End-to-end encryption for securing communications in Industry 4.0

End-to-End Encryption for Securing Communications in Industry 4.0

Sandeep Gupta
DISI
University of Trento
Trento, Italy
sandeep.gupta@ex-staff.unitn.it

Tommaso Sacchetti
DISI
University of Trento
Trento, Italy
tommaso.sacchetti@studenti.unitn.it

Bruno Crispo
DISI
University of Trento
Trento, Italy
bruno.crispo@unitn.it

***Abstract*—In Industry 4.0 (I4.0), reliable data sharing between multiple entities is profoundly significant for both the business-level and the manufacturing operations-level collaboration. Eventually, machine-to-machine (M2M) communication technology can be a key underlying technology for I4.0, where devices (e.g., sensors, actuators, and gateways) can exchange information with each other autonomously, ensuring data confidentiality and integrity. In this paper, we propose a transparent message level end-to-end encryption layer for Message Queue Telemetry Transport (MQTT). We exploit Ciphertext-Policy Attribute-based encryption (CP-ABE) that is implemented using the Fast Attribute Message Encryption (FAME) CP-ABE scheme. We also evaluate the time and space performance of the FAME CP-ABE scheme.**

***Index Terms*—Industry 4.0, M2M communication, Encryption, Secure MQTT**

I. Introduction

Industry 4.0 (I4.0) aims at automating the entire industrial ecosystem by establishing nexus between the physical world (i.e., engineering, manufacturing, and supply chain), and the enterprise business (i.e., information, services, and applications), illustrated in Figure 1, to improve overall performance and productivity. Naturally, robust and reliable communication for a safe and secure nexus between the physical world and enterprise business can be a key to reaping the benefits of Industry 4.0.

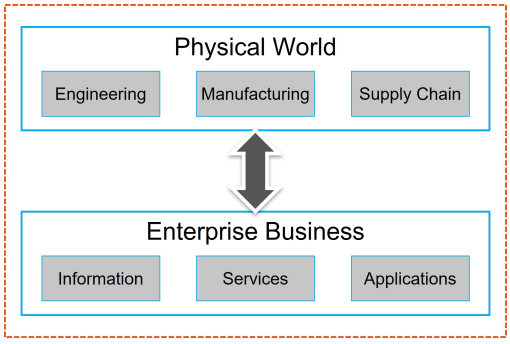


Fig. 1. Illustration of physical world and enterprise business

In an I4.0 environment, physical world devices like industrial PLCs, sensors, actuators, robots, and gateways need to communicate asynchronously for exchanging correct information in real time. Santos et al. [1] described that the communication protocol for data transmission is one of the vital elements of a learning and simulation environment in I4.0. Moreover, reliable data sharing between multiple entities (e.g., stakeholders, machines, etc.) is profoundly significant for both business-level collaboration and manufacturing operations-level collaboration.

Raptis et al. [2] explained that data integrates the physical and cyber worlds enabling digital twins to interact, represent their physical counterparts, and extract knowledge. The authors investigated data management aspects, such as data presence (i.e., specifically defined, localized sources, or from pervasive data generators), data coordination (i.e., hierarchical management of industrial processes based on data input), and data computation (i.e., use of concentrated or distributed resources for data processing) to study the recent trends Industry 4.0.

Data sharing can also be useful for implementing automatic learning systems using AI and big-data technologies to manage inventory or predict maintenance requirements that in turn can improve the manufacturing processes and eliminate bottlenecks. Thus, data-sharing activities between multiple entities spanning across the shop floor, manufacturing zone, enterprise, and external suppliers or any third party are required for horizontal and vertical integration.

Profanter et al. [3] studied machine-to-machine (M2M) protocols like Message Queue Telemetry Transport (MQTT), Open Communication Standard Unified Architecture (OPC UA), Robot Operating System (ROS), and Data Distribution Service (DDS). Table I compares the basic features of these M2M protocols. Both MQTT and OPC UA support authentication via username and password or by using a private key infrastructure (PKI). ROS and DDS only support authentication via MAC Address using third-party packages and PKI authentication, respectively.

