New group-key-based Over The Air (OTA) Update Model Facilitating security and efficiency Using MQTT 5

Hung-Yu Chien1[0000-1111-2222-3333], Nian -Zu Wang 2[1111-2222-3333-4444], Yuh-Min Tseng3, Ruo-Wei Hung4

1,2 National Chi-Nan University, Taiwan, ROC
3 Department of Mathematics, National Changhua University of Education, Taiwan
4 Department of Computer Science and Information Engineering, Chaoyang University of Technology, Taiwan
hychien@mail.ncnu.edu.tw

Abstract. The booming development of Internet-of-Things (IoT) has deployed many IoT systems globally, and this trend is continuously accelerating. However, as many IoT devices are widely deployed, the system-update maintenance is a huge challenge. Over The Air (OTA) update is one promising mechanism for securely updating the firmware of the remote IoT devices.
Message Queue Telemetry Transport (MQTT) is one of the most adopted IoT communication protocols globally. It has also been popularly adopted as the communication protocol for delivering the OTA update messages, in addition to delivering normal IoT messages. This paper focuses on MQTT-based OTA models. Even though there exist several MQTT-based OTA models and schemes, we find that no one can simultaneously satisfying user convenience, efficiency and high security. Some sacrifices the privacy against the MQTT broker to achieve user convenience, and some focuses on the privacy while sacrificing the convenience. This paper sorts out the existent models and proposes a new model that distributes the group keys among the manager and the IoT devices, allows the manager deposit the group-key-encrypting firmware on the broker, and then each device can separately access the encrypted OTA images from the broker. We design the scheme using MQTT 5.0 (the new MQTT standard). The analysis and the evaluation show that the new model achieves better privacy protection and gains efficient communication performance.

Keywords: Internet of Things (IoT), MQTT, Over the Air, privacy, security, Amazon, group key.

1 INTRODUCTION

IoT systems have been widely deployed globally in many application areas, and the number of deployments is continuously increasing. However, as the number of deployed devices increases very fast, so does the challenge of maintaining these remote devices.

In these days, IoT systems need to frequently update the firmware/software to meet the fast new-function release cycle and the requirement of security patches. The mechanisms of the OTA update allow the system manager remotely update the firmware/software of devices via the communications, without recalling the devices back to the companies or personal inspect the devices on-site [1-3]. This approach not only greatly improves the efficiency but also accelerates the product life cycle.

MQTT is a very popular IoT communication protocol [4, 5]. There exist several open-source platforms (like Mosquitto [6], HiveMQ [7], Mosca [8]) and several commercial IoT platforms (for example, Amazon platform [9, 10]). MQTT model adopts the publish-then-forward approach: some devices (called publishers) publish the messages with the specified topic to a broker, and then the broker forwards the messages to those devices (called the subscribers) that have subscribed the same topic. As it is easy-to-use and very efficient, it soon becomes one of the most popular IoT communication protocols. However, the precedent MQTT standards (MQTT 3.1 [4] and its earlier versions) only support account-and-password as their authentication support, and do not provide any encryption by themselves; they assume the deployments would enable SSL/TLS in the underlying layers to protect the privacy of the transmission. The new MQTT standard called MQTT 5.0 [5] adds several new functions (including the User properties and the Enhanced Authentication framework), which greatly improve the flexibility and the security support.

As the MQTT protocol is very efficient, it has also been adopted to deliver the OTA messages, in addition to the normal IoT messages. Several MQTT platforms (like Amazon and Infineon [11]) have included this function in their services. Chien and Wang [12] recently design a new MQTT-based OTA scheme in which a publisher separately builds an End-to-End key with each of its subscribers, and then the publisher securely distributes the encrypted OTA data to these subscribers; in this arrangement, their scheme can protect the privacy against the broker; however, during the OTA update phase, the publisher is required to have a reliable connection with the broker. After surveying the existent MQTT-based OTA solutions, we find that none of the existent models could simultaneously satisfy the requirements of high privacy support, efficiency, and low publisher-burden. Therefore, this paper will propose a new MQTT-based model that our model achieves better privacy protection and gains efficient communication performance. We will leverage the new functions of MQTT 5.0 to achieve this goal.

