electronics

MDPI

Article

A Novel MQTT 5.0-Based Over-the-Air Updating Architecture Facilitating Stronger Security

Hung-Yu Chien * and Nian-Zu Wang

Department of Information Management, National Chi-Nan University, Nantou County 54561, Taiwan
* Correspondence: hychien@mail.ncnu.edu.tw

Abstract: Over-the-air (OTA) updating is a critical mechanism for secure internet of things (IoT) systems for remotely updating the firmware (or keys) of IoT devices. Message queue telemetry transport (MQTT) is a very popular internet of things (IoT) communication protocol globally. Therefore, MQTT also becomes popular in facilitating the OTA mechanism in many IoT platforms, such as the Amazon IoT platform. In these IoT platforms, the MQTT broker acts as the message broker and as an OTA server simultaneously; in these broker-based OTA architectures, it is quite common that an IoT application manager not only uploads the new firmware/software to the broker but also delegates his signing authority on the firmware/software to the same broker. If the broker is secure and trusted, this OTA model works well; however, it incurs lots of security concerns if the broker is not fully trusted or if it is curious. Many MQTT deployments do not own their own brokers, but rely on a third-party broker, which sometimes is a freeware program or is maintained by a curious third party. Therefore, a secure OTA process should protect privacy against these brokers. This paper designs a novel MQTT-based OTA model in which an IoT application manager can fully control the OTA process through an end-to-end (E2E) channel. We design the model using MQTT 5.0's new features and functions. The analysis shows that the new model greatly enhances security and privacy properties while maintaining high efficiency.

Keywords: internet of things (IoT); MQTT; over-the-air; privacy; security; Amazon; mobile payment; end-to-end security; enhanced authentication

check for updates

Citation: Chien, H.-Y.; Wang, N.-Z. A Novel MQTT 5.0-Based Over-the-Air Updating Architecture Facilitating Stronger Security. *Electronics* **2022**, *11*, 3899. https://doi.org/10.3390/electronics11233899

Academic Editor: Esteban Tlelo-Cuautle

Received: 25 October 2022
Accepted: 22 November 2022
Published: 25 November 2022

1. Introduction

Conventionally, to upgrade the firmware/software of computers, technicians either physically go to the sites or recall the computers to facilitate the jobs; both ways are inefficient and costly. As the industries need to renew firmware/software more often these days in order to patch security weaknesses or to upgrade new functions, the OTA mechanism has become very popular for replacing the conventional firmware/software update approach. In the OTA approach [1–3], an application manager remotely upgrades the firmware/software. The OTA mechanism is more crucial for the IoT systems because there are lots of devices deployed remotely in IoT scenarios, and these resource-limited devices are commonly deployed in non-protected areas.

For IoT systems, the merits of adopting an OTA mechanism contributes several key aspects: (1) it lowers the cost in terms of time, money, and efforts; (2) it facilitates the easy upgrade of new functions; (3) it enables remote securing of patches in a large-scale manner.

MQTT [4,5] is a very popular IoT communication protocol and many well-known IoT platforms (for example, the Amazon platform [6,7]) support MQTT. A MQTT system consists of three kinds of entities: a broker, several publishers, and several subscribers, where the broker forwards the messages from a publisher to a set of subscribers. Each MQTT message is associated with a specific topic, and the message is forwarded to the subscribers which have subscribed the same topic. The standard organization has released the newest MQTT standard, MQTT 5.0, in 2019. Several new features (for example, enhanced authentication [8], user properties [9], and request-response [10]) are included in MQTT 5.0 to extend its flexibility and security support.

Many of the MQTT platforms also support OTA services for their users to remotely upgrade their software/firmware; Amazon is one of them. In these platforms, the MQTT brokers not only act as message brokers, but also serve as the servers for the OTA mechanism; when they serve as OTA servers, an application manager can upload his new firmware/software to the platforms and the platforms notify the IoT devices to download the new firmware/software. In an OTA mechanism, the uploaded firmware/software should be signed such that a device can verify the authenticity and integrity of the software/firmware. However, to ease the job of the manager, many platforms also offer delegation functions, in which the manager can delegate his signing authority to the platforms. This model eases the job of the manager, but it also incurs big security and privacy concerns. If the platform is not fully trusted and secure, the signing delegation could be misused, and the privacy of the new firmware/software/keys could be tampered. Many MQTT-based IoT systems do not have their own brokers but rely on the MQTT services of some third party; these third-party brokers might be compromised or they may be curious about the contents of their users. Amazon AWS is one of these third parties.

Therefore, this paper designs a new MQTT-based OTA model in which the application manager could securely control the OTA mechanism. We design an end-to-end (E2E) channel for the publisher–subscriber interaction using MQTT 5.0. Based on the E2E channel, we design an OTA scheme in which the manager securely renews the firmware/software/keys of the IoT devices remotely. The analysis shows that the new model greatly enhances security and privacy while maintaining high efficiency.

The contributions of this paper include (1) an E2E key agreement customized for the OTA update, (2) a new OTA architecture which protects privacy against curious brokers, and (3) both the E2E key agreement and the OTA flow are fully compatible with the newest MQTT 5.0 standards.

The rest of this article is organized as follows. Section 2 introduces the related work. Section 3 introduces some new features of MQTT 5.0 and the conventional MQTT-based OTA model. Section 4 introduces our design. Section 5 evaluates the performance, analyzes the security, and discusses two future works. Section 6 states our conclusions.

2. Related Work

MQTT has gained its popularity, owing to its simplicity, efficiency, and ease of use. It has been widely deployed in many IoT systems globally. These popular deployments also attract both industry and academia concern. The weaknesses of the precedent standards are as follows: MQTT 3.1 [4] and its earlier versions only support using account and password as their authentication support, and do not provide any encryption by themselves; they assume the users would enable SSL/TLS in the underlying layers to protect the privacy of the transmission. In such deployments, two TLS/SSL links are separately built: one is for the publisher–broker pair, and the other is for the broker–subscriber pair. Upon receiving the messages, the broker decrypts the messages and re-encrypts the messages for each subscriber. This way, it can glimpse at the contents and violates the privacy of the publisher–subscriber transmission. The precedent MQTT standards also do not support the conventional request–response interactions. These weaknesses/limitations could be summarized in two aspects: poor security support and poor scalability/flexibility. The main weaknesses include its limitation of security support and its inflexibility of supporting application-oriented functions and interactions. To enhance the security support, MQTT 5.0 supports the enhanced authentication framework; to enhance its flexibility, it supports a new type of property, namely user properties, with which an IoT application can design its application-related fields and use these fields to share information among the entities (publishers, brokers, and subscribers).

There exist many works aiming at enhancing the security of MQTT 3.1 and its earlier versions. Lesjak et al. [11] integrated an IoT device with their design of TLS-capacity-embedded hardware to relieve IoT devices from TLS-connection loading. There also exist several efforts such as [12–20] on incorporating MQTT systems with the customized key agreement schemes. Many of the previous MQTT-aware key agreement schemes do not consider the compatibility with the existent MQTT application interfaces (API), for example MQTT 3.1. Chien et al. [18] proposed a two-phase authentication mechanism for any secure key agreement schemes to be compatible with MQTT 3.1 APIs. In [19,20], Chien et al. extended the conventional MQTT architecture into a group-based model and a hierarchical MQTT broker architecture to enhance the scalability and improve the latency performance of local MQTT clients, respectively.

The conventional TLS interactions between a broker and its clients allow the broker to peek at the content of the messages; therefore, Mektoubi et al. [21] adopted the asymmetric-key-based approach to achieve E2E security for MQTT systems; however, the scheme adopts the costly public-key-based encryption. Chien et al. [20] designed a modified MQTT 3.1 system in which the broker distributes a group key to all the publishers and subscribers of the same group such that group communication is supported; however, the broker can still peek at the message contents. The MQTLS scheme [22] was proposed to distribute an E2E key via a publisher in the MQTT 3.1-like systems; however, all the subscribers must issue their CONNECT requests before the publisher's CONNECT request. Otherwise, the publisher fails to issue the E2E key for that client. This requirement limits the practical application of the scheme. A MQTT 5.0-based challenge–response authentication scheme has been designed and implemented by Ciou and Chien [23], but it does not consider E2E security. In 2022, Chien [24] designed the first E2E security schemes using MQTT 5.0 features. Inspired by Chien's work, we propose a modified E2E security support which is customized for the OTA mechanism. Table 1 summarizes the properties of the related work.

Table 1. Summary of related work.

Properties Scheme	Target at MQTT 3.1 or 5.0	Goals & Functions	MQTT-Standard Compatible	E2E Security	OTA Support
Lesjak et al. [11]	3.1	TLS-embedded hardware	Yes	No	No
[12–17]	3.1	Customized key agreement	No	No	No
[18–20]	3.1	Customized key agreement	Yes	No	No
Lee et al. [22]	3.1	Customized key agreement	No	Yes	No
Ciou–Chien [23]	5.0	Customized key agreement	Yes	No	No
Chien [24]	5.0	Customized key agreement	Yes	Yes	No
Ours	5.0	Customized key agreement OTA scheme	Yes	Yes	Yes

3. Preliminaries of MQTT 5.0 New Features and of OTA General Architecture

Before proposing our design, we introduce some key concepts/functions/terms of MQTT 5.0 and the conventional OTA mechanisms.

3.1. New Features of MQTT 5.0

Here, we introduce the MQTT 5.0 features which are related to our design.

Enhanced authentication. The enhanced authentication framework consists of a new framework and some APIs for users to design and implement their MQTT-aware authentication schemes. New fields are introduced to specify the authentication method (we denote it as auth_id in this paper) and to share the authentication data (denoted as auth_data). A new API called AUTH, accompanied by the existent CONNECT/CONNACK API with new fields (auth_id and auth_data), is used to negotiate the specified authentication method with the authentication data auth_data. Figure 1 shows the protocol stack of the enhanced authentication framework.

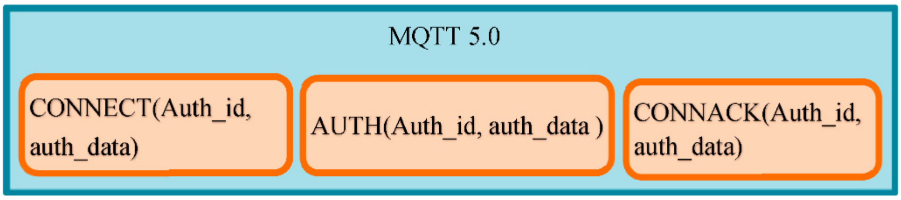
MQTT 5.0
CONNECT(Auth_id, auth_data)
AUTH(Auth_id, auth_data)
CONNACK(Auth_id, auth_data)

Figure 1. Protocol stack of MQTT 5.0-enhanced authentication.

User properties. "User properties" is a new field used to covey user-defined metadata among publishers, brokers, and subscribers. A user-defined metadata is a UTF-8 (key, value) pairs. For example, PUBLISH(topic = 'Publisher/Pub1', retain = TRUE, UserProperties = {"ClientId:Pub1", "Certificate: Pub1.cert", . . . }; the retain flag = TRUE indicates to the broker to keep this message until the next PUBLISH message with "retain = TRUE"; this setting lets the broker deliver this message to those clients which connect online later.

Request–response interaction via response topic. MQTT 5.0's "ResponseTopic" property facilitates the implementation of the conventional request–response interaction much more easily in MQTT 5.0. A PUBLISH packet or a CONNECT packet can specify ResponseTopic (an optional UTF-8 string) to notify its peers to respond their messages in the specified topic.