All the aforementioned protocols use the publish/subscribe (pub/sub) model which is an asynchronous communication paradigm. Publishers do not send events directly to subscribers, instead, a network of interconnected brokers is responsible for events delivery. In fact, publishers do not know who receives their events and subscribers are not aware of the source of information. To receive events, subscribers need to register interest with a broker through a filter. When a new event is published, brokers forward it to all subscribers which expressed a filter that matches the event. The full decoupling of the communicating entities enables dynamic and flexible

TABLE I
A COMPARISON BETWEEN M2M PROTOCOLS [3]

	MQTT	OPC UA	ROS	DDS
Communication	TCP	TCP, UDP	TCP, UDP	TCP, UDP, SHM
Patterns	Pub/Sub	Pub/Sub, RPC	Pub/Sub, RPC	Pub/Sub, RPC
QoS	Yes	No	No	Yes
Authentication	User, PKI	User, PKI	Mac	PKI
Encryption	Yes	Yes	No	Yes
APIs	Rest	ANSI C	Robotics (Java)	C, C++, C#, Java
Semantic Data	No	Yes	No	No

information exchange between a large number of entities. Among all the protocols in Table I, MQTT is the most widely used and de-facto standard for M2M communication [4]. It offers also confidentiality, but only between the publisher and the broker and between the broker and the subscriber but it does not support end-to-end encryption between publisher and subscriber.

In this paper, we propose a transparent message level end-to-end encryption layer (from publisher to subscriber and vice-versa) exploiting Ciphertext-Policy Attribute-based encryption (CP-ABE) [5] for MQTT protocol securing communication in I4.0. The implementation uses the Fast Attribute Message Encryption (FAME) CP-ABE scheme [6]. The proposed MQTTS is implemented entirely in Golang, leveraging the MQTTv3 and MQTTv5 libraries to offer full support for the protocol. Our design can transparently realize data confidentiality and integrity for protocols relying on publish/subscribe model without any modification of the original protocol. Finally, we evaluate the FAME CP-ABE scheme time and space performance.

The rest of the paper is organized into the following sections: Section II gives the background of MQTT protocol and Attribute-based encryption. Section III presents the design and implementation of the proposed secure MQTT protocol. Section IV presents the setup details and performance performance results. Section V presents the related work. Finally, Section VI concludes the paper.

II. BACKGROUND

In this section, we briefly explain the MQTT protocol which is the most commonly used messaging protocol for the Internet of Things (IoT) and the Attribute-based encryption schemes.

A. Message Queue Telemetry Transport (MQTT)

MQTT is a lightweight application-level communication protocol that connects devices using `publish/subscribe` design pattern depicted in Figure 2. The protocol assembles rules that define how IoT devices can publish and subscribe to data over the Internet to establish asynchronous communication. The sender (Publisher) and the receiver (Subscriber) communicate via Topics and the connection between them is coupled by an MQTT broker, which is the only entity having full knowledge of the MQTT network composition. The MQTT broker filters all the incoming messages and distributes them correctly to the Subscribers. Publishers and subscribers can access more than one topic, and topics can be organized hierarchically.

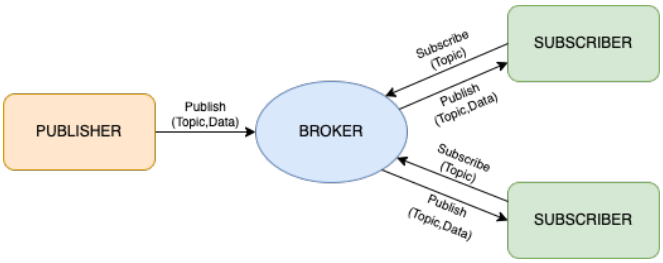


Fig. 2. Publisher-Subscriber model

Figure 3 illustrates the MQTT packet structure that consists of a Fixed header, a Variable header, and the Payload. The two bytes long fixed header is always present in each packet. However, the variable header and the payload size can vary and can be present in some packets only. Retain flag is only used on PUBLISH messages. QoS level decides for messages to be delivered at most once, at least once, or exactly once. DUP flag can be set to zero to indicate the message is being sent for the first time. The first four most significant bits are used for defining the command types, i.e., *CONNECT* (0x10), *CONNACK* (0x20), *UNSUBSCRIBE* (0xA0), etc.

