

Article

Signcryption Facilitating End-to-End Security with Source Authentication in Broker-Based Internet of Things (IoT) Systems

Hung-Yu Chien

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

Abstract

Broker-based Internet of Things (IoT) systems, such as Message Queuing Telemetry Transport (MQTT), typically focus on authentication, privacy, and integrity protection between a client and its broker. Some enhancements to broker-based systems address end-to-end (E2E) security against an honest-but-curious broker. However, these conventional E2E solutions overlook a critical limitation: they do not provide source authentication that enables a subscriber to explicitly authenticate the sender. This study modifies the Mohamed–Elkamchouchi signcryption scheme and applies it to achieve E2E security with source authentication. The analysis shows that the proposed approach enhances security functionality while preserving efficiency.

Keywords: Internet of Things; end-to-end security; MQTT; authentication; privacy; source authentication; CoAP

1. Introduction

Broker-based Internet of Things (IoT) systems are attractive for many IoT scenarios because of their message–transmission decoupling property. MQTT (Message Queuing Telemetry Transport) [1–3] and CoAP Pub-Sub (Constrained Application Protocol Publish-Subscribe) [4] are two popular broker-based IoT protocols. An MQTT/CoAP Pub-Sub system consists of three types of entities: publishers, brokers, and subscribers. Publishers post messages to specific topics/resources, and brokers forward the messages to subscribers that have subscribed or registered for those topics/resources. Their lightweight designs provide simplicity and efficiency while offering limited native security support. The MQTT and CoAP Pub-Sub standards recommend adopting TLS [5] or DTLS [6] at the transport layer to provide basic security properties, such as authentication, privacy, and integrity, between a client (publisher or subscriber) and its broker. Although this approach provides basic security support, it does not provide end-to-end (E2E) security between a publisher and a subscriber. Therefore, MQTT E2E security mechanisms, such as those in [7–17], have been studied. Conventional MQTT E2E security mechanisms, such as [14–17], focus on authentication, privacy, and key agreement between publishers and subscribers, as well as privacy against an honest-but-curious broker. Chien et al. [13] further designed group-key-based E2E solutions to address potential broker overhead. Their study highlights the double-encryption issue and the broker–decrypt–re-encrypt (BDR) overhead of conventional MQTT E2E models. The double-encryption issue arises when a publisher first encrypts a message using an E2E channel key and then re-encrypts the

Academic Editor: Guosheng Xu

Received: 28 April 2026

Revised: 3 July 2026

Accepted: 12 July 2026

Published: 15 July 2026

Copyright: © 2026 by the author.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

resulting ciphertext using the publisher–broker channel key; the same issue occurs in the broker–subscriber channel. BDR overhead refers to the cost incurred when a broker decrypts ciphertext received from a publisher using the publisher–broker channel key and then re-encrypts the plaintext using the broker–subscriber channel key. They explored four approaches to improving security and efficiency; however, these solutions do not address source authentication. Source authentication concerns authenticating the sender of a message when it is received by a broker or subscriber. In group-key-based E2E solutions, a receiver, especially a subscriber, cannot verify the precise source of a message because any authorized group member can generate the same authentication data. Source authentication is particularly important in Industrial Internet of Things (IIoT) scenarios, where applications must verify the origin of messages.

Regarding source authentication, a publisher could attach a digital signature to its message and then follow the MQTT/TLS framework to provide source authentication and privacy protection. However, this approach does not provide the desired E2E security because the broker can access the plaintext content.

Yuliang Zheng (1997) [18] proposed the first signcryption scheme, whose goal is to perform digital signing (authenticity, integrity, and non-repudiation) and encryption (confidentiality) in a single logical step, replacing the traditional signature-then-encryption workflow. By integrating these two functions, signcryption can reduce both the computational cost and communication overhead. Later, Mohamed and Elkamchouchi (2009) [19] proposed an elliptic curve cryptography (ECC)-based signcryption design that, in addition to the core properties of signcryption, provides forward secrecy and encrypted-message authentication/ciphertext authentication (i.e., verifiability while the message remains encrypted). These features align well with the E2E security requirements, including source authentication and privacy against an honest-but-curious broker, of the broker-based IoT systems considered in this work. However, directly applying signcryption to an MQTT message would require a publisher to prepare an individual signcryption for each subscriber, resulting in two weaknesses: publisher–subscriber coupling and inefficiency. We therefore modify the Mohamed–Elkamchouchi signcryption scheme and apply it to the conventional MQTT/TLS model.

The rest of this paper is organized as follows. Section 2 discusses the related work. Section 3 investigates potential models and proposes our new approach. Section 4 examines performance and provides analytical comparisons. Section 5 presents the conclusions and future work.

2. Related Work

MQTT (Message Queuing Telemetry Transport) is a publish/subscribe messaging protocol implemented in a client–server architecture, where a broker acts as an intermediary responsible for maintaining client connections, receiving published messages, and forwarding them to the appropriate clients according to subscription relationships. This design decouples “data producers” from “data consumers,” thereby reducing endpoint coupling and lowering communication complexity in unreliable network environments. MQTT is designed to be lightweight and easy to implement. From a standardization perspective, MQTT v3.1.1 [2] is an OASIS standard and has been adopted as the international standard ISO/IEC 20922:2016; MQTT v5.0 [1] introduces more fine-grained error reporting and extensible properties to improve observability and extensibility.

MQTT security, as defined by the standards, largely follows a “transport-layer protection plus implementation-defined endpoint authentication/authorization” paradigm. The MQTT specifications do not define new cryptographic algorithms; instead, they recommend using TLS [5] to provide confidentiality and integrity protection between a client and the broker. They also permit client authentication through TLS client certificates, ei-

ther in combination with or as a replacement for application-layer credentials, such as the Username and Password fields in the MQTT CONNECT packet. MQTT v5.0 introduced Enhanced Authentication, including the AUTH packet, to support challenge–response and multi-round authentication exchanges, thereby providing greater flexibility for authentication and key-establishment workflows within MQTT. However, authorization, especially topic-level access control, is intentionally left to broker implementations. Although the OASIS MQTT standard defines the operation of SUBSCRIBE and SUBACK packets, it does not specify how authorization decisions are made. In RFC 9431, the ACE/OAuth profile for MQTT and OAuth 2.0 [20], the broker is responsible for authorizing SUBSCRIBE requests.

In addition to standardized TLS, studies such as [15–17] investigate ultra-lightweight cryptography for resource-constrained devices to reduce computational cost and protocol overhead. Another research direction focuses on MQTT E2E security [7–14]. Conventional MQTT models typically assume a trusted broker. Because TLS mainly provides hop-by-hop protection between a publisher and broker and between a broker and subscriber, studies such as [7–13] propose message-level end-to-end encryption for environments in which the broker is not fully trusted. Singh et al. [17] use ciphertext-policy attribute-based encryption (CP-ABE) [21] to achieve fine-grained access control with end-to-end payload encryption. Chien et al. [9,13] designed, implemented, and evaluated an MQTT E2E scheme using MQTT 5.0. Mektoubi et al. [11] adopt a topic certificate whose public key is used to encrypt messages for the topic, whereas the corresponding private key is used to decrypt the ciphertexts. In MQTSL [22], a publisher initiates E2E key distribution using ephemeral Diffie–Hellman session keys shared between the publisher and subscriber, and OpenSSL-based extensions facilitate E2E topic-key distribution. However, these schemes do not consider source authentication and do not maintain efficiency as the system scales.

The SEEMQTT E2E scheme [12], which is based on secret sharing, allows a publisher to delegate its decryption authority to a pool of key stores. Authorized subscribers contact the key stores, obtain verification, and then reconstruct the decryption key. Although this approach delegates subscriber verification to a pool of key stores, it introduces substantial additional overhead.

In 2025, Chien et al. [13] discussed two performance weaknesses of the conventional broker-based MQTT E2E model: double encryption on client–broker channels and the BDR burden. Using a group-key mechanism and a client–broker-channel-integrity-only mechanism, they proposed a new scheme that satisfies both E2E security and efficiency requirements. However, their scheme does not address source authentication. To the best of our knowledge, no existing solution simultaneously provides source authentication, E2E security (privacy and integrity), client–broker security (privacy and integrity), and efficiency.

