

A Secure Secret Key-Sharing System for Resource-Constrained IoT Devices using MQTT

Taku Noguchi*, Masato Nakagawa†, Masami Yoshida† and Alberto Gallegos Ramonet‡

*College of Information Science and Engineering, Ritsumeikan University

†Graduate School of Information Science and Engineering, Ritsumeikan University

‡Graduate School of Technology, Industrial and Social Sciences Tokushima University

Email: *noguchi@is.ritsumei.ac.jp

Abstract—The MQTT (Message Queue Telemetry Transport) protocol has garnered significant attention as a communication protocol for a variety of IoT applications. Although it is a lightweight and energy-efficient communication protocol, it is not equipped with sufficient security mechanisms by default. Usually, secure socket layer/transport layer security (SSL/TLS) is used as a security mechanism in the MQTT protocol. However, it is not suitable for resource-constrained IoT devices because of the huge computational load involved in public key cryptography. In this paper, we propose a lightweight secure secret key-sharing system based on a secret-sharing scheme for resource-constrained IoT devices. The proposed system uses a (k, n) -threshold secret-sharing scheme to securely share a secret key for data encryption between the publisher and its subscriber hosts without compromising the lightweight nature of the MQTT protocol. A prototype of the proposed system is implemented using real IoT devices and its effectiveness and performance are evaluated. The experimental results demonstrate that the proposed system outperforms existing public key-based systems in terms of key-sharing delay.

Index Terms—IoT, MQTT, secret sharing, secret key cryptosystem

I. INTRODUCTION

IN the Internet of Things (IoT), “things” are not only connected to the Internet directly, but also interconnected to each other, forming networks that are, in turn, connected to the Internet. With the widespread utilization of the Internet of Things (IoT), the communication protocol, message queue telemetry transport (MQTT)[1], has also become widely popular. It is a lightweight and energy-efficient communication protocol developed by IBM that was released in 1999 and standardized by the Open Architecture System (OASIS) in 2013. The MQTT protocol is one of the most widely used protocols for IoT applications because it uses less bandwidth to send messages than other protocols.

SHODAN[2] is a search engine used to explore IoT devices. The SHODAN search results reveal nearly 150,000 devices worldwide connected to the Internet using the MQTT protocol. However, the MQTT protocol is not equipped with sufficient security mechanisms by default—it does not adopt any encryption mechanisms to protect the data being transmitted. Instead, the MQTT standard specification strongly recommends the use of secure socket layer/transport layer security (SSL/TLS) for encryption[3]. The SSL/TLS protocols provide a secure communication channel between pairs of hosts. They use sym-

metric (secret key) and asymmetric (public key) encryption techniques to provide security for authentication, authorization, and encryption of the data being transmitted.

Although the SSL/TLS solution has become the standard mode of security for communication over the Internet, most MQTT-based IoT systems do not use the SSL/TLS solution. The main reason behind this is that the SSL/TLS protocols entail a significant amount of additional communication and computational overhead that cannot be supported by resource-constrained IoT devices[5]. The SSL/TLS protocols consume more than 100 KB of memory [4] and entail considerable resource consumption. Thus, although the computation of SSL/TLS security solutions is possible on smartphones and laptops with sufficient memory, it is not feasible on devices with limited computational resources, such as those on RFC7228 [7].

In order to solve the security deficit of the MQTT protocol used in most IoT applications, we propose a lightweight key-sharing system in this paper based on a secret-sharing scheme. A secret-sharing scheme is a technique to share a secret piece of information among a group of participants. It is primarily used to protect confidential information from being stolen, destroyed, or modified. The proposed system uses a secret-sharing scheme to securely share a secret key to encrypt the data being transmitted from a publisher to its subscriber hosts without compromising the lightweight nature of the MQTT protocol in IoT environments. Further, the viability of the proposed system is verified by implementing a prototype on real IoT devices and its effectiveness and performance are evaluated in terms of execution time of sharing and reconstruction of a secret key between the publisher and subscriber hosts.

Our proposed system was initially introduced in our previous work[8]. In this paper, we considerably extend the previous study by reimplementing a prototype on new other IoT devices (Raspberry Pi Zero WH board) and evaluating it from various perspectives through experiments.

The remainder of this paper is structured as follows. A brief outline of related works is provided in Section II. The fundamentals of the MQTT protocol is introduced in Section III. The proposed system is discussed in great detail in Section IV and evaluated in Section V. Finally, some concluding remarks are provided in Section VI.