2 RELATED WORK

Conventionally, MQTT 3.1 [4] and its precedent versions assume that the users would enable SSL/TLS to encrypt the transmission privacy. This approach has several weaknesses. First, the support of SSL/TLS is a burden for some simple IoT devices. Second, even though the transmission between a publisher and its broker and between a subscriber and its broker is encrypted, the broker can still peek at the content of its clients; that is, the SSL/TLS does not protect the clients' privacy against the broker. There exist several publications like [28] elaborating on the performance of MQTT.

In light of the observations on the weak security support, there exist many efforts to improve the security support of MQTT 3.1 and its precedent versions. These proposals could be roughly classified into several categories. One is designing special hardware (for example, Lesjak et al. [13]) to assist IoT devices handle the SSL/TLS overhead. The second category is designing some customized key agreement schemes (like [14-23]) for MQTT systems. The third category, based on the application context, defines the capacities of a IoT device; for example, [24] defines each device's capacity to be publish- only, subscribe-only, and both-publish-subscribe, according to the device's function and location.

The standard organizations also notice the weaknesses and limitations of the precedent MQTT versions, and ratify the new MQTT standard called MQTT 5.0 to enhance both the security support and the flexibility. Regarding the security, the new standard designs the Enhanced Authentication framework in which new Application Programming Interfaces (APIs) and new packet fields are proposed to facilitate users design their own authenticated key agreement schemes and the negotiation of encryption methods. Regarding the flexibility, several new features are introduced; a new packet field called "User Properties" is included to share application data between publisher-broker, between subscriber-broker, and between publisher-subscriber. These application-aware data is very useful for users to embed application-aware message in the MQTT interactions.

Ciou and Chien [25] design a Challenge-Response authentication using the Enhanced Authentication framework of MQTT 5.0; the scheme builds a secure channel between a client with its broker. Chien [26, 12] designs the first End-to-End (E2E) secure channel between a publisher and a subscriber, using the MQTT 5.0 features.. SEEMQTT [27] also concerns the end-to-end security where a publisher delegates its encryption authority to a pool of keystores, via secret sharing, so that those designated subscribers can recover the decryption key; this arrangement aims at those scenarios where it is difficult that the publishers can directly verify their subscribers and can protect the privacy against a curious broker. However, in the OTA application, the publishers which release the new firmware are usually resource-abundant devices and can directly verify their subscribers; SEEMQTT model is too complicated and not efficient enough for the OTA applications.

Amazon Web Services (AWS) provides several popular cloud-based services, and the MQTT-based IoT service is one of them. In its MQTT-based IoT service, the broker can deliver both the normal messages and the OTA messages via the MQTT interactions. In the OTA service of the AWS IoT service, an application manager prepares the new firmware, uploads the new firmware to the cloud, and creates an OTA job to take care of the OTA update process; each designated IoT device then interacts with the broker, and the OTA job on the broker is responsible for delivering the OTA messages and the firmware to the device. To ensure the authenticity and the integrity of the firmware, the manager can personally sign the firmware or delegates the authority to the cloud.

Fig. 1 shows the AWS MQTT-based OTA model. In this model, the device manager creates Things (that is, the IoT devices in the AWS services) and certificates; he uploads the new firmware; he may sign the firmware on his local computer or delegates the authority to the AWS server; he also creates an AWS IoT job on the broker to handle the OTA process. During the OTA process, the OTA update agent on the device interacts with the broker to process the OTA update. Because the broker handles the OTA process on behalf of the manager and the firmware is already on the server, this model does not require the local computer of the manager be on-line during the OTA update process. however, it has two critical weaknesses: one is the delegation of the signing authority, and the second is the lack of privacy protection against the broker (the broker can peek at the content of the firmware).

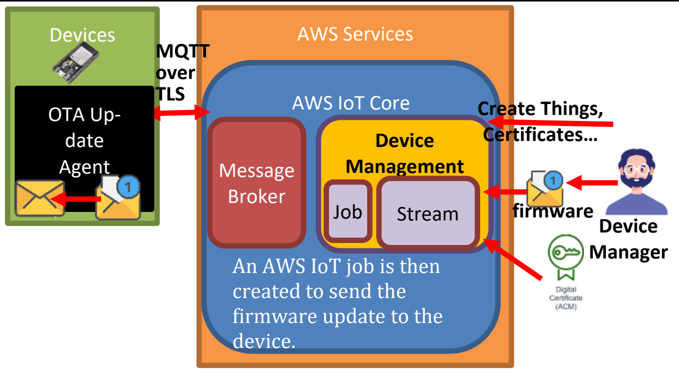
Devices
OTA Up-date Agent
MQTT over TLS
AWS Services
AWS IoT Core
Message Broker
Device Management
Job
Stream
An AWS IoT job is then created to send the firmware update to the device.
Create Things, Certificates...
firmware
Device Manager
Digital Certificate (ACM)

Figure 1. The AWS MQTT-based OTA model

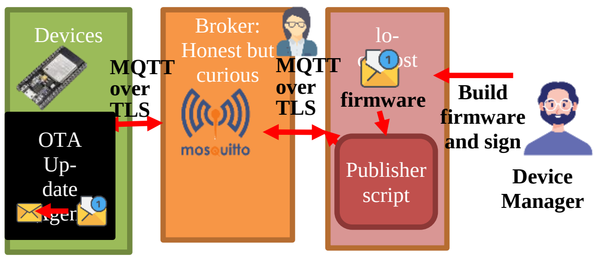

Figure 2. The Infineon MQTT-based OTA model

Infineon [11] also proposes a MQTT-based OTA model. In the model (depicted in Fig. 2), the manager builds the firmware, signs it, but keeps the firmware on the local host. On the local host, a publisher (which runs the OTA-publisher script) handles the OTA process, and interacts with the IoT devices via the MQTT broker. In the figure, we use the mosquito broker [6] as the broker. Of course, one can implement both the broker and the publisher on the same computer (that is, merging the broker and the local host); but, logically, they are separate machines, as they interact using MQTT and the manager of the broker and the device manager might belong to two different authorities. In this model, the device manager can locally handle the OTA process; however, it requires the localhost be reliably on-line during the process; the broker also can peek at the content of the firmware.

Based on the above survey and observations, we can see that, up to now, there is no one model that simultaneously satisfies all the requirements of high privacy (here it refers to the privacy against the broker), efficiency (less communication overhead during the OTA process), and less publisher-broker connection requirement (it refers to the publisher needs not to be on-line during the OTA process).

3 NEW MQTT-GROUP-KEY-BASED OTA MODEL

Our new OTA model is based on building a group key among the publisher (here it refers to the device manager's host), using MQTT 5.0. Section 3.1 first introduces the new OTA model. Section 3.2 introduces the new design in details, using MQTT 5.0. Table 1 introduces the notations.

Table 1. The notations.

M.Cert, D1.Cert	M.Cert denotes Publisher M's certificate; D1.Cert denotes D1's certificate. M.Cert has the public key as g^a; D1.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.
DH_{key}	$DH_{key} = g^{ab}$ is the Diffie-Hellman key between the publisher (the device manager) and one of its subscribers.
$group_{key}$	The group key chosen by the device manager- in this paper, we use the term the device manager to refer to the device manager (the user) and his host. The group key will be shared among the device manager and its subscribers (the IoT devices).
$Enc_{key}[]$,$Dec_{key}[]$	Encryption/Decryption using the key *key*.

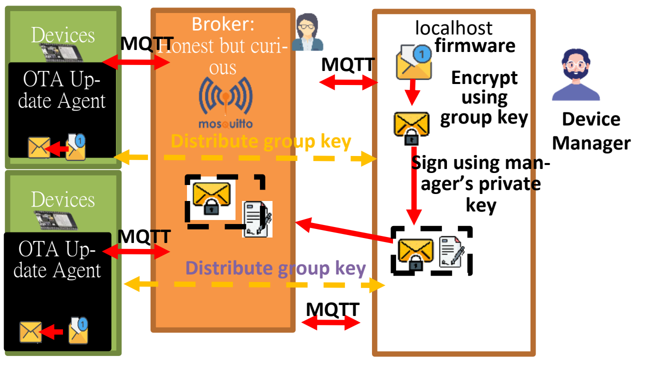

Figure 3. The group-key-based OTA model

3.1 The group-key-based OTA model

Fig. 3 depicts the new model. The new model consists of four phases: the client-broker authentication phase, the group-key distribution phase, the firmware-encrypt-sign phase, and the OTA update phase. In the client-broker authentication phase, each client and the broker mutually authenticate each other to access the MQTT services. In the group-key distribution phase, the device manager distributes the group key to all of the designated subscribers (the IoT devices). In the firmware-encrypt-sign phase, the device manager first, using the group key, encrypts the firmware, and then sign the encrypted firmware using its private key; he then uploads both the encrypted firmware and the signature to the broker. Finally, in the OTA update phase, the devices interact with the broker to access the encrypted firmware and the signature, verify the signature, and decrypt the encryption.

In this model, we note several improvements. First, the encrypted firmware is deposited on the broker such that the manager's local host is not required to be on-line when the IoT devices perform the OTA update phase. Second, the firmware is encrypted using the group key such that the broker cannot peek at the content of the firmware.

3.2 The detailed design using MQTT 5.0

Fig. 4 depicts the message interactions of the four phases.

The client-broker authentication phase

A client and the broker just perform any secure authentication schemes (for example, SSL/TLS or Chien et al.'s scheme [20]) to mutually authenticate each other.

The group-key distribution phase

Step 1(a) and 1(b): the publisher (the device manager) and the subscriber (the device) respectively subscribe the topic=OTA/M/devices and the topic=OTA/M to properly receive the expected messages later.

Step 2(a) and 2(b): The publisher publishes the message "Publish(topic=OTA/M, retain=True, Userproperties= {certificate:M.Cert,…}, ResponseTopic= OTA/M/ devices)", and the broker forwards the message to the subscribers. "M.Cert" in this message is the publisher's certificate, and "ResponseTopic= OTA/M/devices" notifies the subscribers to reply their certificates in the topic= OTA/M/devices. The field "retain=true" is used to notify the broker to keep the message until the next "retain" message replaces the old one: this mechanism facilitates the designated receivers get the message later even if they were not on-line when the retain message was published.

Step 3. The subscriber D1 subscribes the topic "OTA/M /devices/D1" to receive the group-key message for it.

Step 4(a)(b): the subscriber publishes its certificate D1.Cert to the broker in 4(a), and the broker forwards it to the publisher in 4(b).

When the publisher and the subscriber get the two certificates from each other, they can compute the Diffie-Hellman key 〚DH〛_key=g^ab.

Step 5(a)(b): the publisher chooses the group key, uses the 〚DH〛_key to encrypt the group key, and publishes the encryption to the broker; the broker forwards the encryption to the subscriber which decrypts the encryption to get the group key. This completes the group-key distribution phase.

The firmware-encrypt-sign phase

The publisher prepares the new firmware, encrypts the firmware using the group key, and signs on the encrypted firmware using its private key. He then uploads the encrypted firmware and the signature to the broker.

The OTA update phase

All the designated IoT devices (the subscribers) will get the notification from the broker, and directly get the OTA data from the broker.

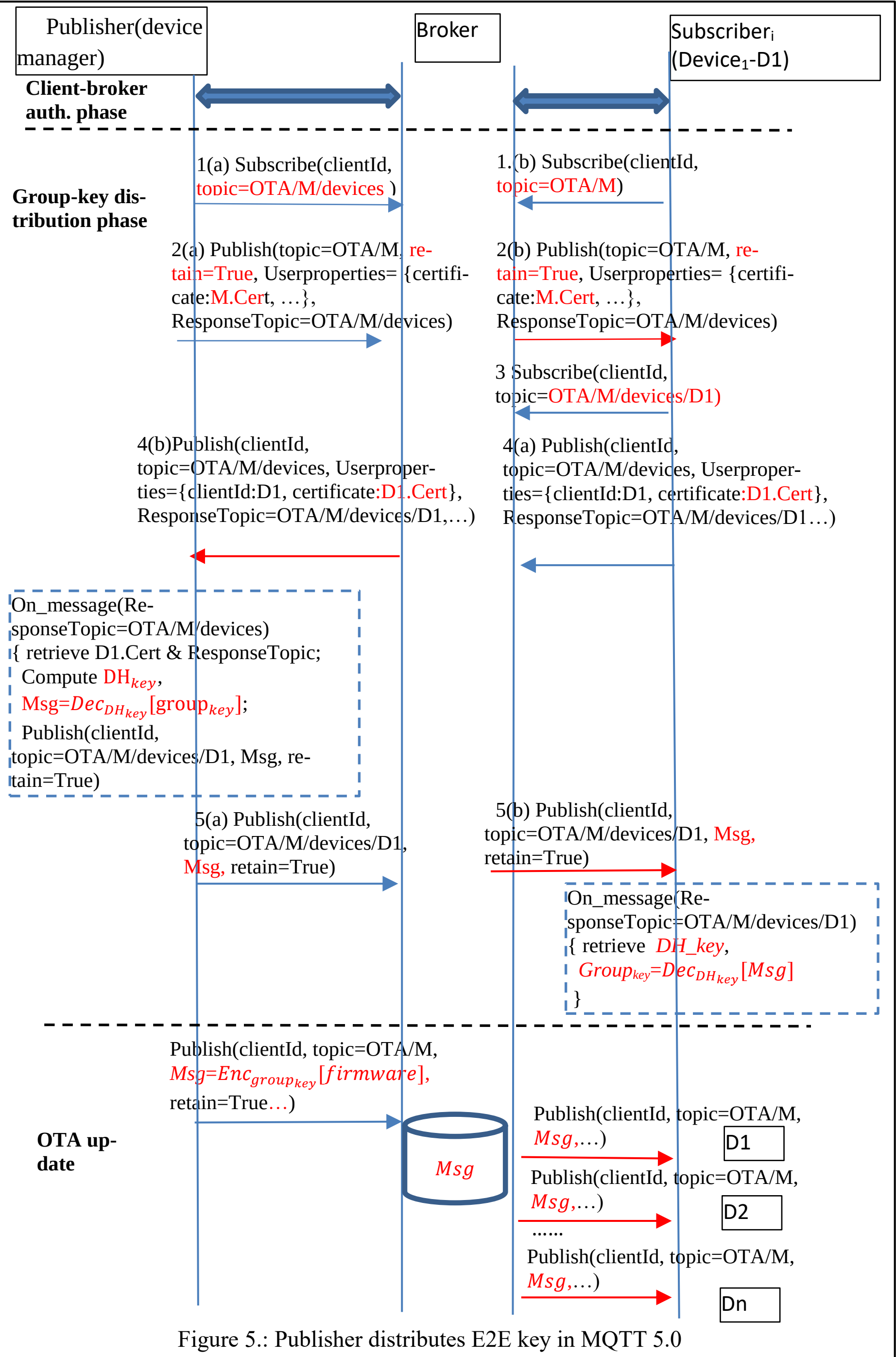
Publisher(device manager)
Broker
Subscriber$_i$ (Device$_1$-D1)
Client-broker auth. phase
Group-key distribution phase
1(a) Subscribe(clientId, topic=OTA/M/devices)
1.(b) Subscribe(clientId, topic=OTA/M)
2(a) Publish(topic=OTA/M, retain=True, Userproperties= {certificate:M.Cert, …}, ResponseTopic=OTA/M/devices)
2(b) Publish(topic=OTA/M, retain=True, Userproperties= {certificate:M.Cert, …}, ResponseTopic=OTA/M/devices)
3 Subscribe(clientId, topic=OTA/M/devices/D1)
4(b)Publish(clientId, topic=OTA/M/devices, Userproperties={clientId:D1, certificate:D1.Cert}, ResponseTopic=OTA/M/devices/D1,…)
4(a) Publish(clientId, topic=OTA/M/devices, Userproperties={clientId:D1, certificate:D1.Cert}, ResponseTopic=OTA/M/devices/D1…)
On_message(ResponseTopic=OTA/M/devices)
{ retrieve D1.Cert & ResponseTopic;
Compute DH$_{key}$,
Msg=$Dec_{DH_{key}}$[group$_{key}$];
Publish(clientId, topic=OTA/M/devices/D1, Msg, retain=True)
5(a) Publish(clientId, topic=OTA/M/devices/D1, Msg, retain=True)
5(b) Publish(clientId, topic=OTA/M/devices/D1, Msg, retain=True)
On_message(ResponseTopic=OTA/M/devices/D1)
{ retrieve DH_key,
$Group_{key}$=$Dec_{DH_{key}}[Msg]$
}
OTA update
Publish(clientId, topic=OTA/M, Msg=$Enc_{group_{key}}[firmware]$, retain=True…)
Msg
Publish(clientId, topic=OTA/M, Msg,…)
D1
Publish(clientId, topic=OTA/M, Msg,…)
D2
……
Publish(clientId, topic=OTA/M, Msg,…)
Dn

Figure 5.: Publisher distributes E2E key in MQTT 5.0
→: orignal message; →: forwarded message;

4 SECURITY ANALYSIS AND PERFORMANCE EVALUATION

4.1 Security analysis

The 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 attackers can actively manipulate any messages on the publisher-broker channel and the subscriber-broker channel, and the goal is to violate the authenticity, the integrity, and the privacy.

