

MQTLS: Toward Secure MQTT Communication with an Untrusted Broker

Hyunwoo Lee
Computer Science and Engineering
Seoul National University
Seoul, Korea
hwlee2014@mmlab.snu.ac.kr

Junghwan Lim
Computer Science and Engineering
Seoul National University
Seoul, Korea
jhlhm@mmlab.snu.ac.kr

Ted “Taekyoung” Kwon
Computer Science and Engineering
Seoul National University
Seoul, Korea
tkkwon98@gmail.com

Abstract—The publisher–subscriber (pub/sub) paradigm is one of the promising communication models to meet the requirements such as interoperability and support of heterogeneity for the Internet-of-Things (IoT). Message Queueing Telemetry Transport (MQTT), one of the protocols based on the pub/sub model, is widely used in practice with public brokers by Eclipse Mosquitto or HiveMQ. A broker in the pub/sub model, however, is intrinsically a man-in-the-middle between clients (i.e., publishers and subscribers); thus, a broker can read and alter all the messages delivered from publishers to subscribers. Therefore, both publishers and subscribers should not exchange confidential messages via an untrusted broker or should use it in-between them at risk.

We propose MQTLS, an MQTT-aware secure communication protocol among publishers, subscribers, and brokers, while restricting brokers not to read any messages from publishers, except topics that need for message delivery. The MQTLS protocol is secure, based on our novel security definition called client-to-broker-to-client (CBC) security semantics for the pub/sub model. Our OpenSSL-based MQTLS proof-of-concept shows that MQTLS increases the delay time at the initial setup due to asymmetric signature/verification. However the delay time becomes marginal – only 4.06% at the high-end device and 10.04% at the low-end device compared with the standard TLS protocol – after the key is distributed.

Index Terms—publisher-subscriber model, MQTT, TLS

I. INTRODUCTION

The Internet-of-Things (IoT) is in the spotlight as the technology that enriches the future of mankind [6], [8]. With IoT, we expect automated sensing and actuating systems that lead to smart cities, smart factories, or smart homes. However, these raise several challenges such as interoperability and support of heterogeneity that should be resolved [5]. Therefore, the publisher-subscriber (pub/sub) paradigm becomes one of the promising communication models to integrate heterogeneous networks and interconnect many devices. For example, a broker that is located between all the publishers and subscribers may have diverse network interfaces such as WiFi or Zigbee; thus, devices with different interfaces can exchange data if they support the same application protocol. Note that among the protocols that adopt the pub/sub model, Message Queueing Telemetry Transport (MQTT) [15] is widely used in practice such as in Facebook Messenger [17] or BMW Mobility Services [9] due to its simplicity.

In MQTT, a broker gets all the messages from publishers and forwards the messages to subscribers. Therefore, both publishers and subscribers should not exchange confidential messages via an unreliable broker, or should use it at risk noting that a broker can read and manipulate all the messages. If a broker is compromised, the adversary can learn and leak all the messages that may include sensitive information (e.g., healthcare data [16]); not to mention, a compromised broker can manipulate sensing data that might lead to wrong decisions. Unfortunately, the security of the off-the-shelf broker such as HiveMQ [2] or Eclipse Mosquitto [1] mainly focuses on secure communications or authentication/authorization only between a broker and a client (i.e., a publisher or a subscriber), not that of between clients or that of a broker itself [3], [4].

We propose MQTLS, an MQTT-aware secure communication protocol in-between publishers and subscribers while limiting a broker not to access payloads in messages. With MQTLS, subscribers can authenticate a publisher when they subscribe to a topic and the confidentiality and the integrity of messages from the publisher are guaranteed. Therefore, MQTLS provides a way to communicate with peers even over untrusted brokers.

To show feasibility of MQTLS, we implement MQTLS with OpenSSL and evaluate the performance overhead compared with the standard TLS protocol. Numerical results show that the overhead is concentrated at the initial setup stage and after setup the load became marginal.

In sum up, our contribution points are as follows:

- We define a new client-to-broker-to-client (CBC) security semantics for the publish/subscribe model.
- We design an MQTT-aware TLS protocol, dubbed MQTLS, based on the CBC security semantics.
- We implement MQTLS with the OpenSSL library and show the performance overhead by MQTLS is practical.