II. RELATED WORKS

Several studies have proposed different approaches to improve the security measures adopted by MQTT in IoT environments. An extension of MQTT, secure MQTT (SMQTT), was proposed in [9] to provide lightweight key/ciphertext policy-attribute-based encryption (KP/CP-ABE). SMQTT uses an elliptic curve in a bilinear map to reduce computational overhead. However, even in this case, a broker host inherently suffers from the computational overhead required to generate the master secret key and access policy. Further, the evaluation of the performance of SMQTT did not include resource-constrained IoT devices | therefore, its viability for these classes of devices has not been proven.

Niruntasukrat et al. [10] proposed an authorization mechanism for MQTT by considering the limited computational resources of IoT devices. The proposed mechanism is based on the OAuth 1.0a protocol and adopts several modifications to accommodate the restrictions of IoT devices. The authors tested the design on a real MQTT-based IoT service platform and verified its successful operation with minimal authorization delay and message overhead without affecting user experience. However, this mechanism only provides authorization services and does not support data encryption between the publisher and subscriber hosts.

Sanjuan et al. [11] proposed an authentication and encryption mechanism using cryptographic smart cards to secure MQTT communications. In the proposed mechanism, each host is assigned a public and a private key for data encryption. Although the proposed mechanism provides authentication and creates a secure communication channel without modifying the MQTT standard specification, computational and communication overheads due to the use of public key cryptography are still high for resource-constrained IoT devices.

Munster et al. [12] proposed an iterative secret share propagation scheme with Intel's Secure Guard Extensions (SGX) named HyShare to share secret information between publishers and subscribers. HyShare uses hybrid broker networks composed of two types of brokers: normal brokers and SGX-enabled brokers to reduce the number of messages required to share a secret. Although HyShare provides an efficient secret sharing scheme for an abstracted pub/sub infrastructure, it is a SGX-based hardware-centric approach and not designed for resource-constrained IoT devices using the MQTT protocol.

To the best of our knowledge, this paper is the first attempt to propose a secret key-sharing system based on a secret-sharing scheme for secure MQTT communication.

III. FUNDAMENTALS OF MQTT

MQTT is a lightweight application-layer messaging protocol based on the publish/subscribe (pub/sub) model (Fig. 1). The MQTT protocol uses TCP as the underlying transport layer protocol for transmission. It usually requires a fixed header of 2-bytes with small message payloads. It comprises three major components—publishers, subscribers, and brokers. A publisher is a device (e.g., sensors) that sends messages to the broker. A subscriber (e.g., applications interested in

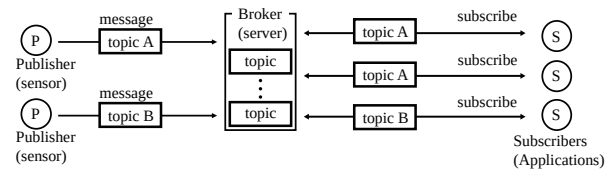


Fig. 1. MQTT Architecture.

certain topics or sensory data) specifically subscribes to the topic on which the publisher publishes a message. A broker is an intermediary device (e.g., a central server) that collects the message from the publisher and supplies it to the subscriber.

The MQTT protocol allows bidirectional communication between a publisher and subscribers via a broker. This helps in data sharing and managing and controlling the participating devices.

The MQTT protocol defines three levels of quality of service for message delivery—QoS 0, 1, and 2. QoS 0 (at most once) is the lowest tier. In this case, the sender (possibly the publisher or the broker) is not concerned if a transmitted message reaches the receiver and does not set up any retransmission mechanism. An example of the application of QoS 0 is an ambient sensing system, where the loss of an individual sensing data does not matter as the next one is published soon after. In QoS 1 (at least once), the message arrives one or more times at the receiver. In this case, after the transmission of a message, the sender waits for the receiver's acknowledgement. If it does not receive an ACK, it resends the message. In QoS 2 (exactly once), it is ensured that each message is received by the receiver exactly once. An example of the application of QoS 2 is a billing system, in which duplicate or lost messages could lead to the application of incorrect charges.

In the MQTT protocol, the provision of appropriate security features is delegated to the implementer. In general, security solutions can be achieved in multiple layers and are required to ensure various security options, e.g., authentication, authorization, and data encryption. TLS/SSL is an expedient solution to the security requirements of MQTT. It provides a secure communication channel between a publisher and a broker, or between a broker and a subscriber. However, the heavy cryptographic computation required by SSL/TLS security solutions is not feasible on IoT devices with limited computational resources.

IV. A KEY-SHARING SYSTEM USING A SECRET-SHARING SCHEME FOR MQTT

Public key cryptosystems such as TLS/SSL are not suitable for IoT environments because of the heavy computational requirements of prime number generation. Unfortunately, the execution time of prime number generation does not depend on the bit size of the prime number. This can be attributed to the randomness of prime number generation algorithms. The variation in the execution times of prime number generation in resource-constrained IoT devices is particularly large. This variability severely affects IoT systems that perform real-time processing. Lightweight cryptography with a low processing

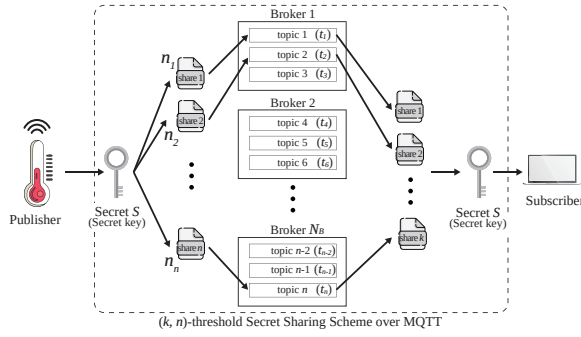


Fig. 2. An overview of the proposed system.

time variation is required for security in IoT environments. In this paper, we propose a new key-sharing system to share a secret key for data encryption without using a public key cryptosystem. As the proposed system does not use a public key cryptosystem, it manages to share the secret key based on a low amount of computation. Further, it is light weight and exhibits low variation in processing time.

Figure 2 depicts an overview of the proposed secret-sharing system. In the proposed system, a publisher is taken to be an IoT device that collects data (e.g., a sensor that measures temperature, movement, etc.). A subscriber is taken to be a user who monitors and observes the data collected by IoT devices (publishers) in real time. We assume the brokers to be in a secure and trusted environment. Before transmitting the acquired data, the publisher ensures that the communication is encrypted by sharing a secret key with the subscribers using the threshold secret-sharing scheme. Secure encrypted communication between the publisher and subscribers becomes possible after the secret key is shared.

A. Key-Sharing Phase

A secret-sharing scheme is a method to share a piece of confidential information among a group of nodes. The first secret-sharing scheme was proposed by Shamir in 1979 [13]. In such schemes, a secret, S , is encoded into n pieces called shares. The n shares are subsequently shared among n shareholders. Such a scheme is also called a (k, n) -threshold scheme if it satisfies the following basic security requirements of a secret-sharing scheme: (1) S can be reconstructed from any subset of k or more shares, and (2) S cannot be reconstructed from any subset of $k - 1$ or fewer shares. In this scenario, the integer k is called the threshold. This scheme is information-theoretically secure because it satisfies the two aforementioned requirements without making any computational assumptions.

In the proposed system, the key-sharing phase is initiated first. An IoT device (publisher) generates a secret key, S , for secure encrypted communication between itself and the desired users (subscribers). This secret key is shared using the MQTT protocol. The publisher encodes S into n shares ($n_1, n_2, \dots, n_n$) by using the (k, n) -threshold secret-sharing scheme and publishes them corresponding to n topics ($t_1, t_2, \dots, t_n$) on multiple brokers (Fig. 2). The key is periodically updated by the publisher to enhance security.

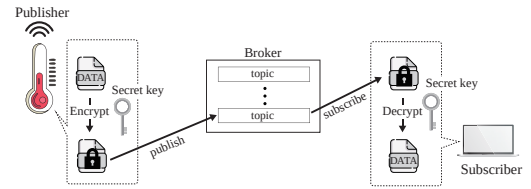


Fig. 3. Secure Data Delivery Phase.

The number of brokers N_B is set adaptively based on the threshold values, (k, n) . For example, if $(k, n) = (3, 5)$ and only two brokers (B_1 and B_2) are available, the five shares are distributed among B_1 and B_2 . In this case, all possible combinations of the numbers of shares held by B_1 and B_2 are $(\beta_1 = 0, \beta_2 = 5)$, $(\beta_1 = 5, \beta_2 = 0)$, $(\beta_1 = 1, \beta_2 = 4)$, $(\beta_1 = 4, \beta_2 = 1)$, $(\beta_1 = 2, \beta_2 = 3)$, and $(\beta_1 = 3, \beta_2 = 2)$. Here, β_1 and β_2 represent the number of shares held by B_1 and B_2 , respectively. In any of these scenarios, if the broker holding the higher number of shares is attacked and the shares are stolen by a malicious third party, the latter can obtain at least k shares and reconstruct S . Therefore, N_B must be taken to be at least k , i.e., $N_B \geq k$. Further, the maximum number of shares held by each broker must be less than or equal to $k - 1$ to prevent the reconstruction of S by a malicious third party if one broker is robbed of its shares in an attack.