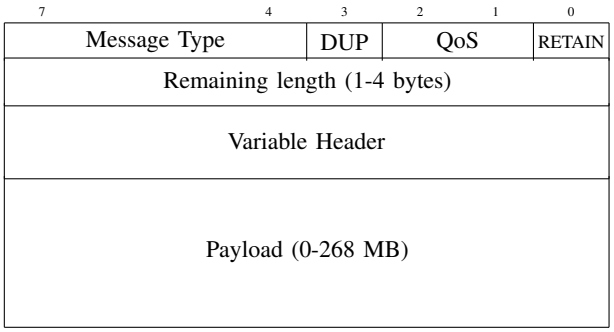


Fig. 3. MQTT header structure

B. Attribute-Based Encryption

Attribute-based encryption (ABE) can be specified as asymmetric or public-key encryption. ABE is well suited for a large-scale dynamic ecosystem like I4.0 because both the senders and receivers do not need a shared secret key. Thus, ABE offers simplified key management using a set of attributes or policies for encrypting messages. Furthermore, ABE is collusion resistance which is a critical security feature. The primary ABE schemes are Key Policy Attribute-Based Encryption (KP-ABE) and Ciphertext Policy Attribute-Based Encryption (CP-ABE).

In KP-ABE, the plaintext is encrypted using a set of attributes, and a user's secret key defines an access policy. In CP-ABE, a user secret key is defined based on attributes, and a ciphertext specifies an access policy. The access policy

structure is usually a tree and allows expressing any monotone access formula consisting of AND, OR, or threshold gates.

III. Proposed Solution

In this section, we specify the threat model assumed by the proposed secure MQTT (MQTTS) design, MQTTS design, workflow, message format, and compliance with MQTT. MQTTS deploys the Fast Attribute Message Encryption (FAME) CP-ABE scheme, a lightweight, secure system based on Elliptic Curves Cryptography (ECC).

A. Threat Model

While there are settings where publishers, brokers, and subscribers all belong to the same organization and can be assumed to be trusted, that's not the general scenario. As I4.0 scenarios are getting more complex, it is more realistic to assume first of all the possibility to have a network of brokers and not a single one, furthermore, some of these brokers can be external to one's security domain, or even provided as a third party service. In that case, brokers cannot be assumed to be trusted. MQTTS assumes an honest-but-curious attacker model for publishers, brokers, and subscribers, as in [7]. This means that the entities follow the protocol, but may be curious to find out information by analyzing the messages that are exchanged. For example, a broker may try to read the content of an event or try to learn the filtering con- strains of subscribers. Subscribers may want to read the events delivered to other subscribers. We also assume that a passive attacker outside the MQTTS system may be able to listen to the communication aiming at breaching the confidentiality of the participants.

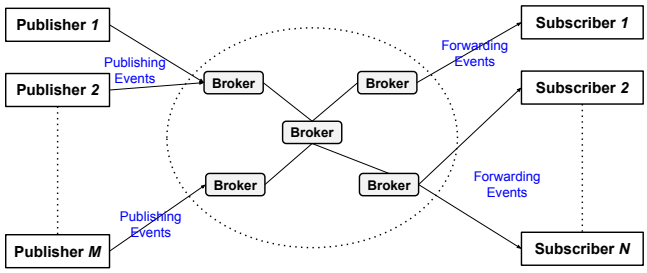


Fig. 4. Message exchange between the clients, i.e., publishers and subscribers, and the broker

MQTT implementations typically rely on other layers, such as TLS, certificates, or IPSec at the network layer, to adhere to security requirements. However, such ad-hoc security mechanisms adversely impact the non-functional requirements [8]. Particularly, the performance of IoT devices with limited computational and power supply get affected due to additional workload and overhead introduced by ad-hoc security mechanisms.

B. Assumptions and Constrains

We assume that an appropriate key management system is already employed that can provide keys to the devices using a secure channel. The implementation comes in the form of a wrapper for the existing Golang MQTT library, including a keygen and an encoder/decoder for the FAME CP-ABE keys. Further, in this work, only CP-ABE was used for encrypting MQTT payloads, while for worst-case analysis, the "AND" logic for access policy is applied. The selection of CP-ABE over KP-ABE enables easier attribute revocation. However, this flexibility is more expensive in terms of encryption performance.

C. Design

We design a secure MQTT by exploiting a lightweight Elliptic Curve Cryptography CP-ABE scheme that enables publishers and subscribers to perform encryption and decryption operations automatically and transparently. Consequently, a publisher or subscriber that can only satisfy the access policy will be able to decrypt a message, hence during the communication, the MQTT broker or any other entity cannot eavesdrop on the content of the message.