The paper is organized as follows. We first introduce the concepts (§II) and explain the problems (§III). Next, we propose MQTLS (§IV), followed by its security analysis (§V) and performance evaluation (§VI). Finally, we discuss related work (§VII) and finalize with concluding remarks (§VIII).

II. BACKGROUND

A. Message Queueing Telemetry Transport (MQTT)

Message Queueing Telemetry Transport (MQTT) [15], one of the pub/sub communication protocols, is a simple and lightweight messaging protocol to be used in various situations such as constrained environments. In MQTT, there are three participants called *publishers*, *subscribers*, and *a broker* (or an MQTT server). We refer to both publishers and subscribers as *clients*. Publishers send messages, each of which consists of *a topic* and *a payload* (an actual content), to a broker and *subscribers* receive messages based on the topic they have subscribed, while a broker is used to store and forward messages from publishers to subscribers. There can be many subscribers per topic; thus, MQTT provides an efficient way to distribute messages in *one-to-many* communications.

Specifically, a client (a publisher or a subscriber) initiates the protocol with `CONNECT` to a broker. The broker then replies with `CONNECT ACK` to confirm that the connection is established. After that, the client sends a message indicating his intention according to his role. For example, the subscriber sends `SUBSCRIBE` that contains a topic to a broker, followed by the broker's `SUBSCRIBE ACK` to acknowledge the subscriber's subscription. Then, the subscriber can receive all the messages related to the topic via the confirmed connection until the subscriber explicitly sends `DISCONNECT` to the broker to close the connection. On the other hand, the publisher sends a `PUBLISH` message that includes a topic and a payload to the broker. On receiving the message, the broker inspects a topic in the message and forwards it to all the associated subscribers. Note that the broker needs to know only a topic, not a payload, to deliver messages.

B. Transport Layer Security

Transport Layer Security (TLS) [7], [14] is the most widely used security protocol on the Internet. The protocol guarantees end-to-end security with three main goals [14] – namely, *authentication*, *confidentiality*, and *integrity*. Authentication is a property that guarantees that the secure channel is established only with the intended peer. Confidentiality means that no one except two endpoints can see the messages between them, while integrity ensures that no one except two endpoints can modify the message on-the-fly.

To achieve the above goals, TLS consists of two sub-protocols called the TLS handshake protocol and the TLS record protocol. The former is the protocol of which the purpose is to establish shared secrets for encryption while authenticating each other. The latter is the protocol that is to exchange messages between two endpoints with confidentiality and integrity. Note that the RFC5246 document (TLS 1.2) [7] specifies how to extend the TLS protocol (i.e., by `ClientHello` and `ServerHello`).

III. PROBLEM SCOPE

Our scenario of interest. We consider a scenario that a group of clients, e.g., a user or a device, wants to communicate with each other via a broker. The messages can include

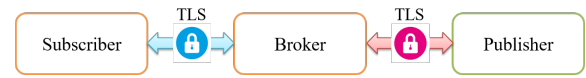


Fig. 1: **The current security model in the pub/sub model with TLS.** The security of the pub/sub model mainly relies on the TLS protocol only between a broker and a client. We call this scheme the standard TLS.

private messages; thus, publishers want to authorize subscribers. Examples can be online private chat rooms, healthcare sensors, or other sensing data in which the confidentiality and integrity of the data are critical requirements. We assume clients have their credentials, such as X.509 certificates or passwords, to identify themselves.

Status-quo. The current security in the MQTT system relies on the TLS protocol between a broker and a client [3], [4], as described in Figure 1. That is, when a publisher wants to distribute his message, he initially establishes a TLS session with a broker while authenticating the broker and sends the message to the broker. A subscriber who wants to fetch the message also establishes a TLS session with the broker and then gets the message with the relevant topic.

