

Article

# A Novel Secure End-to-End IoT Communication Scheme Using Lightweight Cryptography Based on Block Cipher

Agus Winarno \* and Riri Fitri Sari \*

Department of Electrical Engineering, University of Indonesia, Depok 16424, Indonesia

\* Correspondence: agus.winarno01@ui.ac.id (A.W.); riri@ui.ac.id (R.F.S.)

**Abstract:** Personal data security is a cybersecurity trend that has captured the world's attention. Governments, practitioners and academics are jointly building personal data security in various communication systems, including IoT. The protocol that is widely used in IoT implementation is MQTT. By default, MQTT does not provide data security features in the form of data encryption. Therefore, this research was carried out on the design of Secure End-to-End Encryption MQTT with Block Cipher-Based Lightweight Cryptography. The protocol is designed by utilizing the Galantucci secret sharing scheme and a lightweight cryptographic algorithm based on a block cipher. The algorithms used include AES-128 GCM mode, GIFT-COFB, Romulus N1, and Tiny JAMBU. We tested the Secure End-to-End for MQTT protocol on the ARM M4 and ESP8266 processors. Our testing results on NodeMCU board, Tiny JAMBU have an average encryption time of 313  $\mu$ s and an average decryption time of 327  $\mu$ s. AES-128 GCM mode has an average encryption time of 572  $\mu$ s and an average decryption time of 584  $\mu$ s. GIFT-COFB has an average encryption time of 1094  $\mu$ s and an average decryption time of 1110  $\mu$ s. Meanwhile, Romulus N1 has an average encryption time of 2157  $\mu$ s and an average decryption time of 2180  $\mu$ s. On STM32L4 discovery, Tiny JAMBU had average encryption of 82  $\mu$ s and an average decryption time of 85  $\mu$ s. AES-128 GCM mode has an average encryption time of 163  $\mu$ s and an average decryption time of 164  $\mu$ s. GIFT-COFB has an average encryption time of 164  $\mu$ s and an average decryption time of 165  $\mu$ s. Meanwhile, Romulus N1 has an average encryption time of 605  $\mu$ s and an average decryption time of 607  $\mu$ s. Our experiment shows that the fastest performance is produced by Tiny JAMBU, followed by AES-128 Mode GCM, GIFT-COFB and Romulus N1.

**Citation:** Winarno, A.; Sari, R.F. A Novel Secure End-to-End IoT Communication Scheme Using Lightweight Cryptography Based on Block Cipher. *Appl. Sci.* **2022**, *12*, 8817. <https://doi.org/10.3390/app12178817>

Academic Editors: Chun-Yen Chang, Charles Tijus, Teen-Hang Meen and Po-Lei Lee

Received: 28 July 2022

Accepted: 27 August 2022

Published: 1 September 2022

**Publisher's Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Copyright:** © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

**Keywords:** MQTT; block cipher; lightweight cryptography; secret sharing; IoT

## 1. Introduction

The implementation of 5G is a catalyst in the growth of the Internet of Things (IoT). Based on data from Ericson, the number of IoT devices connected to the internet network will reach 14.6 billion devices in 2021 and will increase to 30.2 billion devices in 2027 [1]. The McKinsey Global Institute also stated that the global IoT market could grow to 6.2 trillion dollars by 2025. The IoT market share is dominated by devices from the healthcare sector, which are worth 2.5 trillion dollars, and the manufacturing sector, which is worth around 2.3 trillion dollars [2,3]. By 2027, the 5G market potential for IoT devices will reach \$297.8 billion [4]. The enormous IoT market also creates issues related to personal data protection [5–7]. