A MQTT-API-compatible IoT security-enhanced platform

Hung-Yu Chien* and Yi-Jui Chen

Department of Information Management,
National Chi Nan University,
PuLi, Nantou 54561, Taiwan
Email: hychien@ncnu.edu.tw
Email: a25237780a@gmail.com
*Corresponding author

Guo-Hao Qiu, Jian Fu Liao and Ruo-Wei Hung

Department of Computer Science and Information Engineering,
Chaoyang University of Technology,
Wufeng, Taichung 41349, Taiwan
Email: z55121255@gmail.com
Email: tt159753tt@gmail.com
Email: rwhung@cyut.edu.tw

Pei-Chih Lin, Xi-An Kou and Mao-Lun Chiang

Department of Information and Communication Engineering,
Chaoyang University of Technology,
Wufeng, Taichung 41349, Taiwan
Email: qazz6411@gmail.com
Email: kandy841011@gmail.com
Email: mlchiang@cyut.edu.tw

Chunhua Su

Division of Computer Science,
The University of Aizu,
Aizuwakamatsu-shi, Fukushima 965-8580, Japan
Email: chsu@u-aizu.ac.jp

Abstract: Owing to its lightweight and easiness, the message queue telemetry transport (MQTT) has become one of the most popular communication protocols in the internet-of-things (IoT). However, the security supports in the MQTT are very weak. In this paper, we systematically examine the security requirements of a MQTT-based IoT system, identify the gap between the requirements and the supported functions, and design a security-enhanced MQTT framework. The framework facilitates device authentication, key agreement, and policy authorisation. Additionally, it is desirable that any MQTT-security enhancements should be compatible with existent MQTT Application Programming Interfaces (API). We propose a two-phase authentication approach that can smoothly integrate secure key agreement schemes with the current MQTT-API. To evaluate its effectiveness and efficiency, we implement prototype. Compared to its counterparts, the results show the merits of improved communication performance, MQTT-API compliance, and security robustness.

Keywords: transport layer issues; security and privacy; MQTT; message queue telemetry transport; internet of things; authentication.

Reference to this paper should be made as follows: Chien, H-Y., Chen, Y-J., Qiu, G-H., Liao, J.F., Hung, R-W., Lin, P-C., Kou, X-A., Chiang, M-L. and Su, C. (2020) 'A MQTT-API-compatible IoT security-enhanced platform', *Int. J. Sensor Networks*, Vol. 32, No. 1, pp.54–68.

Biographical notes: Hung-Yu Chien is a Professor of National Chi Nan University Taiwan. His research interests include cryptography, networking, network security, ontology, and Internet-of-Things.

Yi-Jui Chen obtained his Master degree from the Department of Information Management, National ChiNan University, Taiwan. His interests include system development and system integration.

Guo-Hao Qiu obtained his BS from the Department of Computer Science and Information Engineering, Chaoyang University of Technology, Taiwan in 2019. He is now pursuing his Master degree at Chaoyang University of Technology.

Jian Fu Liao obtained his BS from the Department of Computer Science and Information Engineering, Chaoyang University of Technology, Taiwan in 2019. He is now pursuing his master degree at Chaoyang University of Technology.

Ruo-Wei Hung is a Professor of Chaoyang University of Technology, Taiwan. His research interests include graph algorithms, Sensor Networks, and Linux System.

Pei-Chih Lin obtained his BS from the Department of Information and Communication Engineering, Chaoyang University of Technology, Taiwan. He is now pursuing his Master degree at Chaoyang University of Technology.

Xi-An Kou obtained his BS from the Department of Information and Communication Engineering, Chaoyang University of Technology, Taiwan. He is now pursuing his Master degree at Chaoyang University of Technology.

Mao-Lun Chiang is an Associate Professor of Chaoyang University of Technology, Taiwan. His research interests include cloud computing, mobile computing, distributed systems, and network security.

Chunhua Su is an Associate Professor of The University of Aizu, Japan. His research interests include preserving technology in big data, security and privacy for IoT devices, cryptanalysis, and cryptographic protocols.

1 Introduction

The IoT technologies are expected to improve many sectors of our daily life, and various IoT applications have been booming very fast. One of the key components in a IoT system is its communication protocols that facilitate the machine-to-machine (M2M) communication and the data transmission. Among several IoT communication protocols, the message queue telemetry transport (MQTT) is the most popular one, owing to its lightweight and easiness to use. There are many MQTT applications deployed globally.

However, MQTT itself does not provide the security protection like authentication, integrity, and confidentiality. It assumes the use of secure sockets layer (SSL) in the underlying layer. However, SSL demands more computational resources, and deploying certificates to every IoT devices is very effort-demanding. Additionally, MQTT itself does not enforce some desirable security properties and functions like topic access control, secure multicast, and simple device management. In 2018, Avast (https://press.avast.com/avast-research-finds-at-least-32000-smart-homes-and-businesses-at-risk-of-leaking-data) reported that there are 49,000 mis-configured MQTT servers and more than 32,000 servers with no password protection, putting them at risk of leaking data.

Therefore, in this paper, we propose a MQTT-security-enhanced framework and a two-phase authentication approach that facilitates designers an easy integration of any secure key agreement schemes (like the challenge-and-response (CR) mechanism) into the platform and being compatible with existent MQTT APplication interface (API). We implement the enhanced security functions and the CR in our platform. The results show that the proposed framework and the two-phase authentication mechanism is easy to use and does enhance the security functions. This paper has the following contributions.

- A systemic discussion of desirable MQTT security properties and functions
- A proposed MQTT-security-enhanced framework that can effectively improve the security and easiness to use
- A two-phase authentication approach which can easily integrate any secure key agreement schemes into MQTT systems
- A prototype of our proposed framework and the two-phase authentication using the CR key agreement has been implemented and evaluated
- Both the implementation-based evaluations and the analysis show that our approach outperforms the

conventional SSL approach in terms of computation cost, communication overhead, and easiness of deployment and maintenance.

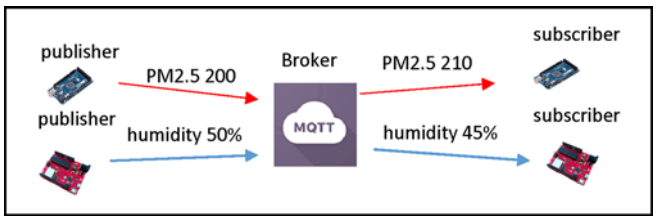
The rest of this paper is organised as follows. Section 2 discuss the related publications and platforms. Section 3 systematically identifies the security requirements and functions for enhancing MQTT security and management. Section 4 proposes our MQTT framework and the two-phase API-compatible key agreement mechanism, and describes our prototype implementation. Section 5 evaluates the communication performance of our implementation, and summarises the comparisons with related works. Section 6 states our conclusions and some future works.

2 Related work

There are several popular IoT transmission protocols such as MQTT (http://mqtt.org/), advance message queuing protocol (AMQP) (https://www.amqp.org/), constrained application protocol (CoAP) (http://coap.technology/), Extensible messaging and presence protocol (XMPP) (https://www.omgwiki.org/dds/), and data distribution service (DDS) (https://www.omgwiki.org/dds/). Among them, MQTT is the most popular one in the consumer internet of things (CIoT) applications, owing to its lightweight and easiness to use. There are many MQTT-based IoT systems worldwide. MQTT has been ratified as the ISO standard (ISO/IEC 20922: 2016) (https://www.iso.org/standard/69466.html) and the OASIS standard (https://www.oasis-open.org/committees/mqtt/), respectively.

MQTT is a message-oriented protocol and is based on the publish/subscribe interaction pattern. A MQTT system consists of a set of clients and a broker who acts as an intermediary among the clients. 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. The broker receives the messages from the publishers, and forward them to those subscribers. Figure 1 depicts the MQTT architecture, where 'PM25' and 'humidity' represent two topics. To keep the MQTT protocol lightweight and easy, the MQTT standard itself does not specify how to keep the transmission and the access secure; instead, it suggests the use of SSL/TLS and AES/DES for client authentication and for message encryption in the SSL layer. It is this simple principle that makes it so popular today, and it is also this principle that puts the MQTT-based systems in such a high risky situation or poor security-related functions (https://en.wikipedia.org/wiki/Mirai_(malware)). The security support in MQTT is very weak; we, therefore, focus on the security enhancement of the MQTT systems in this paper.