Threats in the status-quo. Note that the current security model only considers a passive network adversary that can monitor the network and an active network adversary that can insert, drop, alter, or reorder the packets on-the-fly, which are assumed threats in the TLS protocol. We should consider the security regarding the broker that stores and forwards all the messages between publishers and subscribers as well if messages include any sensitive information. There are two main problems, assuming an honest-but-curious broker as well as a passive and active network adversary on the current pub/sub model, as follows:

- **Passive attacks:** A broker can read all the messages received. In other words, a broker not only can *know* sensitive information but also can *leak* messages to the others who are not authorized on a specific topic.
- **Active attacks:** A broker can insert/delete/alter/reorder a message received from publishers. For example, a broker can *insert* false messages to subscribers, impersonating publishers; *delete* important control messages from publishers; *alter* messages with wrong information; or reorder messages to make a subscriber confused.

Out-of-scope attacks. We do not consider denial-of-service (DoS) attacks. In this regard, a modification on topics by a broker is not a concern since topics are only to be used for delivery and if they are modified, then subscribers cannot receive the message. We believe this is just a DoS attack without other threats.

IV. THE MQTLS PROTOCOL

In this section, we introduce MQTLS short for an MQTT-aware TLS protocol. Before specifying the protocol, we propose client-to-broker-to-client (CBC) security semantics that address the threats discussed in §III. The MQTLS protocol implements

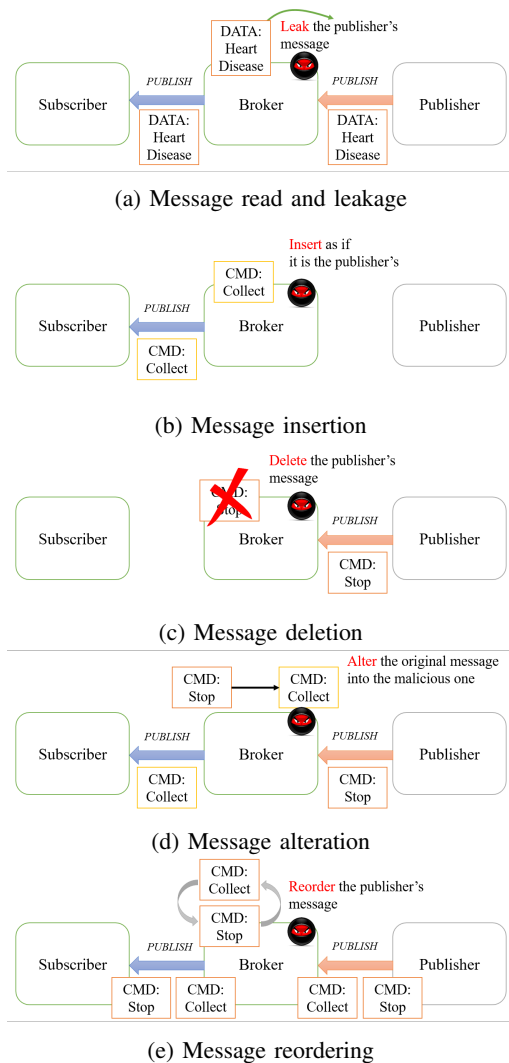


Fig. 2: **Threats in the current pub/sub model.** We consider both a passive adversary and an active adversary on a broker.

CBC security semantics in-between publishers and subscribers while giving the least privilege to a broker. Figure 3 shows the protocol, including the MQTLS handshake protocol as well as the MQTLS record protocol.

A. Client-to-broker-to-client (CBC) Security Semantics

Unlike a standard TLS scenario that includes only two parties, we consider three participants in communications; thus, we need to answer the following question:

What is the “end-to-end security” with a broker?

Answering the question, we consider communication between one publisher and one subscriber via a broker for brevity.

Client-to-broker. We can think “end-to-end” security between a client (i.e., a publisher or a subscriber) and a broker, which is the end-to-end security model that is generally applied in current practice (cf. Figure 1). No network attacker can see or modify messages between a client and a broker with a TLS

session, but a broker can still completely monitor payloads as well as topics in the messages. Note that the broker has excessive privilege since a broker does not need to read a payload to perform its functionality.