The security of the model depends on several modular blocks: the TLS-based client-broker channel, the group-key-based OTA channel, and the encryption-signature firmware protection. Now we analyze the modular design of our scheme as follows.

The TLS-based client-broker channel. Each client and the broker should mutually authenticate each other before the client can access the MQTT services. The TLS channel protects the authenticity, the privacy, and the integrity of the client-broker channel.

The group-key-based OTA channel. The group-key-based OTA channel is built on top of the publisher-broker-TLS channel, the subscriber-broker-TLS channel, and the publisher-subscriber-DH-key-encryption of the group key. The DH_key is derived from the Diffie-Hellman key using the (public key, private key) pairs of the entities. As long as the TLS channels and the computational Diffie-Hellman problem are secure, this ensures the privacy of the group key against both the attackers and the curious broker.

The encryption-signature firmware protection. The firmware is encrypted using the group key, and then is signed by the device manager using its private key. This protection protects the privacy and the authenticity of the firmware.

Here, we summarize the security properties. The model provides the authenticity, the integrity, and the privacy of the normal MQTT messages; the attackers cannot access the content of the MQTT messages. The model ensures the privacy of the firmware against a curious broker. Both the broker and the attackers cannot violate the authenticity, the integrity, and the privacy of the firmware.

4.2 Performance evaluation

The new model consists of four phases. The first phase, the MQTT connection phase, could adopt any secure authentication schemes (example, the classic TLS or any other secure authentication) which is not the focus of this paper; therefore, we focus on evaluating the rest three phases. As our new model builds the End-to-End channel, the model can adopt any secure authentication schemes (for example, the identity-password authentication or the Challenge-Response authentication [20]), and can rely on the End-to-End channel to protect the transmissions.

Before examining the schemes, we first define some metric symbols. Let T_{sig} denotes the computation cost for one digital signature, T_{ver} denotes that for one signature verification, T_{enc} denotes that for one symmetric encryption or decryption, and T_{exp} denotes that for one modular exponentiation. Let Lat_{TLS} denotes the authentication time using TLS, Lat_{CR} denotes the authentication time using the Challenge-Response

[20], Lat_{E2E} denotes the latency for Chien's End2End channel building [26], Lat_{group} denotes the latency for the group key distribution in our new scheme, Lat_{MQTT} denotes the latency for either publisher-broker MQTT interaction or broker-subscriber interaction, Lat_{OTACTL} denotes the latency for the OTA control message exchanges. The OTA control messages consist of OTA message like notification, response, and confirmation could be exchanged to ensure the smooth OTA interactions; since this part is independent of the model design, we assume this part is the same for all models to simplify the comparison. $Lat_{broker(n)}$ denotes the forwarding delay caused by the broker for *n* simultaneous subscribers.

Table 3. Comparison of the related models

Scheme / Properties	AWS	Infineon	Ours
Privacy firmware against the broker	No	No	Yes
Firmware signing authority	Broker/manager	manager	manager
Number of MQTT interactions in the 2nd phase	NA[1]	NA[1]	9
Computations in the 2nd phase on the subscriber side	NA[1]	NA[1]	1 T_{enc}+ 1 T_{ver} ver. + 1T_{exp}
Computations in the 2nd phase on the publisher side (for *n* subscribers)	NA[1]	NA[1]	n*(1 T_{enc} + 1 T_{ver} + 1 T_{exp})
Computations in the OTA update phase on the publisher side[2]	NA[1]	n*T_{enc}	NA[4]
Computations in the OTA update phase on the broker side (for *n* subscribers)[2]	n*T_{enc}	2n*T_{enc}	None
Total comm. Cost for *n* subscribers[3]	Lat_{TLS} + Lat_{OTACTL} + $Lat_{broker(n)}$+ n*T_{enc}+ Lat_{MQTT}	Lat_{TLS} + Lat_{OTACTL} + $Lat_{broker(n)}$ +2n* T_{enc} + 2Lat_{MQTT}	Lat_{CR}+ Lat_{OTACTL}+ $Lat_{broker(n)}$ + Lat_{MQTT}

1. For the AWS model and the Infineon model, they do not build the End-to-End channel.
2. Here, we assume that the firmware is small-size and can be delivered in one MQTT message.
3. We note that here we can only roughly estimate the total latency, due to it involves three kinds of entities (the broker, the publisher, and n subscribers), and the broker's loading is greatly affected by the number of the subscribers in the AWS model and the Infineon model.
4. In our model, the publisher pre-encrypts the firmware and upload the encrypted firmware before the OTA update phase.