Subscribers interested in the data generated by the publisher specifically subscribe to at least k topics corresponding to k shares. They receive k shares on k topics from the brokers and reconstruct S based on these k shares.

The system parameters, n , k , and N_B , depend on the IoT applications and IoT system environments. These parameters are determined carefully in advance based on the number of available brokers and the application's security and performance requirements. Information about $(t_1, t_2, \dots, t_n)$, e.g., the name of the topic, t_i , and the addresses of the brokers that hold t_i , is also pre-shared between the publisher and the subscribers via a secure offline/online channel. For example, the information can be pre-written in the memory of an IoT device (publisher) and pre-installed in applications (subscribers). Compared to the direct pre-sharing of the secret key, S , only pre-sharing the information about $(t_1, t_2, \dots, t_n)$ is more flexible to update the key. As the secret key, S , is not pre-shared and only the topic information, $(t_1, t_2, \dots, t_n)$, that is required to reconstruct S is pre-shared, subscribers who join the system at any given time can obtain the latest secret key.

B. Secure Data Delivery Phase

Following the key-sharing phase, a secure data delivery phase is initiated (Fig. 3). During this phase, the publisher encrypts its data using the secret key shared during the key-sharing phase and transmits the encrypted data to the broker corresponding to a specific topic. Subscribers interested in this topic subscribe to it and receive the encrypted data from the broker. The subscribers then decrypt the data received using the secret key. If a subscriber is subscribed to a particular topic, the broker automatically transmits any new data on that topic to the subscriber.

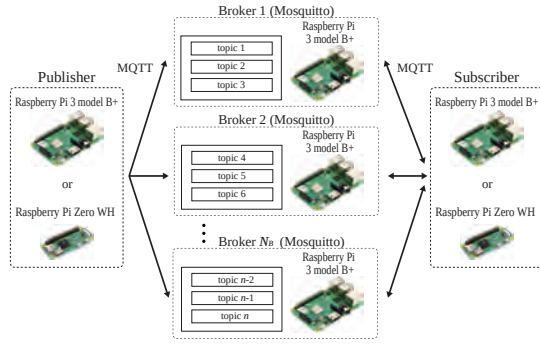


Fig. 4. Emulated IoT Environment.

TABLE I
PUBLISHER/BROKER/SUBSCRIBER TECHNICAL SPECIFICATION

Publisher	Raspberry Pi 3 model B+, Raspberry Pi Zero WH
Broker	Raspberry Pi 3 model B+
Subscriber	Raspberry Pi 3 model B+, Raspberry Pi Zero WH
MQTT Broker	Mosquitto (C)
Multi-precision arithmetic library	GMP-6.1.2

The topic names and addresses of brokers used during the secure data delivery phase do not need to be pre-shared. The publisher can create new topics and even use brokers that were not used during the key-sharing phase.

V. EXPERIMENTAL EVALUATION

A. Experimental Setting

In order to evaluate the performance of the proposed system, a small and emulated IoT environment comprising a Raspberry Pi 3 model B+ and Raspberry Pi Zero WH boards is constructed. Figure 4 depicts the emulated IoT environment for the proposed system. As is evident from the figure, one Raspberry Pi 3 model B+ board or one Raspberry Pi Zero WH board is used as both the publisher and the subscriber. N_B Raspberry Pi 3 model B+ boards are used as brokers. They communicate with each other using the MQTT protocol. An open-source Mosquitto-based MQTT broker [14] that can be implemented on all devices, from low-power computational devices to full servers, is used. The specifications of the publisher, brokers, and subscribers used in our experiments are tabulated in Table I. The detailed technical specifications of the Raspberry Pi Boards are recorded in Table II.

TABLE II
RASPBERRY PI BOARD TECHNICAL SPECIFICATION.