Client-to-client. On the other hand, let assume there is an “end-to-end” security session between a publisher and a subscriber. In this case, MQTT messages are delivered over a TLS session. Therefore, both a topic and a payload are encrypted between the publisher and the subscriber. Although a broker in the middle cannot perform any threats mentioned before, the broker cannot read a topic; thus, it cannot deliver any messages to a subscriber.

As can be seen above, it is difficult to define “end-to-end” with either of the above. Therefore, we propose a new notion of “end-to-end” security for the pub/sub model, called *client-to-broker-to-client (CBC) security*.

Client-to-broker-to-client. In this security model, we focus on the fact that a broker can perform its role without reading a payload. Hence, we apply the notion of end-to-end security differently for a topic and a payload. A topic should be protected between a client and a broker, while a payload should be encrypted between a publisher and a subscriber.

B. Overview of MQTLS

We propose MQTLS that implements CBC security semantics in MQTT. MQTLS is designed based on the following security goals to make CBC security semantics feasible.

- **Entity Authentication (G1):** Clients should be able to authenticate a broker as well as a peer client on subscribing to a topic (to confirm and to authorize each other).
- **Payload confidentiality and integrity (G2):** Confidentiality and integrity should be guaranteed between clients.

Together with the above security goals, we further consider the followings as well.

- **Minimal networking overhead (G3):** A new protocol should not increase the number of round trips in executing the protocol.
- **Minimal impact on applications (G4):** Revisions on applications to deploy a new protocol should be minimal.

Before explaining the MQTLS handshake protocol and the MQTLS record protocol in detail, we specify essential factors used in the protocol below:

- **Payload encryption key (🔑):** A payload encryption key is used to encrypt a payload. The key is generated by a publisher at the initial state and should be delivered to all the subscribers who want to subscribe to the topic.
- **Payload sequence number:** A payload sequence number is used to count the number of messages sent by a publisher. Each PUBLISH message includes a unique payload sequence number.
- **Topic encryption key (🔑, 🔑):** A topic encryption key is used to encrypt a topic (and also an encrypted payload and a payload sequence number); thus, the key is established between a client and a broker.
- **One-time delivery key (🔑):** A one-time delivery key is uniquely established per a pair of a publisher and a