MQTT-SN (MQTT for Sensor Networks) [23] is an optimized version of the MQTT protocol specifically designed for resource-constrained devices and low-bandwidth wireless networks. A conventional MQTT-SN system consists of clients, gateways, and brokers. An MQTT-SN publisher first communicates with an MQTT-SN gateway over UDP, typically protected by DTLS. The gateway translates MQTT-SN messages into standard MQTT semantics and forwards them to the MQTT broker, which then distributes the message to subscribed clients. In practice, MQTT-SN gateways can expose UDP and DTLS listeners, while the gateway-to-broker side commonly uses standard MQTT over TCP/TLS. From a security perspective, this architecture naturally provides channel security, but not automatically end-to-end security. Client–gateway security is typically achieved with DTLS, protecting the local transport hop between the MQTT-SN device and the gateway, while gateway–broker security is typically achieved with TLS, protecting the backend MQTT hop. However, this does not mean the publisher and subscriber have end-to-end confidentiality because the gateway and broker still see the application payload in

plaintext during forwarding or protocol translation. To achieve true publisher–subscriber end-to-end security, the payload must be protected at the application layer before it reaches the gateway so that the gateway and broker only relay ciphertext, while authorized subscribers decrypt it at the endpoint. In MQTT-SN gateway deployments, DTLS and TLS mainly address the hop-to-hop security, but not the E2E security and source authentication.

Another approach with similar goals combines publish-over-UDP, a pre-shared key (PSK), and authenticated encryption with associated data (AEAD) [24]. A publisher sends messages directly over UDP using a PSK and protects each message at the application layer with AEAD. Instead of relying on a connection-oriented secure channel, the publisher encrypts and authenticates the payload before transmission, while optional metadata, such as a topic ID, publisher ID, or sequence number, can be authenticated as associated data. A gateway or broker can forward the message based on visible routing information without accessing the plaintext. Authorized subscribers that hold the same security context or shared key verify the authentication tag, check freshness or anti-replay information, and then decrypt the protected payload. Therefore, a UDP + PSK + AEAD design may provide publisher–subscriber confidentiality and integrity at the application layer, even when no end-to-end secure transport channel is established. However, as the system scales, this approach introduces substantial PSK-management overhead.

The Constrained Application Protocol (CoAP) [4] is a lightweight web-transfer protocol designed for constrained devices and low-power, lossy networks. It follows a REST-style model similar to HTTP and uses methods such as GET, POST, PUT, and DELETE, but it is optimized for IoT environments with small messages and low overhead. CoAP Pub-Sub with a broker extends CoAP from a request/response model to a publish–subscribe architecture. In this design, publishers and subscribers interact through a broker that hosts topic resources, enabling loosely coupled communication and supporting devices with intermittent connectivity or limited uptime. This architecture is currently specified in an IETF Internet-Draft rather than a finalized RFC. Object Security for Constrained RESTful Environments (OSCORE) [25] is an application-layer security mechanism for CoAP. It protects CoAP messages end to end using COSE and provides confidentiality, integrity, and replay protection, even when messages pass through proxies or protocol translators. Authentication and Authorization for Constrained Environments (ACE) is an IoT-oriented authorization framework based on OAuth 2.0 and adapted for constrained environments such as CoAP-based systems. It defines how clients obtain access tokens and authorized security parameters so that resource-constrained devices can securely access protected services. With CoAP Pub-Sub, a broker, OSCORE, ACE, and Group OSCORE, end-to-end security is achieved through group-based object security rather than a direct secure channel between each publisher and subscriber [25,26]. The KDC/Group Manager in the ACE framework authorizes clients and provides the keying material required to establish a shared Group OSCORE security context. The publisher encrypts and authenticates the content before publishing it, the broker only forwards the protected message, and authorized subscribers decrypt and verify it upon receipt. Thus, the broker supports message distribution without requiring access to plaintext content. CoAP Pub-Sub is currently specified in an IETF Internet-Draft, whereas OSCORE and ACE are standardized components of the IETF constrained-environment security stack. Source authentication could be added to such a framework, but it is not a primary design objective.

3. Signcryption-Based MQTT E2E with Source Authentication

3.1. System Architecture and Threat Model

We aim to design a novel approach for a broker-based publish–subscribe IoT model that satisfies the following requirements:

- (1) Client–broker channel security, including authentication, privacy, and integrity.
- (2) Publisher–subscriber E2E security, including authentication, privacy, and integrity protection for E2E messages, as well as protection of sensitive data from an honest-but-curious broker.
- (3) Source authentication: In many group-key-based solutions, any group member, whether a publisher or subscriber, can generate group-key-based authentication codes; therefore, a receiver cannot verify the precise source of a message. Here, source authentication means that any receiver, including a broker or subscriber, can verify the source of a message.
- (4) Efficiency: The system maintains high performance as it scales.

The system consists of four types of entities: clients (publishers and subscribers), a broker, a key distribution center (KDC), and adversaries. The KDC is trusted, whereas the broker is honest-but-curious. An honest-but-curious broker faithfully executes the protocol but may attempt to inspect message content for its own benefit.

Adversaries may compromise clients and perform attacks such as eavesdropping, message modification, replay, and impersonation.

Our approach combines TLS-based client–broker connections with a modified Mohamed–Elkamchouchi signcryption scheme. Section 3.2 introduces the preliminaries, and Section 3.3 presents the proposed ideas and schemes. For the remainder of this section, we use MQTT as the underlying protocol to demonstrate the proposed approach.

3.2. Preliminaries

The notations used in this paper are introduced in Table 1.

Table 1. The notation.

P: publisher; S: subscriber; B: broker.	G: the generator for the elliptic curve field. q : a large prime. C: an elliptic curve over GF .
S_i : i th subscriber. KDC: key distribution center.	
v_P/PK_P : publisher's private/public key, where $v_P \in_R [1, \dots, q-1]$, and $PK_P = v_P G$. Similarly, v_S/PK_S and v_B/PK_B respectively denote private/public keys of subscriber/broker.	
$E_k(), D_k()$: symmetric key encryption/decryption using key k .	$Token_{client, KDC}$: an authorized token which is an encryption of data like the client ID, authorized topic/period, or the issuer's signature; the content is encrypted using the KDC public key.
$Sig_{v_P}()$: signature by P .	
$T_P/seq\#$: Timestamp/sequence number of P ;	K_{sess} : the session key between KDC and S_i .
$HMAC()$: message authentication code; $h()$: secure hash.	K_{KDCS_i} : the temporary key for S_i to authenticate itself to KDC.
CHL_{PB}/CHL_{BS_i} : the channel between P and B /between B and S_i . CHL_{PS_i} : the E2E channel between P and S_i .	
Topic: MQTT topic; PK_{Topic}/v_{Topic} : public/private key of Topic.	$MAC_{K_{KDCS_i}}$: a K_{KDCS_i} – based message authentication code.

3.2.1. Schnorr Digital Signature

Schnorr is an efficient digital signature scheme. Here, we review the ECC version of the Schnorr signature shown in Figure 1. The scheme consists of two APIs: Sign and Verif.

Sign of m by Publisher:= $\text{Sign}(m, v_P) \rightarrow (R, s)$ $\{ r \in_R [1, \dots, q-1], R = rG, c = h(R, m)$ $s = r + cv_P \bmod q, \text{ return } \sigma = (m, R, s)\}$	Verification of (m, R, s) by a recipient:= $\text{Verf}(m, R, s, PK_P) \rightarrow \text{TRUE/FALSE}$
--	--

Figure 1. ECC version of Schnorr digital signature.

3.2.2. The Modified Mohamed–Elkamchouchi Signcryption

Mohamed and Elkamchouchi (2009) [19] proposed an ECC-based signcryption design that, in addition to message confidentiality, non-repudiation, and unforgeability, provides forward secrecy and encrypted-message/ciphertext authentication (i.e., verifiability while the message remains encrypted). They designed the scheme for scenarios in which (1) a judge can resolve disputes by directly verifying the sender's signature on signcrypt messages without assistance from the sender or decryption of the message, or (2) a firewall can securely filter signcrypt messages without performing full unsigncryption to verify the sender's identity. We review the original Mohamed–Elkamchouchi scheme in Figure 2 and describe its steps below.