Now we examine the second phase. For each pair of (publisher, subscriber), the second phase takes nine MQTT interactions of which four messages involve the publisher, and five messages involve the subscriber. Because these MQTT messages are all simple-and-short MQTT messages, they demand only little overhead.

Now we examine the 3rd phase. The device manager prepares the firmware, encrypts it, signs it, and uploads it to the broker in our model. The AWS model in this phase is similar, but it does not keep privacy against the broker. The Infineon model keeps the firmware on the local host and does not keep privacy against the broker.

The final OTA update phase of our scheme and the AWS model allow the IoT devices directly access the firmware from the broker, while the Infineon model requires the publisher be on-line to handle the firmware delivery. Because the firmware for IoT devices is usually small size, here we assume it takes only one MQTT message to deliver it.

Table 3 summarizes the performance comparison of the related models. We can see that our model out-performs the AWS/Infineon models in terms of firmware privacy protection, at the extra cost of nine MQTT interactions and the extra computations at the second phase. We note that these nine interactions only be executed once, and they are very efficient MQTT interactions. In the last raw, we estimate the rough total latency for *n* subscribers. Here we can see that our model is expected to have shorter latency; that is because our model's last phase does not require the publisher-broker interactions and the firmware is pre-encrypted once only before the OTA update phase. However, we should note that the inter-impact among the broker's loading, the number of subscribers, and the aggregated latency is quite complicated; here we only capture the possible asymptotic behavior. To have accurate comparison, we plan to implement these models to evaluate their performance in the real scenarios.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we have designed the group-key support for the MQTT5.0 system and the new OTA update model. In the model, the device manager can securely distribute the group key to his IoT devices. The manager encrypts the firmware and uploads it to the broker. During the OTA update phase, the devices directly access the encrypted firmware from the broker. This arrangement greatly enhances the privacy protection of the firmware and keeps the interactions during the TOA update phase simple and efficient.

Through simple analytic comparison, we can see that our model achieves better privacy protection and gains efficient communication performance. In the future work, we plan to implement the four models and evaluate their performance in the real scenarios.

Acknowledgement: This research was funded by the Ministry of Science and Technology, Taiwan, R.O.C. grant number MOST 111-2221-E-260-009-MY3, MOST110-2221-E-018-006-MY2, MOST 110-2221-E-324-007-MY3.

References

1. Karim Hamdy, "Over-the-Air (OTA) Updates: What is it and How to do it simply, efficiently with ZDM", https://itskarim.medium.com/over-the-air-ota-updates-what-is-it-and-how-to-do-it-simply-efficiently-with-zdm-db613ea29678, accessed 2022/08/30.
2. Mohammad Afaneh, " Implementing Over-the-Air Device Firmware Update (OTA DFU) – Part 1", https://www.novelbits.io/ota-device-firmware-update-part-1/, accessed 2022/08/30.
3. Wikipedia, "Over-the-air programming", https://en.wikipedia.org/wiki/Over-the-air_programming, accessed 2022/08/30.
4. ISO/IEC 20922:2016, Information technology -- Message Queuing Telemetry Transport (MQTT) v3.1.1, https://www.iso.org/standard/69466.html, last access 2022/03/25.
5. OASIS, MQTT Version 5.0, 07 March 2019. https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html, last access 2022/04/01.
6. Eclipse Mosquitto, https://mosquitto.org/, accessed 2023/01/04.
7. HiveMQ Homepage, Enhanced Authentication. https://www.hivemq.com/blog/mqtt5-essentials-part11-enhanced-authentication/, last access 2022/04/02.

