the interactions among system components like the daemon, the web pages, and the MGT in Figure 2, the SSL protection is activated to ensure the authenticity and the privacy of the interactions. These components are implemented on our server which has enough resources to support SSL.

3.3 The Two-phase CR-based Key Agreement for Establishing Secure Session Keys (Figure 4)

The current MQTT CONNECT API standards [32-33] support only two parameters, identity and password, for clients to initiate connection with a broker. If a device does not afford SSL/TLS to protect the transmission of the two parameters, then the parameters would be disclosed to attackers. To be compatible with the current API standards and to eliminate the burden of supporting SSL/TLS for these devices, Chien et al. [26] have designed and implemented a two-phase CR-based key agreement scheme for authentication and session key generation; The scheme runs a challenge-response protocol on network sockets to authenticate each other and establish a session key $K_{sess} = h(ID_{topic}, K_{topic}, ID_{device}, K_{device}, C1, C2)$ in Phase 1, and provides the hash of the session key $h(K_{sess})$ as the second parameter in the CONNECT API as $CONNECT[\text{ “ } ID_{device}, \ ID_{topic} \text{ ”}, \ h(K_{sess})]$. In this arrangement, even if we assume there in no SSL/TLS protection in the lower layer, the attacker can only eavesdrop $h(K_{sess})$ but not the session key K_{sess}. Owing to the freshness and randomness of the session key, this ensures the authentication and the privacy of the following MQTT transmissions.

3.4 The Group Key Distribution Scheme

In the proposed group communication framework, each publisher uses the active group key to encrypt its messages. The group keys are periodically updated and distributed to all the authorized and active clients for that topic. This ensures the freshness and randomness of the group keys. The group key is encrypted as $E_{Ksess}[timestamp, group\ key]$, using the receiver's session key. This ensures the secure delivery of the group keys to only authorized clients.

The above four main security components ensure the security of the whole system. We now discuss the impact of device-compromise on our system as follows.

3.5 The Impact of Device-compromise

In our design, each device would be authenticated, based on its knowledge of a topic key and its device key; so when a device is compromised, both the topic key and its device key are disclosed. However, the attacker who compromises the device cannot impersonate any other devices using the keys. So our design cooperating with a sound intrusion detection system can effectively enhance the robustness of MQTT system security.

4 Performance Evaluation

Chien at al.'s work [26] has compared the communication latency of MQTT connection requests between the CR-based scheme and the SSL-based scheme. The experiments confirmed that the CR-based authentication can greatly improve the communication latency. In this session, we will compare the performance of message transmissions in three mechanisms: our group-key-based end-to-end encryption (group-key-E2E for short), the session-based encryption using CR-authentication (session-encryption-CR for short), the session-based encryption using SSL authentication (session-encryption-SSL for short). Among them, end-to-end encryption refers to the approach of which publishers encrypt messages, the broker forwards the encryptions, and subscribers decrypt the encryptions (as specified in Figure 1(b)); session-based encryption refers to the approach of which publishers encrypt messages using session keys, the broker decrypts and then re-encrypt the messages using session keys shared with subscribers, and finally subscribers decrypt the encryptions (as specified in Figure 1(a)). The session-encryption-CR refers to Chien et al.'s scheme [26] in which clients and the broker apply the CR-based key agreement to generate the session keys. The session-encryption-SSL refers to those cases in which they apply SSL to authenticate each other and share the session keys.

Regarding session-encryption-SSL, there are two points worthy being noticed:

(1) Because we target resource-limited IoT devices, we only activate the server authentication using SSL but not the client's SSL authentication in our current experiments; even so, we still find that the average communication latency of the session-encryption-SSL is still the longest one among the above three mechanisms.

(2) SSL handshake protocol in the SSL protocol suite is the most computation-and-communication demanding protocol. Before two parties establishing a SSL channel, they should perform SSL handshake protocol. To reduce the overhead of SSL handshake protocol, SSL standards and implementations supports SSL session re-use. A SSL session re-use is a simplified SSL authentication process in which the participants reduce the cost by re-using the previous connection's parameters and keying material without sending certificates and exchanging keying parameters [34].