ABE deployment requires 1) setup, 2) encryption, and 3) decryption operation. During the setup phase, a trusted third party is selected that can perform the role of Public Key Generator (PKG) for generating the master secret key and public/private keys. A sending device encrypts the data using the public key and the access policy provided by the PKG. While a receiving device decrypts the ciphertext using the private key corresponding to the attributes contained in the policy. An MQTTS packet can be distinguished by prefixing a 12-byte internal header to the encrypted payload shown in Figure 5.

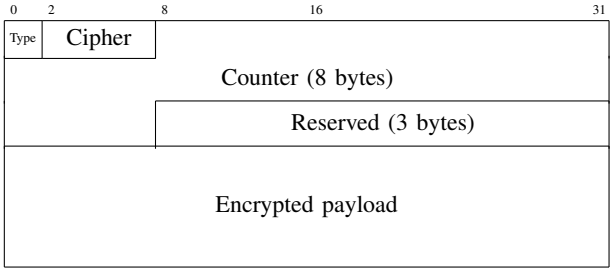


Fig. 5. MQTTS header structure

D. MQTTS Workflow

Figure 6 illustrates a sequence diagram for the proposed MQTTS protocol workflow. Publisher devices, subscriber devices (or devices), and PKG (trusted third party) are the three main entities. There can be four stages, i.e., setup, encryption, publish, and decryption. The setup stage requires registration with the PKG and key management scheme. In the encryption stage, the publisher device encrypts the data using the given public key provided and an access policy that can be self-generated or injected by the PKG. In the publishing stage, the encrypted data is appended to the 12-bytes header, then the flow is identical to the standard MQTT protocol, with the message being sent to the broker with a specific topic and then forwarded to all the subscribers to that topic. In the

decryption phase, the subscriber device decrypts the ciphertext using its private attribute key if it satisfies the access policy of the ciphertext.

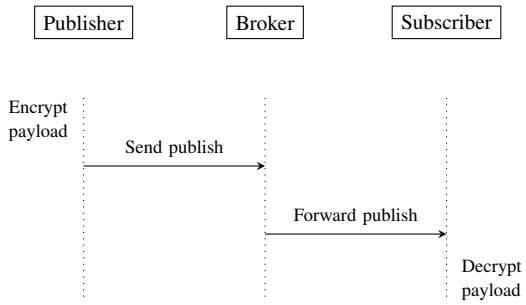


Fig. 6. Sequence diagram for a secure message flow

The use of a CP-ABE scheme fulfills the non-interactive and offline requirements of the PKG. Theoretically, the PKG is not required after the setup stage for CP-ABE. However, KP-ABE or other encryption schemes would require PKG to handle the key management at all stages. Thus, CP-ABE schemes are more generic and support the scalability requirement for I4.0. Furthermore, as there are no changes to the MQTT header or the protocol specifications, an MQTTS packet is transparent to any MQTT standard-compliant implementation. The MQTTS-aware peers can identify the encryption mechanism using the header placed before the encrypted payload.

E. Message Format

An encryption-aware client can identify the encryption scheme using the first byte of the internal header shown in Figure 5. The existing MQTTS implementation supports two symmetric encryption schemes (AESGCM, CHACHA20Poly1305) and one asymmetric (FAME CP-ABE).

- **Type** (2 bits) indicates the type of encryption: symmetric or asymmetric.
- **Cipher** (6 bits) indicates the concrete encryption algorithm used.
- **Nonce** (64 bits) is the nonce used in symmetric AEAD ciphers.
- **Reserved** (24 bits) reserved for future possibilities

When using asymmetric encryption, the nonce should be set to zero. Sending the nonce along with the ciphertext is a necessary design choice because keeping track of the nonce in an IoT scenario with one-way asymmetric communication would be impossible. The nonce being public does not raise any security concerns since it is how AEAD ciphers work.

F. Discussion on MQTT Compliance

Compliance with the MQTT standard is guaranteed because the header is not modified. Since the extra header is contained in the MQTT payload, it is unnecessary to modify anything else in the MQTT settings since the underlying library will handle everything transparently. Also, enabling options in the MQTT configuration will not change the protocol's behavior since the payload is treated separately. The MQTT implementation is, in fact, unaware of the content of the payload, and this allows for full transparency and compliance with the standard. If no encryption method is set, MQTTS will fall back to the standard unencrypted MQTT protocol.