Figure 1 The MQTT architecture (see online version for colours)



There are several open-source/commercial MQTT platforms such as Amazon Web Services (https://docs.aws.amazon.com/iot/latest/developerguide/iot-security-identity.html), Mosquitto (http://projects.eclipse.org/projects/technology.mosquitto), Arduino cloud (https://cloud.arduino.cc/), Shiftr.io (https://shiftr.io/), Mosca (https://github.com/mcollina/mosca/), and there are many publications such as Chien and Chen (2018), Andy et al. (2017), Firdous et al. (2017), Shin et al. (2016), Shin and Kobara (2012), Bhawiyuga et al. (2017), Mektoubi et al. (2016), Espinosa-Aranda et al. (2015), Rizzardi et al. (2016), Lesjak et al. (2015), Singh et al. (2015), Neisse et al. (2014), Niruntasukrat et al. (2016) discussing or improving the security of the MQTT systems. However, these enhancements are limited and they might not be compatible with the current MQTT APIs.

The platforms such as Arduino cloud, Shiftr Cloud, and MOSCA provide web pages for users to create a topic (they respectively call it a 'thing' or a 'name space'). They provide a topic identity and a corresponding key to those registered clients so that the broker can later validate whether a client owns the knowledge of the secrets. This can be viewed as the topic-level authentication. Since all the users (and their devices) who are authorised to access the topic share the same secrets, a single compromised device would endanger the access of the topic, and the system does not authenticate each individual device. Amazon's IoT security solution (https://docs.aws.amazon.com/iot/latest/developerguide/iot-security-identity.html) supports TLS/SSL client authentication, but it does not support MQTT-level encryption, multicast, and dynamic multicast. It supports customised authentication, but there is no enough information to tell whether it is compatible with MQTT APIs.

Andy et al. (2017) demonstrated several attack scenarios on the MQTT platforms. Firdous et al. (2017) evaluated the vulnerability of the MQTT protocol under the denial-of-service (DOS) attacks. Chien and Chen (2018) evaluated the security vulnerability of several Arduino products (https://www.arduino.cc/en/Guide/ArduinoUnoWiFi; https://www.arduino.cc/en/Main/ArduinoMKR1000?s_tact=C3970CMW; https://wiki.wemos.cc/products:d1:d1_mini) acting as MQTT clients, and they find several IoT devices (https://www.arduino.cc/; https://www.raspberrypi.org/; https://www.arduino.cc/en/Guide/ArduinoUnoWiFi; https://

www.arduino.cc/en/Main/ArduinoMKR1000?s_tact=C3970CMW; https://wiki.wemos.cc/products:d1:d1_mini) are vulnerable to various attacks and compromise.

Shin et al. (2016) noticed that some existent secure MQTT solutions highly depend on public key infrastructure (PKI), which requires expensive computations and heavy communication overhead of accessing/verifying the certificate revocation lists (CRLs). Therefore, they, based on the Mosquitto 1.4.9 platform (http://projects.eclipse.org/projects/technology.mosquitto) and the AugPAKE protocol (Shin and Kobara, 2012), designed the AugMQTT platform which provides device authentication and establishes session keys between a client and the MQTT broker. The AugPAKE protocol uses simple passwords to establish Diffie-Hellman keys; it is efficient. However, they did not consider how the system authorises a subscription and how the AugPAKE protocol could be integrated with the MQTT standard API.

Bhawiyuga et al. (2017) noticed the default authentication mechanism of using username and password in the MQTT API would have poor security and poor scalability; therefore, they propose their token-based authentication solution; an authentication server generates a token for a client which keeps it for later requesting connections. However, the token just simply encodes the username and the password without using any encryptions, and there is no session key generation for the connections.

Mektoubi et al. (2016), based on the PKI system and the symmetric key encryption, implemented the client authentication and the topic-related message encryptions, using several standard algorithms such as RSA, elliptic curve cryptosystem (ECC), and advanced encryption standard (AES). One key feature of the scheme is that there is one specific certificate for each topic so that the messages for one topic can be encrypted using the public key of the certificate and can be decrypted using the corresponding private key. This feature facilitates the possible multicast of the topic messages. They conclude that the feasibility of the proposed solution in practice; however, they also emphasise that the management of the lifecycle of the certificates/keys, the scalability challenge for a large number of clients, and the evaluation of other lightweight algorithms should be further investigated. Espinosa-Aranda et al. (2015) designed a specialised hardware to help an IoT device handle the SSL connection. This extra hardware solution is costly for many IoT deployments.

Rizzardi et al. (2019) and Neisse et al. (2014) respectively enhanced the security of the MQTT platform from the architecture point of view; they enhance the platform by augmenting it with the key management framework and a policy management framework so that the messages could be flexibly and dynamically encrypted and accessed according to the flexible policies. They all focused on a framework for policy enforcement while our study focusing on efficient-and-MQTT-API compatible authenticated key agreement design. Both approaches complement each other.

Lesjak et al. (2015) designed a specialised hardware called the meditator to be integrated with an IoT device and to help the device handle the TLS server authentication with a MQTT broker. A performance comparison with the native OpenSSL authentication exhibits no significant negative impact, given the specified industrial scenario. However, this extra hardware solution is costly and might not be affordable to many CIoT scenarios, and it only provides limited security enhancement.

Singh et al. (2015) proposed their secure MQTT (SMQTT) and Secure MQTT for sensor networks (SMQTT-SN) protocols, by augmenting exist MQTT protocols with Key/Ciphertext Policy-Attribute Based Encryption (KP/CP-ABE) (Wang et al., 2014; Goyal et al., 2006; Bethencourt et al., 2007). This attribute-based-encryption-based approach has the potential advantage of providing flexible access policy for 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. They only evaluate the encryption/decryption time, but not including the communication time. 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.

Niruntasukrat et al. (2016) adopted OAuth1.0a (Hammer-Lahav, 2010) as the authorisation mechanism for MQTT system. Like our scheme, their system considers device authentication by incorporating device identity and device key. But their scheme requires the devices to get an authorisation permission from the users during the authentication process. This requirement of user involvement during the authentication process not only increases the communication delay but also significantly increases the inconvenience of IoT applications.

3 Identifying the security requirements and functions

Even though both the academia and the industry has paid great attention to the security challenges of the MQTT and there are several publications/platforms devoting to enhance the security, the desirable security requirements and the security-related functions of a MQTT system have not yet been well discussed. Most of the works all agree that authentication, integrity, and confidentiality of the transmission are essential, but there are still some fundamental or subtle requirements being neglected. Here, we systematically examine the requirements.

There are several high-end and security-aware IoT devices/platforms on the market, but many developers might choose some affordable IoT platforms such as Arduino and Raspberry pi to develop their systems, owing to the cost competitiveness, the open architecture, and the resourceful supports. These inexpensive IoT devices are vulnerable to various attacks once they are captured. Capturing IoT

devices is much easier, compared to the conventional servers and workstations, as they are often deployed in unprotected areas. Therefore, *individual authentication of each device* and generating MQTT-level session keys for the connection are crucial. We name the requirements the device authentication (DA) and the MQTT-session key (MSK), respectively. Unfortunately, many MQTT platforms (Avast, https://press.avast.com/avast-research-finds-at-least-32000-smart-homes-and-businesses-at-risk-of-leaking-data; Amazon Web Services, https://docs.aws.amazon.com/iot/latest/developerguide/iot-security-identity.html; Mosquitto, http://projects.eclipse.org/projects/technology.mosquitto; Arduino cloud, https://cloud.arduino.cc/; Shiftr.io, https://shiftr.io/; Mosca, https://github.com/mcollina/mosca/) just simply authenticate the privilege of accessing a topic but not the devices, and do not provide any MQTT-level session keys for encrypting the MQTT payloads. Of course, the detection of possible device compromise and intrusion is very important; but it can be independently developed for all IoT systems and integrated with a MQTT system. We, therefore, do not include it in the following discussion.

The access of the MQTT messages is based on the publication and the subscription to a specified topic, where a topic is a designated string that differentiates one topic from another. For example, let the string 'environment' be a topic, and the string 'environment/home' and the string 'environment/office' could be two subtopics under the upper level topic 'environment'. The authorisation of accessing a topic and possible accessing the sub-topics is policy-dependent: each MQTT application might have quite different policies, depending the characteristics of the applications and the topics. Therefore, we differentiate the authorised topic access (ATA) and the fine-grained authorised topic access (FGATA); here the ATA refers to a coarse access control that, once a client is authorised to a topic, the client is authorised to access all of the sub-topics. On the other hand, the FGATA refers to a fine-grained access control where the authorisation of accessing each sub-topic is separately examined and authorised, even if the upper topic is authorised to the client. Of course, the choice of which authorisation policy to be applied is highly application-dependent. However, a secure MQTT system should provide the function of policy management (PM) and some kind of mechanisms to enforce the policy.

The MQTT standard suggests the use of TLS connections for protecting the transmissions and even the possible client authentication. However, to authenticate a client using a certificate, it requires each client being equipped with a certificate and each client has to process the signature generation, the signature verification, and the checking of CRL. It is too costly in terms of computations and communications for many IoT devices. An SSL connection encrypts a message on the transport layer, but not on the MQTT layer. It would be desirable that a MQTT system could encrypt the messages by itself instead of depending on the underlying TLS encryption only; we call this property the MQTT message encryption (MME).

We also notice that many clients would subscribe to the same topic. If a MQTT broker should use each individual key to encrypt the message for each subscriber, then it would be very inefficient. Therefore, it is desirable that a MQTT system could support secure multi-cast (SMC); that is, the broker (or a publisher) just sends one encrypted message instance and all the authorised subscribers can decrypt the encryption directly. The membership of a topic group might be static or quite dynamic. For those dynamic topic group, it is desirable that a MQTT system supports dynamic secure multi-cast (DSMC).

Finally, it is desirable that an enhanced security solution is compatible with the MQTT standard APIs so that an existent MQTT platform can apply the solution without modifying the Standard MQTT APIs: We call this the standard-MQTT-API compatible (SMAC).

Among the above requirements, some requirements like DA/MSK/ATA/PM/MME are crucial, because they are closely related to the security robustness. The others like FGATA/SMC/DSMC/SMAC are desirable because these functions could improve the efficiency, the convenience of usage, or the MQTT-API compatibility. In our system, we implement these crucial functions and the SMAC criteria, and plan to tackle the other challenges for further improving the efficiency and the convenience.

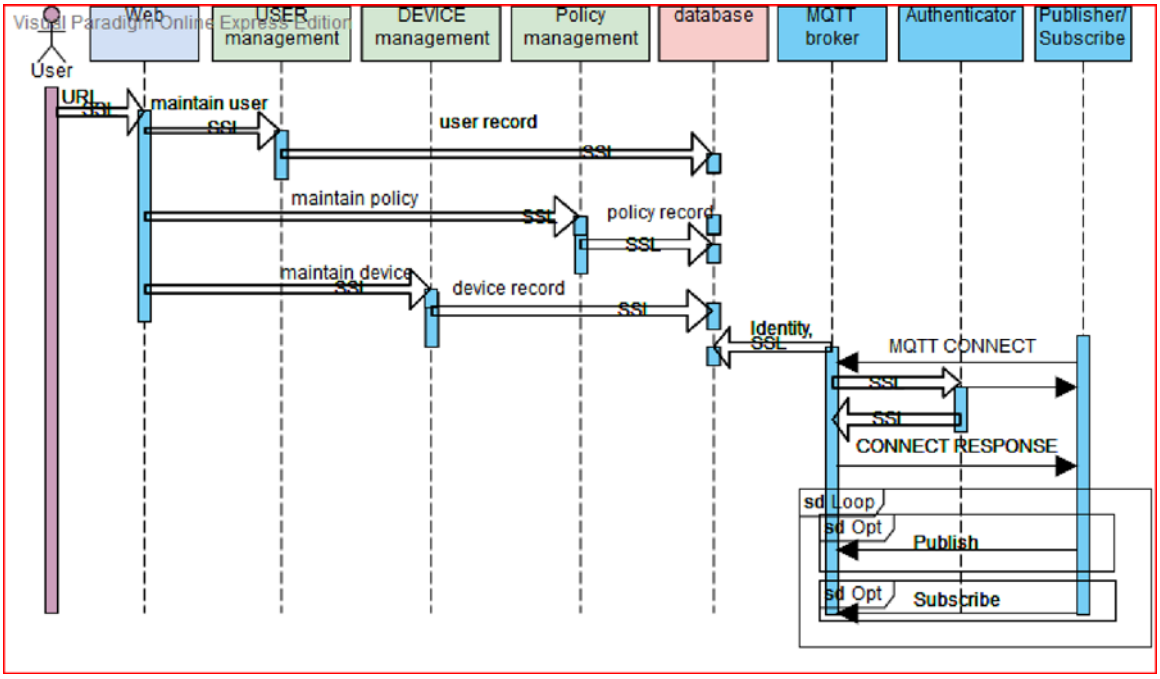
4 The proposed architecture, two-phase device authenticated key agreement, and functions

This section describes our MQTT security architecture, the two-phase authentication approach, and our prototype implementation. The two-phase authenticated key agreement facilitates the integration of any secure key agreement scheme into existent MQTT platforms, and it is compatible with the current MQTT standard APIs.

4.1 The system architecture

Figure 2 shows the system sequence diagram of our security-enhanced MQTT platform. The system provides web interfaces for the manager and users to manage user accounts, to register their devices, to specify and manage policies and topics. The user-system interactions and the interactions between the system modules and the database are conducted via secure SSL channels ('↑' in Figure 2). The interactions with the symbol '↑' in Figure 2 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.

Figure 2 The system sequence diagram of our security-enhanced MQTT platform (see online version for colours)



The policy management module allows users to specify their policies and to authorise other users' requests for accessing the topics. Each topic is associated with an identity and a corresponding key. A user who creates a topic is responsible for authorising other users' requests for that topic.

Once a user has been authorised to access a topic and his advice registration has been authorised, 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. During the authentication, the broker and the device, based on the knowledge of the thing key and the device key, will negotiate a session key which will be used to encrypt the messages of that session.

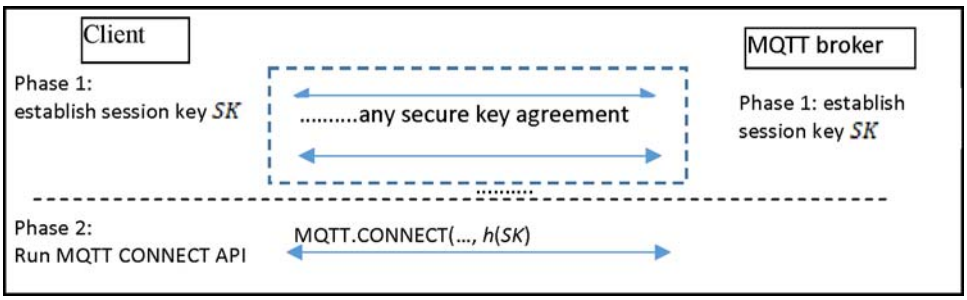
4.2 The two-phase MQTT authentication

In the current MQTT standards, v 3.1 (ISO/IEC 20922:2016, https://www.iso.org/standard/69466.html; Locke, 2010), the packet CONNECT and the CONNECT API are used for a client to specify the broker to be connected and to pass the authentication parameters for this connection request. The CONNECT API has the two fields for passing the authentication data; the two fields are the 'username' field and the 'password' field. The current standards do not consider the extension/flexibility to incorporate other authentication options. In many MQTT platforms such as Mosquitto (http://projects.eclipse.org/projects/technology.mosquitto), Arduino cloud (https://cloud.arduino.cc/), Shiftr.io (https://shiftr.io/), Mosca (https://github.com/mcollina/mosca/), these two fields are used to convey the topic-level identity and the corresponding key (when a user creates a topic in the platform, the system generates an identity string to represent the topic and a corresponding key). Even though the principle of the standards is open to any possible authentication mechanisms; however, the design of the CONNECT packet and the CONNECT APIs makes it very difficult to incorporate other authentications. That is one of the reasons why so many platforms such as Mosquitto (http://projects.eclipse.org/projects/technology.mosquitto), Arduino cloud (https://cloud.arduino.cc/), Shiftr.io (https://shiftr.io/), Mosca (https://github.com/mcollina/mosca/) still restrict to the topic-level authentication. Additionally, MQTT itself would not generate any session keys.