Due to the limited resources we have, we use one notebook to host many MQTT clients in our experiments, instead of using 30~50 standalone IoT devices. The comparisons of the performance evaluations are still valid, as we concern the communication latency improvement of our proposed scheme. For all the three mechanisms, we use one desktop computer as the broker and one notebook hosting several clients. Table 2 lists the hardware specifications. Table 3 lists the software specifications. The parameters of the experiments of the three mechanisms are listed in Table 4. For each mechanism, we evaluate several metrics: the average communication latency between a publisher and a subscriber, the CPU utilization of the broker, the memory utilization of the broker, the received message overhead at the broker, and the accumulated message overhead of a publisher.

Table 2. The hardware for the experiments

	client	Broker
Operation system	Windows10 professional 64-bit	Windows10 professional 64-bit
CPU	Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz	Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
Memory	16GB	32GB
GPU	NVDIA GeForce GTX 1060	NVDIA GeForce GT610
Network interface	Qualcomm Atheros AR8121/8175 PCI-E Gigabit	Intel(R) 82579LM Gigabit Network Connection

Table 3. The software for the experiments

	client	Broker
SSoftware	Node 12.4.1, mqtt 2.15.1, sha1 1.1.1, crypto-js 3.1.9	Node 12.4.1, mongoose 5.4.1, mosca 2.8.3, passport-local 1.0.0, OpenSSL 1.1.1c, sha1 1.1.1, crypto-js 3.1.9

Table 4. The parameters of the experiments

mechanisms	Group-key-E2E, session-encryption-CR, session-encryption-SSL
Message frequency at publisher	1Message/sec, 1Message/5sec, 1Message/10sec.
Number of Subscribers	10, 30, 50.
Duration of each experiment	30 minutes
metrics	• Communication latency between a publisher and a subscriber, • CPU utilization of broker, memory utilization of a broker, received message overhead of broker, • accumulated message overhead of a publisher.

Table 5 summarizes the average communication latency between a publisher and a subscriber, the standard derivation of the latency, and the number of runs of the three approaches. From the table, we can see that the average latency increases as the number of nodes (subscribers) increases, especially for the session-encryption-CR and for the session-encryption-SSL; the reason for this is quite obvious that the broker needs to decrypt once and re-encrypt the messages many times; the number of re-encryptions increases linearly as the number of nodes increases; it also explains why the group-key-E2E's latency is the least among the three approaches: the broker in the group-key-E2E does not decrypt and does not re-encrypt messages. In Table 5, we also notice that the relation between the average latency and the frequency of messages sent is not so obvious for the three approaches with the parameters we tried. We also notice that the standard derivation of the session-encryption-SSL approach is quite large; we check the log files and find that it is because SSL will run a complete SSL handshake after several times of SSL-re-use sessions.

Figure 6 shows the average latency trend as the numbers of subscribers increases for the case of frequency being 1 message per second. Here, we highlight two points. First, the latencies of the session-encryption-CR and the session-encryption-SSL are significantly larger than that of the group-key-E2E. Second, the trend of the increasing of latency as the number of nodes increases in the group-key-E2E approach is not so obvious as that of the other two approaches. This is because the broker in the group-key-E2E does not decrypt and re-encrypt the messages.

Figure 7 depicts the average latency and the standard derivation of the latency of the three approaches when

Table 5. The average communication latency between a publisher and a subscriber, the standard derivation, and the number of runs in the experiments

Number of nodes	Message/sec.	Group-key-E2E Avg(ms)/Run/Std(ms)	Session-encryption-CR Avg(ms)/Run/Std(ms)	Session-encryption-SSL Avg(ms)/Run/Std(ms)
10	1Message/s	15.76/2236/15.75	38.39/2179/ 38.52	82.59/1806/456.85
	1Message/5s	16.52/450/8.87	38.31/429/10.38	86.14/356 /191.1
	1Message/10s	15.89/234/8.4	38.32/200 /11.0	63.55/188/54.74
30	1Message/s	17.24/1896/12.38	61.42/1829/31.62	100.56/1697/105.73
	1Message/5s	17.62/480/3.44	62.70/443/28.2	107.49/355/170.4
	1Message/10s	16.98/200/3.36	65.38/196/13.11	113.10/177/115.64
50	1Message/s	25.35/1860/19.05	109.73/1800/66.02	137.95/1652/86.45
	1Message/5s	25.55/410/12.03	105.12/378/50.95	163.21/358/403.6
	1Message/10s	25.08/200/10.13	91.69/190/32.93	151.91/181/103.47

Note. Avg stands for average latency; Run stands for the number of runs in the experiments; Std stands for the standard derivation.

