Design, Implementation, and Evaluation of MQTT5.0-Based End-to-End Security

Hung-Yu Chien[1], An-Tong Shih[2], Yuh-Ming Huang[2]*

[1] Department of Information Management, Natonal ChiNan University, Taiwan
[2] Department of Computer Science & Information Engineering, Natonal ChiNan University, Taiwan
hychien@ncnu.edu.tw, s112321902@mail1.ncnu.edu.tw, ymhuang@csie.ncnu.edu.tw

Abstract

MQTT has become one of the most common communication protocols for various Internet of Things (IoT) applications. The MQTT 5.0 standard has been proposed to support better security and functional designs. However, the existent MQTT End-to-End (E2E) mechanisms were not designed for the new MQTT 5.0; so, up to now, there is no MQTT5.0-based E2E design, implementation, and security analysis.

This study designs and implements MQTT5.0-based E2E scheme, adopting the MQTT 5.0 enhanced authentication framework and the newly added extension field - User Property. The evaluation of the implementation shows that the design is efficient and scalable. It shares some potentials of MQTT5.0-based designs and implementations.

Keywords: MQTT 5.0, Mosquitto, Enhanced authentication, End to end

1 Introduction

With the maturation of various sensors and the development of Internet of Things (IoT) systems, IoT has been rapidly and vigorously expanding across numerous industries. In IoT systems, rapid and efficient communication plays a crucial role. Among the many IoT communication protocols, MQTT (Message Queuing Telemetry Transport) [1-2] has been widely adopted due to its speed, efficiency, and ease of use. As MQTT gains widespread adoption, its security support limitations and enhancements have become a focal point of attention in both academia and industry.

An MQTT system is composed of three types of roles: publisher, broker, and subscriber. Publishers post information on specific topics, and broker forwards the messages to subscribers who have subscribed to those topics. Versions of MQTT prior to MQTT 5.0 [2] only provided a simple username and password method for brokers to authenticate clients (either publishers or subscribers), although developers could also initiate SSL/TLS authentication under MQTT to protect client-server data transmission. With the widespread adoption of MQTT, there have been many efforts in academia and industry to improve the original MQTT security support.

Most efforts to enhance MQTT security (such as [4-16]) focus on improving the authentication functions and privacy protection between the client and the broker. These improvements use mechanisms including symmetric encryption systems, asymmetric encryption systems, additional hardware support, or the support of additional proxy servers. In light of the generic limitations, the standard organizations have released the newest MQTT standard-MQTT 5.0 for designers and system developers to design new security mechanisms and new functions.

However, because the MQTT architecture allows the broker to relay messages, it also enables the broker inspect the content of these messages. The security and privacy vulnerability lead to research on MQTT End-to-End (E2E) security: that is, establishing authentication and secure private communication between publishers and subscribers and preventing broker from inspecting the content.

There exist many works addressing the challenges of MQTT security. However, only few of them (for example, [17-19]) tackle the MQTT E2E challenges, and few of them (for example, [20-21]) conform to MQTT5.0. Furthermore, those MQTT5.0-based designs do not practically implement the designs using MQTT5.0 APIs and evaluate the performance on-field. Regarding those MQTT E2E proposals [18-19], they are inefficient either in the architectures or in the computations, suitabile to only very limited scenarios, or non-compatible with MQTT5.0 standards.

Therefore, this study extends our previous work [23], redesigns and implements the MQTT5.0-based E2E scheme using MQTT5.0 APIs, evaluates the performance of the implementation, and analyzes the security. The contributions of this study include: (1) a MQTT5.0-based E2E design, (2) the first implementation using Mosquitto 5.0 package, (3) thorough performance evaluations (latency, CPU, memory, SWAP) of the implementations confirming its scalability and efficiency.

The rest of this study is organized as follows. Section 2 discusses the related work. Section 3 introduces the high-level designs, while Section 4 describes the detailed designs using Mosquitto 5.0 APIs. Section 5 analyzes the security and evaluate the performance. Section 6 states our conclusions and future work.

*Corresponding Author: Hung-Yu Chien; Email: hychien@ncnu.edu.tw
DOI: https://doi.org/10.70003/160792642026092705002

2 The Related Work

The discussion of the related work is organized as follows, according to each subtopic. Section 2.1 discusses the security proposals for the conventional MQTT. Section 2.2 sorts out the MQTT 5.0-based works. Section 2.3 compares the MQTT E2E schemes.

2.1 The Security-enhanced Proposals for the Conventional MQTT

Most of the existent MQTT security discussions focus on improving the conventional MQTT standards. The works like [4] report the security weaknesses and threats. Many proposals like [5-16] provide authenticated key agreement schemes, using symmetric-key-based approaches or asymmetric-key-based approaches.

Some of the MQTT-security-enhancement proposals like [14] propose an add-on hardware to support TLS/SSL functions. Rizzardi et al. [15] proposes the key management framework and Neisse et al. [16] propose a policy management framework to support flexible and dynamical policies. The inclusion of OAuth mechanism is proposed in Niruntasukrat et al. scheme [7]. The works [12-13] aim at efficient anonymous MQTT authentication.

2.2 The MQTT 5.0-based Security Works

Even though MQTT5.0 has been released for several years, only very few designs leverage the new functions and the new security supports of MQTT 5.0. One main reason for that is the lack of the knowledge and understanding of the new standards. Another reason is the slow release of MQTT 5.0 open-source projects. However, the limitations have been improved; it is time to develop and promote MQTT5.0-based designs and implementations to accelerate the proliferation of MQTT5.0-based systems.

Chien [17] design an E2E scheme for MQTT 5.0, but the work does not provide the implementation. Ciou and Chien [20] recently design and implement a Challenge-Response authentication using MQTT 5.0 Enhanced Authentication framework, but it does not support E2E security. The work [21-22] designs the Over-The-Air (OTA) protocol using MQTT 5.0, but the implementation does not leverage the MQTT5.0 new APIs.