We therefore, design a two-phase authentication for MQTT. This design has two merits. First, it facilitates designers an easy way to integrate any secure key agreement schemes into MQTT. Second, this approach is compatible with the current MQTT standards and APIs (ISO/IEC 20922:2016, https://www.iso.org/standard/69466.html; Locke, 2010).

Figure 3 depicts the two-phase approach. In Phase 1, the client and the broker can run any secure authenticated key agreement schemes to authenticate each other and to establish a secure session key SK. In Phase 2, the client initiates the CONNECT API and use the hashed key $h(SK)$ to replace the password field; the broker authenticates the client, based on the knowledge of $h(SK)$. This approach is easy and perfectly compatible with the APIs.

Figure 3 Two-phase MQTT authenticated key agreement (see online version for colours)



4.2.1 An implementation of the two-phase approach using the challenge-and-response (CR) technique

To validate our proposed approach and to evaluate the performance, we implement the popular and well-studied Challenge-and-Response (CR) mechanism in Phase 1, and embed the hashed key in the MQTT.CONNECT API in Phase 2. A client and the broker first, via a web socket connection, run the CR protocol, establish a session key, and then the client passes the hashed key as the 'password' in the CONNECT API (packet). The notations used are introduced in Table 1.

We note that our CR implementation verifies not only the knowledge of the topic key but also the knowledge of the device key. We describe the details of both the device registration phase and the MQTT authentication phase as follows.

Table 1 The notations and abbreviations

C; B	C denotes one client; B denotes one broker.
ID_{thing}, K_{thing}	ID_{thing} denotes the identity of a 'thing'. K_{thing} denotes the corresponding key assigned by the system.
ID_{device}, K_{device}	ID_{device} denotes the identity of a 'device'. K_{device} denotes the corresponding key assigned by the system.
$C1, C2$, $R1, R2$	$C1, C2$: random challenges. $R1, R2$: responses
$h()$	Cryptographic hash function.
CONNECT[ID,PW]	The standard MQTT packet for the connection request, which contains two authentication-related parameters (ID and PW).
T_h , $T_{RSA-sig}$, $T_{RSA-ver}$, $T_{RSA-enc}$, $T_{RSA-dec}$	$T_h / T_{RSA-sig} / T_{RSA-ver} / T_{RSA-enc} / T_{RSA-dec}$ respectively denotes the computational cost for hashing/RSA signature/RSA verification/RSA encryption/RSA decryption.

4.2.2 The device registration

The user registers his device using the device management module, and the system returns the device identity ID_{device} and the secret key K_{device}. The user creates a 'thing' using the policy management module, and the system returns the thing identity ID_{thing} and a secret key K_{thing}. Please note that a 'thing' is an IoT application that roughly corresponds to a topic in the MQTT context; it is called a 'thing' in the Arduino MQTT cloud (Arduino cloud, https://cloud.arduino.cc/), and is called a 'namespace' in the Shiftr MQTT platform (Shiftr.io, https://shiftr.io/).

4.2.3 The two-phase authentication process

This process is executed between a client (an IoT device) and a broker to authenticate each other and to establish session keys, based on the knowledge of (ID_{thing}, K_{thing} , ID_{device}, K_{device}). Figure 4 depicts the sequence diagram of the authentication process, and we describe the steps as follows. The first three interactions are conducted on the socket channel, and the others are MQTT APIs.

1 $C \rightarrow B$: $ID_{device}, ID_{thing}, C1$

The client first chooses a random challenge $C1 \leftarrow_R \{0,1\}^{|q|}$, where q is the system security parameter. It sends the identities and the challenge to the broker.

2 $B \rightarrow C$: $C2$, $R1$

Upon receiving the client's request, the broker first loops up the keys (K_{device}, K_{thing}) from its database, chooses a random challenge $C2$, and then computes $R1 = h(K_{thing} \,||\, K_{device}, C1)$ and $R2 = h(K_{thing} || K_{device}, C2)$. It sends ($C2$, $R1$) back to the client.

3 $C \rightarrow B$: $R2$

The client first validates the received $R1$. If the verification succeeds, then it computes $R2 = h(K_{thing} \,||\, K_{device}, C2)$. The client sends back the value $R2$, and it computes $SK = h(ID_{thing}, K_{thing}, ID_{device}, K_{device}, C1, C2)$.

The broker performs the validation of the received value $R2$. It then computes SK, if the verifications succeed.

At this point, the client and the broker share the same key SK, if they properly follows the above process.

Now the client can wrap the identities and the key into the standard CONNECT API as follows.

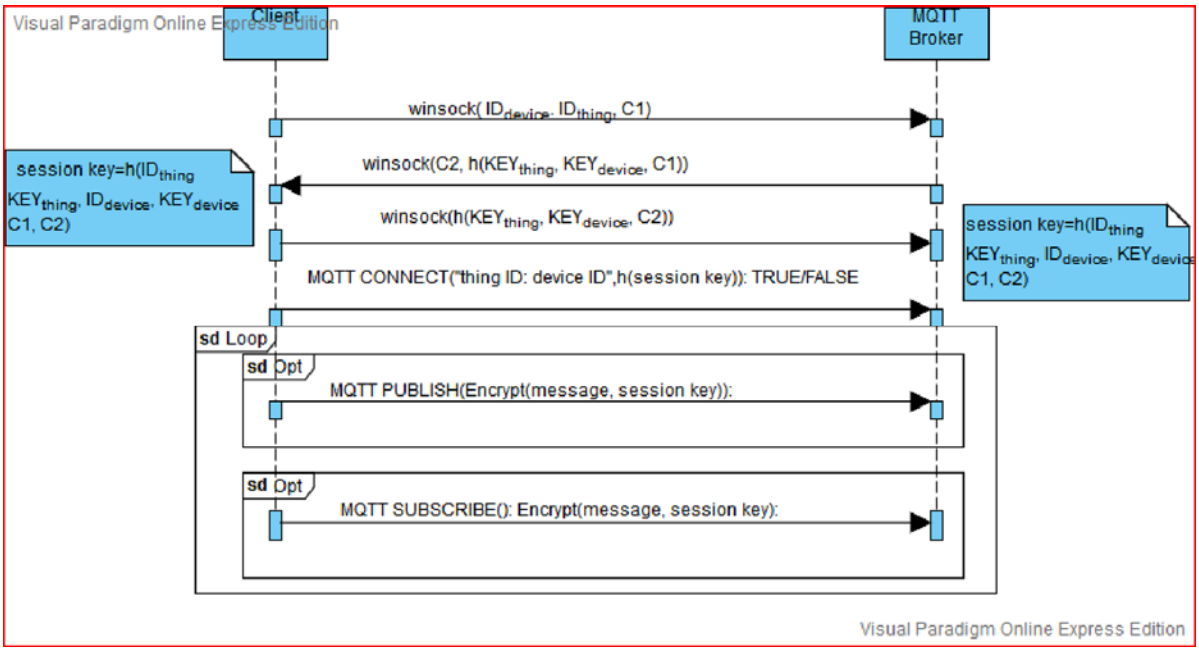
4 $C \rightarrow B$: $CONNECT[ID, PW]$

The client sets $ID := "ID_{thing} : ID_{device}"$ and $PW := h(SK)$, and sends the CONNECT packet to the broker.

Upon receiving the CONNECT request, the broker first parses the parameters to separate the two identities and the hashed key. Then, it verifies the hashed key. If the verifications succeed, then the client and the broker successfully authenticate each other and share the same session key.

Please note that the phase is executed between a client and a broker, and the SSL connection is not required in this phase. This property has a significant meaning: a resource-limited IoT device can easily connect the broker and establish session keys, without the overhead of certificate verification, without the overhead of the CRL checking, without the cost of signature generations/verifications, and without the overhead of installing certificates on clients.

Figure 4 Integration of the CR authentication with the MQTT connect API (see online version for colours)



4.3 The functions and implementations

Based on the open source Mosca platform (Mosca, https://github.com/mcollina/mosca/), web socket, JSON (https://www.json.org/), Arduino (https://www.arduino.cc/), and node.js (http://www.debugrun.com/a/cZomeQJ.html/), we have implemented the proposed security-enhanced framework (depicted in Figure 2) and the proposed CR-based key agreement (described in Section 4.2). Here, we introduce some major functions of our extension of the Mosca platform. We respectively introduce these functions, based on the categories.

1 The device management module

In this module, a user can register his devices here (Figure 5(b)). The administrator is responsible for authorising a registered device, and an authorised device will get a device key for it (Figure 5(c)).

2 The policy management module

In this module, a user can create 'things' (a 'thing' roughly corresponds to a 'topic' in the MQTT context) in Figure 5(a). An authenticated user can browse the listing of all the 'things', and clicks the button to request for accessing the 'thing' in Figure 5(d). The user who creates a 'thing' is responsible for authorising other users' requests for the 'thing'.

3 The device-thing authenticator module

We have implemented the proposed integration of the CR key agreement scheme and the MQTT CONNECT API. The channel for initiating the CR key agreement mechanism is built on a web socket, and the packets are wrapped as a JSON object.

Next, we show some key functions using the implementation outputs. Figure 6(a) shows the JSON output of a created thing. Figure 6(b) shows the JSON output of an authorised device, where both the device identity and the device key are generated. Figure 6(c) shows the interaction output between a client and a broker; they perform the CR protocol first, the client then issues the CONNECT API and gets connected, and finally the client publishes a message. On the client side, it prints out the data sent and received in a concise form. On the server (broker) side, it only prints out the results of each step.

Figure 5 Some functions (user interfaces) of the security-enhanced system: (a) creation of a 'thing'; (b) a registered and authorised 'thing'; (c) one device gets authorised, and the other is not and (d) a user clicks the request button of an interesting 'thing' (see online version for colours)

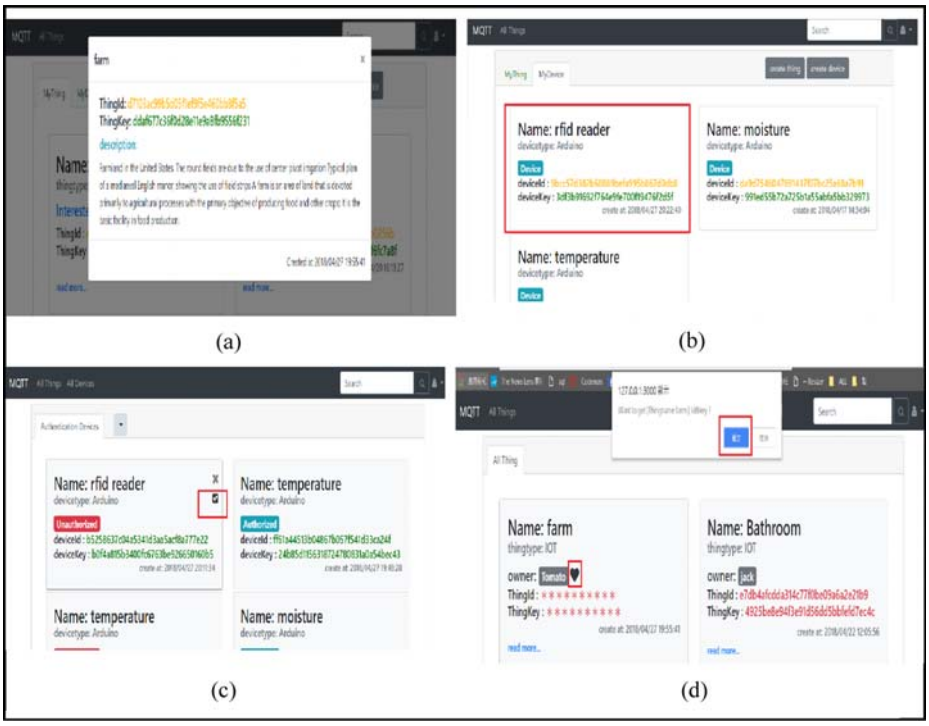
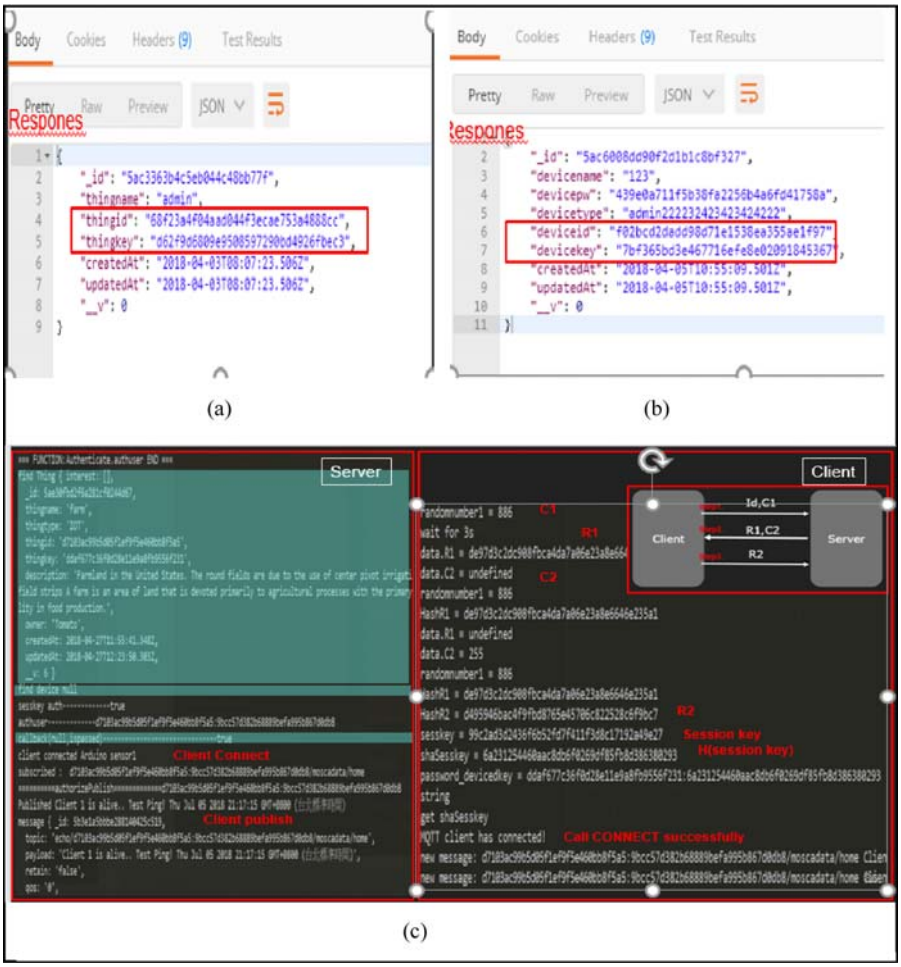


Figure 6 The MQTT-API-compatible key agreement output: (a) JSON output of a thing; (b) JSON output of an authorised device and (c) client-server interaction using the CR protocol and CONNECT API (see online version for colours)



ATA: the access control for course topic access

For implementing the access control of topic access, we propose to extend the parameter passed in the publish and subscribe API. In the original API, only the topic path is included (say '/PM25'). Now we append it with 'thing id||device id' as 'thing id||device id/PM25' so that the broker can verify the access request.