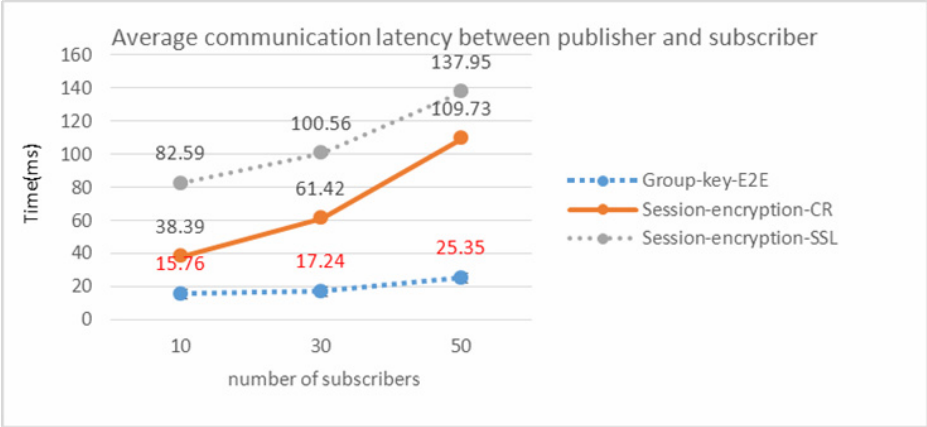

Figure 6. The average latency of three approaches when the publisher publishes 1 message/sec

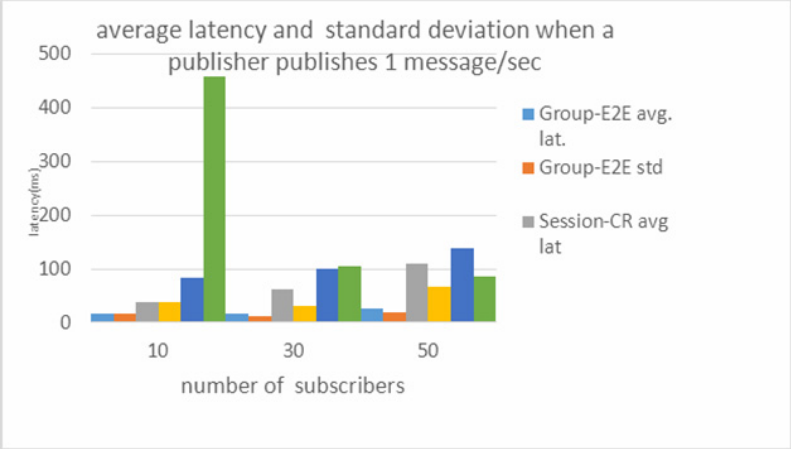

Figure 7. The average latency and the standard deviation when a publisher publishes 1 message/sec

a publisher publishes 1 message per second. Here, we notice that the standard derivations of the latencies of the Session-Encryption-SSL is obviously larger than the other two approaches; this is because SSL will run a complete SSL handshake after several runs of the SSL re-use sessions. This phenomenon is specifically obvious when the publisher sends more messages per second.

Table 6 summarizes some performance metrics of the three approaches for the case where a publisher publishes 1 message/sec. Regarding the received message overhead at the broker, the session-encryption-SSL's overhead is the largest, because it needs to send certificates; Figure 8 shows the message overhead sent to the broker of the three approaches. We can see that the group-key-E2E generates the least message overhead at the broker, and the broker of the session-encryption-SSL receives the largest amount of messages overhead. Figure 9 shows the message overhead trend at a publisher of the three approaches; among them, the session-encryption-SSL demands the most message overhead; but the differences are not so significant.

Table 6. Performance summary for 1Messages/sec

Node		Group-key-E2E	Session-encryption-CR	Session-encryption-SSL
10	CPU utilization at broker	0.981%	2.938%	1.424%
	Messages at broker (bytes/sec)	5807.43	5663.86	7215.43
	Messages at publisher (bytes/sec)	1254.95	1957.37	3906.78
	Memory at broker (MB)	3411	3823	2971
30	CPU utilization at broker	1.130%	2.604%	1.258%
	Messages at broker (bytes/sec)	7015.81	6977.80	8664.23
	Messages at publisher (bytes/sec)	3456.57	5438.29	6460.08
	Memory at broker (MB)	3512	3853	3100
50	CPU utilization at broker	1.224%	2.539%	3.828%
	Messages at broker (bytes/sec)	8055.76	8667.11	10127.05
	Messages at publisher (bytes/sec)	5593.34	8967.03	9038.37
	Memory at broker (MB)	3497	3907	3160

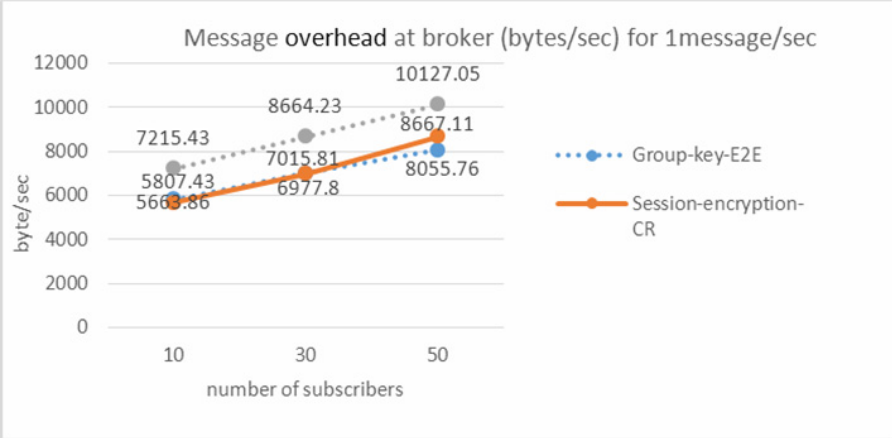

Figure 8. The received message overhead (bytes/sec) at the broker

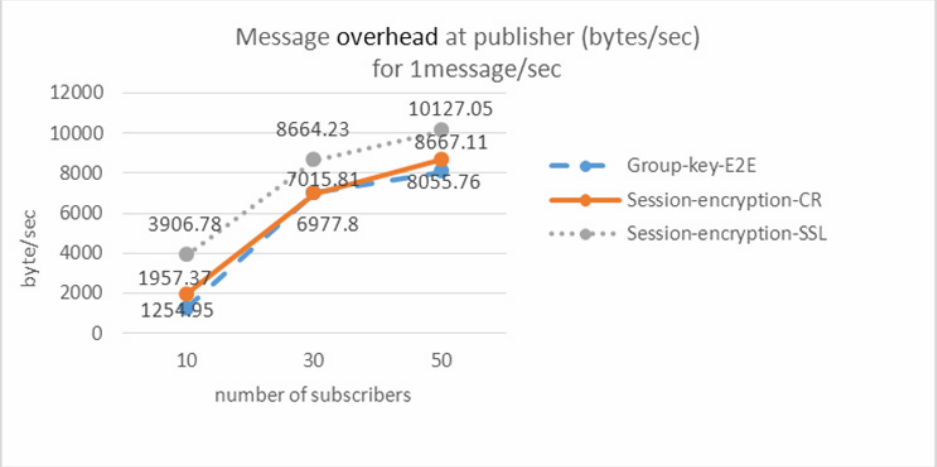

Figure 9. The message overhead (byte/sec) at a publisher

Regarding the broker's CPU utilization and the broker's memory overhead, we expected, before the experiments, that the session-encryption-SSL should have the largest CPU utilization and the largest memory usage; but the results show that its utilization and its memory usage are smaller than that of the Session-encryption-CR (except for its CPU utilization at broker when there are 50 clients)**;** we speculate that it is because the provider's implementation of the session-encryption-SSL has been optimized while our implementations of both the session-encryption-CR and the group-key-E2E could be further improved. Nonetheless, the group-key-E2E still shows the best performance in terms of communication latency.

In a short summary, our group-key-E2E has the least communication delay, because it generates the least message overhead and it does not decrypt-and-then-re-encrypt at the broker. The latency improvement becomes much larger as the number of clients increases. The session-encryption-SSL has the largest delay, and the message overhead at its broker is the largest.

5 Conclusions

In this paper, we have proposed and implemented the MQTT group communication framework. This framework supports group key distribution and facilitates secure group communication of MQTT messages. The experiments confirm that (1) the session-encryption-SSL generates the largest message overhead at the broker and has the largest communication delay; (2) our group-key-E2E generates the least message overhead at the broker and has the least delay; (3) the elimination of the decrypt-and-re-encrypt at the broker is the main reason of the performance improvement from our group-key-E2E; (4) the improvements become more significantly as the number of clients increases. The implementation and the experiments confirm that the proposed MQTT group communication framework can greatly improve the communication latency of MQTT systems; and this latency improvement is very important for many MQTT IoT scenarios.

Acknowledgements

This project is partially supported by the Ministry of Science and Technology, Taiwan, R.O.C., under grant No. MOST 108-2221-E-260-009-MY3.

References

[1] Y. Yang, L. Wu, G. Yin, L. Li, H. Zhao, A Survey on Security and Privacy Issues in Internet-of-Things, *IEEE Internet of Things Journal*, Vol. 4, No. 5, pp. 1250-1258, October, 2017.

[2] S. Y. Moon, J. H. Park, J. H. Park, Authentications for Internet of Things Security: Threats, Challenges and Studies, *Journal of Internet Technology*, Vol. 19, No. 2, pp. 349-358, March, 2018.

[3] S. N. Firdous, Z. Baig, C. Valli, A. Ibrahim, Modelling and Evaluation of Malicious Attacks against the IoT MQTT Protocol, *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, Exeter, UK, 2017, pp. 748-755.

[4] The HiveMQ Team, TLS BENCHMARKS, HiveMQ 3.1.0 on AWS, https://www.hivemq.com/downloads/hivemq_tls_benchmarks.pdf.

[5] T. Dierks, *The Transport Layer Security (TLS) Protocol Version 1.2*, IETF RFC 5246, August, 2008.

[6] The HiveMQ Team, TLS and MQTT: How is the performance affected?, https://www.hivemq.com/blog/how-does-tls-affect-mqtt-performance/.

[7] T. Yokotani, Y. Sasaki, Comparison with HTTP and MQTT on Required Network Resources for IoT, *2016 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*, Bandung, Indonesia, 2016, pp. 1-6.

[8] Amazon Web Services, Security and Identity for AWS IoT, https://docs.aws.amazon.com/iot/latest/developerguide/iot-security-identity.

[9] Mosquitto, http://projects.eclipse.org/projects/technology.mosquitto, 2018/11/07 access.

[10] Arduino Cloud, https://cloud.arduino.cc/, 2018/11/07 access.

[11] Shiftr.io, https://shiftr.io/, 2018/11/07 access.

[12] Mosca, https://github.com/mcollina/mosca/, 2018/11/07 access.

[13] H. Y. Chien, Y. J. Chen, Security Evaluation on Various Arduino-Compatible IoT Devices, *Cryptology and Information Security Conference 2018 (CISC 2018)*, Taipei, Taiwan, 2018.

[14] S. Andy, B. Rahardjo, B. Hanindhito, Attack Scenarios and Security Analysis of MQTT Communication Protocol in IoT System, *Proc. of 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI 2017)*, Yogyakarta, Indonesia, 2017, pp. 1-5.

[15] S. H. Shin, K. Kobara, C. C. Chuang, W. C. Huang, A Security Framework for MQTT, *2016 IEEE Conference on Communications and Network Security (CNS): International Workshop on Cyber-Physical Systems Security (CPS-Sec)*, Philadelphia, PA, USA, 2016, pp. 432-436.

[16] S. H. Shin, K. Kobara, *Efficient Augmented Password-Only Authentication and Key Exchange for IKEv2*, IETF RFC 6628, June, 2012.

[17] A. Bhawiyuga, M. Data, A. Warda, Architectural Design of Token based Authentication of MQTT Protocol in Constrained IoT Device, *2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA)*, Lombok, Indonesia, 2017, pp. 1-4.

[18] A. Mektoubi, H. L. Hassani, H. Belhadaoui, M. Rifi, A. Zakari, New Approach for Securing Communication over MQTT Protocol A Comparison between RSA and Elliptic Curve, *2016 Third International Conference on Systems of Collaboration (SysCo)*, Casablanca, Morocco, 2016, pp. 1-6.

[19] J. L. Espinosa-Aranda, N. Vallez, C. Sanchez-Bueno, D. Aguado-Araujo, G. Bueno, O. Deniz, Pulga, A Tiny Open-source MQTT Broker for Flexible and Secure IoT Deployments, *2015 IEEE Conference on Communications and Network Security (CNS)*, Florence, Italy, 2015, pp. 690-694.