8. Mosca, https://github.com/moscajs/mosca, accessed 2023/01/04.
9. Amazon, "How to perform secondary processor over-the-air updates with FreeRTOS", https://aws.amazon.com/tw/blogs/iot/how-to-perform-secondary-processor-over-the-air-updates-with-freertos/, accessed 2022/08/30.
10. Amazon, "AWS IoT Over-the-air Update", 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 2022/08/30.
11. Implementing MQTT Client Using AnyCloud Libraries, https://community.infineon.com/t5/Blogs/Implementing-MQTT-Client-Using-AnyCloud-Libraries/ba-p/246975, accessed 2023/01/04.
12. Hung-Yu Chien, Nian -Zu Wang, "A Novel MQTT 5.0-Based Over-the-Air Updating Architecture Facilitating Stronger Security", MPDI Electronics 2022 Nov 25, 11(23). https://www.mdpi.com/2079-9292/11/23/3899.
13. Lesjak, C., Hein, D., Hofmann, M., Maritsch, M., Aldrian, A., Priller, P., Ebner, T., Ruprechter, T., and Pregartne, G.: Securing Smart Maintenance Services: Hardware-Security and TLS for MQTT. 2015 IEEE 13th International Conference on Industrial Informatics (INDIN), Cambridge, UK, Cambridge, pp. 1243-1250 (2015).
14. Andy, S., Rahardjo, B., Hanindhito, B.: Attack Scenarios and Security Analysis of MQTT Communication Protocol in IoT System. Proc. EECSI 2017, Yogyakarta, Indonesia, 19-21 September 2017, pp. 19-21 (2017).
15. Firdous, S. N., Baig, Z., C. Valli, Ibrahim A.: 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), pp. 748-755 (2017).
16. Andy, S., Rahardjo, B., Hanindhito, B.: Attack Scenarios and Security Analysis of MQTT Communication Protocol in IoT System. Proc. EECSI 2017, Yogyakarta, Indonesia, 19-21 September, pp. 19-21 (2017).
17. Rizzardi, A., Sicari, S., Miorandi, D., Coen-Porisini, A.o: AUPS: An Open Source Authenticated Publish/Subscribe system for the Internet of Things. Information Systems 62, pp. 29-41 (2016).
18. Neisse, R., Steri, G., Baldini, G.: Enforcement of security policy rules for the internet of things. In 2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), Larnaca, pp. 165-172 (2014).
19. Shin, S. H., Kobara, K.: Efficient Augmented Password-Only Authentication and Key Exchange for IKEv2. IETF RFC 6628, Experimental, June 2012. https://tools.ietf.org/rfc/rfc6628.txt, last access 2022/02/05.
20. 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, International Journal of Sensor Networks, Vol. 32, No.1, pp. 54-68 (2020).
21. Chien, H.-Y., Lin, P.C., Chiang, M.L.: Efficient MQTT Platform Facilitating Secure Group Communication. Journal of Internet Technology, Dec., 21(7), pp. 1929~1940 (2020).
22. Chien, H.Y., Qiu, G.H., Hung, R.W., Shih, A.T., Su, C.H.: Hierarchical MQTT with Edge Computation. The 10th International Conference on Awareness Science and Technology (iCAST 2019), Morioka, Japan, pp. 1-5 (2019).
23. A. Mektoubi, H. Lalaoui, H. Belhadaoui, M. Rifi, A. Zakari, "New approach for securing communication over MQTT protocol A comparison between RSA and Elliptic Curve", 2016 Third International Conference on Systems of Collaboration (SysCo), Casablanca, Morocco.

24. Prajit Kumar Das, Sandeep Narayanan∗, Nitin Kumar Sharma, Anupam Joshi, Karuna Joshi, Tim Finin, “Context-Sensitive Policy Based Securityin Internet of Things”, 2016 IEEE International Conference on Smart Computing (SMARTCOMP), St. Louis, MO, USA.
25. Ciou, P.-P., Chien, H.-Y. Chien: An Implementation of Challenge-Response Authentication for MQTT 5.0 IoT System. the 2021 International Conference on Emerging Industry and Health Promotion (EIHP2021), Puli on July 3-4 (2021).
26. H.-Y. Chien, “Design of End-to-End Security for MQTT 5.0”, The 4th International Conference on Science of Cyber Security - SciSec 2022, August, 10-12, 2022 | Matsue city, Shimane, Japan.
27. SEEMQTT: Secure End-to-End MQTT-Based Communication for Mobile IoT Systems Using Secret Sharing and Trust Delegation. IEEE Internet Things J. 10(4): 3384-3406 (2023).
28. Performance evaluation of CoAP and MQTT with security support for IoT environments. Comput. Networks 197: 108338 (2021).