2.3 The MQTT E2E Schemes

There exist some works tackling the E2E security of MQTT. Mektoubi et al.'s E2E proposal [10] is based on topic-certificate design of which the messages are encrypted using the public key of the corresponding topic certificate while a legitimate subscriber using the corresponding private key to decrypt the messages; this approach demands very huge overhead for the clients.

MQTLS scheme [23] enables a publisher securely share its E2E keys via publisher's-subscriber's ephemeral Diffie-Hellman session keys; this approach effectively facilitates a publisher the job of E2E key distribution to its subscribers; however, it requires all the subscribers issue their CONNECT request before the publisher issue its CONNECT request; this requirement is impractical.

Chien [20], based on MQTT5.0, designs an E2E channel model, but the work does not implement and evaluate the design. Wang and Chien [21-22] implement an MQTT-based OTA scheme, but the implementation is based on MQTT 3.1. Shih et al. [23] implement an MQTT 5.0 E2E scheme, but only initial evaluation is provided.

SEEMQTT [3] facilitates the E2E security where a publisher delegates its decryption authority to a pool of the key stores, via secret sharing mechanism. Those designated subscribers should contact the pool of key stores, and get verifications from the stores to recover the decryption key. This arrangement aims at those scenarios where it is difficult for the publishers to directly verify their subscribers, but it incurs lots of extra overhead.

Table 1 summarizes the related studies. This study aims at designing, implementing, and evaluating MQTT5.0-based E2E scheme. From the table, only this study simultaneously satisfies the goals.

Table 1. Summary of MQTT-based security schemes

Property / Scheme	Goals	MQTT<3.1 or 5.0 design	MQTT 5.0 impl.	Eval. of MQTT5.0-impl.	E2E
[5-16]	Key agreement etc	<=MQTT 3.1	NO	NO	NO
[18-19]	Key agreement etc	<=MQTT 3.1	NO	NO	YES
[17]	Key agreement etc	MQTT5.0	NO	NO	YES
[20]	Key agreement etc	MQTT5.0	NO	NO	NO
[21-22]	OTA	MQTT5.0	NO	NO	YES
[23]	Key agreement etc	MQTT5.0	YES	Very minor	YES
This work	Key agreement etc	MQTT5.0	YES	YES	YES

3 The Design of MQTT 5.0-based E2E Scheme

3.1 Preliminary of MQTT 5.0

MQTT 5.0 provides several new features and framework that facilitates designers design and implementations of MQTT-aware security and rich-and-flexible function support. The Enhanced Authentication framework facilitates design of MQTT-aware security. "User properties" enable the inclusions of custom-design parameters between a client and its broker and between a publisher and its subscribers.

Enhanced Authentication. The MQTT 5.0 Enhanced Authentication Framework aims at facilitating designers design and implementation of MQTT-aware authenticated key agreement schemes by providing new AUTH APIs and new fields. The new AUTH API and CONNECT/CONNACK API with the new fields can be used to initiate the corresponding authentication method and passing the corresponding authentication data in AUTH/CONNECT/CONNACK API.

User Properties. User properties are user-defined metadata shared among publishers, brokers, and subscribers. They are represented as UTF-8 key/value pair. One example of our design of sharing clientId/ topic/certificate via a PUBLISH API is specified as follows. PUBLISH(topic='IoT/P1', UserProperties = { "ClientId:P1", "Topic:IoT/P1", "Certificate: xxx", …}. When the subscribers get the messages, they will get the metadata shared by the publisher P1.

3.2 The High-level E2E Protocol

Before describing our MQTT5.0-based E2E scheme, we first introduce its high-level protocol description without involving its MQTT5.0 details. Via the high-level protocol description, we can get the core ideas of the scheme. Table 2 introduces the notation. PK_P/ SK_P respectively denotes the long-term public/private key of the publisher. $TPK1_P$/ $TSK1_P$ respectively denotes the 1st temporary public/private key of the publisher. $Sig_{SK_P}()$ denotes a digital signature using the private key of the publisher. MAC()/ENC() respectively denotes Message Authentication Code/ ENCryption using symmetric-key.

Table 2. The notation

P: Publisher; S: Subscriber; BR: Broker.	PK_P/ SK_P : long-term public/private key of *P*. Similar notations apply for *S* and *BR*.
$E2E_{key}$: the session key used in the E2E channel; ie. SK_{SP} in our detailed scheme.	$TPK1_P$/ $TSK1_P$, $TPK2_P$/ $TSK2_P$: temporary public/ private key of *P*. Similar notations apply for *S* and *BR*. For example, $TPK1_{BR} = g^{TSK1_{BR}}$ modp.
$Sig_{SK_P}()$: signature by *P*.	*MAC*(): message authentication code which could be implemented using HMAC. *ENC*(): symmetric key encryption
SK_{PBR}/ SK_{SBR}/ SK_{SP}: the session key between *P* and *BR*/ *S* and *BR* / *S* and *P*. $SK_{PBR} = g^{TSK1_{BR}*TSK1_P}$ modp . $SK_{SBR} = g^{TSK2_{BR}*TSK1_S}$ modp . $SK_{SP} = g^{TSK2_S*TSK2_P}$ modp .	

The scenario of this design assume that a publisher generates the messages of a specific topic, and there are many subscribers subscribing the topic. The main idea of the protocol is described as follows. A publisher and a subscriber use their certificates and the signatures on the temporary public keys to mutually authenticate each other; they establish their Computational-Diffie-Hellman-Problem (CDHP)-based session key SK_{SP} using their temporary public key and temporary private key. This session key SK_{SP} is used to encrypt the E2E channel messages between a publisher and a subscriber. The mutual authentication between a client (a publisher or a subscriber) and a broker is also achieved by exchanging their long-term certificates and signatures. The session key between a client and a broker is by deriving the CDHP-based key using their temporary public/private key pair.

The protocol flow of the main idea is depicted in Figure 1. In this demonstrated scenario, the two clients (a publisher and a subscriber) build a client-broker authenticated channel in the first dash-line block and in the second dash-line block respectively. After that, the subscriber initiates the E2E channel request and builds a shared E2E key with the publisher in the third flow bock. Finally, the subscriber sends a confirmation message back to the publisher in the fourth block. In the following, we detail the functions and the messages in each block. Even though we let the subscriber initiate the E2E request in this demonstration, it is easy to change the initiator to fit the application scenario.

Block 1: In this block, the publisher and the broker respectively select a random number $TSK1_P$ and $TSK1_{BR}$, and compute $TPK1_P = g^{TSK1_P}\ modp$ and $TPK1_{BR} = g^{TSK1_{BR}}$modp, where the temporary public keys $TPK1_P$/ $TPK1_{BR}$ are used both as the CDHP keying materials and as the challenges. They exchange their certificates $Cert_P$/ $Cert_{BR}$, their temporary public keys, and their signatures $Sig_{SK_P}()$/ $Sig_{SK_{BR}}()$ on the temporary public keys. Based on the signatures, they can verify the authenticity of the temporary public keys, and computes their shared key $SK_{PBR} = g^{TSK1_{BR}*TSK1_P}$ modp. This session key SK_{PBR} would be used as the encryption key and the hash key in the publisher-broker channel.

Block 2: In this block, the subscriber and the broker respectively select a random number $TSK1_S$ and $TSK2_{BR}$, and compute $TPK1_S = g^{TSK1_S}\ modp$ and $TPK2_{BR} = g^{TSK2_{BR}}$modp, where the temporary public keys $TPK1_S$/ $TPK2_{BR}$ are used both as the CDHP keying materials and as the challenges. They exchange their certificates $Cert_S$/ $Cert_{BR}$, their temporary public keys, and their signatures $Sig_{SK_S}()$/ $Sig_{SK_{BR}}()$ on the temporary public keys. Based on the signatures, they can verify the authenticity of the temporary public keys, and computes their shared key $SK_{SBR} = g^{TSK2_{BR}*TSK1_S}$modp. This session key SK_{SBR} would be used as the encryption key and the hash key in the subscriber-broker channel.

Block 3: The subscriber selects a random number $TSK2_S$, and computes $TPK2_S = g^{TSK2_S}modp$; it sends $Cert_S$, $TPK2_S$, $Sig_{SK_S}(TPK2_S)$, $ENC_{SK_{SBR}}(...)$ back to the broker, where $ENC_{SK_{SBR}}(...)$ denotes the encryption of the session data using the key SK_{SBR}, which serves as a response (Message Authentication Code- MAC) to the broker's challenge $TPK2_{BR}$. The broker forwards the message to the publisher but replaces the message authentication code $ENC_{SK_{SBR}}(...)$ with a new one $ENC_{SK_{PBR}}(...)$ which involves the key SK_{PBR} to convince the publisher the authenticity.

The publisher verifies the signatures and the MAC. If the verification satisfies, it chooses a new random number $TSKs_P$, computes $TPK2_P = g^{TSK2_P}modp$ and its signature, and derives the E2E key $E2E_{key} = TPK2_S^{TSK2_P} = g^{TSK2_P*TSK2_S}$.

It sends back the message $\{Cert_P, TPK2_P, Sig_{SK_P}(TPK2_P), ENC_{SK_{SP}}(...), MAC_{SK_{PBR}}(...)\}$ where $ENC_{SK_{SP}}(...)$ serves as a MAC back to the subscriber and $MAC_{SK_{SBR}}(...)$ serves as a MAC to the broker. Upon receiving the message, the broker verifies the MAC $MAC_{SK_{PBR}}(...)$, replaces it with a new MAC $MAC_{SK_{SBR}}(...)$, and forwards the message to the subscriber.

The subscriber verifies the signature and the MAC codes, and then computes $E2E_{key} = TPK2_P^{TSK2_S} = g^{TSK2_P * TSK2_S}$. At this point, the publisher and the subscriber have established the shared key $E2E_{key}$.

Block 4: This block is optional. It serves as the subscriber's confirmation message back to the publisher. The subscriber sends $\{ENC_{SK_{SP}}(...), MAC_{SK_{SBR}}(...)\}$ to the broker, where $ENC_{SK_{SP}}(...)$ serves as a MAC to the publisher and $MAC_{SK_{SBR}}(...)$ serves as a MAC to the broker. The broker verifies $MAC_{SK_{SBR}}(...)$ and the publisher verifies $ENC_{SK_{SP}}(...)$.