[20] A. Rizzardi, S. Sicari, D. Miorandi, A. Coen-Porisini, AUPS: An Open Source Authenticated Publish/Subscribe System for the Internet of Things, *Information Systems*, Vol. 62, pp. 29-41, December, 2016.

[21] C. Lesjak, D. Hein, M. Hofmann, M. Maritsch, A. Aldrian, P. Priller, T. Ebner, T. Ruprechter, G. Pregartner, Securing Smart Maintenance Services: Hardware-Security and TLS for MQTT, *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, Cambridge, UK, 2015, pp. 1243-1250.

[22] M. Singh, M. A. Rajan, V. L. Shivraj, P. Balamuralidhar, Secure MQTT for Internet of Things (IoT), *2015 Fifth International Conference on Communication Systems and Network Technologies*, Gwalior, India, 2015, pp. 746-751.

[23] R. Neisse, G. Steri, G. Baldini, Enforcement of Security Policy Rules for the Internet of Things, *2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Larnaca, Cyprus, 2014, pp. 165-172.

[24] A. Niruntasukrat, C. Issariyapat, P. Pongpaibool, K. Meesublak, P. Aiumsupucgul, A. Panya, Authorization mechanism for MQTT-based Internet of Things, *2016 IEEE International Conference on Communications Workshops (ICC)*, Kuala Lumpur, Malaysia, 2016, pp. 290-295.

[25] H. Y. Chien, X. A. Kou, M. L. Chiang, C. H. Su, Secure and Efficient MQTT Group Communication Design, *20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD 2019)*, Toyama, Japan, 2019.

[26] H. Y. Chien, Y. J. Chen, G. H. Qiu, J. F. Liao, R. W. Hung, X. A. Kou, P. C. Lin, M. L. Chiang, C. H. Su, A MQTT-API-Compatible IoT Security-Enhanced Platform, *International Journal of Sensor Networks*, Vol. 32, No. 1, pp. 54-68, January, 2020.

[27] H. Y. Chien, G. H. Qiu, R. W. Hung, A. T. Shih, C. H. Su, Hierarchical MQTT with Edge Computation, *2019 IEEE 10th International Conference on Awareness Science and Technology (iCAST)*, Morioka, Japan, 2019, pp. 1-5.

[28] H. Y. Chien, Novel Attacks and Novel Efficient Three-Party Authenticated Key Agreement Schemes for Resource-Limited Devices, *Journal of Internet Technology*, Vol. 20, No. 7, pp. 2177-2188, December, 2019.

[29] V. Goyal, O. Pandey, A. Sahai, B. Waters, Attribute-based Encryption for Fine-grained Access Control of Encrypted Data, *13th ACM Conference on Computer and Communications Security (CCS '06)*, Alexandria, VA, USA, 2006, pp. 89-98.

[30] J. Bethencourt, A. Sahai, B. Waters, Ciphertext-Policy Attribute-Based Encryption, *2007 IEEE Symposium on Security and Privacy (SP '07)*, Berkeley, CA, USA, 2007, pp. 321-334.

[31] E. Hammer-Lahav, *The OAuth 1.0 Protocol*, IETF RFC 5849, April, 2010.

[32] MQTT, http://mqtt.org/, 2018/04/07 access.

[33] OASIS, OASIS Message Queuing Telemetry Transport (MQTT) TC, https://www.oasis-open.org/committees/mqtt/, 2018/11/07 access.

[34] E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.3*, IETF RFC 8446, August, 2018.

Biographies

Hung-Yu Chien received the B.S. degree from NCTU, Taiwan, 1988, the M.S. degree from NTU, Taiwan, 1990, and the doctoral degree in applied mathematics at NCHU 2002. He is a professor of National Chi Nan University since 199808. His research interests include cryptography, networking, network security, ontology, and Internet-of-Things.

Pei-Chih Lin is currently studying a master's degree in the Department of Information and Communication at the Chaoyang University of Technology in Taiwan. His research topic is Message Queuing Telemetry Transport and network security.

Mao-Lun Chiang received the Ph.D. degree in Department of Computer Science from National Chung-Hsing University, Taiwan. He is a professor in the Department of Information and Communication Engineering at the Chaoyang University of Technology, Taiwan. His current research interests include mobile computing, IoT, fault tolerant computing, and cloud computing.