G. Implementation Details

We use the Go language (Golang[1]) for the MQTTS implementation and third-party libraries for the CP-ABE schemes and MQTT. The third-party libraries provide a **crypto** library containing the code to generate, encode and decode the FAME CP-ABE keys. Each client maintains its *Connection State* with a status, a cipher, a message counter, and a boolean value indicating whether or not it is the publisher. The latter is required to know what to use for CP-ABE operations. The main interface is *mqtts.go* which acts upon *mqttv3* and *mqttv5* making the wrapper compatible with both the version of the protocol. The source code of the proposed solution can be accessed on GitHub[2].

IV. Performance Evaluation

This section presents the setup details and the performance evaluation results. It also covers the relevant studies that evaluated CP-ABE overhead. Waters and BSW CP-ABE are some of the reference algorithms for CP-ABE [5], [9].

A. Setup Details

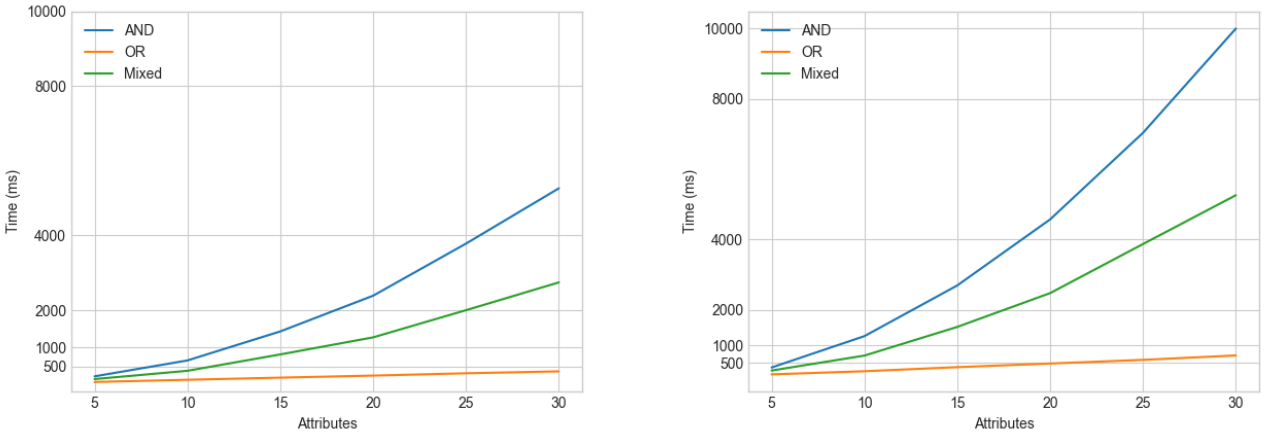
We conduct experiments on a Raspberry Pi 4b (Broadcom BCM2711, 2GB LPDDR4) and a Librecomputer "La Frite" (Amlogic S805X, 1GB SDDR4) to evaluate the protocol performances. Zickau et al. [10] perform multiple tests using a Raspberry Pi 2. The tests were performed only with CP-ABE encryption, which is the primary encryption algorithm for MQTTS and requires more resources. Girgenti et al. [11] analyzed the feasibility of using ABE in resource-constrained devices (ESP32). In our experiments, we consider decryption performances secondary because we focus on low-power IoT devices that will only encrypt and send the data. If it works for low-power IoT devices, it will be adequate for the more powerful ones that decrypt the data.

B. Results

For evaluating CP-ABE performances, we consider the number of attributes and the type of access policy (AND-policy, OR-policy) as the two most important parameters. The policy length and type originate the encryption overhead. A monotone span program (MSP) input is generated as per the selected policy and then used to encrypt the plaintext.

Figure 7 depicts the relations between policy size, encryption time and ciphertext size. When using the FAME CP-ABE scheme, the minimum obtainable ciphertext length with a single-attribute policy on a 100 bytes payload is approximately 3,9 KB. The difference between the two policy types is clear, with the OR-policy encryption being dramatically faster than the AND-policy. It can be observed in Figure 8b that the policy type