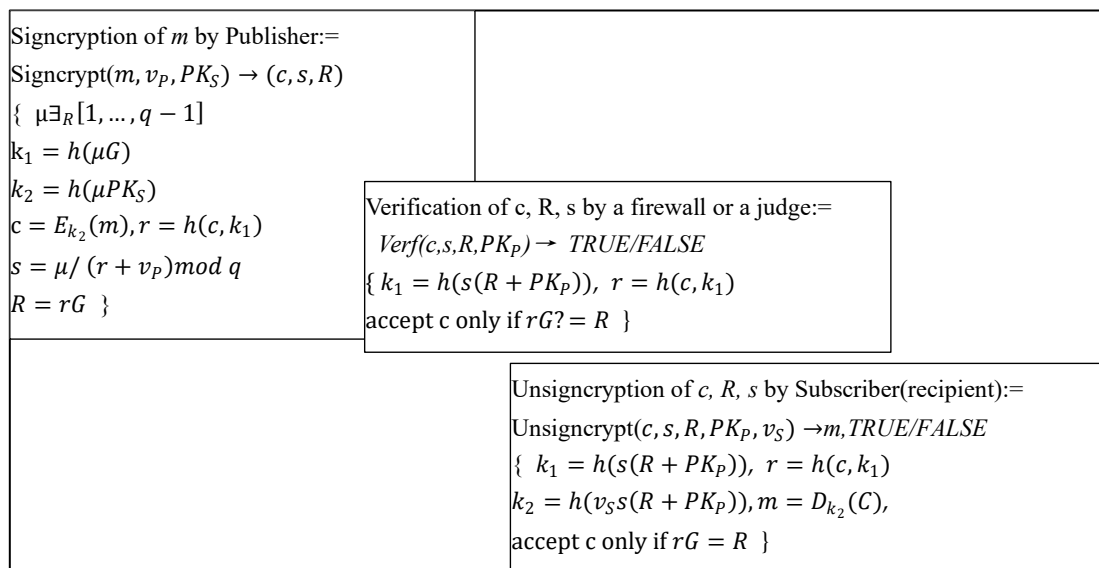


Figure 2. Mohamed–Elkamchouchi (ME) signcryption scheme.

Steps of the Mohamed–Elkamchouchi Signcryption Scheme

The publisher generates the signcryption on a message m as follows: (1) It chooses a random number $\mu \in_R [1, \dots, q-1]$ and computes $k_1 = h(\mu G)$ and $k_2 = h(\mu PK_S)$, where PK_S is the subscriber's public key. (2) It generates the ciphertext $c = E_{k_2}(m)$ using the key k_2 and has the hash $r = h(c, k_1)$. (3) It then uses its private key v_p to produce the signature $s = \mu / (r + v_p) \bmod q$ and $R = rG$. The signcryption is $\{c, R, s\}$.

A firewall (or a judge or a broker) performs the verification without decrypting the encryption as follows: (1) It computes $k_1 = h(s(R + PK_P))$ and $r = h(c, k_1)$, where PK_P is the publisher's public key. (2) It accepts the encryption by checking whether $rG = R$.

The designated receiver (subscriber) decrypts and verifies the signcryption as follows: (1) It calculates $k_1 = h(s(R + PK_P))$, $r = h(c, k_1)$, and $k_2 = h(v_S s(R + PK_P))$, where v_S is the subscriber's private key. (2) It decrypts the encryption as $m = D_{k_2}(C)$. It accepts the message by checking whether $rG = R$.

Based on the abstract signcryption APIs shown in Figure 2—($\text{Signcrypt}(m, v_a, P_b) \rightarrow (c, s, R)$; $\text{Verf}(c, s, R, P_a) \rightarrow \text{TRUE/False}$; $\text{Unsigncrypt}(c, s, R, P_a, v_b) \rightarrow m, \text{TRUE/FALSE}$)—we propose a new MQTT E2E source-authentication scheme. Although MQTT is used as the underlying IoT protocol to demonstrate the proposed ideas, the approach can also be applied to other IoT protocols.

Modified Mohamed–Elkamchouchi (ME) Signcryption

The preceding paragraph reviews the original Mohamed–Elkamchouchi signcryption scheme. In our application, however, the subscriber public key is replaced with the topic public key PK_{Topic} during signcryption generation, and the subscriber private key

is replaced with the topic private key v_{Topic} during unsigncryption. This arrangement provides two advantages: publisher–subscriber decoupling and the ability to prepare a single signcryption for all authorized subscribers.

3.3. The Three Models

We propose applying the modified Mohamed–Elkamchouchi (ME) signcryption to the MQTT/TLS model to achieve this study’s goals. To demonstrate the performance enhancement of the proposed model (approach), we will explore three models, of which two of them are classic models, and the third one is our proposed approach.

3.3.1. High-Level Designs of the Three Models

We introduce the three models (depicted in Figure 3).

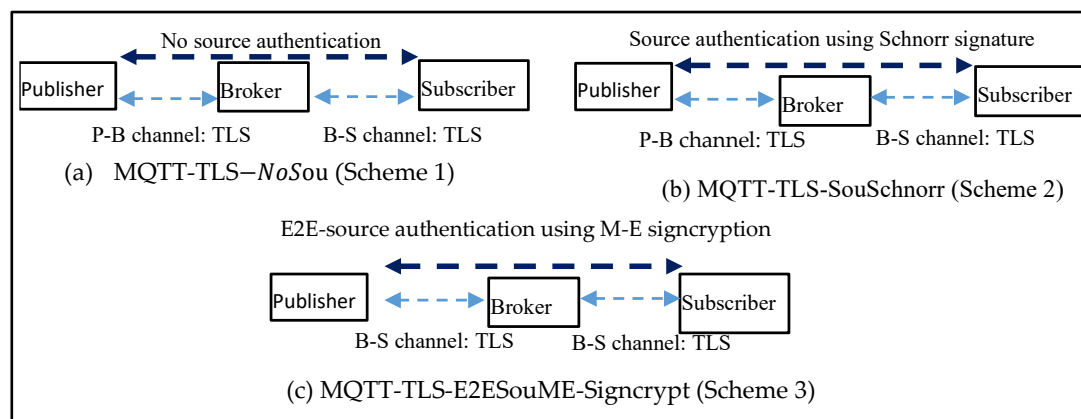


Figure 3. The three MQTT E2E-source-AUTH approaches.

Scheme 1 (MQTT-TLS-noSou): This scheme uses TLS to establish secure client–broker channels that provide client/broker authentication and confidentiality/integrity protection for message transmission. The broker decrypts data received over the publisher–broker channel and re-encrypts the message for transmission over the broker–subscriber channel. This approach represents the conventional MQTT/TLS model, which provides neither E2E security nor source authentication. We include this scheme as a baseline for comparison.

Scheme 2 (MQTT-TLS-SouSchnorr): Like Scheme 1, this scheme uses TLS for client–broker channels. In addition, the publisher attaches a Schnorr signature to each MQTT message so that receivers can verify its source. Thus, Scheme 2 adds source authentication to the conventional MQTT/TLS model but does not provide E2E confidentiality because the broker can access the plaintext content.

Scheme 3 (MQTT-TLS-E2ESouME-Signcrypt): Like Scheme 1, this scheme uses TLS for client–broker channels. In addition, the publisher applies the modified ME-Signcrypt scheme to each MQTT message so that (1) one signcryption can serve all authorized subscribers, (2) the broker can verify the source of the ciphertext without decrypting it, (3) each subscriber can decrypt the ciphertext and verify the source of the message, and (4) E2E security is provided because an honest-but-curious broker cannot access the plaintext content.

Scheme 1 represents the conventional MQTT/TLS approach, which provides neither E2E security nor source authentication. Scheme 2 adds a Schnorr signature to each MQTT message to provide source authentication but not confidentiality against an honest-but-curious broker. Scheme 3 applies the modified ME-Signcrypt scheme to the conventional MQTT/TLS model. In the following subsections, we omit the detailed design of Scheme 1 and describe only Schemes 2 and 3 for comparison, as shown in Figures 4 and 5.

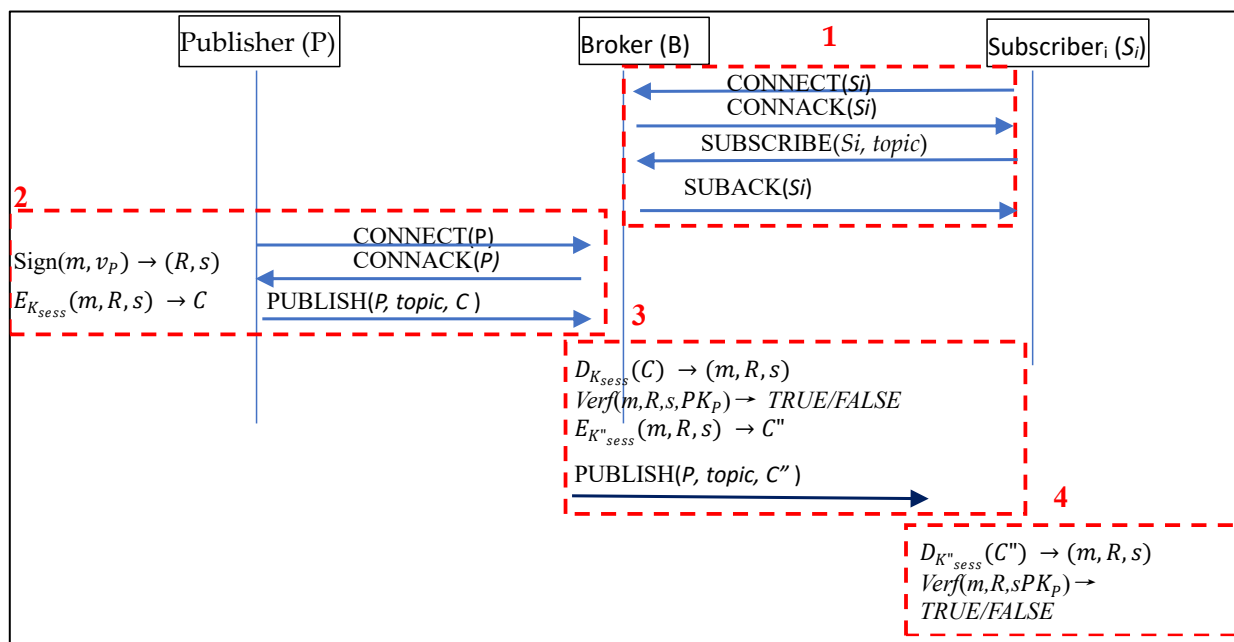


Figure 4. The steps/flow of Scheme 2.

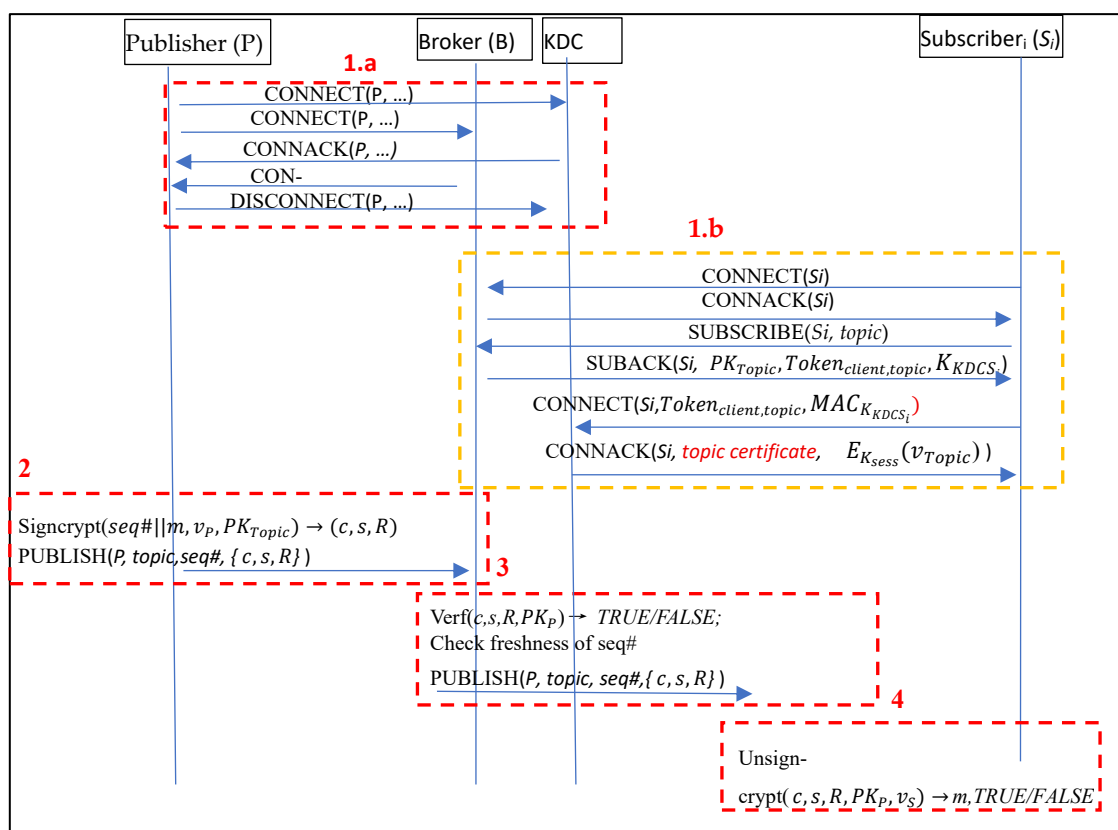


Figure 5. The steps/flow of Scheme 3.

3.3.2. Scheme 2 (MQTT-TLS-SouSchnorr)

In Scheme 2, as depicted in Figure 4, a subscriber first establishes a TLS-based subscriber–broker channel through the CONNECT procedure and then subscribes to a topic using SUBSCRIBE. In Step Block 2, a publisher establishes a publisher–broker channel and shares a session key K_{sess} with the broker. The publisher prepares message m , generates a Schnorr signature $\text{Sign}(m, v_p) \rightarrow (R, s)$ on m , encrypts the message and signature $E_{K_{sess}}(m, R, s) \rightarrow C$, and publishes the resulting ciphertext using a PUBLISH packet.

In Step Block 3, the broker decrypts the ciphertext and verifies the signature via $Verf(m, R, s, PK_p)$. If the verification returns TRUE, the broker re-re-encrypts the message and signature via $E_{K''_{sess}}(m, R, s) \rightarrow C''$, where K''_{sess} is the session key for the broker–subscriber channel.

In Step Block 4, the subscriber decrypts the encryption to derive the message and signature via $D_{K''_{sess}}(C'') \rightarrow (m, R, s)$, and then verifies them via $Verf(m, R, s, PK_p)$.

3.3.3. Scheme 3 (MQTT-TLS-SouME-Signcrypt)

We describe Scheme 3 in two phases: the connection phase for topic-key distribution and the publish–receive phase. In the connection phase, a publisher first establishes a TLS connection with the KDC to configure the topic-related parameters, including the topic name, authorization period, topic public/private key pair, sequence number, sliding window, and other related settings. The publisher defines the topic, while the KDC is responsible for generating the topic private key and for creating, distributing, and renewing the corresponding certificate.

Based on the local security policy, the KDC determines the authorization period. A compact representation of the valid period is then appended to the topic string, i.e., topic name || authorized period so that authorization validity and certificate renewal can be efficiently enforced during MQTT operations. The KDC is trusted by all entities to generate and distribute these settings. After receiving the settings from the KDC, the publisher connects to the broker and notifies it of the public part of the topic-related settings.

In addition, intrusion detection mechanisms are recommended to detect a possible device compromise. Topic certificates and keys are renewed according to both time-based policies and event-based policies, such as member joining, leaving, or removal. To distribute certificates and keys efficiently to legitimate members, the KDC and the broker are required to synchronize the membership list of each topic.

The following paragraphs introduces the message flows of MQTT-TLS-SouME-Signcrypt, the KDC operations and topic-certificate renewal policies, and one possible implementation for replay-attack prevention.

Message Flows of MQTT-TLS-SouME-Signcrypt

In the connection phase, a subscriber builds a TLS connection with the broker; it sends a SUBSCRIBE command to a specific topic, and then the broker verifies and returns a secret key K_{KDCS_i} and $Token_{client, KDC}$. K_{KDCS_i} is for the subscriber to authenticate itself to the KDC, and the authorized token $Token_{client, KDC}$ is for the KDC to derive the client's information and the key K_{KDCS_i} . $Token_{client, KDC}$ is an encryption of content like the client ID, authorized topic, signature, and K_{KDCS_i} ; the content is encrypted using the KDC public key. The client then builds a TLS channel with the KDC and sends a request that contains a K_{KDCS_i} -based Message Authentication Code (MAC), along with $Token_{client, KDC}$; upon verifying the token and the MAC, the KDC responds with the topic certificate and the topic private key- v_{Topic} in a secure client–KDC channel.

In the publish–receive phase, a publisher performs signcrypt on its messages and publishes the topic name, the sequence number, and the signcrypts in the publisher–broker channel. The broker verifies the signcrypts/sequence number and forwards them to the legitimate subscribers. The subscribers verify the sequence number and verify/decrypt the signcrypts.

Conventionally, when a user applies ME-Signcrypt, it would use its private key and the recipient's public key. Here, when we apply the modified ME-Signcrypt in our schemes, the recipient's public key is replaced by the topic public key PK_{Topic} . All the legitimate subscribers use the topic private key to decrypt the messages.

Step-block 1.a: The publisher first obtains the initial publishing authorization from the broker and acquires the key K_{KDCS_i} and $Token_{client,KDC}$. Using K_{KDCS_i} and $Token_{client,KDC}$, the publisher establishes a TLS connection with the KDC to configure the topic-key settings, including the topic name, authorization period, sequence number, sliding window, PK_{Topic} , v_{Topic} , and other related parameters. In this design, the KDC is responsible for generating these settings. The publisher then notifies the broker of the topic certificate, which contains the topic name, authorization period, PK_{Topic} , sequence number, sliding window, and other public topic-related parameters.