	Raspberry Pi 3 model B+	Raspberry Pi zero WH
CPU type	ARM Cortex-A53	ARM1176JZF-S
No. Of Cores	4	1
CPU clock	1.4GHz	1GHz
RAM	1GB	512MB
OS	Raspbian	Raspbian
Network	802.11/g/n WLAN	802.11/g/n WLAN
Power consumption	459mA (2.295W)	150mA (0.75W)

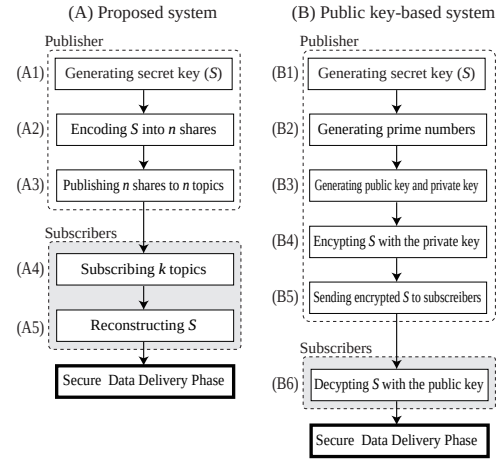


Fig. 5. System Flowchart.

In order to verify the effectiveness of the proposed system, it is compared with a simple public key-based system. In a public key-based system, the publisher generates both a public key and a private key using the Rivest–Shamir–Adleman (RSA) algorithm [15]. This key generation process involves heavy computational requirements for prime number generation. The publisher encrypts the secret key for data encryption between the publisher and the subscribers using its private key and transmits the encrypted secret key to the subscribers. The subscribers then decrypt the encrypted key using the publisher's public key. Subsequently, the subscribers obtain the secret key.

B. Evaluation Metrics

Figure 5 depicts the system flowcharts of both the proposed system and the public key-based system. The following three quantitative metrics are used to measure system performance:

- *Encoding delay of secret sharing*: This is defined to be the time taken by the publisher to encode the secret key into n shares, i.e., the time taken to execute (A2) in Fig.5.
- *Reconstruction delay of secret sharing*: This is defined to be the time taken by the brokers to reconstruct the secret key from k shares, i.e., the time taken to execute (A5) in Fig.5. Neither the encoding delay nor the reconstruction delay includes communication delays between the publisher and the brokers. Further, these delays are purely dependent on computational resources, such as CPU and memory.
- *Key sharing delay*: This is defined to be the time taken to share the secret key between the publisher and the subscribers. This delay includes the communication delay between the publisher and the brokers. In the case of the proposed system, this is equal to the duration between the encoding of the secret key into n shares by the publisher and the reconstruction of the secret key by the subscribers, i.e., the time taken to execute steps (A2)–(A5) in Fig.5. In the case of the public key-based system, this is equal to the duration between encoding

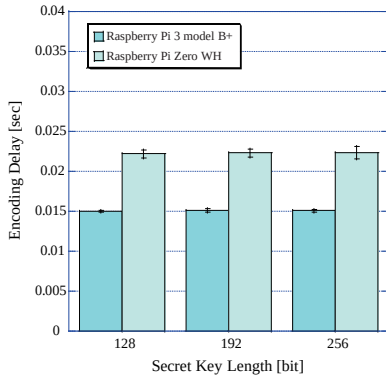


Fig. 6. Encoding delay performance of each of the two types of Raspberry Pi boards $((k, n) = (3, 5))$.

the prime numbers by the publisher and the decryption of the secret key by the subscribers, i.e., the time taken to execute steps (B2)–(B6) in Fig.5. The network latency (B5) is assumed to be zero for the public key-based system. Please note that this is an ideal environment for the public key-based system.

C. Experimental Results

In the following experiments, two scenarios are tested: i) the scenario in which both the publisher and the subscribers are taken to be Raspberry Pi 3 model B+ boards, and ii) that in which both are taken to be Raspberry Pi Zero WH boards. The average over 30 measurements were accepted as the experimental results.

1) *Different Secret Key Lengths*: In this experiment, the (k, n) -threshold scheme corresponding to $(3, 5)$ is used in the proposed system. The RSA algorithm is used to generate the secret key, whose key lengths are taken to be 128, 192, and 256 bits.

Figure 6 presents the encoding delay performances in the two scenarios. The error bars in the figure capture the standard deviation corresponding to the 30 measurements. As is evident from the figure, the encoding delay remains almost constant even when the key length is varied. In the proposed system, the length of the secret key, S , does not affect the encoding delay significantly. The encoding delay and the delay variance of the Raspberry Pi Zero WH board is observed to be greater than those of the Raspberry Pi 3 model B+ board owing to the lower CPU performance and smaller memory size of the former.