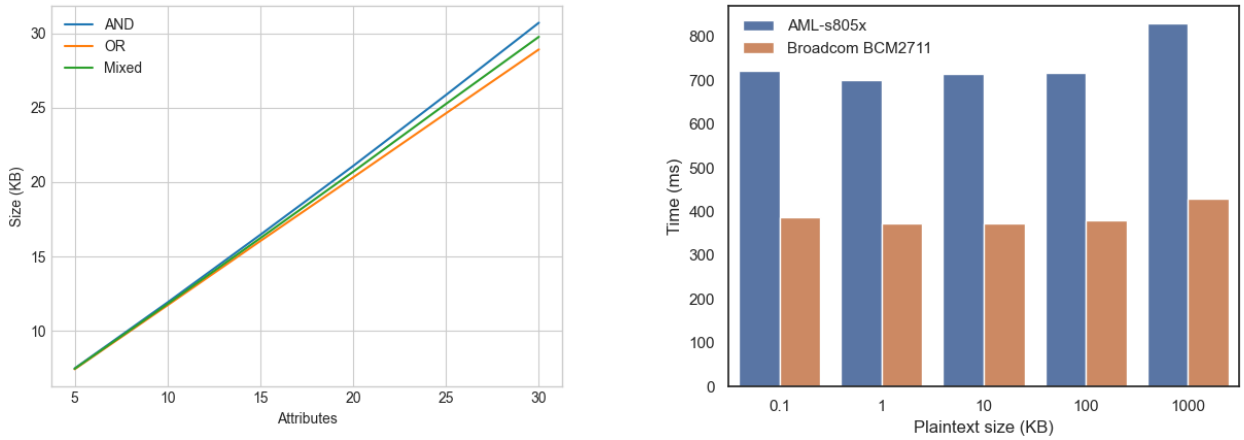
[1]https://go.dev/
[2]https://github.com/sacca97/unitn-mqtts



(a) Raspberry Pi 4 (Broadcom BCM2711)

(b) Libre Computer "La frite" (AML-S805X)

Fig. 7. FAME CP-ABE time performances



(a) Policy size - Ciphertext size relation

(b) Plaintext size - Encryption time relation

Fig. 8. FAME CP-ABE space performances

does not significantly affect the ciphertext size. On the other hand, Figure 8b shows that the plaintext size has a negligible effect on the encryption time. The test was performed using a mixed policy with up to ten attributes, which is comparable with the time of mixed policy in Figure 7.

C. Synopsis

For an I4.0 ecosystem, the CP-ABE scheme is more suitable for MQTTS to establish communication for systems/devices composed of a single board computer (SBC), however, sufficient computing power and storage is a prerequisite for dynamic access policies implementation. Moreover, managing the CP-ABE scheme requires less interaction and communication overhead (w.r.t. key distribution and management). Theoretically, the FAME CP-ABE scheme performs better than other known CP-ABE schemes guaranteeing more security, and thus can be a candidate for a real secure pub-sub architecture for IoT. On the other hand, KP-ABE is suitable for the access policies that are fixed and known apriori and the interaction with a key distribution center is feasible.

V. Related Work

In this section, we review research that focused on securing MQTT without adding overhead to the communication. Studies have described that the security issues like network mutual trust mechanism, encrypted transmission, DoS attacks, replay attacks, man-in-the-middle attacks, or anomaly detection, are of great concern for MQTT [12].

Mathur et al. [13] proposed end-to-end encryption using AES-GCM with a 256-bit key to secure MQTT. However, key management performed through Elliptic Curve Diffie-Hellman or HKDF is computation exhaustive and unsuitable for dynamic

scenarios involving resource-constrained devices. Gu et al. [14] proposed a solution based on Proxy re-encryption (PRE) is proposed. It leverages symmetric encryption, which does not overload the IoT devices like ABE. On the other hand, the middleware proxy is overloaded with homomorphic encryption computations. The second main drawback is that PRE security is based on trusting the proxy, in most cases, the broker. In a zero-trust environment, it is not a viable solution without adding another hop to the connection and thus adding more overhead.

Singh et al. [15] proposed a secure version of MQTT using both KP/CP-ABE, with Waters [5] scheme for the latter. The implementation's source code is not open, so it is impossible to know the work's details. Moreover, the empirical tests reported were not detailed enough to properly understand the feasibility of the work. Lastly, the FAME CP-ABE scheme used in this work supersedes theirs in both efficiency and security.

Su et al. [9] proposed a solution to secure MQTT by adding an end-to-end transparent encryption layer by modifying the MQTT Paho C library and adding symmetric encryption and CP-ABE, which is the baseline for this work. However, we propose different algorithms and extend the compatibility to MQTTv3 and MQTTv5 without directly modifying the library, thus, our implementation supports a more manageable library code. Moreover, our performance evaluation results show that the selection of the FAME CP-ABE scheme is more secure than the solution proposed by Galbraith [16] and more efficient than the one used by Su et al. [9].

VI. Conclusions and Future Work

In this paper, we discussed the importance of reliable data sharing in I4.0 and the pressing issue of missing E2E security in the MQTT protocol that is essential for centrally managed message brokers. We designed and implemented a secure MQTT wrapper (i.e., MQTTS) with a publish command that publishes encrypted data based on the FAME CP-ABE scheme using ECC. The mechanism is transparent to the MQTT standard and guarantees interoperability with existing MQTT deployments, thus, enables data confidentiality and integrity for data sharing in I4.0. In future work, we will continue analyzing the security aspects of MQTTS, such as key revocation or the feasibility of applying a distributed offline CP-ABE scheme to optimize the performances.

References

[1] O. Gonçalves dos Santos, U. Ibusuki, and K. Schützer, "Development of the communication with mqtt protocol for industry 4.0," in *Proceedings of the 11th Conference on Learning Factories*, SSRN, 2021.

[2] T. P. Raptis, A. Passarella, and M. Conti, "Data management in industry 4.0: State of the art and open challenges," *IEEE Access*, vol. 7, pp. 97052–97093, 2019.

[3] S. Profanter, A. Tekat, K. Dorofeev, M. Rickert, and A. Knoll, "Opc ua versus ros, dds, and mqtt: performance evaluation of industry 4.0 protocols," in *Proceedings of the IEEE International Conference on Industrial Technology (ICIT)*, pp. 955–962, IEEE, 2019.

[4] M. Dahlmanns, J. Pennekamp, I. B. Fink, B. Schoolmann, K. Wehrle, and M. Henze, "Transparent end-to-end security for publish/subscribe communication in cyber-physical systems," in *Proceedings of the 2021 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems*, pp. 78–87, 2021.

[5] B. Waters, "Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization," in *Proceedings of the International workshop on public key cryptography*, pp. 53–70, Springer, 2011.

[6] S. Agrawal and M. Chase, "Fame: fast attribute-based message encryption," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pp. 665–682, 2017.

[7] A. Shikfa, M. Önen, and R. Molva, "Privacy-preserving content-based publish/subscribe networks," in *Emerging Challenges for Security, Privacy and Trust* (D. Gritzalis and J. Lopez, eds.), (Berlin, Heidelberg), pp. 270–282, Springer Berlin Heidelberg, 2009.

[8] S. Gupta, "Non-functional requirements elicitation for edge computing," *Internet of Things*, vol. 18, p. 100503, 2022.

[9] W.-T. Su, W.-C. Chen, and C.-C. Chen, "An extensible and transparent thing-to-thing security enhancement for mqtt protocol in iot environment," in *Proceedings of the Global IoT Summit (GIoTS)*, pp. 1–4, IEEE, 2019.

[10] S. Zickau, D. Thatmann, A. Butyrtschik, I. Denisow, and A. Küpper, "Applied attribute-based encryption schemes," in *Proceedings of the 19th International ICIN Conference-Innovations in Clouds, Internet and Networks*, pp. 1–3, 2016.

[11] B. Girgenti, P. Perazzo, C. Vallati, F. Righetti, G. Dini, and G. Anastasi, "On the feasibility of attribute-based encryption on constrained iot devices for smart systems," in *Proceedings of the IEEE International Conference on Smart Computing (SMARTCOMP)*, pp. 225–232, IEEE, 2019.

[12] F. Chen, Y. Huo, J. Zhu, and D. Fan, "A review on the study on mqtt security challenge," in *Proceedings of the IEEE International Conference on Smart Cloud (SmartCloud)*, pp. 128–133, IEEE, 2020.

[13] A. Mathur, T. Newe, W. Elgenaidi, M. Rao, G. Dooly, and D. Toal, "A secure end-to-end iot solution," *Sensors and Actuators A: Physical*, vol. 263, pp. 291–299, 2017.

[14] Z. GU, Y. GUO, and C. FANG, "End-to-end security solution for message queue telemetry transport protocol based on proxy re-encryption," *Journal of Computer Applications*, vol. 41, no. 5, p. 1378, 2021.

[15] M. Singh, M. Rajan, V. Shivraj, and P. Balamuralidhar, "Secure mqtt for internet of things (iot)," in *Proceedings of the fifth international conference on communication systems and network technologies*, pp. 746–751, IEEE, 2015.

[16] S. Galbraith, "New discrete logarithm records, and the death of type 1 pairings," 2014.