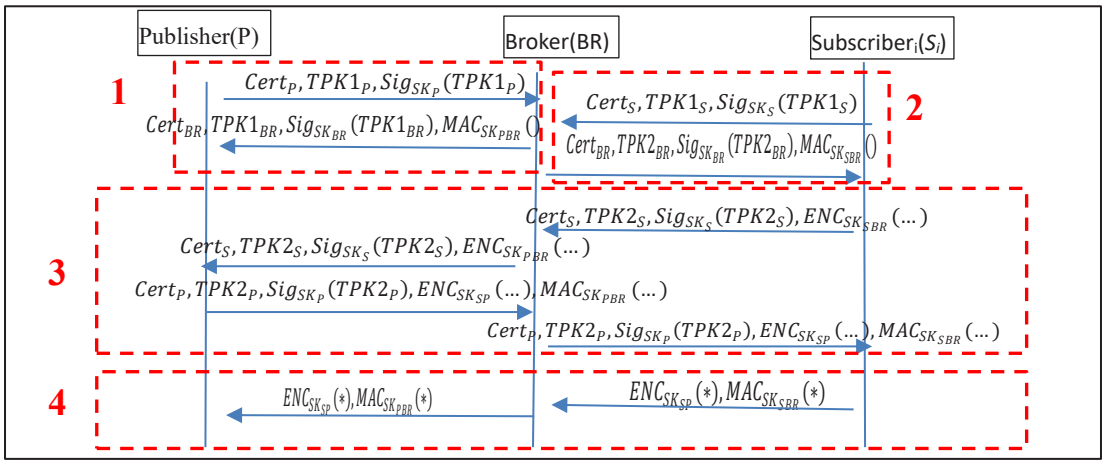


Figure 1. The high-level E2E channel building flow

4 The Design and Implementation of MQTT5.0-based E2E using Mosquitto

The previous section describes the high-level logic protocol flow of our MQTT5.0-based E2E, and this section introduces the design and implementation using Mosquitto [24] APIs. Mosquitto is a popular open-source MQTT package, and the newest versions support MQTT5.0. Mosquitto's plugins facilitate developers implement some customized programs to work with the broker. The clients from Paho [3] package is used, and OpenSSL [25] is adopted for security functions. json-c library [26] and VMware [27] are adopted.

4.1 The Topic Design

In order to facilitate the efficient distribution of the E2E requests and responses, we design the following topic tree. The message publishing and subscriptions in MQTT is based on the concept "topic". Only those subscribers who subscribe the specified topic can receive the forwarded messages from a broker. The topic tree is hierarchical (Figure 2). The first level "iot" represents this application. The second level consists of "P01" and "broker", where "P01" is the identity of the publisher and "broker" is the broker. The third level "S01/S02/S03" respectively denote three subscribers. The "message" topic serves as the channel for the general MQTT messages.

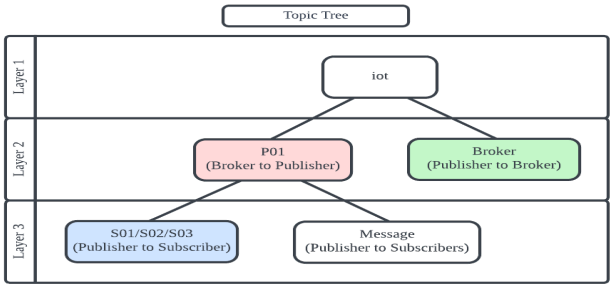


Figure 2. The topic tree

The topic "iot/P01/S01" is used as the specific communication channel for the publisher and the subscriber S1. The topic "iot/P01/message" serves as the channel for the publisher to broadcast its general MQTT messages for all subscribers. The publisher can securely send messages to each individual subscriber, or it can broadcast messages to all subscribers by using topic-based group keys as shown in Figure 3. We do not show the group key distribution (which is optional in Figure 3).

4.2 The Implemented E2E Message Flow using MQTT APIs

Here, we depict a simplified version of our E2E scheme implementations using the MQTT APIs. The differences between this version with that of the above include two parts. First, this implementation drops the fourth block of the E2E scheme in Figure 1, where the fourth block is sending back the subscriber's confirmation.

Second, we merge the subscriber-broker confirmation and the subscriber-publisher-E2E confirmation in the same CONNACK message (Step 14 of Figure 4) to evaluate the E2E connection latencies. The client-broker authentication is implemented using the MQTT Connect packet and the Connack packet (Step 1~2).

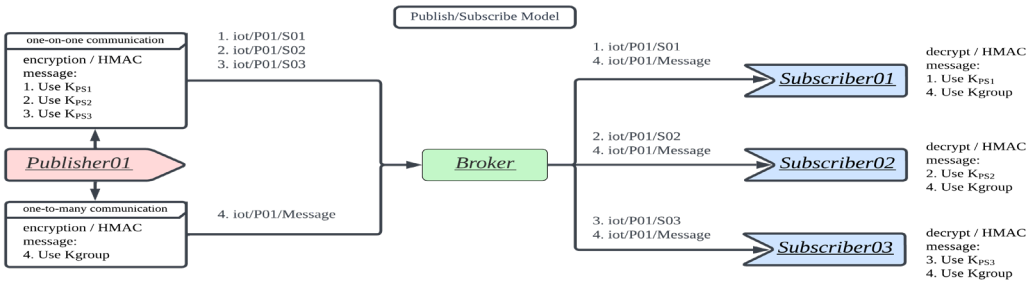


Figure 3. The Publish/Subscriber topic-key relation

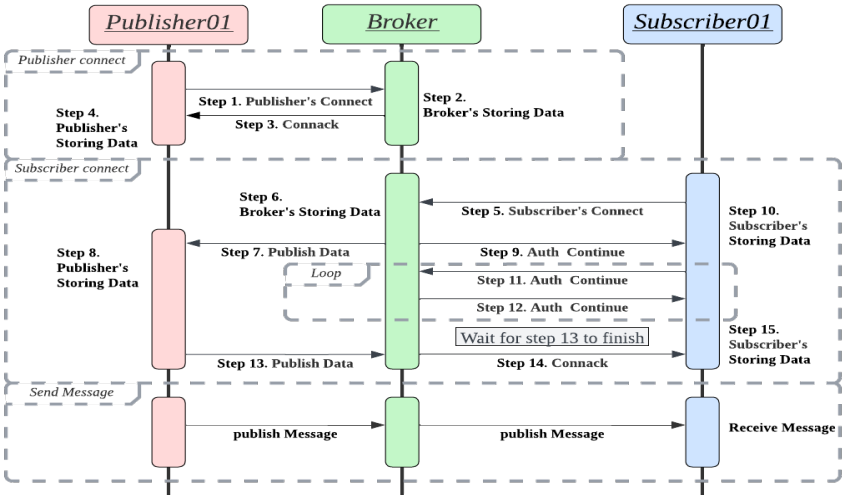


Figure 4. An implementation of MQTT5.0-based E2E scheme

The subscriber initiates a connect request and the E2E request from Step 5, and the two authentication requests finish at Step 14. In the Figure, Step 9, 11, 12 consist of a series of MQTT Auth packets, where the Auth packets are used by the broker to notify the subscriber keep on waiting the final response (Connack). The exact number of the Auth packets depends on how long it takes to get the final response from the publisher (Step 13).