Step-block 1.b: A subscriber builds a TLS channel, and then sends a SUBSCRIBE request; upon authorization, the broker returns the topic certificate (which contains the authorization period, the topic public key, K_{KDCS_i} , sequence number, sliding window, etc.) and an authorization token $Token_{client,topic}$. It then builds a TLS connection request with the KDC, where the request contains the client ID, topic, K_{KDCS_i} -based MAC, and the authorized token. The KDC first verifies the request, the K_{KDCS_i} -based MAC, and the token; if the verifications succeed, then it returns the encrypted topic private key $E_{K_{sess}}(v_{Topic})$, where K_{sess} is the session key shared during the KDC–subscriber channel-building phase.

Step-block 2: A publisher generates a signcryption on its message m and the seq# via $\text{Signcrypt}(seq\#||m, v_P, PK_{Topic}) \rightarrow (c, s, R)$ and publishes its MQTT message via PUBLISH($P, topic, seq\#, \{c, s, R\}$).

Step-block 3: The broker verifies the signcryption $\text{Verf}(c, s, R, PK_P)$ and the freshness of seq#. If the verifications succeed, it forwards the message to subscribers.

Step-block 4: The subscribers decrypt and verify the signcryption via $\text{Unsigncrypt}(c, s, R, PK_P, v_{Topic})$ and checks the freshness of seq#.

Operations and Topic Certificate Renewal Policies of KDC

The KDC is trusted and is responsible for topic-certificate generation, renewal, and distribution. Since the topic private key is shared between authorized clients, the compromise of a single client may illegally disclose the message contents. Therefore, we propose that the topic certificate and its associated public/private key pair should be renewed according to both time-based and event-based policies.

The time-based policy defines the renewal interval. Under this policy, the KDC regularly triggers the renewal process according to the configured time period. The selection of the renewal interval is a trade-off between computational/communication cost and the desired security level. The event-based policy defines the events that trigger the renewal process, even when the current certificate remains valid. Several typical event-based settings are introduced as follows, but they are not limited to these cases:

1. Member joining: A member joining refers to the case where a new subscriber successfully subscribes to a specific topic. After joining, the subscriber acquires a secret key K_{KDCS_i} and $Token_{client,KDC}$ to access the KDC. After successfully verifying the client, the KDC determines, according to the policy settings, whether to distribute the current topic certificate or immediately launch the renewal process. This decision depends on the security requirements of the application. If the KDC renews the keys and certificate, it returns the new settings to the new subscriber and securely distributes the renewed certificates and keys to the publisher, the broker, and the legitimate subscribers. This renewal provides the desired backward secrecy, ensuring that new members cannot access previous messages for which they were not authorized.
2. Member leaving: A member leaving refers to the case where a subscriber unsubscribes from a specific topic. When the broker successfully verifies a valid unsubscribe request, it notifies the KDC of the unsubscribe request together with the client identity. The KDC then renews the certificate and keys and updates the membership list accordingly. The KDC securely distributes the new settings to the broker and the

remaining legitimate subscribers. This renewal provides the desired forward secrecy, ensuring that the leaving member cannot access future messages.

3. Member removal: If a client compromise is detected, the intrusion detection system notifies the broker to remove the compromised subscriber or publisher from its list. It also notifies the KDC to launch the renewal and distribution processes.

Next, we discuss one possible approach for implementing topic-certificate/key distribution by the KDC. We propose that a trusted KDC can host an MQTT broker that is separate from the broker responsible for regular MQTT messaging. The KDC defines specific system topics, and all entities in the system subscribe to these topics to communicate with the KDC. Since the MQTT messages sent by the KDC are related to topic-certificate/key renewal and distribution, QoS 2, i.e., exactly once delivery, is required.

Discussions of Replay Attack Prevention

We propose adopting sequence numbers and a sliding window to mitigate replay attacks. Unlike the conventional approach, in which the sequence number and sliding-window mechanism are maintained on a per-session basis, this scheme requires the sequence number and sliding window to be maintained on a per-topic-certificate basis. That is, each topic certificate has its own sequence number state and sliding window state. Each publisher, the broker, and the corresponding subscribers maintain this information in their local states.

To implement the sequence-number and sliding-window mechanism for a topic certificate, one possible design includes the highest accepted sequence number, denoted as high-seq; the window size, denoted as win-size; and the associated bitmap. We use the following example to illustrate the idea. Assume that high-seq = 100, win-size = 4, and bitmap = {(97,0), (98,1), (99,0), (100,0)}; this means that sequence numbers in the range ([97,100]) are within the acceptable window and that the message associated with sequence number 98 has already been received.

4. Performance Evaluation and Discussion

This section evaluates the performance of the three schemes and discusses several key performance characteristics of related E2E schemes.

We analytically examine the computational overhead incurred by the publisher, broker, and subscriber in each scheme, with particular emphasis on the broker because existing studies identify broker performance as a bottleneck as the system scales.

We assume that TLS uses a full handshake, Elliptic Curve Diffie–Hellman Ephemeral (ECDHE) for key exchange, and the Elliptic Curve Digital Signature Algorithm (ECDSA) for mutual authentication.

Table 2 defines the notations used in this evaluation. Here, we define the notations for basic operations that Schemes 1–3 use; for example, $T_{\text{Signcrypt}}/T_{\text{Signcrypt,verf}}/T_{\text{UnSigncrypt}}$ respectively denote the computational costs for Signcrypt/Verify/UnSigncrypt of ME-Signcrypt; $T_{\text{ECDSA,sign}}/T_{\text{ECDSA,verf}}$ respectively denote those for sign/verify of ECDSA; T_{ECDHE} denotes that for a ECDHE computation.

Table 2. The notations for the computational performance evaluation.

$T_{\text{Signcrypt}}/T_{\text{Signcrypt,verf}}/T_{\text{UnSigncrypt}}$: the cost for one Signcrypt/Verf/Unsigncrypt in ME-Signcrypt.		
$T_{\text{ECDSA,sign}}/T_{\text{ECDSA,verf}}$: the cost for one signature generation/verification in ECDSA.	T_{ECDHE} : the cost for one ECDHE computation of a client.	T_{AES} : the cost for one AES 256 bit with ECB mode.
$T_{G,dbl}/T_{G,add}/T_{G,m}$: the cost for one ECC point doubling/point addition/scalar point multiplication.	$T_{F,m}, T_{F,s}, T_{F,add}, T_{F,I}$: the cost for one multiplication/squaring/addition/inverse in a prime field.	

To estimate the cycles, running times, and computations required for key operations in TLS/ECDSA/ME-Signcrypt/AES, we refer to the work of Tobias Markmann [27], which provides the required number of ECC operations/field operations, and to the work of Seo et al. [28], which provides the estimated cycles/times for the 256-bit prime-field operations on the STM32F4 Cortex-M4 168 MHz hardware. Table 3 summarizes these estimations.

Table 3. Numbers of ECC/prime-field computations, cycles, and execution times required for the evaluated operations on an STM32F4 Cortex-M4 processor operating at 168 MHz.

Operation	Metric Computations	Number of Cycles	Time (ms)
$T_{\text{Signcrypt}}$	$3T_{G,m} + T_{F,m} + T_{F,l} + 3T_h + T_{\text{AES}}$	5,117,341	30.26
$T_{\text{Signcrypt,verf}}$	$2T_{G,m} + T_{G,add} + T_{F,m} + 2T_h$	3,382,467	20.08
$T_{\text{UnSigncrypt}}$	$3T_{G,m} + T_{G,add} + T_{F,m} + 3T_h + T_{\text{AES}}$	5,090,144	30.10
$T_{\text{ECDSA,sign}}$	$T_{G,m} + 1T_h$	1,686,288	10.04
$T_{\text{ECDSA,verf}}$	$2T_{G,m} + T_{G,add} + 1T_h$	3,382,467	20.08
T_{ECDHE}	$2T_{G,m}$	3,372,576	20.07
T_{AES}	- ¹	21,389	0.0127
$T_{G,dbl}$	$3T_{F,m} + 5T_{F,s} + 8T_{F,add}$	4945	0.03
$T_{G,m}$	$255T_{G,dbl} + 43T_{G,add}$	1,686,288	10.01
$T_{G,add}$	$11T_{F,m} + 5T_{F,s} + 9T_{F,add}$	9891	0.06
$T_{F,m}$	-	608	0.0036
$T_{F,s}$	-	493	0.0029
$T_{F,add}$	-	82	0.0005
$T_{F,l}$	-	36,480	0.2171

¹ The symbol “-” indicates a basic computation that does not need to be expressed as a combination of other operations.