Figure 7 depicts the reconstruction delay performances in the two scenarios. As in the case of encoding delay, the reconstruction delay of the Raspberry Pi 3 model B+ board is also observed to remain almost constant even when the key length is varied. As the key length is increased, the reconstruction delay of the Raspberry Pi Zero WH board is observed to increase slightly. The reconstruction delay and delay variance of the Raspberry Pi Zero WH board is greater than those of the Raspberry Pi 3 model B+ board.

2) *Different (k, n) -threshold* : In the (k, n) -threshold scheme, it is reasonable to assume that both the encoding

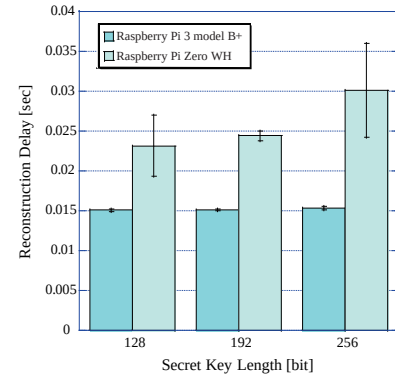


Fig. 7. Reconstruction delay performance of each of the two types of Raspberry Pi boards $((k, n) = (3, 5))$.

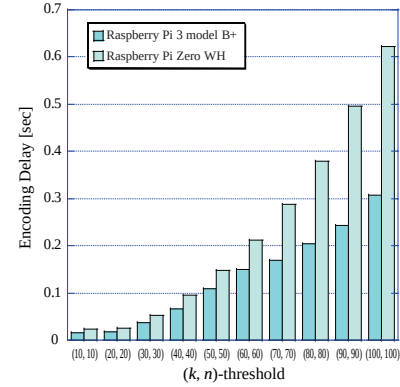


Fig. 8. Encoding delay performance of each of the two types of Raspberry Pi boards (secret key length = 128 bits).

delay and the reconstruction delay increase when either n is constant and k is increased or k is constant and n is increased. Therefore, k and n are assumed to increase from $(k, n) = (10, 10)$ to $(k, n) = (100, 100)$ with $k = n$. A 128-bit RSA secret key is used in this experiment.

Figure 8 depicts the encoding delay performances in the two scenarios. As (k, n) is increased, the encoding delay is observed to increase exponentially. Compared to that of the Raspberry Pi 3 model B+ board, the rate of increase of the encoding delay of the Raspberry Pi Zero WH board is

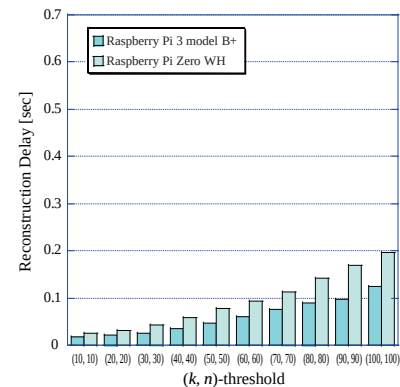


Fig. 9. Reconstruction delay performance of each of the two types of Raspberry Pi boards (secret key length = 128 bits).

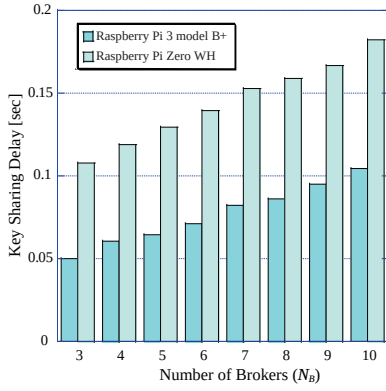


Fig. 10. Key-sharing delay performance of each of the two types of Raspberry Pi boards (secret key length = 128 bits, $k = n = N_B$).

significantly higher due to its lower CPU performance and smaller memory size.

Figure 9 depicts the reconstruction delay performances in the two scenarios. As in the case of encoding delay, the reconstruction delay is observed to increase exponentially as (k, n) is increased. From Figures 8 and 9, it is evident that the rate of increase of the encoding delay is much higher than that of the reconstruction delay. Therefore, determination of the values of (k, n) based on the encoding delay is essential.

3) *Different Numbers of Brokers*: The number of brokers, (N_B), is an important parameter of the proposed system. In this experiment, the effect of the number of brokers on key-sharing delay is examined. To this end, (N_B) is varied between 3 and 10. $k = n = N_B$ is assumed. This satisfies the condition, $N_B \geq k$, described in Section IV-A.