Examples of the message contents

Figure 5 depicts the fields of the messages in Step 1~4. The authentication method in Step1 and 3 specifies the method "CR_Auth", which corresponds to our Challenge-Response method. The content (Auth data) of step1 and 3 consists of user_id (here, P1), pub_key_sender (P'1 long-term public key), time, signdata_to_broker (P1's signature on its data), channel (refer to it is client-broker or E2E channel), role (publisher/subscriber/broker), and cert_sender (P1's certificate). Step 2 and 4 respectively depict the stored data in publisher and in broker.

Figure 6 and Figure 7 show the contents of a published message (after E2E channel in built) in Figure 4. Here, the title "HMAC message" specifies that its content is authenticated with a HMAC code which is kept in the User Property.

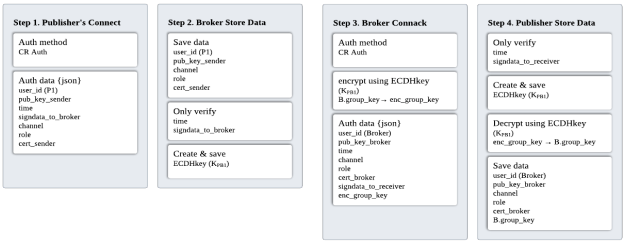


Figure 5. The fields of the message contents for Step 1~4.

```
Payload:{
    "title": "HMAC message",
    "content": "abcdefghijklmnopqrstuvwxyz\n",
    "sender": "Publisher01",
    "receiver": "topic/P01/Message",
    "cipher_opt": "HMAC",
    "time": "2023-11-22 13:01:19"
}
User Property:{
    "digest_HMAC":
    "2120024A9C7D7E1C503DA63088F69471E0C5B98E63C-
    1CA959A213FD306AC5CF8"
}
```

Figure 6. Message payload and User Property content in a published message

5 Security Analysis and Performance Evaluation

We analyze the security properties in Section 5.1. Section 5.2 evaluates the performance.

5.1 Security Analysis

The security goals of the proposed scheme include: (1) the mutual authentication between a client and the broker and the privacy of the shared keys; (2) the mutual authentication between a publisher and a subscriber and the privacy of the E2E key; (3) the scheme is secure against replay attack, side channel attack, and server compromise.

The E2E scheme is built based a modular approach. For each E2E channel between a publisher and a subscriber, it consists of two client-broker authentication and one publisher-subscriber authentication.

The authentication/privacy of client-broker connection. A client and a broker perform the connection authentication by exchanging their certificates, their temporary public keys (as challenges), and their signatures on the challenges. After verifying the signatures, they calculate the Diffie-Hellman key as the session key. This ensures the authentication of the connection and the privacy of the session key.

The mutual authentication of publisher and subscriber/the privacy of the E2E channel. Like the client-broker authentication, the publisher and the subscriber exchange their certificates, their temporary public keys (as challenges), and their signatures on the challenges. When the exchanged messages being transmitted on the publisher-broker channel, the messages are appended with MACs. This ensures the broker forwards authenticated messages. When the broker forwards the messages from the publisher to the subscriber, the broker adds its MAC codes to the messages, where the MAC code calculations involve the CDH key derived from the broker-subscriber's temporary public keys and private keys. It ensures all the three entities (publisher, broker, subscriber) can verify the authenticity of the messages. The CDH key for the E2E channel ensures the security and privacy of the E2E channel.

Theorem 1. The client-broker session key and the publisher-subscriber E2E session key are secure against replay attacks. The two channel session keys are based on authenticated D-H key calculation using the exchanged temporary public/private keys (with signature using the long-term public/private keys. The temporary public keys are renewed randomly in each session and are used as challenges (nonce) to the communicating parties, therefore, the calculation of the D-H key is authenticated, fresh and secure; therefore, it can resists replay attacks.

Regarding a server (broker) compromise, we respectively two situations as follows.

Theorem 2. For an honest-but-curious broker, our E2E scheme can protect the privacy of the publisher-subscriber connection using the E2E key. An honest-but-curious broker would follow the protocol honestly but still being curious about the contents of clients' transmissions. Because the DH-based E2E session key is freshly calculated based on the signed temporary public/private keys (renewed in each session), an honest-but-curious broker cannot derive the E2E session key; therefore, it protects the privacy against a curious broker. Interested readers are referred to the formal proofs in [21-22].

If server compromise refers to the situation that the broker's long-term private key (certified in its certificate) is compromised, then the attacker who breaks the long-term private key can impersonate the broker and break the system. For such kind of server compromise, there is no solution to any PKI-based systems, to the best of our knowledge.

Theorem 3. Resistance to side channel attacks. A side-channel attack is any attack based on extra information that can be gathered because of the fundamental way a computer protocol or algorithm is implemented; the extra information might be derived from the transmissions or from the physical hardware. Regrading the extra information from the transmission, our scheme only transmits long-term/short-term public keys with the signatures, therefore, an attacker can only get the public keys, which does not help the attackers. Of course, if the private key is stored in memory/USB/cards, then some extra information might be derived from the side channel; but, the hardware implementation is beyond the scope of this study.

Discussion of downgrade attacks. In a downgrade attack, an attacker forces the target system to switch to a low-quality, less secure mode of operation; for example, in our context, one could refer it as forcing the system to switching to precedent MQTT versions (like MQTT v3.1, etc). Here, we discuss this challenge is two parts. Part1: as a MQTT5.0-based implementation, our implementations require the clients and brokers all support MQTT5.0, and does not support backward compatibility.

Part2. If one takes the core concepts and implements it using the precedent-MQTT-version APIs, of course, it is feasible to have the similar functions in the precedent versions (some MQTT-3.1-based enhancements could be found [22]). However, it would take much more efforts/ cost in the deign phase and in the running phase. One might integrate the two systems and allow backward compatibility. Does it, therefore, facilitate downgrade attacks? It depends on the designs and implementations, and it is beyond the scope of this study.

5.2 Performance Evaluation

Figure 7 shows the deployment of our implementations, where there are two virtual machines- one is for the broker and the other is for one publisher and 5 subscribers. Table 3 summarizes the hardware and software used in the experiments.

Table 4 sorts out the connection latencies of the publisher and the subscribers, where each client initiates 50 runs of connection requests. Here, we have to emphasize the differences between the connection latencies of the publisher and the subscribers. The connection latencies of the publisher consist of only these interactions for publisher-broker authentications while that of a subscriber

consisting of both the client-broker authentication interactions and the publisher-subscriber E2E interactions. Therefore, the latency difference between the two can get some insight about how much performance impact for building a E2E channel. The fifth row of the table indicates the numbers of AUTH packets sent before the subscriber get the final response, which is the current implementation we adopt to keep the subscriber waiting for the final response; it could be further enhanced in the future implementation optimization.

In this implementation, the interval between two Auth(continue) packets is 25 ms. From the table, we see that the average connection time of the latencies subscribers is 153.8 ms while the average connection time of the publisher being 135ms; that is, the average E2E channel building time is 153.8-135=18.8 ms, which is non-significant, compared to the conventional client-broker connection. This insignificant amount of E2E overhead gives us an important insight that the inclusion of E2E channel does not add significant overhead while gaining end-to-end privacy and security.

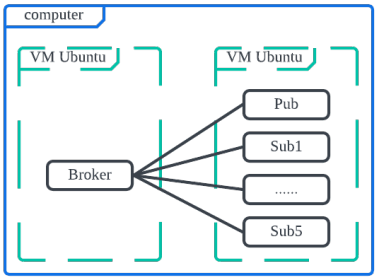


Figure 7. The 1st deployment of our implementation

Table 3. The hardware/software setting of the experiments

Client & Broker VM environment	Virtualization Software	VMware Workstation Player
	Operating System	Ubuntu 22.04.3 LTS
	CPU	2 vCPUs
	Memory	4, 2.4 GB RAM
	Network Configuration	NAT mode
Software for the MQTT E2E	Client VM	Broker VM
	Paho.MQTT.C Version 2.8.12 OpenSSL Version 3.0.2 15	Mosquitto Version 2.0.15 OpenSSL Version 3.0.2 15

Note: Network Description: All VMs are run on the same host, configured in a NAT environment allowing VMs to communicate with the external network through the host's interface while maintaining network isolation among VMs.

Table 4. Summary of 50 connection latency of publisher/subscribers (ms)

	Pub	Sub 1	Sub 2	Sub 3	Sub 4	Sub 5	
AVERAGE	135	163	142	155	150	159	AVG(Sub1~5)=153.8
MAX	191	282	149	186	186	193	MAX(Sub1~5)=282
MIN	39	57	52	74	50	47	MIN(Sub1~5)=47
MAX(w)		4	1	1	1	1	

Note: w = Wait count, w=1(+25ms), w=2(+50ms); Pub: publisher, Sub: subscriber.

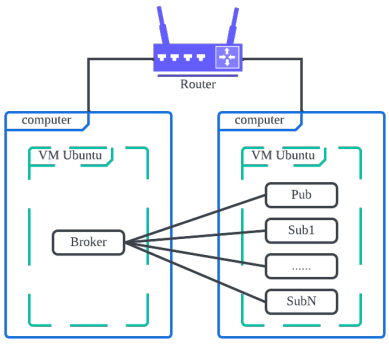


Figure 8. The 2nd deployment of our implementation

To evaluate the impact of the number of the clients, we design and deploy the second scenario in Figure 8, where the number of subscribers ranges from 10, 20, 50, 100 and 200, and two computers interacts via router connection; the publisher sends 1000 messages and we average the latency, the CPU utilization, and the memory usage.

Table 5. The latencies of pub/sub connection

Number of subs	10	20	50	100	200
Latencies of pub/sub	151/239	137/259	177/260	189/261	157/277

Table 5 shows the average latencies of pub/sub connection for 1000 connections. Figure 9 depicts the trend chart. Here, we can see that the latency of publisher connection is quite stable while that of a subscriber connection slightly increasing, as the number of subscribers increases, where we speculate that the little fluctuation is due to the LAN network fluctuation. When the number of subscribers is 200, the average E2E delay is 277-157=120 ms, which is still smaller than the publisher-

broker connection latency; it means that the inclusion of E2E channel does not significantly increase the overhead.

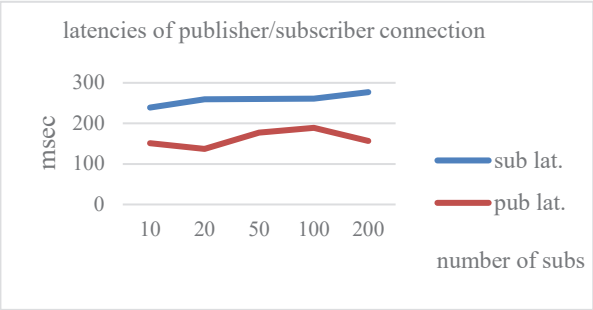
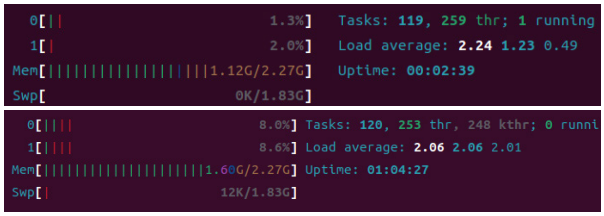
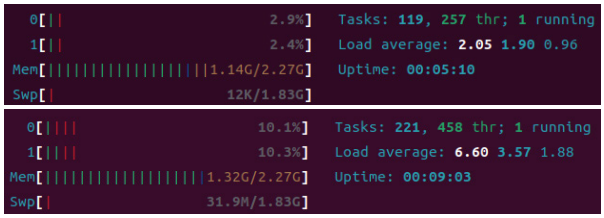


Figure 9. The latencies of pub/sub connections in Scenario2



(a.1)/(a.2) broker VM before/after subscribers launch



(b.1)/(b.2) client VM before/after subscribers launch

Figure 10. CPU/memory/SWAP usage before/after client launch

Figure 10 shows the CPU/memory/SWAP utilization percentage conditions before/after 200 subscriber launch. On the broker VM, the CPU utilization roughly increases 6~7% while that increases roughly 6~7% on the client VM; the memory usage increases from 1.12G to 1.6G while that increases from 1.14G to 1.32G on the client VM; the SWAP increases from 0K to 12K when a broker starts, and that increases from 12K to 31.9M on the client VM. From these data, we can see that our implementation does not impact the CPU/memory/SWAP usage significantly since we just add the Enhanced authentication and the E2E building. Compared to the existent functions of a broker/client, the inclusion of our functions does not have significant impact on the CPU/memory/SWAP utilization.

Some conclusions from the above experiment observations are summarized as follows. (1) On a single computer deployment, the E2E latency is only 18.8 ms which is very small; (2) On the 2-computer-2-VM-LAN experiments, the increase of the number of clients on a single VM only slightly impacts the latency performance a little; (3) even when we deploy 200 subscribers on a single VM, the increase of CPU/memory/SWAP utilization after 200 client launch is roughly 6~7%. All these observations confirm that our E2E design and implementation is very efficient; of course, one might wonder what will be the situation when there are 2000~10000 clients: of course, the performance might have non-neglectable degrade; but, we should notice that, in the real deployments, it is almost impossible to deploy so many clients on a single VM (or a PC).

In a short summary, the experiments confirm that our design and implementation does not cause significant overheads in terms of latency/CPU/memory/SWAP performance; and it supports scalability. However, we still notice that the performance of some new platforms (like HiveMQ [28]) out-performs that of Mosqutiio platform. However, it is beyond the scope of this study.

6 Conclusions and Future Work

This study has designed and implemented MQTT E2E scheme, using Mosquitto MQTT5.0 APIs. We have evaluated the performance of the implementations. The E2E channel building enforces the privacy protection against a curious broker. The evaluation confirms that building an E2E channel between a publisher and a subscriber only add non-significant latency. The contributions include: (1) It is the first MQTT E2E implementation using MQTT 5.0, (2) the performance of the implementation confirms that the extra E2E channel building in MQTT enhances the privacy protection while preserving the high efficiency of connection latency, (3) thorough evaluations on the latency, CPU, memory, SWAP utilization and the analysis on the impacts of increasing the number of clients confirm its scalability and efficiency.

We hope this study can encourage more further research on MQTT end-to-end security, MQTT 5.0 designs, MQTT5.0 implementations. Some interesting design challenges include E2E handling across multi-broker systems (MQTT supports bridging), MQTT resource discovery, and security challenges for integrating MQTT with other IoT schemes (like Apache NiFi).

Acknowledgements

We sincerely appreciate reviewers' insights of pointing out the presentation weaknesses and sharing their insights for future wok. This project is partially supported by the Ministry of Science and Technology, Taiwan, R.O.C. grant number MOST 111-2221-E-260 -009 -MY3.

References

[1] ISO/IEC 20922:2016, *Information technology -- Message Queuing Telemetry Transport (MQTT) v3.1.1*. https://www.iso.org/standard/69466.html, 2024/01/25 accessed.

[2] OASIS, *MQTT Version 5.0*, 07 March 2019. https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html, 2024/03/01 accessed.

[3] Eclipse, *An Eclipse Paho C client library for MQTT for Windows, Linux and MacOS*. https://github.com/eclipse/paho.mqtt.c, 2024/03/01 accessed.

[4] 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, 2024/03/07 accessed.

[5] S. Andy, B. Rahardjo, B. Hanindhito, Attack Scenarios and Security Analysis of MQTT Communication Protocol in IoT System, *Proc. 2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, Yogyakarta, Indonesia, 2017, pp. 1-5. https://doi.org/10.1109/EECSI.2017.8239179

[6] S. N. Firdous, Z. Baig, C. Valli, A. Ibrahim, Modelling and Evaluation of Malicious Attacks against the IoT MQTT Protoco, *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, Exeter, UK, 2017, pp. 748-755. https://doi.org/10.1109/iThings-GreenCom-CPSCom-SmartData.2017.115

[7] A. Niruntasukrat, C. Issariyapat, P. Pongpaibool, K. Meesublak, P. Aiumsupucgul, A. Panya, Authorization mechanism for MQTT-based Internet of Things, *2016 IEEE International Conference on Communications Workshops (ICC)*, Kuala Lumpur, Malaysia, 2016, pp. 290-295. https://doi.org/10.1109/ICCW.2016.7503802

[8] S. H. Shin, K. Kobara, C. C. Chuang, W. C. Huang, A Security Framework for MQTT, *2016 IEEE Conference on Communications and Network Security (CNS): International Workshop on Cyber-Physical Systems Security (CPS-Sec)*, Philadelphia, PA, USA, 2016, pp. 1-5. https://doi.org/10.1109/CNS.2016.7860532

[9] A. Bhawiyuga, M. Data, A. Warda, Architectural Design of Token based Authentication of MQTT Protocol in Constrained IoT Device, *2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA)*, Lombok, Indonesia, 2017, pp. 1-4. https://doi.org/10.1109/TSSA.2017.8272933