Correlation data. This optional binary data is accompanied with the specified response topic to synchronize their communications between a requestor and a responder.

3.2. OTA Architectures

Figure 2 shows one OTA architecture, where the Amazon cloud with the FreeRTOS [25] (an open-source IoT operation system) serves as the cloud for the MQTT messaging and for OTA updating. Other examples of the conventional OTA architectures might differ in some minor aspects. The architecture (in Figure 2) consists of the Amazon cloud and IoT devices (the hardware here is ESP32 [26], the freeRTOS OS, and the communication protocols (MQTT and http)). The Amazon cloud provides many services, such as the Amazon Web Service (AWS), the OTA services, etc. The AWS server serves as the server for managing users, devices, and the firmware/software images; MQTT acts as the protocol for sharing both the ordinary messages and the OTA messages; freeRTOS is the free open-source operation system on the IoT devices (here, the ESP32 platforms). Some key processes are explained as follows:

Stage 1. The application manager who is responsible for maintaining the firmware/software/key-updating process can create a code-signing certificate; in this case, the manager delegates his signing authority to the AWS platform which signs the files on behalf of the manager. The other option on AWS allows the user to sign the image files on his local computers and then later sends the images files and the signatures to the AWS platform.

Stage 2. The manager prepares the files to be downloaded by IoT devices, and send the files and the related meta data to the AWS server.

Stage 3. The AWS server creates an IoT–OTA job which manages the related files, sends the messages through a particular MQTT topic (we can view a MQTT topic as a particular channel here), and maintains the communications and the records.

Stage 4. A specialized software on those designated IoT devices would communicate with the AWS server to handle the OTA messages and the files via the MQTT interactions. Note that, on the left side of the figure, the CoreMQTT represents the basic MQTT capacity on the IoT devices to handle basic MQTT interactions, while the OTA update agent is the software responsible for handling the OTA tasks.

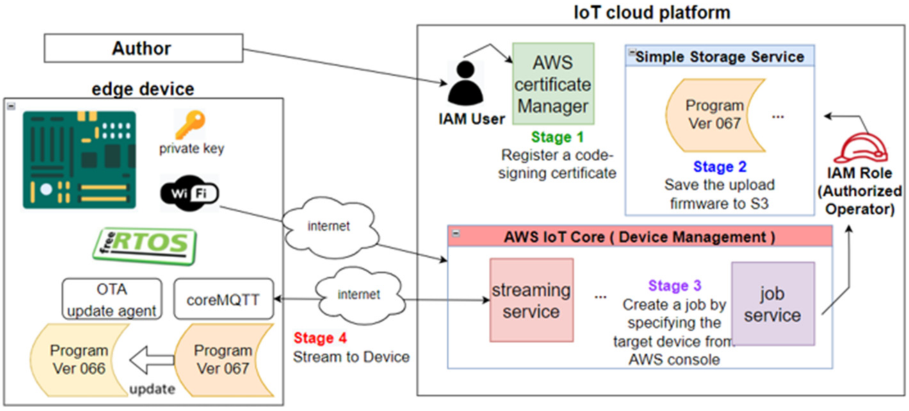

Figure 2. An IoT device OTA architecture by AWS.

4. New OTA Model Using MQTT 5.0

Our new OTA model is based on an E2E security support system using MQTT 5.0. Section 4.1 first introduces the E2E security design, and Section 4.2 proposes the new OTA model. Table 2 introduces the notations.

Table 2. The notations.

P1.Cert, S1.Cert	P1.Cert denotes Publisher P1's certificate; S1.Cert denotes S1's certificate. P1.Cert has the public key as g^a; S1.Cert has its public key as g^b. Here we eliminate the specification of the underlying fields, and any secure fields like Elliptic Curve Cryptographies could be used.
$E2E_{key}$	The End-to-End session keys between publisher-subscriber E2E session key. Here, $E2E_{key} = g^{ab}$.
$\text{Enc}_{key}[]$,$\text{Dec}_{key}[]$	Encryption/Decryption using the key *key*.

4.1. The Overview of the Model and the Topic Tree Design

Figure 3 shows the entities and the relations in the new OTA model. The device manager refers to the manager who maintains an MQTT-based IoT system. In the model, the certificate authority (CA) is responsible for issuing the certificates to all the entities, and it is trusted.

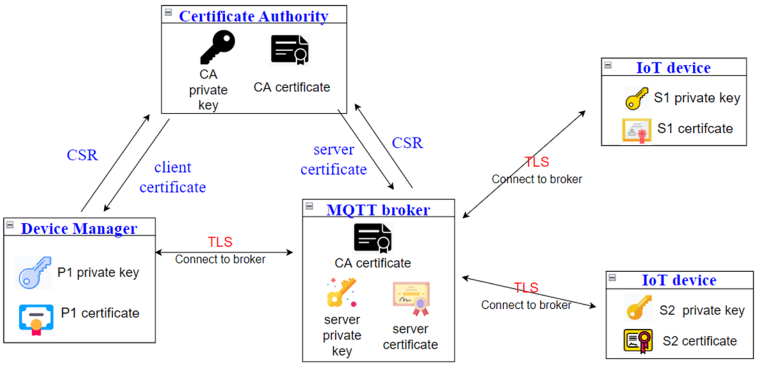

Figure 3. Entities in the new OTA model.

The whole model consists of three phases: the outer-channel establishment phase, the inner-channel establishment phase, and the OTA-update phase. In the outer-channel establishment phase, each client authenticates itself to the broker to access the MQTT services. In the inner-channel establishment phase, the manager and each client establish the $E2E_{key}$ key for encrypting the messages in the End2End channel. In the OTA update phase, the manager initiates the OTA update process, and the sensitive data in the exchanged messages is encrypted using the key $E2E_{key}$. The exchanged messages in the OTA update phase mainly consist of the new certificate, the new private key, and the related data in our example later.

It is assumed that the clients and the broker authenticate their peers using the certificates. In MQTT, a client has to authenticate itself to the broker before it accesses the forwarding services of the broker; the channel between a client and its broker is then protected by encrypting the transmissions. This channel is called an outer channel in this paper. On the contrary, an inner channel refers to the channel between two clients (a publisher and a subscriber—of course, in the MQTT model, all messages are relayed by a broker). In the model, the manager builds an inner channel between itself with each IoT device; this inner channel is built by deriving a Diffie–Hellman key based on their public keys and private keys. This inner channel facilitates the manager to securely deliver the new certificates and the new keys to the IoT devices, without the privacy threats from either the broker or the outsiders. Figure 4 shows the relations between the two channels.

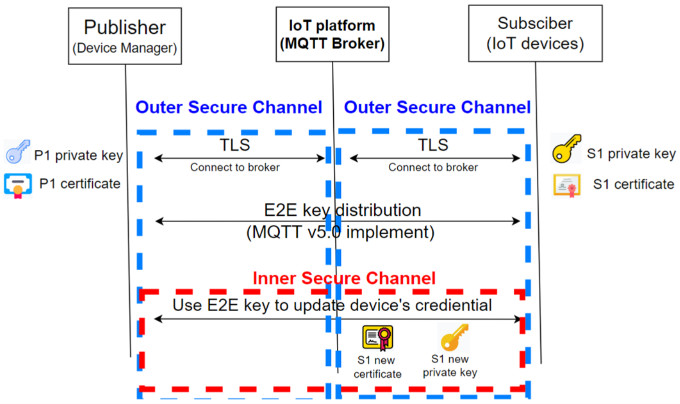

Figure 4. The two channels of the OTA model.

Now we are ready to introduce the design of the E2E channel. Before initiating the inner E2E channel, each client should authenticate itself to the broker; here, we propose two approaches for the outer channel process: one is the conventional TLS authentication above the TCP layer, and the other is to initiate an TLS authentication within MQTT 5.0's enhanced authentication. The latter approach enables the two entities (the broker and the client) to be aware of each other's certificates. Since the outer-channel building process has been well studied and it is not the topic of this paper, we skip the details here and focus on the inner-channel building process as follows.

Before depicting the inner-channel building process, we first introduce the MQTT topic tree for the process. Figure 5 shows the topic tree design. The 1st layer "$iot" denotes this application. The 2nd layer "P1" denotes the manager for this application; in this application, we assume there is only one manager. There are many subscribers (IoT devices) in this application, and they are respectively denoted as S1, S2, . . . , etc. Each device has three sub-topics in the 4th layer, and they are respectively denoted as "ready", "notify", and "update"; "ready" is for each device to respond whether it is ready for the upgrading process; "notify" is for the manager to notify the device the channel for updating; "update" channel (topic) is for the device to receive the update messages, and the "status" channel is for the device to report its status during the updating process.

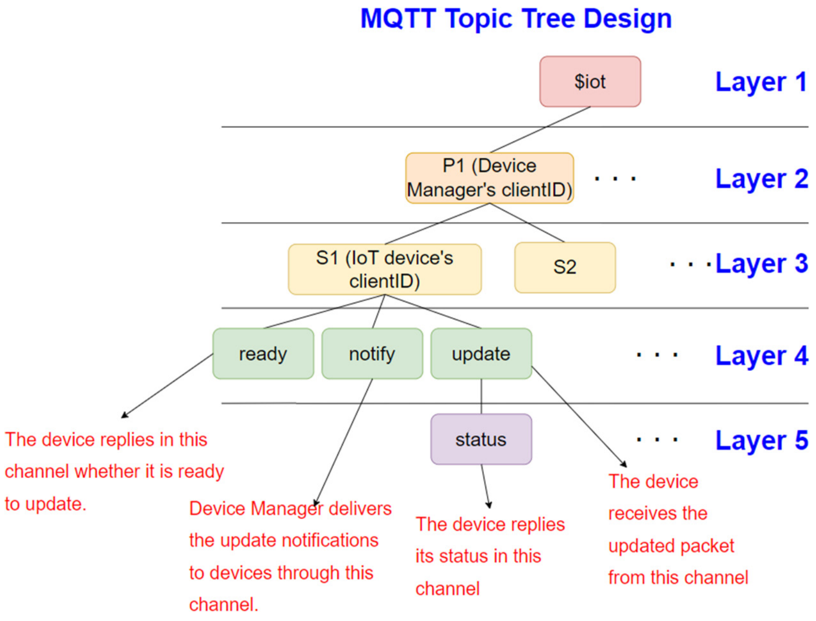
MQTT Topic Tree Design
$iot
Layer 1
P1 (Device Manager's clientID)
Layer 2
S1 (IoT device's clientID)
S2
Layer 3
ready
notify
update
Layer 4
status
Layer 5
The device replies in this channel whether it is ready to update.
Device Manager delivers the update notifications to devices through this channel.
The device replies its status in this channel
The device receives the updated packet from this channel

Figure 5. The MQTT topic tree design used in the E2E channel building.

4.2. The Message Flows of the End2End Building and the OTA Phase

Because the first phase (the outer-channel building) could be implemented using the well-studied TLS or the MQTT 5.0′ enhanced authentication with TLS, we skip the details here. Instead, we focus on the inner-channel establishment phase and the OTA update phase.

Figure 6 shows the message interactions during the E2E channel building process. In Figure 6, P1.Cert denotes the certificate of P1, and S1.Cert denotes the certificate of the device S1. $Enc_{key}[]$ and $Dec_{key}[]$, respectively, denote the encryption and decryption using the key "*key*".

Initially, the manager subscribes the channels as (topics) $iot/P1/+/ready, where the "+" denotes the devices in this application and the "+" will be replaced by each device's identity (for example, S1, S2, . . . , etc.); this subscription lets the manager receive each device's ready status. Likewise, each device (say S1) needs to subscribe to the channel "$iot/P1/S1/notify" in which the manager will notify the device with the update instructions. The instructions include the manager's certificate and digital signatures; please note that the instruction message would set the "retain" field equal to "true" such that any later joining subscribers can still receive the manager's certificate. The message also contains the ResponseTopic = "$iot/P1/S1/ready", and it notifies the device to send its ready status in this channel. In this channel, the device also sends its certificate S1.Cert and notifies the manager the ResponseTopic = "$iot/P1/S1/update". The channel "$iot/P1/S1/update" is for the manager to send the updated certificate and the updated private key.

The key mechanism for the manager securely delivering the certificate and the private key is by encrypting the messages using the E2Ekey; the E2Ekey is the Diffie–Hellman key derived from the two entities' public/private key pairs.

In this example (shown in Figure 7), we assume the private key delivery is the sensitive data that need to be protected from both the broker and the attackers. Before delivering the OTA update messages, both the manager and the device (say S1) need to subscribe the channels "$iot/P1/S1/update" and "$iot/P1/S1/update/status" in order to properly receive the messages.

The inner-channel establishment phase

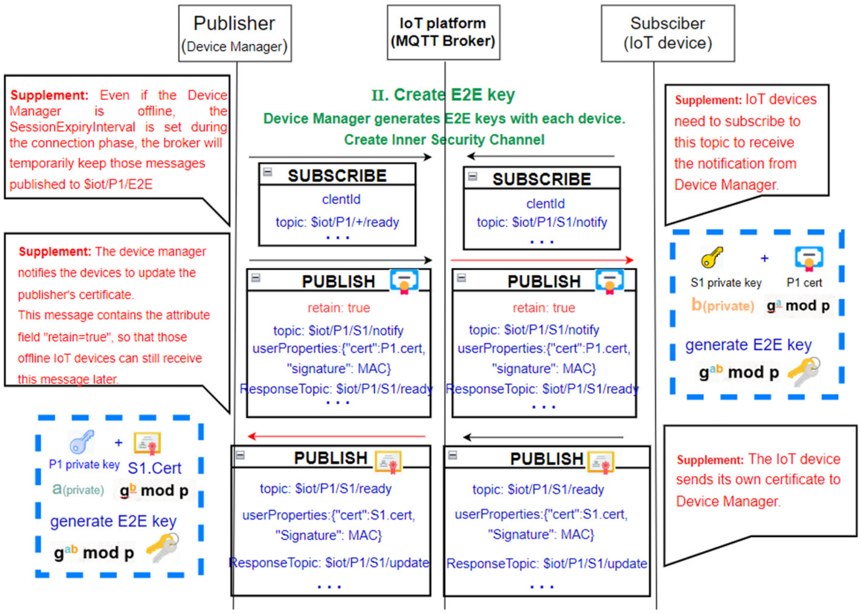
Publisher
(Device Manager)
IoT platform
(MQTT Broker)
Subsciber
(IoT device)
II. Create E2E key
Device Manager generates E2E keys with each device.
Create Inner Security Channel
Supplement: Even if the Device Manager is offline, the SessionExpiryInterval is set during the connection phase, the broker will temporarily keep those messages published to $iot/P1/E2E
Supplement: The device manager notifies the devices to update the publisher's certificate.
This message contains the attribute field "retain=true", so that those offline IoT devices can still receive this message later.
Supplement: IoT devices need to subscribe to this topic to receive the notification from Device Manager.
Supplement: The IoT device sends its own certificate to Device Manager.
SUBSCRIBE
clientId
topic: $iot/P1/+/ready
SUBSCRIBE
clientId
topic: $iot/P1/S1/notify
PUBLISH
retain: true
topic: $iot/P1/S1/notify
userProperties:{"cert":P1.cert,
"signature": MAC}
ResponseTopic: $iot/P1/S1/ready
PUBLISH
retain: true
topic: $iot/P1/S1/notify
userProperties:{"cert":P1.cert,
"signature": MAC}
ResponseTopic: $iot/P1/S1/ready
PUBLISH
topic: $iot/P1/S1/ready
userProperties:{"cert":S1.cert,
"Signature": MAC}
ResponseTopic: $iot/P1/S1/update
PUBLISH
topic: $iot/P1/S1/ready
userProperties:{"cert":S1.cert,
"Signature": MAC}
ResponseTopic: $iot/P1/S1/update
S1 private key
P1 cert
b(private)
g^a mod p
generate E2E key
g^ab mod p
P1 private key
S1.Cert
a(private)
g^b mod p
generate E2E key
g^ab mod p

Figure 6. The message interactions during the End2End channel building.

The manager sends (publishes) the new certificate "S1.new_cert", the encrypted private key file $Enc_{E2E_{key}}[private\ key\ file]$, and the ResponseTopic in the inner channel. This arrangement ensures that only the designated device can properly receive the certificate and the private key. During delivery, it might require several MQTT packets. The device then reports its status in the "$iot/P1/S1/update/status" channel. During the process, both the manager and the devices might need to contact the CA when it is necessary.

The OTA update phase

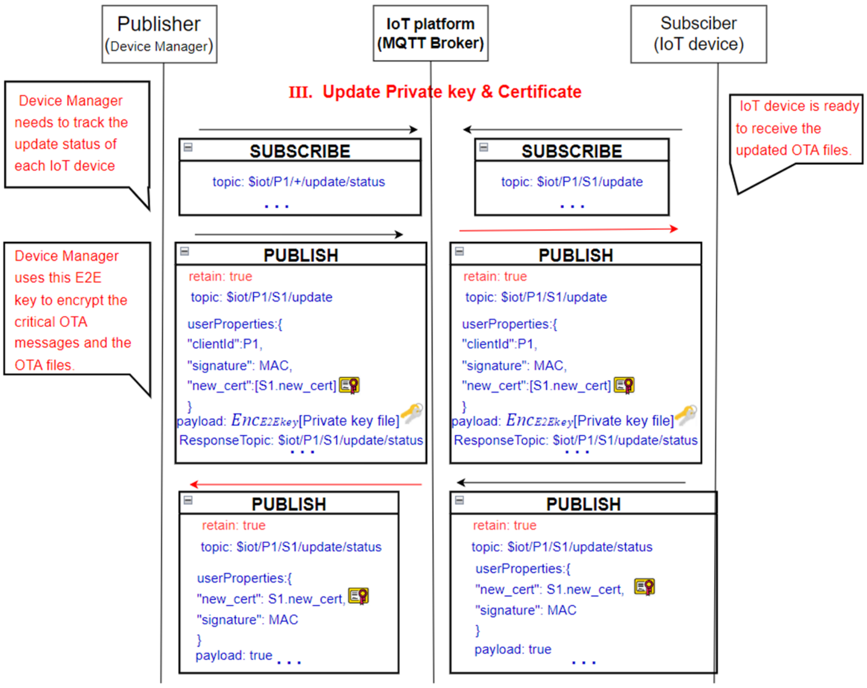

Figure 7. The message flows of the OTA update phase.

5. Security Analysis, Performance Evaluation, and Future Works

5.1. Analysis of the Security

We now examine the security of the proposed scheme. Both the Amazon AWS and our scheme are based on the modular design approach: the two schemes all adopt the well-studied security algorithms (such as ECDSA, ECDH, and so on) to design the system. If the implementations are securely ensured, then the security of the two schemes can be ensured by a modular analysis. Our model consists of four kinds of entities: publishers, subscribers, the broker, and attackers. The broker is assumed to be honest but curious; that is, the broker follows the protocols but is curious about the content. The security model here is to examine the authenticity/integrity of the OTA update messages (files), and the privacy of the OTA update keys against both the attackers and the broker. On the contrary, the conventional models (for example, the Amazon AWS model) assume the broker is fully trusted; however, we should note that the Amazon AWS can easily peek at the MQTT contents of their users. Now, we analyze the modular design of our scheme as follows.

The authenticity/integrity/privacy of the normal MQTT messages via the TLS outer channel. Our scheme is designed in a modular way, and the security modules consist of three parts. The first part is the outer-channel, built by the conventional TLS connection between a client and the broker. The outer-channel is built to ensure the authenticity/privacy/integrity of the messages between a client and the broker. As long as the certificates are properly maintained, the outer channel (built through the classic TLS process) ensures the authenticity of the client and the broker and the privacy/integrity of the transmissions.

The normal messages from a publisher to a subscriber (via the broker) are protected by two outer channels (one from the publisher to the broker and the other from the broker to the subscriber); as long as the two outer channels are secure and the broker is honest, it ensures the security (authenticity, integrity, and privacy) of the transmissions from a publisher to a subscriber; the privacy is protected from the attackers. The security is based on the TLS/SSL implementation.

The authenticity/integrity/privacy of the OTA update files/messages against the attackers via the inner channel. The second module is the inner-channel, built between a publisher and a subscriber, and the E2E key is used to encrypt the critical OTA messages and the OTA files. The E2E key is derived from the Diffie–Hellman key based on the publisher–subscriber's public/private key pairs; this ensures the privacy of the key from both the attackers and the broker. Therefore, it ensures the authenticity/integrity/privacy of the critical OTA update files/messages. The security is based on the computational Diffie–Hellman problem and the implementations.

The privacy of the OTA update keys to the broker via the inner channel. The $E2E_{key} = g^{ab}$ is the Diffie–Hellman key derived from a publisher key pair (g^a, a) and a subscriber's key pair (g^b, b). As long as the Diffie–Hellman key derived from both clients' certificates is secure, the security of the $E2E_{key}$ is secured; this ensures the privacy of the key from both the attackers and the broker. The E2E key is used to encrypt the OTA private key transmission in our example. It ensures the authenticity/integrity/privacy of the OTA private key transmissions against both the attackers and the broker.

5.2. Evaluation of the Performance

The scheme consists of three phases: the outer-channel establishment phase, the inner-channel establishment phase, and the OTA update phase. The first phase is a TLS client–broker channel establishment phase, which is the same as the classic MQTT connection phase using TLS; therefore, we will not compare the performance of this phase.

Now we examine the second phase. For each publisher–subscriber pair, the second phase takes six MQTT interactions of which two are the subscriptions, two are the publish interactions for delivering the publisher's certificate, and the other two are the publish messages from the subscriber for distributing the certificate. Among the transmissions, only the certificate demands larger overhead (amounting to 2~4 K bytes); luckily, there is some research on small-size IoT certificates. Because the AWS system does not support the end-to-end channel building, there is no corresponding process in the AWS system. In addition to the encryption of the transmissions, the main computation in this phase of our scheme is the computation of the Diffie–Hellman key, which incurs only one modular exponentiation.

Now we examine the 3rd phase. The OTA update phase of our scheme takes 6 MQTT interactions, of which two are the subscriptions, two are the publish interactions for delivering the certificate and the encrypted private key, and the other two are the publish interactions for the subscriber to report its status. The subscription interactions and the publish interactions from the subscriber are simple and would only take few bytes. Only the publish interaction from the publisher to deliver the certificate would take few kilobytes. Therefore, it is very efficient. The third phase of the AWS system requires much more MQTT interactions; however, we should note that this is because the AWS system takes more job controls in its process. We also note that more transmissions mean more energy consumption, which is crucial for resource-limited IoT devices.

Table 3a summarizes the security performance of the proposed schemes and of the AWS system. The merits of our scheme include (1) an E2E key agreement customized for the OTA update, (2) a new OTA architecture which protects privacy against curious brokers, (3) both the E2E key agreement and the OTA flow are fully compatible with the newest MQTT 5.0 standards.