5 Experiments and evaluations

To evaluate the performance, we implement the two-phase authentication using the CR mechanism and the SSL settings in two environments.

5.1 The experiments

We have implemented our solutions using two different sets of environments. One is using the node.js on a notebook as a platform (we name it Lab1), and the other is using an Arduino-like platform WeMos D1 as the client (we name it Lab2). Both Lab 1 and Lab 2 only test one client with one broker. The Lab1 is to evaluate the performance for resource-abundant clients, and it is conducted in a wired LAN to avoid possible communication disturbance of wireless links. The Lab2 is to evaluate the performance on resource-limited IoT devices in the Internet environment, where the clients have to connect to the Internet to access the remote broker.

Table 2 depicts the settings for the hardware and software in the Lab1 experiment. Table 3 summarises the comparison of the communication performance. *Here, we should note that the SSL setting does not authenticate the client device while our CR implementation providing mutual authentication. For SSL to authenticate a client, it requires each client to be equipped with an individual certificate and the communication in the SSL handshake protocol would take longer, because the client has to send its certificate and the server has to verify the certificate.* From Table 3, we can see that, even if the SSL does not authenticate the client and our CR mechanism is implemented in the application layer, the SSL connection time is still longer than ours. We also notice that SSL/TLS with client-rebooting would really execute the SSL handshake protocol in every session so that it even takes up to 355.31 ms; otherwise, SSL without rebooting, would resume the old session keys, which do not execute the SSL handshake for each session, and it takes 74.65–102.08 ms.

Table 2 Hardware settings for Lab1 experiment in a LAN environment

	Client	*Server*
CPU	intel® Core™ i7-4790 CPU @ 3.60GHz	intel® Core™ i7-4702MQ CPU @ 2.20GHz
OS	Windows 7	Windows 10
RAM	6 GB	16 GB
Model	Acer Veriton M6630G	HP Probook 450 G1
Network card	Intel® Ethernet Connection I217-LM	Intel® Dual Band Wireless-AC 3160
Router	D-Link® DIR-809 Wireless AC750	D-Link® DIR-809 Wireless AC750
Software	Node.js 10.13.0, mqtt 2.18.3	Node 8.9.3, mongoose 5.4.1, mosca 2.8.3, passport-local 1.0.0

Table 3 The communication performance comparison of Lab1

	Two-phase with CR	*SSL*
Functions	Mutual authentication between device and broker Thing authentication	Only broker authentication Thing authentication
Conn. Time after rebooting	74.65–102.08 ms; average 78.65 ms	70.21–355.31 ms; Average 87.74 ms
Conn. time without reboot	36.96–43.46 ms; average 39.30 ms	41.48–47.33 ms; Average 43.73 ms

Table 4 Hardware settings for Lab2 experiment in an Internet environment

	Client	*Server*
CPU	Arduino WeMos D1 R1	intel® Core™ i7-4790 CPU @ 3.60GHz
OS	Arduino	Windows 10 enterprise
RAM		8 GB
Network card	Intel® Ethernet Connection I217-LM	Intel® Dual Band Wireless-AC 3160
Router	D-Link® DIR-809 Wireless AC750	D-Link® DIR-809 Wireless AC750
Software	Arduino 1.8.8 IDE, AsyncMqttClient.h, SocketIOClient.h	mosca: 2.8.0, Node.js v10.15.0, mongoose: 5.0.11, openSSL: 1.1.1a

Table 4 depicts the hardware and software settings for the Lab2 experiment. Table 5 summarises the communication performance comparison. The results show that the SSL connection take longer than ours, and the difference become larger in the Lab2 experiment. We think that it might be caused by the longer transmission latency of the server certificate in the Internet environment.

Table 5 The communication performance comparison of Lab2

	Two-phase with CR	*SSL*
Functions	Mutual authentication Thing authentication	Only server authentication Thing authentication
Conn. time	Average 551 ms	Average 719 ms

Next, we extend the experiments to test the performance for 1 broker with 10 clients, of which each client initiate 50 sessions. We follow Lab 2 hardware setting, but change the network environment from the Internet to the LAN environment. The average connection latency for our CR scheme is 41.487 ms while that for the SSL version being 995.31 ms. The fast increasing in the SSL version might result from the increasing of the transmissions of many certificates.

Finally, we follow Lab 1 hardware setting to simulate 190 clients using Node.js software. We separate these clients on different hardware platforms. Client group 1 of 50 clients runs on the same sever which hosts the broker. Client group 2 with 60 clients runs on a Personal Computer (PC) with a LAN interface. Client group 3 with 60 clients runs on a notebook with a wifi interface. Client group 4 with 20 clients runs on a smart phone with a wifi interface. Table 6 summarises the settings and the results. From the table, we have three observations.

- The resource constraint does have great impact on the latency of both aooroaches. The latency of clients on the smart phone, even only 20 clients, is much larger than other groups.
- Client group 1 running on a server that also hosts the broker have larger latency than Group 2 and Group 3, because the broker's loading become much larger when the number of clients increases.
- The SSL version takes longer latency than the CR version, and the difference become much significantly when the clients are running on resource-constrained devices (here, we use a smart phone to simulate many clients).

Table 6 Connection latency of 190 clients using different hardware

190 clients	*1* (*local host:50 clients*)	*2* (*PC+LAN: 60 clients*)	*3* (*NB+wifi: 60 clients*)	*4* (*smart phone +wifi: 20 clients*)
CR	9–13 ms	6–10 ms	9–15 ms	44–65 ms
SSL	9–15 ms	11–13 ms	17–20 ms	70–100 ms

5.2 Analytic approach to compare the CR-based version with the SSL-based version

Now we compare the performance of our CR-based two-phase authentication with the SSL-based version, using an analytic approach. We examine their performance in computation, in communication latency, in message overhead, and ease of maintenance. For the SSL-based approach, the SSL handshake protocol is the most computation-intensive part, and we will focus on this part. Here, we assume a simple and common setting for most SSL-based deployments: a certificate of 2048-bit RSA and 256-bit AES.

Regarding the computational complexity, our approach just demands three hashing operations ($3\,T_h$); while the SSL-based approach requires at least three signature verifications, four hashing operations, one signature generation, one public-key encryption, and one public-key decryption- ($4\,T_h + 1\,T_{RSA-sig} + 3\,T_{RSA-ver} + 1\,T_{RSA-enc} + 1\,T_{RSA-dec}$). Based on the figures from the computation testing on hardware implementations (Tschofenig and Pegourie-Gonnard, 2015; Sinha et al., 2013), the ratio between our approach cost and the SSL approach is around 0.03; that is, the computational cost of our scheme is around 3% of the SSL approach.

Now we examine the message overhead. Several factors affect the message overhead of the SSL approach, like the number of supported algorithms, the size of the keys, the number of certificates in the certificate chain, and so on (netsekure rng, http://netsekure.org/2010/03/tls-overhead/; Narens, http://narendrasharma.blogspot.com/2018/01/tls-and-mutual-tls-handshake-data.html). For a 2048-bit-RSA-and-256-bit-AES certificate with only one certificate in the certificate chain, the message overhead is around 1949 bytes. For a 2048-bit-RSA-and-256-bit-AES certificate with only one certificate in the certificate chain, the message overhead is around 6449 bytes. On the contrary, the message overhead of our approach is 72 bytes, because the scheme only transmits two identifications, two challenge numbers, and two hashes. The ratio between the message overhead of our approach and the SSL approach is 0.03 for the case of one certificate in the certificate chain, and it is 0.01 for the case of six certificate in the certificate chain.

Finally, we analyse the message latency. Because both the SSL approach and our CR-based approach all need to initiate the MQTT connection, here we only count the message flow before the MQTT connection. According to the TLS specification (Internet Engineering Task Force, https://tools.ietf.org/html/rfc8446), a TLS mutual authentication requires 12 message steps in four round trips.

On the contrary, our C-R based approach only needs three messages in 1.5 round trips.

Regarding the ease of maintenance, the SSL approach requires much larger maintenance overhead because it needs the maintenance of the PKI and the client certificate installation. Table 7 summarises he above comparisons. From the table, we can easily understand why the SSL approach takes longer latency in the experiments.

Table 7 Comparison between our CR approach and the SSL approach

	Our CR approach	*SSL approach*	*Our CR-based approach/ SSL approach*
Computation	$3\,T_h$	$4\,T_h + 1\,T_{RSA-sig} + 3\,T_{RSA-ver} + 1\,T_{RSA-enc} + 1\,T_{RSA-dec}$	0.03[1]
Message overhead	72 bytes	1949 (6449) bytes[2]	0.03 (0.01)[2]
Message steps (round-trip)	3(1.5)	12(4)	NA
Conn. Time Lab 1	39.48 ms	45.3 ms	
Conn. Time Lab 2	551 ms	719 ms	
Maintenance Overhead	Just a shared key between clients and broker	PKI maintenance and client certificate installation	

NA: Not applicable.

1 The ratio is based on a 2048-bit-RSA-and-256-AES certificate and the hardware testing (Tschofenig and Pegourie-Gonnard, 2015; Sinha et al., 2013).

2 The message overhead and the ratio varies, depending on the number of certificates in a certificate chain. Here, we assume 1 and 6 respectively.

Table 8 Comparison of several MQTT security-enhancement proposals and platforms

Platforms/enhancements	*DA*	*MSK*	*PM*	*MME*	*ATA*	*FGATA*	*SMC*	*DSMC*	*SMAC*	*Comments*
Arduino Cloud (https://cloud.arduino.cc/)	X	X	X	X	V	X	X	X	X	Only ATA is provided.
Shiftr Cloud (https://shiftr.io/)	X	X	X	X	V	X	X	X	X	Only ATA is provided.
MOSCA (https://github.com/mcollina/mosca/)	X	X	X	X	V	X	X	X	X	Only ATA is provided.
AugMQTT (Shin et al., 2016)	V	V	X	V	V	X	X	X	X	Messages are encrypted by a publisher. The broker decrypts the message and re-encrypts it, using each subscriber's session key.
Bhawiyuga et al. (2017)	X	X	X	X	X	X	X	X	X	Only an un-encrypted token is proposed to save the storage of the broker.
Mektoubi et al. (2016)	V	X	X	V	V	X	V	X	X	A client is responsible for authenticating all the devices that request for a topic it creates, which is an unbearable burden for many IoT devices. Expensive PKI computations are required.
AUPS (Rizzardi et al., 2016), SecKit (Neisse et al., 2014)	X	X	V	V	V	X	X	X	X	They designed a framework without specifying the authenticated key algorithms and other required functions.
Lesjak et al. (2015)	V	X	X	V	X	X	X	X	X	They designed a special hardware to help a device perform SSL connection.
AWS (https://docs.aws.amazon.com/iot/latest/developerguide/iot-security-identity.html)	V	X	V	X	V	V	X	X	?	AWS IoT supports TLS/SSL client authentication. There is no enough information to tell whether the supported customised authentication is compatible with MQTT API.
Singh et al. (2015)	X	V	V	V	V	V	V	X	X	Attribute-based encryption can easily support multicast encryption.
Niruntasukrat et al. (2016)	V	V	V	V	V	?	X	X	X	The scheme requires the devices to get an authorisation permission from the users during the authentication process. This requirement of user involvement during the authentication process not only increases the communication delay but also significantly increases the inconvenience of IoT applications.
Ours	V	V	V	V	V	X	X	X	V	Up to now, we do not implement secure multicast.

Authorised topic access (ATA); Device authentication (DA); MQTT session key (MSK); Fine-grained authorised topic access (FGATA); Policy management (PM); MQTT message encryption (MME); Secure multi-cast (SMC); Dynamic secure multi-cast (DSMC); Standard-MQTT-API compatible (SMAC).

5.3 The performance comparison of related works

In this section, we compare the key features of several security-enhanced MQTT systems and some popular platforms. The comparison is summarised in Table 8.

The platforms like Arduino cloud, Shiftr Cloud, and MOSCA only provide ATA. They do not provide DM, MSK, PM, FGATA, SMC, DSMC, and SMAC. If a single device is compromised, then the system cannot tell which one disclose the secrets and the attacker can gain the content of this topic.

AugMQTT (Shin et al., 2016) does not handle the policy management, the fine-grained authorised topic access, the secure multicast, and the dynamic secure multicast. Bhawiyuga et al.'s (2017) token-based solution does not provide any encryption or any protection to protect the content of a token.

The weaknesses of Mektoubi et al.'s (2016) scheme include

- a publisher is responsible for both the authentication of the subscribers and the authorisation of the request for the topic, which is a very big burden for many IoT devices
- the IoT devices need to perform the expensive PKI computations such as the signature generations and verifications for every MQTT connections.

AUPS (Rizzardi et al., 2016) and SecKit (Neisse et al., 2014) all focus on proposing a framework for the policy management and enforcement, instead of designing efficient-and-API-compatible key agreement for MQTT; These framework designs are complementary to our work.

Lesjak et al. (2015) just designed a specified hardware to help an IoT device to handle the SSL connections.

Our system provides the DM, the DA, the PM, the MME, the ATA, and the SMAC, But, it does not provide the SMC and the DSMC.

We give a short summary as follows. The basic MQTT platforms such as the Arduino cloud, the Shiftr could, and the MOCSA platform only authenticate a client, based on the shared key of the interested topic. No device authentication is considered. Even though some schemes such as Shin et al. (2016) considered the MQTT-level session key for each client, they did not consider how could it be compatible with the MQTT standard APIs. In Mektoubi et al.'s (2016) scheme, there is one certificate associated with each topic, and the public key of a topic could be used to encrypt the messages and to achieve secure multicast. But, they did not consider the relation between the dynamic membership changing and the lifetime of a multicast key. AUPS focuses on a framework to support the required functions; however, no any specific algorithms are specified to achieve the functions. AWS IoT (Amazon Web Services, https://docs.aws.amazon.com/iot/latest/developerguide/iot-security-identity.html) supports SSL client authentication, policy specification and policy enforcement; but, it does not support MQTT-level encryption and multicast. There is no enough information to tell whether AWS's customised authentication support is compatible with MQTT-APIs. Our scheme provides desirable functions of policy management, device management, key agreement, and MQTT-API compatibility. Compared to SSL/TLS approach, our approach is more efficient in terms of communication overhead, communication delay, and computational cost; our approach is also more convenient, compared to SSL/TLS approach, because our approach does not need to maintain the PKI, does not install client certificates in IoT devices (which is a challenge for many low-cost IoT device), and does not require the clients periodically verifying the CRLs.

We also note that none of the above schemes have considered the dynamic secure multicast requirement. It is one of our future works.

6 Conclusions and future work

In this paper, we have systematically examined the desirable functions for enhancing the MQTT security, and have introduced our system design and the proposed two-phase MQTT-API-compatible key agreement solution. Several experiments and an analytic evaluation have been conducted. Both the experiment results and the analytic evaluation show that our communication performance is better than the SSL version, even if the SSL connection only supports server authentication. The contributions of this paper include

- a systematic examination of the desirable functions for enhancing the MQTT systems
- a secure framework for MQTT systems
- a MQTT-API-compatible key agreement solution
- experiments showing that our CR-based authentication solution having better communication performance than the SSL solutions.

However, we also note that none of any existent platforms or enhancements consider the dynamic secure multicast, which is a very interesting future work.

Acknowledgements

This project is partially supported by the National Science Council, Taiwan, R.O.C., under grant no. MOST 107-2218-E-260-001 and MOST 108-2221-E-260-009-MY3, and Chunhua Su is supported by JSPS Kiban(B) 18H03240 and JSPS Kiban(C) 18K11298.