[10] 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, 2016, pp. 1-6. https://doi.org/10.1109/SYSCO.2016.7831326

[11] M. Singh, M. A. Rajan, V. L. Shivraj, P. Balamuralidhar, Secure mqtt for internet of things (iot), *2015 Fifth International Conference on Communication Systems and Network Technologies*, Gwalior, India, 2015, pp. 746-751. https://doi.org/10.1109/CSNT.2015.16

[12] H. Y. Chien, Highly Efficient Anonymous IoT Authentication Using Composite Hashing, *The 2021 IEEE Conference on Dependable and Secure Computing*, Aizuwakamatsu, Fukushima, Japan, 2021, pp. 1-7. https://doi.org/10.1109/DSC49826.2021.9346263

[13] H. Y. Chien, Two-Level-Composite-Hashing Facilitating Highly Efficient Anonymous IoT and D2D Authentication, *MDPI Electronics*, Vol. 10, No. 7, Article No. 789, April, 2021. https://doi.org/10.3390/electronics10070789

[14] C. Lesjak, D. Hein, M. Hofmann, M. Maritsch, A. Aldrian, P. Priller, T. Ebner, T. Ruprechter, G. Pregartne, Securing Smart Maintenance Services: Hardware-Security and TLS for MQTT, *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, Cambridge, UK, 2015, pp. 1243-1250. https://doi.org/10.1109/INDIN.2015.7281913

[15] A. Rizzardi, S. Sicari, D. Miorandi, A. Coen-Porisini, AUPS: An Open Source Authenticated Publish/Subscribe system for the Internet of Things, *Information Systems*, Vol. 62, pp. 29-41, December, 2016. https://doi.org/10.1016/j.is.2016.05.004

[16] R. Neisse, G. Steri, G. Baldini, Enforcement of security policy rules for the internet of things, *2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Larnaca, Cyprus, 2014, pp. 165-172. https://doi.org/10.1109/WiMOB.2014.6962166

[17] H. Y. Chien, Design of End-to-End Security for MQTT 5, *The 4th International Conference on Science of Cyber Security - SciSec 2022*, Matsue city, Shimane, Japan, 2022, pp. 353-363. https://doi.org/10.1007/978-3-031-17551-0_23

[18] H. Lee, J. Lim, T. Kwon, MQTLS: Toward Secure MQTT Communication with an Untrusted Broker, *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, Jeju, Korea (South), 2019, pp. 53-58. https://doi.org/10.1109/ICTC46691.2019.8940001

[19] M. Hamad, A. Finkenzeller, H. Liu, J. Lauinger, V. Prevelakis, S. Steinhorst, SEEMQTT: Secure End-to-End MQTT-Based Communication for Mobile IoT Systems Using Secret Sharing and Trust Delegation, *IEEE Internet of Things Journal*, Vol. 10, No. 4, pp. 3384-3406, February, 2023. https://doi.org/10.1109/JIOT.2022.3221857

[20] H. Y. Chien, P. P. Ciou, Design and Implementation of Efficient IoT Authentication Schemes for MQTT 5.0, *Journal of Internet Technology*, Vol. 24, No. 3, pp. 665-674, May, 2023. https://doi.org/10.53106/160792642023052403012

[21] H. Y. Chien, N. Z. Wang, Y. M. Tseng, R. W. Hung, New group-key-based Over The Air (OTA) Update Model Facilitating security and efficiency Using MQTT 5.0, *13th International Conference on Frontier Computing (FC 2023)*, Tokyo, Japan, 2023, pp. 317-328. https://doi.org/10.1007/978-981-99-9342-0_34

[22] N. Z. Wang, H. Y. Chien, Design and Implementation of MQTT-based Over-The-Air Updating Against Curious Brokers, *IEEE Internet of Things Journal*, Vol. 11, No. 6, pp. 10768-10777, March, 2024. https://doi.org/10.1109/JIOT.2023.3327447

[23] A.-T. Shih, H.-Y. Chien, Y.-M. Huang, Enhanced Authentication and End-to-End Encryption in MQTT 5.0: Using Elliptic Curve for Design Optimization and Analysis, *2024 IEEE 4th International Conference on Electronic Communications, Internet of Things and Big Data (ICEIB)*, Taipei, Taiwan, 2024, Oral Paper B240017.

[24] Mosquitto Eclipse, *Eclipse Mosquitto - An open source MQTT broker*. https://github.com/eclipse/mosquitto, 2024/02/11 accessed.

[25] OpenSSL, *OpenSSL*. https://www.openssl.org/, 2024/03/01 access.

[26] GitHub, *GitHub - json-c/json-c*. https://github.com/json-c/json-c, 2024/03/01 accessed.

[27] VMware, *VMware Workstation Player*. https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html, 2023/08/18 accessed.

[28] HiveMQ vs. Mosquitto: *An MQTT Broker Comparison*. https://www.hivemq.com/blog/hivemq-vs-mosquitto-an-mqtt-broker-comparison/, 2024/12/25 accessed.

Biographies

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

An-Tong Shih received his B.S. degree in Information Management from Chaoyang University of Technology, Taiwan, in 2015, and his M.S. degree in the same field from the same university in 2017. He is currently a Ph.D. candidate in Information Engineering at National Chi Nan University, Taiwan. His research interests are focused on cryptography, machine learning, and IoT security.

Yuh-Ming Huang received his B.S. degree in mathematics from National Tsing Hua University, Hsinchu Taiwan, in 1987 and his M.S. and Ph.D. degrees in computer science and information engineering from National Taiwan University, Taipei Taiwan, in 1989 and 1999, respectively. He is currently an Associate Professor in the Department of Computer Science and Information Engineering at National Chi Nan University, Nantou Taiwan. His current research interests include data compression, error correction coding, joint source/channel coding, data hiding codes, video encryption, and network coding.