Table 3. The performance comparison.

(a) The security performance comparison

Properties Scheme	E2E security	Which entity distributes E2E key	Privacy support of firmware update against the broker
Ours	Yes	Publisher	Yes
AWS	No	NO	No

(b) The communication performance comparison of the proposed scheme

Properties Scheme	Number of MQTT interactions in the second phase	Number of MQTT interactions in the third phase
Ours	6	6
AWS	Not applicable [1]	15 [2,3]

(c) The computation performance comparison of the proposed scheme

Properties Scheme	Computations of MQTT interactions in the second phase	Computations of MQTT interactions in the third phase
Ours	6 encryptions + 1 modular exponentiation	6 encryptions
AWS	Not applicable [4]	15 encryption [5]

[1] Amazon does not provide the E2E security support. [2] According to the information on the AWS website, we think the interactions are similar to those of ours, except that the publisher in AWS deposits its certificates and some users' private keys on the cloud and the cloud delivers the data to the client on behalf of the publisher. [3] From the table, our scheme demands three less messages compared to the AWS system. Assume each message takes the same length in both schemes; this means that the total transmission overhead of our scheme is less than that of the AWS system. However, we should note that the AWS system takes care of more job control tasks, and this comparison is only for obtaining a rough idea of the communication overhead. [4] Amazon does not provide the E2E security support. [5] For each MQTT interaction, the device needs to perform one encryption for the outer-channel transmission.

Table 3b summarizes the communication performance comparison. From the table, our scheme demands three less messages compared to the AWS system. However, we should note that the AWS system takes care of more job control tasks, and this comparison is only for obtaining a rough idea of the communication overhead.

5.3. Discussion of Future Works

Based on the customized E2E security support and the novel OTA scheme, we find several interesting further works. Regarding the E2E security support, we are interested in one promising improvement. In the current design, the sensitive data would be encrypted twice: one is by the outer-channel protection between a client and the broker and the other is the inner-channel protection. Therefore, it is desirable that the outer-channel provides only the data integrity protection but not the privacy protection (the inner-channel has already provided the privacy protection). With this arrangement, the sensitive data would be encrypted only once, potentially reducing computation overhead further. However, the designs and implementations of the existent outer-channels have all included both the integrity protection and the privacy protection. To integrate an integrity-protect-only outer-channel with an E2E inner-channel, we need to further study the compatibility challenges with the existent classic models/standards.

Another interesting future work is applying the new E2E security support and the new OTA mechanism in some potential applications; mobile payment is one of such applications. Some authors [27] have provided a comprehensive survey of the key components and the security challenges of mobile payment systems. As our E2E security support can build a secure E2E channel between two clients via the broker, it is interesting that one new mobile payment system can be developed as follows. Two mobile clients (say two remote users) can rapidly build their E2E channel via a MQTT broker, and perform their secure transactions in the secure inner channel; during the process, the broker can examine the necessary data fields (like the payment amounts) but not some confidential fields. With this new approach, the two clients who perform the transactions can be extended from the conventional mobile-client–bank model to a much more general mobile-client–broker(bank)–mobile-client model.

6. Conclusions

In this paper, we have designed end-to-end security support for the MQTT 5.0 system and a novel OTA update model. In the model, the publisher (the IoT application manager) can securely control the transmissions of the OTA update data (for example, the updated firmware, the certificate, and the private key). With the end-to-end security, the privacy of the transmissions is protected against a curious broker. Some popular MQTT services (such as the Amazon service) do not support end-to-end security support, and a curious broker could peek at the contents of their clients. With the new design of the OTA mechanisms and the E2E key agreement, our new model protects privacy against curious brokers. This new model and scheme could be very attractive because a MQTT application manager could securely co-operate with any public MQTT 5.0 platforms, and would not need to worry about privacy issues. The performance analysis shows that the proposed model out-performs the existent models in terms of security and privacy, and the number of MQTT interactions are quite simple and efficient. All these excellent features make the proposed model much more attractive than the conventional models.

It will be interesting to extend the model or to apply the model in other applications in the future. One possible extension is to eliminate the encryption function from the outer channel and let the inner channel fully take the privacy protection responsibility. Another future work is to apply our model to facilitate secure transactions between two clients via the help of a broker.

Author Contributions: Conceptualization, H.-Y.C. and N.-Z.W.; methodology, H.-Y.C. and N.-Z.W.; validation, H.-Y.C.; formal analysis, H.-Y.C.; investigation, H.-Y.C. and N.-Z.W.; resources, H.-Y.C. and N.-Z.W.; writing—original draft preparation, H.-Y.C.; writing—review and editing, H.-Y.C.; visualization, N.-Z.W.; supervision, H.-Y.C.; project administration, H.-Y.C.; funding acquisition, H.-Y.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Ministry of Science and Technology, Taiwan, R.O.C. grant number MOST 111-2221-E-260-009-MY3.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hamdy, K. Over-the-Air (OTA) Updates: What Is It and How to Do It Simply, Efficiently with ZDM. Available online: https://itskarim.medium.com/over-the-air-ota-updates-what-is-it-and-how-to-do-it-simply-efficiently-with-zdm-db613ea29678 (accessed on 30 August 2022).
2. Afaneh, M. Implementing Over-the-Air Device Firmware Update (OTA DFU)—Part 1. Available online: https://www.novelbits.io/ota-device-firmware-update-part-1/ (accessed on 30 August 2022).
3. Wikipedia. Over-the-Air Programming. Available online: https://en.wikipedia.org/wiki/Over-the-air_programming (accessed on 30 August 2022).
4. ISO/IEC 20922:2016, Information Technology—Message Queuing Telemetry Transport (MQTT) v3.1.1. Available online: https://www.iso.org/standard/69466.html (accessed on 25 March 2022).
5. OASIS, MQTT Version 5.0. 7 March 2019. Available online: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html (accessed on 1 April 2022).
6. Amazon. How to Perform Secondary Processor Over-the-Air Updates with FreeRTOS. Available online: https://aws.amazon.com/tw/blogs/iot/how-to-perform-secondary-processor-over-the-air-updates-with-freertos/ (accessed on 30 August 2022).
7. Amazon. AWS IoT Over-the-Air Update. Available online: https://aws.github.io/amazon-freertos/202107.00/embedded-csdk/libraries/aws/ota-for-aws-iot-embedded-sdk/docs/doxygen/output/html/ota_design.html (accessed on 30 August 2022).
8. HiveMQ Homepage. Enhanced Authentication. Available online: https://www.hivemq.com/blog/mqtt5-essentials-part11-enhanced-authentication/ (accessed on 2 April 2022).
9. HiveMQ Homepage. User Properties—MQTT 5 Essentials Part 6. Available online: https://www.hivemq.com/blog/mqtt5-essentials-part6-user-properties/ (accessed on 24 March 2022).
10. Steve Internet Guide. Understanding and Using MQTT v5 Request Response. Available online: http://www.steves-internet-guide.com/mqttv5-request-response/ (accessed on 14 March 2022).
11. Lesjak, C.; Hein, D.; Hofmann, M.; Maritsch, M.; Aldrian, A.; Priller, P.; Ebner, T.; Ruprechter, T.; Pregartne, G. Securing Smart Maintenance Services: Hardware-Security and TLS for MQTT. In Proceedings of the 2015 IEEE 13th International Conference on Industrial Informatics (INDIN), Cambridge, UK, 22–24 July 2015; pp. 1243–1250.
12. Andy, S.; Rahardjo, B.; Hanindhito, B. Attack Scenarios and Security Analysis of MQTT Communication Protocol in IoT System. In Proceedings of the EECSI 2017, Yogyakarta, Indonesia, 19–21 September 2017; pp. 19–21.
13. Firdous, S.N.; Baig, Z.; Valli, C.; Ibrahim, A. Modelling and Evaluation of Malicious Attacks against the IoT MQTT Protocol. In Proceedings of the 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, 21–23 June 2017; pp. 748–755.
14. Singh, M.; Rajan, M.A.; Shivraj, V.L.; Balamuralidhar, P. Secure MQTT for internet of things (iot). In Proceedings of the 2015 Fifth International Conference on Communication Systems and Network Technologies, Gwalior, India, 4–6 April 2015; pp. 746–751.
15. Rizzardi, A.; Sicari, S.; Miorandi, D.; Coen-Porisini, A.O. AUPS: An Open Source Authenticated Publish/Subscribe system for the Internet of Things. *Inf. Syst.* **2016**, *62*, 9–41. [CrossRef]
16. Neisse, R.; Steri, G.; Baldini, G. Enforcement of security policy rules for the internet of things. In Proceedings of the 2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), Larnaca, Cyprus, 8–10 October 2014; pp. 165–172.
17. Shin, S.H.; Kobara, K. Efficient Augmented Password-Only Authentication and Key Exchange for IKEv2. IETF RFC 6628, Experimental. June 2012. Available online: https://tools.ietf.org/rfc/rfc6628.txt (accessed on 5 February 2022).
18. Chien, H.Y.; Chen, Y.J.; Qiu, G.H.; Liao, J.F.; Hung, R.W.; Kou, X.A.; Lin, P.C.; Chiang, M.L.; Su, C.H. A MQTT-API-Compatible IoT Security-Enhanced Platform. *Int. J. Sens. Netw.* **2020**, *32*, 54–68. [CrossRef]
19. Chien, H.-Y.; Lin, P.C.; Chiang, M.L. Efficient MQTT Platform Facilitating Secure Group Communication. *J. Internet Technol.* **2020**, *21*, 1929–1940.
20. Chien, H.Y.; Qiu, G.H.; Hung, R.W.; Shih, A.T.; Su, C.H. Hierarchical MQTT with Edge Computation. In Proceedings of the 10th International Conference on Awareness Science and Technology (iCAST 2019), Morioka, Japan, 23–25 October 2019; pp. 1–5.
21. Mektoubi, A.; Lalaoui, H.; Belhadaoui, H.; Rifi, M.; Zakari, A. New approach for securing communication over MQTT protocol A comparison between RSA and Elliptic Curve. In Proceedings of the 2016 Third International Conference on Systems of Collaboration (SysCo), Casablanca, Morocco, 8–29 November 2016.
22. Lee, H.; Lim, J.; Kwon, T. MQTLS: Toward Secure MQTT Communication with an Untrusted Broker. In Proceedings of the 2019 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Korea, 16–18 October 2019; pp. 53–58.
23. Ciou, P.-P.; Chien, H.-Y. An Implementation of Challenge-Response Authentication for MQTT 5.0 IoT System. In Proceedings of the 2021 International Conference on Emerging Industry and Health Promotion (EIHP2021), Puli, Taiwan, 3–4 July 2021.
24. Chien, H.-Y. Design of End-to-End Security for MQTT 5.0. In Proceedings of the 4th International Conference on Science of Cyber Security—SciSec 2022, Matsue City, Japan, 10–12 August 2022.

25. FreeRTOS. freeRTOS Running OTA over MQTT. Available online: https://www.freertos.org/ota/ota-mqtt-agent-demo.html (accessed on 30 August 2022).
26. Last Minute Engineers Website. ESP32 Basic Over The Air (OTA) Programming in Arduino IDE. Available online: https://lastminuteengineers.com/esp32-ota-updates-arduino-ide/ (accessed on 30 August 2022).
27. Ahmed, W.; Rasool, A.; Javed, A.R.; Kumar, N.; Gadekallu, T.R.; Jalil, Z.; Kryvinska, N. Security in Next Generation Mobile Payment Systems: A Comprehensive Survey. *IEEE Access* **2021**, *9*, 115932–115950. [CrossRef]