References

Andy, S., Rahardjo, B. and Hanindhito, B. (2017) 'Attack scenarios and security analysis of MQTT communication protocol in IoT system', *Proc. EECSI 2017*, 19–21 September, 2017, Yogyakarta, Indonesia, DOI: 10.1109/EECSI.2017.8239179.

Bethencourt, J., Sahai, A. and Waters, B. (2007) 'Ciphertext-policy attribute-based encryption', *Proceedings of the 2007 IEEE Symposium on Security and Privacy, ser. SP '07*, Washington, DC, USA, pp.321–334.

Bhawiyuga, A., Data, M. and Warda, A. (2017) 'Architectural design of token based authentication of MQTT protocol in constrained IoT device', *2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA)*, 26–27 October, 2017, Lombok, Indonesia, DOI: 10.1109/TSSA.2017.8272933.

Chien, H-Y. and Chen, Y-J. (2018) 'Security evaluation on various Arduino-compatible IoT devices', *CISC2018*, 24–25 May, Taipei.

Espinosa-Aranda, J.L., Vallez, N., Sanchez-Bueno, C., Aguado-Araujo, D., Bueno, G. and Deniz, O. (2015) 'Pulga, a tiny open-source MQTT broker for flexible and secure IoT deployments', *2015 IEEE Conference on Communications and Network Security (CNS)*, 28–30 September 2015, Florence, Italy, DOI: 10.1109/CNS.2015.7346889.

Firdous, S.N., Baig, Z., Valli, C. and Ibrahim, A. (2017) '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)*, DOI: 10.1109/iThings-GreenCom-CPSCom-SmartData.2017.115.

Goyal, V., Pandey, O., Sahai, A. and Waters, B. (2006) 'Attribute-based encryption for fine-grained access control of encrypted data', *Proceedings of the 13th ACM Conference on Computer and Communications Security, ser. CCS '06*, Alexandria, VA, USA, 30 October–3 November, 2006, pp.89–98, DOI: 10.1145/1180405.1180418.

Hammer-Lahav, E. (2010) *The OAuth 1.0 Protocol*, IETF Request for Comments, April, https://tools.ietf.org/html/rfc5849

Lesjak, C., Hein, D., Hofmann, M., Maritsch, M., Aldrian, A., Priller, P., Ebner, T., Ruprechter, T. and Pregartne, G. (2015) 'Securing smart maintenance services: hardware-security and TLS for MQTT', *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, 22–24 July, 2015, Cambridge, UK, DOI: 10.1109/INDIN.2015.7281913.

Locke, D. (2010) *MQ Telemetry Transport (MQTT) V3.1 Protocol Specification*, IBM DeveloperWorks Technical Library, August 2010, http://www.ibm.com/developerworks/webservices/library/ws-mqtt/index.html

Mektoubi, A., Lalaoui, H., Belhadaoui, H., Rifi, M. and Zakari, A. (2016) '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, DOI: 10.1109/SYSCO.2016.7831326.

Neisse, R., Steri, G. and Baldini, G. (2014) 'Enforcement of security policy rules for the internet of things', *2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, October, IEEE, pp.165–172, DOI: 10.1109/WiMOB.2014.6962166.

Niruntasukrat, A., Issariyapat, C., Pongpaibool, P., Meesublak, K., Aiumsupucgul, P. and Panya, A. (2016) 'Authorization mechanism for MQTT-based internet of things', *2016 IEEE International Conference on Communications Workshops (ICC)*, May, IEEE, pp.290–295.

Rizzardi, A., Sicari, S., Miorandi, D. and Coen-Porisini, A.O. (2016) 'AUPS: an open source authenticated publish/subscribe system for the internet of things', *Information Systems*, Vol. 62, pp.29–41.

Shin, S.H. and Kobara, K. (2012) *Efficient Augmented Password-Only Authentication and Key Exchange for IKEv2*, IETF RFC 6628, Experimental, June, Available at https://tools.ietf.org/rfc/rfc6628.txt

Shin, S-H., Kobara, K., Chuang, C-C. and Huang, W-C. (2016) '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, 17–19 October, 2016, DOI: 10.1109/CNS.2016.7860532.

Singh, M., Rajan, M.A., Shivraj, V.L. and Balamuralidhar, P. (2015) 'Secure MQTT for internet of things (IoT)', *In 2015 Fifth International Conference on Communication Systems and Network Technologies*, IEEE, Gwalior, India, 4–6 April 2015, pp.746–751.

Sinha, R., Srivastava, H.K. and Gupta, S. (2013) 'Performance based comparison study of RSA and elliptic curve cryptography', *International Journal of Scientific & Engineering Research*, Vol. 4, No. 5, May, pp.720–725.

Tschofenig, H. and Pegourie-Gonnard, M. (2015) 'Performance investigations', *IETF proceeding92*, ARM, 15 March, 2015, https://www.ietf.org/proceedings/92/slides/slides-92-lwig-3.pdf, access 2019/06/05.

Tschofenig, H. and Pegourie-Gonnard, M. (2015) 'Performance investigations', *IETF Proceeding92*, ARM, 15 March, 2015, https://www.ietf.org/proceedings/92/slides/slides-92-lwig-3.pdf (Accessed 5 June, 2019).

Wang, X., Zhang, J., Schooler, E. and Ion, M. (2014) 'Performance evaluation of attribute-based encryption: toward data privacy in the IoT', *2014 IEEE International Conference on Communications (ICC)*, Sydney, NSW, Australia, June, pp.725–730.

Websites

Arduino UNO Wifi, https://www.arduino.cc/en/Guide/ArduinoUnoWiFi, 2018/04/07 access.

Arduino MKR1000, https://www.arduino.cc/en/Main/ArduinoMKR1000?s_tact=C3970CMW, 2018/04/07 access.

Avast, "Avast research finds at least 32,000 smart homes and businesses at risk of leaking data", https://press.avast.com/avast-research-finds-at-least-32000-smart-homes-and-businesses-at-risk-of-leaking-data, 2018/11/07 access.

Amazon Web Services, "Security and Identity for AWS IoT", https://docs.aws.amazon.com/iot/latest/developerguide/iot-security-identity.html, 2019/1/17 access.

Mirai (malware) – Wikipedia, https://en.wikipedia.org/wiki/Mirai_(malware), 2018/04/07 access.

CoAP – Constrained Application Protocol|Overview, http://coap.technology/, 2018/11/07 access.

DDS Portal – Data Distribution Services – Object Management Group, https://www.omgwiki.org/dds/, 2018/11/07 access.

ISO/IEC 20922:2016, Information technology – Message Queuing Telemetry Transport (MQTT) v3.1.1, https://www.iso.org/standard/69466.html, 2018/11/07 access.

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

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

Narens, TLS and Mutual TLS handshake data overhead, http://narendrasharma.blogspot.com/2018/01/tls-and-mutual-tls-handshake-data.html, Access 2019/06/1.

Internet Engineering Task Force, RFC 8446 – The Transport Layer Security (TLS) Protocol Version 1.3, https://tools.ietf.org/html/rfc8446, Access 2019/06/1.

netsekure rng, TLS overhead, http://netsekure.org/2010/03/tls-overhead/, Access 2019/06/05.

Arduino project, https://www.arduino.cc/, 2018/04/07 access.

Raspberry pi, https://www.raspberrypi.org/, 2018/04/07 access.

WeMos D1, https://wiki.wemos.cc/products:d1:d1_mini, 2018/04/07 access.

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

AMQP: Home, https://www.amqp.org/, 2018/11/07 access.

XMPP|About XMPP, https://xmpp.org/about/, 2018/11/07 access.

Arduino cloud, https://cloud.arduino.cc/, 2018/11/07 access.

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

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

Introducing JSON, https://www.json.org/, 2018/11/07 access.

NODE.JS, http://www.debugrun.com/a/cZomeQJ.html/, 2018/11/07 access.