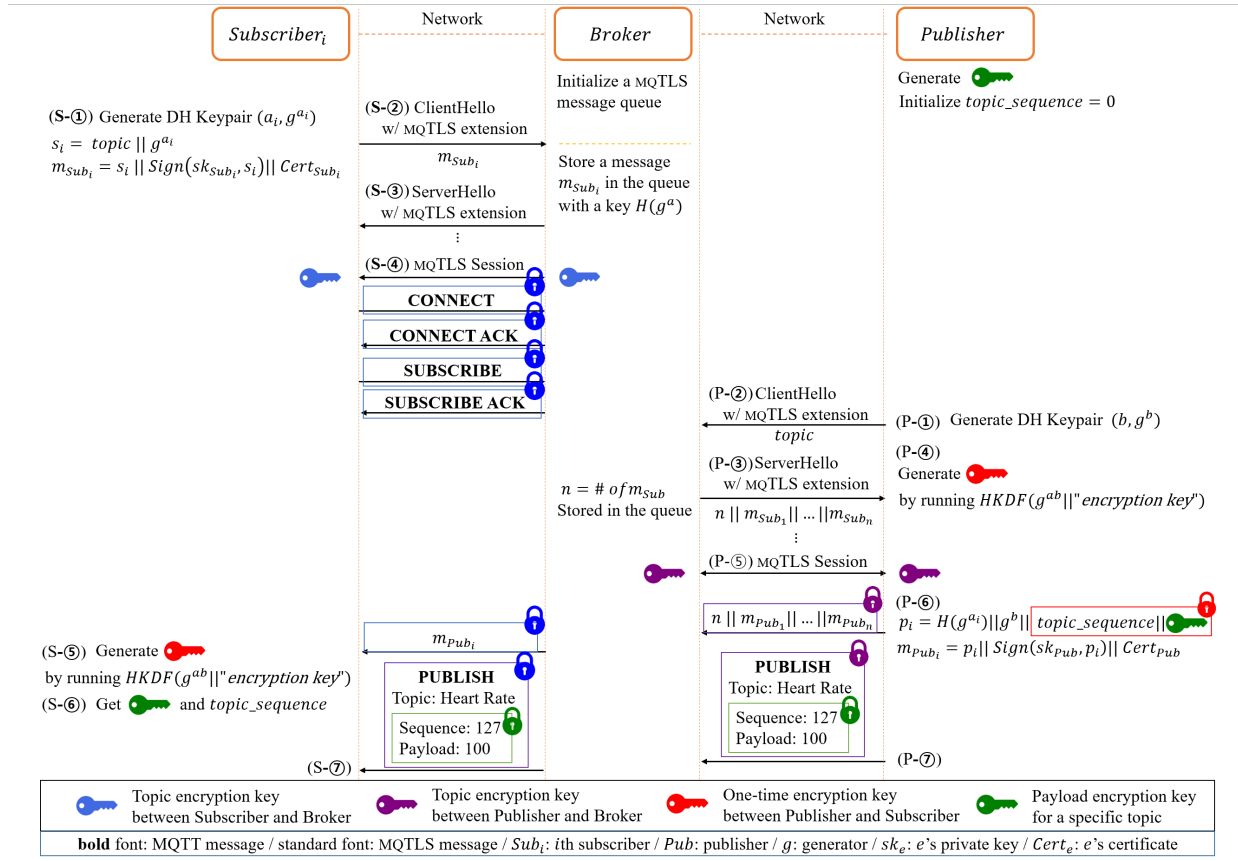


Fig. 3: **The MQTTS Protocol.** During MQTTS handshake protocol, each of a publisher and a subscriber establishes a topic encryption key with a broker while the publisher and the subscriber share a one-time encryption key and a payload sequence number. By using the payload encryption key with the payload sequence number, the publisher can send sensitive messages securely to the subscriber without revealing the messages to the broker.

subscriber. The key is used to encrypt/deliver a payload encryption key with a payload sequence number.

C. The MQTTS Handshake Protocol

The objectives of the MQTTS handshake protocol is to establish a topic encryption key as well as a one-time delivery key and to deliver a payload encryption key and a payload sequence number encrypted with the one-time delivery key while authenticating a broker as well as the peer client. The MQTTS handshake protocol is executed differently depending on the role of the client; thus, we describe the protocol below, separated by the role in chronological order.

Subscriber behavior (S-① – S-④). A subscriber who would subscribe to the topic first generates an ephemeral Diffie-Hellman (DH) keypair (say, (a, g^a)) to establish a *one-time delivery key* with a publisher (S-①). Then, the subscriber sends **ClientHello** (S-②) with an MQTTS extension constant (to negotiate the use of MQTTS with a broker and a subscriber) and a message m_{Sub_i} ¹ that includes his interested topic (*topic*), a DH public key (g^a), a signature over $topic || g^a$, and a subscriber's certificate. Then, the broker inserts m_{Sub_i} into

its message queue and responds with **ServerHello** (S-③) including the MQTTS extension constant. The subsequent handshake messages, such as **Certificate** that contains a broker's certificate, are the same as those in the standard TLS protocol [7], [14] and the TLS session between the subscriber and the broker is finally established. The resulting session key is used as a *topic encryption key*. Now, the subscriber subscribes to the topic with the MQTT control messages followed (S-④).

Publisher behavior (P-① – P-⑥). Initially, a publisher creates a topic with a new payload encryption key and a payload sequence number initialized with 0. Whenever the publisher wants to send a message to a broker, the publisher executes the MQTTS handshake protocol with the broker. Initially, the publisher generates an ephemeral DH keypair (say, (b, g^b)) (P-①) and sends **ClientHello** with the MQTTS extension constant and her topic (*topic*) (P-②). Next, the broker responds with a series of m_{Sub_i} that relates to the *topic* from its message queue (P-③). Let n denote the number of m_{Sub_i} received. A publisher authenticates n subscribers by verifying each signature in m_{Sub_i} and finally generates each one-time delivery key from m_{Sub_i} by running a HMAC-based Extract-and-expand Key Derivation Function (HKDF) [10]

¹ i is an index to represent multiple subscribers

over g^{ab} with a label, “encryption key” (P-④). The subsequent handshake messages, such as `Certificate` that contains a broker’s certificate, follow the standard protocol and finally a topic encryption key is established between the broker and the publisher (P-⑤). A publisher then generates m_{Pub_i} (as a response to m_{Sub_i}) that contains a hash value of the subscriber’s DH public key ($H(g^a)$), her DH public key (g^b), a payload sequence number and a payload encryption key encrypted with an one-time delivery key, a signature over $H(g^a)\|g^b\|payload_sequence\|payload_encryption_key$, and her certificate ($Cert_{Pub}$) (P-⑥).

Subscriber behavior (S-⑤ – S-⑥). The broker forwards m_{Pub_i} to the subscriber i before a PUBLISH message. On receiving m_{Pub_i} , the subscriber generates a one-time delivery key (S-⑤) and finally gets a payload sequence number and a payload encryption key (S-⑥).

D. The MQTLS Record Protocol

The MQTLS record protocol is responsible for encrypting/decrypting a topic and a payload in a PUBLISH message complying CBC security semantics. Whenever a publisher wants to send a message, the publisher first prepends a payload with a payload sequence number ($payload_sequence_number\|payload$), encrypts the resulting message by an authenticated encryption (AE) algorithm (e.g., AES-GCM) with a payload encryption key, and sends it encrypted with a topic encryption key (P-⑦). A broker that receives the message decrypts it with a topic encryption key between a publisher and a broker, checks the topic in the message, and finally forwards the message to all the associated subscribers with the message encrypted with topic encryption keys established between the broker and the subscribers (S-⑦). The receiving subscribers first decrypt the message and then check whether the payload sequence number is right.

Note that a publisher needs to deliver a payload encryption key to a subscriber only once. In other words, if the payload encryption key is delivered to all the subscriber and there is no new subscriber added between two consecutive PUBLISH messages, only P-②, P-③, P-⑤, and P-⑦ are executed.

During the MQTLS handshake protocol, a client authenticates a peer (by `ClientHello` and `ServerHello`) and a broker (by `Certificate`); thus, entity authentication (G1) is achieved. Since all the published messages are encrypted by an authenticated encryption algorithm with a key that a broker does not know, confidentiality and integrity of the messages are guaranteed (G2). Furthermore, we only utilize the existing message flights in TLS and MQTT protocol without any additional round trips (G3). The application should notify the MQTLS layer of its topic and role; thus, revision on the application is required for MQTLS. However, we believe it is minimal since we can implement our MQTLS with Mosquitto [1] only adding four line-of-codes in clients and no line-of-codes in a broker (G4).

V. SECURITY ANALYSIS

In this section, we show how the MQTLS protocol addresses the problems discussed in §III.

Message read and leakage (Figure 2a): A broker cannot read and leak messages. It is natural since all the payloads are encrypted with a payload encryption key of which a broker is not aware. Thus, clients, i.e., publishers and subscribers, need not worry about information leakages.

Message insertion (Figure 2b): A broker cannot insert a message, impersonating a related publisher. It is because the broker does not know the payload encryption key; thus, the broker cannot make encrypted payload.

Message deletion (Figure 2c): A broker can delete an arbitrary message by just discarding or not responding to the message. However, subscribers would know that integrity is broken when they receive the message next to the discarded message since the payload sequence number is mismatched.

Message alteration (Figure 2d): Although a broker can tamper with a message, a subscriber can detect the misbehavior. It is because a payload is secret with authenticated encryption; thus, the MAC value is mismatched if the broker has modified the message.

Message reordering (Figure 2e): Like the message deletion and the message alteration, a broker can reorder the messages but a subscriber quickly detects it. The subscriber can simply check the reordered messages by using the payload sequence numbers inside the messages.

VI. EVALUATION

With addressing the security problems discussed in §V, we evaluate the performance overhead of MQTLS regarding time delay and CPU overhead, focusing on publishing messages at a publisher-side. We also see how the performance overhead increased with regard to the number of subscribers.

Implementation. We implement MQTLS by using OpenSSL-1.0.2o and combine it with Eclipse Mosquitto [1], an open-source MQTT broker and clients. Note that we do not need to revise a broker application, while we only add 4 line-of-codes to client applications. It is to indicate a topic and its role (i.e., a publisher or a subscriber) to the MQTLS layer. The ciphersuite used for our evaluation is ECDHE-ECDSA-AES-128-GCM-SHA256. Furthermore, we generate DH keypairs over a NIST P-256 elliptic curve, use AES-128-GCM for payload encryption, and utilize SHA-256 for the hash algorithm.

Testbed. We build our testbed comprised of two publishers, a broker, and subscribers. We use a laptop with Intel i7 Core at 2.90GHz with 8GB RAM for a high-end publisher, a Raspberry Pi 3B+ with ARM core A53 at 1.4GHz with 1GB RAM for a low-end publisher, a PC with Intel i5 Core at 3.30GHz with 16GB RAM for a broker, and an AWS instance with Intel Xeon at 2.5GHz with 1GB RAM for a subscriber.

Delay time and CPU processing overhead (Figure 4). We first evaluate the time required to send a PUBLISH message from a publisher (P-① to P-⑤) as well as the CPU processing time. The result shows that the overhead is increased 6.58% (20.67ms) at the high-end publisher and 99.88% (106.57ms)

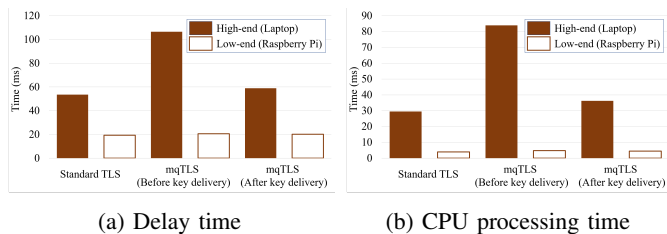


Fig. 4: Delay time and CPU processing overhead. The result shows that the overhead is mainly due to computation.

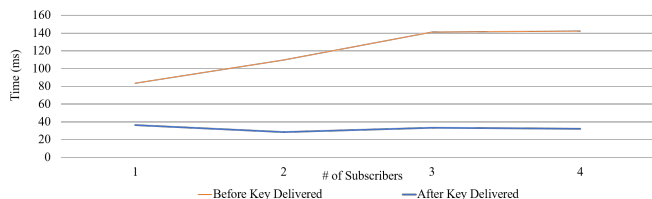


Fig. 5: Delay time with regard to the number of subscribers. The delay time is increased before a payload encryption key is delivered due to the number of asymmetric signatures is increased. However, after the key is delivered, the delay time becomes steady regardless of the number of subscribers.

at the low-end publisher compared with the standard TLS (19.38ms at the high-end, 53.59ms at the low-end) before the payload encryption key is delivered. However, once the payload encryption key is delivered to a subscriber, the increment due to MQTLS compared with the standard TLS is decreased to 4.06% (20.17ms) at the high-end and 10.04% (58.97ms). Note that the trend of the delay time and the CPU processing time is similar, meaning that the overhead is mainly due to computation. The main culprit of the increment is asymmetric signature generation and verification to establish a one-time delivery key, which requires heavy computation. This explains why the low-end device shows further increment.

Delay time with regard to the number of subscribers (Figure 5). We also measure the delay time with regard to the number of subscribers at the low-end device. The result shows that the delay time is increased before a payload encryption key is delivered due to the number of asymmetric signature/verification is increased. However, after the key is delivered, the delay time becomes stable regardless of the number of subscribers.

Takeaway: we can conclude that the overhead of MQTLS is concentrated on delivering a payload encryption key. However, once the key is delivered, MQTLS is performed without a significant load while achieving security goals discussed earlier.

VII. RELATED WORK

There have been several studies to establish a TLS session with multi-parties, including middleboxes, but none of them is based on the pub/sub model. Multi-context TLS (mcTLS) [13] applies the least privilege principle to middleboxes by sharing the MAC keys according to their permission. For example, read-only middleboxes only get a read

MAC key, while read/write middleboxes possess both a read MAC key and a write MAC key. We apply a similar concept to a broker to read only a topic. Middlebox TLS (mbTLS) [12] combines remote attestation with TLS to securely introduce middleboxes. Note that mbTLS requires middleboxes run over trusted execution environment (TEE). Middlebox-aware TLS (maTLS) [11] proposes auditable middleboxes that possess middlebox certificates in the transparency log servers. End-points can be aware of middleboxes by their certificates and verify their behavior with audit mechanisms.

VIII. CONCLUSION

We propose MQTLS that address threats due to untrusted brokers in MQTT. The MQTLS protocol implements CBC security semantics that makes “end-to-end” security feasible between publishers and subscribers, while restricting a broker not to access to any sensitive information. Numerical results show that the performance overhead of MQTLS is large in the early stages, but after a payload encryption key is distributed, we see the performance overhead became marginal.

REFERENCES

- [1] “Eclipse mosquito,” <https://mosquitto.org>, accessed: 2019-07-28.
- [2] “Hivemq: Reliable data movement for connected devices,” <https://www.hivemq.com/>, accessed: 2019-07-30.
- [3] “mosquitto-tls man page,” <https://mosquitto.org/man/mosquitto-tls-7.html>, accessed: 2019-07-23.
- [4] “Mqtt security fundamentals,” <https://www.hivemq.com/mqtt-security-fundamentals/>, accessed: 2019-07-23.
- [5] Z. H. Ali, H. A. Ali, and M. M. Badawy, “Internet of things (iot): definitions, challenges and recent research directions,” *International Journal of Computer Applications*, vol. 975, p. 8887, 2015.
- [6] S. Chen, H. Xu, D. Liu, B. Hu, and H. Wang, “A vision of iot: Applications, challenges, and opportunities with china perspective,” *IEEE Internet of Things journal*, vol. 1, no. 4, pp. 349–359, 2014.
- [7] T. Dierks, “The Transport Layer Security (TLS) protocol version 1.2,” IETF, RFC 5246, 2008.
- [8] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of things (iot): A vision, architectural elements, and future directions,” *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.
- [9] HiveMQ, “Bmw mobility services,” <https://www.hivemq.com/case-studies/bmw-mobility-services/>, accessed: 2019-07-30.
- [10] H. Krawczyk and P. Eronen, “Hmac-based extract-and-expand key derivation function (hkdf),” <https://tools.ietf.org/html/rfc5869>, 2010, accessed: 2019-07-28.
- [11] H. Lee, Z. Smith, J. Lim, G. Choi, S. Chun, T. Chung, and T. T. Kwon, “maTLS: How to make TLS middlebox-aware?” in *Network and Distributed System Security Symposium (NDSS)*, San Diego, USA, February 2019.
- [12] D. Naylor, R. Li, C. Gkantsidis, T. Karagiannis, and P. Steenkiste, “And then there were more: Secure communication for more than two parties,” in *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. ACM, 2017, pp. 88–100.
- [13] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. R. López, K. Papagiannaki, P. R. Rodriguez, and P. Steenkiste, “Multi-context tls (mctls): Enabling secure in-network functionality in tls,” in *SIGCOMM Computer Communication Review*. ACM, 2015.
- [14] E. Rescorla, “The Transport Layer Security (TLS) protocol version 1.3,” IETF, RFC 8446, 2018.
- [15] O. Standard, “Mqtt version 3.1.1,” <http://docs.oasis-open.org/mqtt/mqtt/v3>, 2014, accessed: 2019-07-28.
- [16] R. Wadhwa, A. Mehra, P. Singh, and M. Singh, “A pub/sub based architecture to support public healthcare data exchange,” in *2015 7th International Conference on Communication Systems and Networks (COMSNETS)*. IEEE, 2015.
- [17] L. Zhang, “Building facebook messenger,” <https://www.facebook.com/notes/facebook-engineering/building-facebook-messenger/10150259350998920/>, accessed: 2019-07-30.