Table 4 summarizes the computational costs of the three schemes; we include only computationally intensive operations while omitting lightweight operations. The evaluation is divided into the connection phase and the publish–receive phase because all schemes can use persistent MQTT/TLS connections [29] or TLS session resumption [5] to reduce the impact of the connection-establishment overhead.

Table 4. The required computations of the three schemes.

Scheme	Publisher	Broker	Subscriber
Scheme 1: connection phase	$T_{\text{ECDSA,sign}} + T_{\text{ECDSA,verf}} + T_{\text{ECDHE}}$	$(1+k)(T_{\text{ECDSA,sign}} + T_{\text{ECDSA,verf}} + T_{\text{ECDHE}})$	$T_{\text{ECDSA,sign}} + T_{\text{ECDSA,verf}} + T_{\text{ECDHE}}$
Scheme 1: publish–receive phase	$r \times T_{\text{AES}}$ “ $r \times 0.0127$ ms”	$r \times (1+k)T_{\text{AES}}$ “ $r \times (1+k) \times 0.0127$ ms”	$r \times T_{\text{AES}}$ “ $r \times 0.0127$ ms”
Scheme 2: connection phase	$T_{\text{ECDSA,sign}} + T_{\text{ECDSA,verf}} + T_{\text{ECDHE}}$	$(1+k)(T_{\text{ECDSA,sign}} + T_{\text{ECDSA,verf}} + T_{\text{ECDHE}})$	$T_{\text{ECDSA,sign}} + T_{\text{ECDSA,verf}} + T_{\text{ECDHE}}$
Scheme 2: publish–receive phase	$r \times (T_{\text{ECDSA,sign}} + T_{\text{AES}})$ “ $r \times (10.04 + 0.0127)$ ms”	$r \times (T_{\text{ECDSA,verf}} + (1+k)T_{\text{AES}})$ “ $r \times (20.08 + (1+k)0.0127)$ ms”	$r \times (T_{\text{ECDSA,verf}} + T_{\text{AES}})$ “ $r \times (20.08 + 0.0127)$ ms”
Scheme 3: connection phase	$2(T_{\text{ECDSA,sign}} + T_{\text{ECDSA,verf}} + T_{\text{ECDHE}})$	$(1+k)(T_{\text{ECDSA,sign}} + T_{\text{ECDSA,verf}} + T_{\text{ECDHE}})$	$2(T_{\text{ECDSA,sign}} + T_{\text{ECDSA,verf}} + T_{\text{ECDHE}})$
Scheme 3: publish–receive phase	$r \times (T_{\text{Signcrypt}} + T_{\text{AES}})$ “ $r \times (30.26 + 0.0127)$ ms”	$r \times (T_{\text{Signcrypt,verf}} + (1+k)T_{\text{AES}})$ “ $r \times (20.08 + (1+k)0.0127)$ ms”	$r \times (T_{\text{UnSigncrypt}} + T_{\text{AES}})$ “ $r \times (30.11 + 0.0127)$ ms”

k denotes the number of subscribers that have subscribed to a specific topic. r denotes the message rate (the number of messages per s) of a publisher. Values enclosed in quotation marks denote the execution times obtained by substituting the measurements from Table 3 into the corresponding

expression. For example, “ $(1 + k) \times 0.0127$ ms” denotes the time obtained by substituting $T_{AES} = 0.0127$ ms into the expression $(1 + k)T_{AES}$.

In Table 4, we assume that one publisher publishes a message to a topic subscribed to by k subscribers. We illustrate the selected calculations from Table 4 as follows.

Overhead of the connection phase

In the connection phase of Scheme 1 and Scheme 2, a publisher/subscriber needs to establish a TLS-based client–broker channel while a publisher/subscriber in Scheme 3 establishing two TLS channels: one is for the client–broker channel and the other is for the client–KDC channel. During a TLS-channel-building process, a client takes one ECDSA signing ($T_{ECDSA,sign}$), one ECDSA signature verification ($T_{ECDSA,verf}$), and one ECDHE (T_{ECDHE}). Therefore, a client takes $(T_{ECDSA,sign} + T_{ECDSA,verf} + T_{ECDHE})$ in the connection phase of Scheme 1 and Scheme 2, while it takes $2(T_{ECDSA,sign} + T_{ECDSA,verf} + T_{ECDHE})$ in Scheme 3. A broker needs to establish $(1 + k)$ TLS connections for one publisher and k subscribers.

During the TLS connection phases, all schemes exchange certificates, keying material, and related protocol messages. Consequently, the connection-phase communication cost of Scheme 3 is approximately twice that of Schemes 1 and 2 because each client in Scheme 3 establishes two TLS connections.

Overhead of KD

In Scheme 3, the KDC is responsible for topic certificate/keyings generation/renewal/distribution. The renewal process is triggered according to both the time-based policy and the event-based policy. The selection of a renewal interval involves a trade-off between the overhead and the desired security level. Compared with the publish frequency, certificate renewal frequency is generally much smaller, its overhead is comparable with certificate maintenance in the other schemes and could be amortized during the whole process; therefore, it is not listed separately in Table 4. The validity period can be enforced by appending a compact representation of the authorized period to the topic string (topic name || authorized period), allowing it to be checked directly during MQTT operations. The parameters of the replay prevention mechanism are specified in the topic certificate.

Overhead of the publish–receive phase

Now we examine the computational costs during the publish–receive phase. We assume there are k subscribers for a specific topic, and a publisher publishes MQTT messages at the publish rate r . We list typical message rates of several IoT application scenarios in Table 5. For each application scenario, we adopt the highest rates from the third column in the table to estimate the computational overhead.

Table 5. Message rates of several IoT applications.

Scenarios	Device Types	Message Rate per Device, $r := \#/\text{s}$	Desirable QoS
Agriculture	Various sensors like moisture meter	$1/(60 \times 60) - 1/(15 \times 60)$	QoS 0/QoS 1
Smart Healthcare/Wearables	Heart rate wristband, CGM (Continuous Glucose Monitoring)	$1/(5 \times 60) - 1/(1 \times 60)$	QoS 1/QoS 2
Home IoT	Home environment sensors like temperature monitor	$1/(5 \times 60) - 1/30$	QoS 0 (report) QoS 1 (command)
IoV (Internet of Vehicles)	GPS tracker, etc.	$1/(15 \times 60) - 1/30$	QoS 1
IIoT	Motor vibration sensor, PLC (Programmable Logic Controller)	10–100	QoS 1

A publisher in Scheme 1 encrypts the message (T_{AES}) and a subscriber needs to decrypt the encryption (T_{AES}); when a broker forwards an MQTT message, it takes one decryption and k re-encryptions (total: $(1 + k)T_{AES}$).

Compared with Scheme 1, a publisher/subscriber in Scheme 2 adds one additional signature generation/signature verification. A broker in Scheme 2 takes one signature verification ($T_{ECDSA,verf}$) and $(1 + k)T_{AES}$ for one decryption and k re-encryptions. The additional overhead for source authentication at the broker is just one signature verification ($T_{ECDSA,verf}$, 20.08 ms), which is quite small.

Compared with Scheme 1, a publisher in Scheme 3 additionally adds one signcryption $T_{Signcrypt}$ and a subscriber additionally verifies/decrypts one signcryption ($T_{UnSigncrypt}$). A broker additionally takes one $T_{Signcrypt,verf}$ (20.08 ms), which is quite small. Most importantly, it pays only an additional cost of $T_{Signcrypt,verf}$ to achieve both the E2E function and source authentication.

The publish–receive computational/communication costs at broker of all the three schemes are similar with only a marginal difference (one additional ($T_{ECDSA,verf}$ or one additional $T_{Signcrypt,verf}$).

All the corresponding figures of the publish–receive phases in Table 4 are acquired by multiplying the required operations with the message rate r .

To further estimate the typical performance figures when we consider the number of subscribers and the message rates, we assume an IoT device with the specification “256-bit prime-field operations on the STM32F4 Cortex-M4 168 MHz hardware” and assume an MQTT broker with the specification “AMD’s Turin (Zen 5) the 192-core EPYC 9965” [30]. Under such an assumption, the ratio of the mathematical computing capacities between the two devices (AMD’s Turin (Zen 5)/STM32F4 Cortex-M4 168 MHz) could be roughly estimated as $R = 170,000$. Table 6 shows the estimated computational costs of a publisher/broker/subscriber for Schemes 1–3 during the publish–receive phase. We use some examples to illustrate how we derive the figures. The cost of P for Scheme 1 under the condition ($k = 1$ and $r = 0.00111111$) being 0.00001 ms is derived by calculating $r \times T_{AES}$, where $T_{AES} = 0.0127$ ms from Table 3. The cost of B for Scheme 2 under the condition ($k = 10$ and $r = 0.01666666$) being $0.33697625/R$ ms is derived by calculating $r \times ((1 + k) \times T_{AES} + T_{ECDSA,verf})/R$, where $T_{AES} = 0.0127$ ms and $T_{ECDSA,verf} = 20.08$ ms from Table 3 (which estimates the figures assuming it is an STM32F4 Cortex-M4 device), and we have this result divided by R (170,000 which denotes the computing capacity ratio between the broker and the IoT device).