Figure 10 depicts the key-sharing delay performances in the two scenarios. Besides the encoding delay and the reconstruction delay, key-sharing delay also includes the communication delay between the publisher and the brokers. As N_B is increased, the key sharing delay also increases. The primary reason behind this is the corresponding increase in the computational load of publishing n shares corresponding to n topics (i.e., (A3) in Fig.5). Publishing corresponding to topics requires the connecting, transmitting, and closing operations corresponding to each broker. Thus, the required number of such operations increase in proportion to N_B . The key sharing delay of the Raspberry Pi Zero WH board is observed to be greater than that of Raspberry Pi 3 model B+ board due to the lower CPU performance and smaller memory size of the former.

4) *Comparison with the Public Key-based System*: In order to compare the overall delay performances of the proposed system and the existing public key-based system, we evaluated their key-sharing delays. In this experiment, the system parameters, $k = n = N_B = 10$, are used for the proposed system. As depicted in Fig.10, the key sharing delay is observed to be the worst delay performance in this case. The 1024-bit RSA public key is used for the public key-based system.

The key-sharing delay performances of the proposed system and the public key-based system in the two scenarios are

presented in Table III. The minimum, median, maximum, and standard deviation (SD) of the key sharing delay over a total of 30 measurements are tabulated in Table III. The delay-increasing-ratio (B/A) is calculated by dividing the key-sharing delay of the public key-based system (B) by that of the proposed system (A), which represents the delay performance ratio between the public key-based system and the proposed system. The key-sharing delay of the proposed system is observed to be significantly lower than that of the public key-based system in both Raspberry Pi 3 model B+ and Raspberry Pi Zero WH scenarios. The median of the key-sharing delay of the public key-based system is observed to be approximately 13.5–14.8 times higher than that of the proposed system. The longer delay of the public key-based system is attributed to the heavy computational load of prime number generation. The delay performance of the proposed system is verified to be stable with a small variation. The standard deviations in the Raspberry Pi 3 model B+ and Raspberry Pi Zero WH scenarios are observed to be 0.00658 and 0.01085, respectively. On the other hand, the variation of the delay performance of the public key-based system is significantly larger than that of the other. The standard deviations in the Raspberry Pi 3 model B+ and Raspberry Pi Zero WH scenarios are 0.68271 and 1.54697, respectively. In the scenario using the Raspberry Pi Zero WH board with lower CPU performance and smaller memory size, the variation of the key-sharing delay becomes much larger.

Our experimental results clearly establish the superiority of the proposed system over the public key-based system in an IoT environment consisting of resource-constrained IoT devices as the former exhibits a lower key-sharing delay.

VI. CONCLUSIONS

In order to resolve the end-to-end, i.e., publisher-to-subscriber, security problem of the MQTT protocol, a lightweight key-sharing system is proposed in this study based on a secret-sharing scheme. The proposed system is verified to be particularly suitable for resource-constrained IoT devices as it uses a lightweight and less resource-intensive secret-sharing scheme compared to the heavyweight and resource-intensive public key cryptosystem to share the secret key. Based on experiments on real IoT devices, the performance of the proposed secret-sharing scheme is evaluated by comparing the delay performance of the proposed system with that of the existing public key-based system. The experimental results establish that the proposed secret-sharing scheme improves the key-sharing delay performance.

Relevant topics of further research include the improvement of the delay performance by optimizing the system parameters (k, n , and N_B), and the validation of the proposed system in a variety of IoT environments consisting of various IoT devices with different computational resources. Further, an experimental comparison of performance with existing secure MQTT systems is also required.

TABLE III
KEY SHARING DELAY.

	Raspberry Pi 3 Model B+				Raspberry Pi Zero WH			
	Key sharing delay [s]				Key sharing delay (s)			
	Min	Median	Max	SD	Min	Median	Max	SD
Proposed system (A)	0.09456	0.10315	0.11932	0.00658	0.16640	0.17928	0.20572	0.01085
Public key-based system (B)	0.45010	1.39075	3.28044	0.68271	1.14498	2.65575	8.32784	1.54692
Delay-increasing-ratio (B/A)	4.760	13.483	27.493	—	6.881	14.813	40.481	—