Based on the results in Table 6 and the hardware assumptions, several observations can be made. First, in all considered scenarios, the brokers in Schemes 1–3 can effectively handle the required computations. Second, in some scenarios, the IoT devices cannot afford the required computational load. The values corresponding to these scenarios are marked in blue and bold in Table 6.

For the scenario where $r = 100$ messages/s, the estimated costs required by the publisher/subscriber in Schemes 2 and 3 are 1005 ms, 2009 ms, 3027 ms, and 2009 ms, respectively. These costs are unaffordable for the IoT devices because the required computation time exceeds 1 s when the message rate reaches 100 messages/s. Therefore, this result indicates that adding a source-authentication mechanism, such as a signature or signcryption mechanism, imposes a non-negligible computational burden on IoT devices when the message rate increases beyond a certain level.

Table 6. Estimated computational costs (ms per one publish) of publisher/broker/subscriber for Schemes 1–3.

k	r (#/s)	Scheme 1			Scheme 2			Scheme 3		
		Cost of P	Cost of B	Cost of S	Cost of P	Cost of B	Cost of S	Cost of P	Cost of B	Cost of S
1	0.00111111	0.00001	$(2.8 \times 10^{-5})/R$	0.00001	0.01116681	0.02233808/R	0.02232397	0.03364018	0.02233808/R	0.02232397
10	0.01666666	0.00021	0.002328/R	0.00021	0.16750214	0.33697625/R	0.33485958	0.50460269	0.33697625/R	0.33485958
100	0.03333333	0.00042	0.042756/R	0.00042	0.33500428	0.7120525/R	0.66971916	1.00920539	0.7120525/R	0.66971916
1000	1	0.01270	12.7127/R	0.01270	10.0501285	32.791575/R	20.091575	30.2761619	32.791575/R	20.091575
10,000	100	1.27000	12,701.27/R	1.27000	1005.01285	14,709.1575/R	2009.1575	3027.61619	14,709.1575/R	2009.1575

k denotes the number of subscribers for one topic; # denotes number of messages; r denotes the message rate; R denotes the mathematical computing capacity ratio between a broker and an IoT device. P denotes one publisher; B denotes one broker; S denotes one subscriber.

Tables 4–6 leads to the following observations:

- (1) Schemes 1–3 have similar computational costs in the connection phase, except that the computational and communication overhead incurred by each client in Scheme 3 is approximately twice that in Schemes 1 and 2. The broker-side cost of connection establishment grows linearly with the number of connections. Persistent MQTT/TLS connections and TLS session resumption can amortize the impact of connection-establishment overhead.
- (2) Compared with Scheme 1, a publisher in Scheme 2/Scheme 3 respectively adds one extra signature ($T_{ECDSA,sign}/T_{Signcrypt}$) on its message, while a subscriber respectively needs one extra signature verification ($T_{ECDSA,verf}$)/unsigncrypt ($T_{Unsigncrypt}$). Since the additional cost of $T_{ECDSA,sign}/T_{ECDSA,verf}$ is fixed and not large, the extra overhead is negligible when the message rates are not high; but, the costs are unaffordable when the message rate increases beyond 100 #/s under the hardware assumptions.
- (3) During the publish–receive phase, the broker encryption/decryption cost in all three schemes grows linearly with the number of subscribers. Because one symmetric encryption or decryption operation is inexpensive, this overhead is negligible.
- (4) In summary, Schemes 2 and 3 provide source authentication with negligible additional overhead under low-message-rate conditions, whereas only Scheme 3 provides both E2E security and source authentication.

Table 7 summarizes the performance of several MQTT E2E schemes. Among them, only Scheme 3 simultaneously provides source authentication and E2E security. The other E2E schemes [9,11–13,17,22] incur non-negligible overhead in their E2E mechanisms. Schemes [9,11,13,22] require pairwise E2E key negotiation, the secret-sharing mechanism between a client and a pool of key stores in [12] incurs a substantial communication overhead, and the attribute-based encryption (ABE) used in [17] requires expensive computations and supports only a limited number of attributes efficiently. In summary, Scheme 3 is the only evaluated approach that simultaneously provides E2E security and source authentication with affordable overhead (a publisher with an additional $T_{Signcrypt}$, a subscriber with an additional $T_{Unsigncrypt}$, and a broker with an additional $T_{Signcrypt,verf}$) under low-message-rate conditions.

Table 7. Performance summary of the three schemes and other MQTT-E2E work.

Feature Scheme	C-B Conn.	E2E Mechanism	E2E Overhead at Broker	Source Auth.	Sou. Auth. Overhead at Broker	Sou. Auth. Overhead at Device
Scheme 1	TLS	No	No	No	No	No
Scheme 2	TLS	No	No	Yes (signature)	$T_{ECDSA,verf}$ (20.08 ms)	Unaffordable when the message rate exceeds a certain level

Scheme 3	TLS	Signcryption + topic key	$T_{\text{Signcryption,verf}}$ (20.08 ms).	Yes (signcryption + topic key)	$T_{\text{Signcryption,verf}}$ (20.08 ms).	Unaffordable when the message rate exceeds a certain level
[9,11,13]	TLS	Publisher–subscriber–DH	Heavy pairwise key negotiation overhead	No	No	No
SEEMQTT [12]	TLS	Secret sharing between client and key stores	Very large overhead for secret sharing between client and key stores	No	No	No
Singh et al. [17]	TLS	ABE	Heavy computations involved in ABE operations, not scalable as the number of attributes grows	No	No	No
MQTLS [22]	TLS extension	Publisher–subscriber–DH	Heavy pairwise key negotiation overhead	No	No	No

5. Conclusions and Future Work

This study focuses on E2E security and source authentication in broker-based publish–subscribe systems. We analytically evaluate the security properties and performance of three models. We modify the Mohamed–Elkamchouchi signcryption scheme and apply it to the conventional MQTT/TLS model. The performance analysis shows that only the proposed scheme simultaneously provides E2E security and source authentication with affordable overhead when the message rates are not high. The proposed design is therefore a promising approach for broker-based IoT systems requiring both properties. It also motivates several directions for future work: (1) incorporating signcryption into the ACE/OSCORE framework for the benefit of the relevant research communities; (2) implementing, deploying, and evaluating the proposed scheme in realistic environments; (3) applying the modified Mohamed–Elkamchouchi signcryption scheme to cross-chain IoT data-sharing scenarios [31]; (4) extending the modified Mohamed–Elkamchouchi signcryption scheme using identity-based cryptography and studying the potential applications.

Funding: This research was partially funded by the Ministry of Science and Technology of Taiwan by the grant number NSTC 114-2221-E-260-014.

Data Availability Statement: No new data were created or analyzed in this study.

Conflicts of Interest: The author declares no conflicts of interest.

References

1. OASIS. *MQTT*, version 5.0; OASIS Standard; OASIS Open: Burlington, MA, USA, 2019. Available online: <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html> (accessed on 23 April 2026).
2. OASIS. *MQTT*, version 3.1.1; OASIS Standard; OASIS Open: Burlington, MA, USA, 2014. Available online: <https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html> (accessed on 23 April 2026).
3. ISO/IEC 20922:2016; Information Technology—Message Queuing Telemetry Transport (MQTT) v3.1.1. International Organization for Standardization: Geneva, Switzerland, 2016.
4. Jimenez, J.; Koster, M.; Keränen, A. *A Publish-Subscribe Architecture for the Constrained Application Protocol (CoAP)*; Internet-Draft draft-ietf-core-coap-pubsub-19; Work in Progress; Internet Engineering Task Force: Fremont, CA, USA, 2026. Available online: <https://datatracker.ietf.org/doc/draft-ietf-core-coap-pubsub/19/> (accessed on 23 April 2026).
5. Rescorla, E. *The Transport Layer Security (TLS) Protocol Version 1.3*; RFC 8446; Internet Engineering Task Force: Fremont, CA, USA, 2018. <https://doi.org/10.17487/RFC8446>.
6. Rescorla, E.; Tschofenig, H.; Fossati, T.; Nir, Y. *The Datagram Transport Layer Security (DTLS) Protocol Version 1.3*; RFC 9147; Internet Engineering Task Force: Fremont, CA, USA, 2022. <https://doi.org/10.17487/RFC9147>.
7. Gupta, S.; Sacchetti, T.; Crispo, B. end-to-end Encryption for Securing Communications in Industry 4.0. In *Proceedings of the 2022 4th IEEE Middle East and North Africa COMMunications Conference (MENACOMM)*, Amman, Jordan, 6–8 December 2022; IEEE: Piscataway, NJ, USA, 2023; pp. 153–158. <https://doi.org/10.1109/MENACOMM57252.2022.9998272>.