REFERENCES

- [1] OASIS, "MQTT Version 5.0 OASIS Standard," <http://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>, March 2019, (accessed March 18, 2021).
- [2] SHODAN, <https://www.shodan.io/search?query=MQTT>, (accessed Mar. 18, 2021).
- [3] T. Dierks and E. Rescorla, "RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2," <https://tools.ietf.org/html/rfc5246>, Aug. 2008.
- [4] R. R. Pahlevi, P. Sukarno, and B. Erfianto, "Implementation of event-based dynamic authentication on MQTT protocol," *it Jurnal Rekayasa Elektrika*, vol. 15, no. 2, pp. 125-133, <https://doi.org/10.17529/jre.v15i2.13963>, Aug. 2019.
- [5] A. Gantait, J. Patra, and A. Mukherjee, "Securing IoT devices and gateways," in IBM Developer, <http://www.ibm.com/convey.pro/I/7BQYR5L>, May 2016, (accessed March 18, 2021).
- [6] P. Waher, "Learning Internet of Things," Packt Publishing Ltd. 2015.
- [7] C. Bormann, M. Ersue, and A. Keränen, "RFC7228: Terminology for Constrained-Node Networks," <https://tools.ietf.org/html/rfc7228>, May 2014.
- [8] M. Nakagawa, M. Yoshida, A. G. Ramonet, and T. Noguchi, "A Key Sharing System using a Secret Sharing Scheme for MQTT in an IoT Environment," *IEICE Technical Report*, vol. 120, no. NS-183, pp.63-67, Oct. 2020. (Japanese).
- [9] M. Singh, M. A. Rajan, V. L. Shivraj and P. Balamuralidhar, "Secure MQTT for Internet of Things (IoT)," in *Proceedings of 2015 Fifth International Conference on Communication Systems and Network Technologies*, pp. 746-751, doi: 10.1109/CSNT.2015.16, 2015.
- [10] A. Niruntasukrat, C. Issariyapat, P. Pongpaibool, K. Meesublak, P. Aiumsupucgul and A. Panya, "Authorization mechanism for MQTT-based Internet of Things," in *Proceedings of 2016 IEEE International Conference on Communications Workshops (ICC)*, pp. 290-295, doi: 10.1109/ICC.2016.7503802, 2016.
- [11] E. B. Sanjuan, I. A. Cardiel, J. A. Cerrada and C. Cerrada, "Message Queuing Telemetry Transport (MQTT) Security: A Cryptographic Smart Card Approach," in *IEEE Access*, vol. 8, pp. 115051-115062, doi: 10.1109/ACCESS.2020.3003998, 2020.
- [12] J. Munster and H.-A. Jacobsen, "Secret sharing in pub/sub using trusted execution environments," in *Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems*, pp. 28-39, doi: 10.1145/3210284.3210290, 2018.
- [13] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612-613, DOI:<https://doi.org/10.1145/359168.359176>, Nov. 1979.
- [14] Mosquitto, <https://mosquitto.org/> (accessed Mar. 18, 2021).
- [15] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120-126, DOI:<https://doi.org/10.1145/359340.359342>, Feb. 1978.



Taku Noguchi He received the B.E., M.E., and Ph.D. degrees in communications engineering from Osaka University, Osaka, Japan in 2000, 2002, and 2004, respectively. He joined the College of Information Science and Engineering at Ritsumeikan University in 2004, where he is currently a professor. His research interests include performance analysis and the design of computer networks and wireless networks. He received the best tutorial paper award from IEICE ComSoc in 2012. He is a member of the IEEE, IEICE, and IPSJ.



Masato Nakagawa He received the B.E. and M.E. degrees in Information Science and Engineering from Ritsumeikan University, Shiga, Japan in 2019 and 2021, respectively. His research interests include IoT system, with emphasis on IoT security.



Masami Yoshida He received the B.E. and M.E. degrees in Information Science and Engineering from Ritsumeikan University, Shiga, Japan in 2016 and 2018, respectively. At present he is Ph.D. candidate at the same university. His research interests include mobile ad hoc networks, with emphasis on network coding. He is a student member of IEICE.



Alberto Gallegos Ramonet He received the B.E. degree in Computer Science from Guadalajara University, Jalisco, Mexico in 2005. He later received the M.S. and Ph.D. degrees in engineering from Ritsumeikan University, Shiga, Japan in 2014 and 2018, respectively. In the College of Information Science and Engineering at Ritsumeikan University, he was an Assistant Professor from 2018 to 2021. He is currently an Assistant Professor with the Graduate School of Technology, Industrial and Social Sciences, Tokushima University. His current research interests include but are not limited to wireless sensor networks, network simulations, network visualizers, routing protocols and anything related to IoT and connectivity of small devices.