8. Lin, S.; Cui, L.; Ke, N. End-to-End Encrypted Message Distribution System Using Conditional Proxy Re-Encryption. *Sensors* **2024**, *24*, 438.
9. Chien, H.-Y.; Shih, A.-T.; Huang, Y.-M. Design, Implementation, and Evaluation of MQTT 5.0-Based end-to-end Security. *J. Internet Technol.* **2025**, *in press*.
10. Wang, N.-Z.; Chien, H.-Y. Design and Implementation of MQTT-Based Over-the-Air Updating Against Curious Brokers. *IEEE Internet Things J.* **2024**, *11*, 10768–10777. <https://doi.org/10.1109/JIOT.2023.3327447>.
11. Mektoubi, A.; Lalaoui, H.; Belhadaoui, H.; Rifi, M.; Zakari, A. New Approach for Securing Communication over MQTT Protocol: A Comparison between RSA and Elliptic Curve. In *Proceedings of the 2016 Third International Conference on Systems of Collaboration (SysCo), Casablanca, Morocco, 28–29 November 2016*; IEEE: Piscataway, NJ, USA, 2016; pp. 1–6. <https://doi.org/10.1109/SYSCO.2016.7831326>.
12. Hamad, M.; Finkenzeller, A.; Liu, H.; Lauinger, J.; Prevelakis, V.; Steinhorst, S. SEEMQTT: Secure end-to-end MQTT-Based Communication for Mobile IoT Systems Using Secret Sharing and Trust Delegation. *IEEE Internet Things J.* **2023**, *10*, 3384–3406. <https://doi.org/10.1109/JIOT.2022.3221857>.
13. Chien, H.-Y.; Shih, A.-T.; Huang, Y.-M. Exploring MQTT Broker-Based, end-to-end Models for Security and Efficiency. *Sensors* **2025**, *25*, 5308. <https://doi.org/10.3390/s25175308>.
14. Sengul, C.; Kirby, A. *Message Queuing Telemetry Transport (MQTT)-TLS Profile of Authentication and Authorization for Constrained Environments (ACE) Framework*; Internet-Draft draft-ietf-ace-mqtt-tls-profile-17; Work in Progress; Internet Engineering Task Force: Fremont, CA, USA, 2022. Available online: <https://datatracker.ietf.org/doc/draft-ietf-ace-mqtt-tls-profile/17/> (accessed on 23 April 2026).
15. Nwiah, E.; Kant, S. A Survey on Securing Payload in MQTT and a Proposed Ultra-Lightweight Cryptography. In *Information and Communication Technology for Intelligent Systems*; Senjyu, T., Mahalle, P.N., Perumal, T., Joshi, A., Eds.; Smart Innovation, Systems and Technologies; Springer: Singapore, 2021; Volume 196, pp. 323–335. https://doi.org/10.1007/978-981-15-7062-9_32.
16. Park, C.-S.; Nam, H.-M. Security Architecture and Protocols for Secure MQTT-SN. *IEEE Access* **2020**, *8*, 226422–226436. <https://doi.org/10.1109/ACCESS.2020.3045441>.
17. Singh, M.; Rajan, M.A.; Shivraj, V.L.; Balamuralidhar, P. Secure MQTT for Internet of Things (IoT). In *Proceedings of the 2015 Fifth International Conference on Communication Systems and Network Technologies (CSNT), Gwalior, India, 4–6 April 2015*; IEEE: Piscataway, NJ, USA, 2015; pp. 746–751. <https://doi.org/10.1109/CSNT.2015.16>.
18. Zheng, Y. Digital Signcryption or How to Achieve $\text{Cost}(\text{Signature} \ \& \ \text{Encryption}) \ll \text{Cost}(\text{Signature}) + \text{Cost}(\text{Encryption})$. In *Advances in Cryptology—CRYPTO '97*; Kaliski, B.S., Ed.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 1997; Volume 1294, pp. 165–179. <https://doi.org/10.1007/BFb0052234>.
19. Mohamed, E.; Elkamchouchi, H. Elliptic Curve Signcryption with Encrypted Message Authentication and Forward Secrecy. *Int. J. Comput. Sci. Netw. Secur.* **2009**, *9*, 395–398.
20. Sengul, C.; Kirby, A. *Message Queuing Telemetry Transport (MQTT) and Transport Layer Security (TLS) Profile of Authentication and Authorization for Constrained Environments (ACE) Framework*; RFC 9431; Internet Engineering Task Force: Fremont, CA, USA, 2023. <https://doi.org/10.17487/RFC9431>.
21. Bethencourt, J.; Sahai, A.; Waters, B. Ciphertext-Policy Attribute-Based Encryption. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy (SP 2007), Oakland, CA, USA, 20–23 May 2007*; IEEE: Piscataway, NJ, USA, 2007; pp. 321–334. <https://doi.org/10.1109/SP.2007.11>.
22. Lee, H.; Lim, J.; Kwon, T. MQTLS: Toward Secure MQTT Communication with an Untrusted Broker. In *Proceedings of the 2019 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Republic of Korea, 16–18 October 2019*; IEEE: Piscataway, NJ, USA, 2019; pp. 53–58. <https://doi.org/10.1109/ICTC46691.2019.8940001>.
23. Coppen, R. *Contributed MQTT-SN 1.2 Specification*; OASIS MQTT Technical Committee Document; OASIS Open: Burlington, MA, USA, 2019. Available online: https://groups.oasis-open.org/higherlogic/ws/public/document?document_id=66091 (accessed on 23 April 2026).
24. Rogaway, P. Authenticated-Encryption with Associated-Data. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002), Washington, DC, USA, 18–22 November 2002*; Association for Computing Machinery: New York, NY, USA, 2002; pp. 98–107. <https://doi.org/10.1145/586110.586125>.
25. Selander, G.; Mattsson, J.; Palombini, F.; Seitz, L. *Object Security for Constrained RESTful Environments (OSCORE)*; RFC 8613; Internet Engineering Task Force: Fremont, CA, USA, 2019. <https://doi.org/10.17487/RFC8613>.

26. Palombini, F.; Sengul, C.; Tiloca, M. *CoAP Publish-Subscribe Profile for Authentication and Authorization for Constrained Environments (ACE)*; Internet-Draft draft-ietf-ace-coap-pubsub-profile-04; Work in Progress; Internet Engineering Task Force: Fremont, CA, USA, 2026. Available online: <https://datatracker.ietf.org/doc/draft-ietf-ace-coap-pubsub-profile/04/> (accessed on 23 April 2026).
27. Markmann, T. *Modern Elliptic Curve Cryptography for Constrained Devices*; Project Report PR2; Hamburg University of Applied Sciences: Hamburg, Germany, 2015. Available online: <https://inet.haw-hamburg.de/teaching/ws-2014-15/project-class/tobias-markmann-modern-elliptic-curve-cryptography-for-constrained-devices> (accessed on 23 April 2026).
28. Seo, H.; Kwon, H.; Kwon, Y.; Kim, K.; Choi, S.; Kim, Y.; Jang, K. Fast Number Theoretic Transform for Ring-LWE on 8-bit AVR Embedded Processor. *Sensors* **2020**, *20*, 2039. <https://doi.org/10.3390/s20072039>.
29. Prantl, T.; Iffländer, L.; Herrnleben, S.; Engel, S.; Kounev, S.; Krupitzer, C. Performance Impact Analysis of Securing MQTT Using TLS. In *Proceedings of the 2021 ACM/SPEC International Conference on Performance Engineering (ICPE '21), Virtual Event, France, 19–23 April 2021*; Association for Computing Machinery: New York, NY, USA, 2021; pp. 241–252. <https://doi.org/10.1145/3427921.3450253>.
30. Advanced Micro Devices, Inc. AMD EPYC™ 9005 Series Processors. Available online: <https://www.amd.com/en/products/processors/server/epyc/9005-series.html> (accessed on 26 June 2026).
31. Xiong, R.; Cheng, J.; Pu, J.; Dong, X. Enhancing trust and collaboration: A reputation-driven mechanism for cross-chain IoT data sharing. *Comput. Netw.* **2025**, *264*, 111266. <https://doi.org/10.1016/j.comnet.2025.111266>.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.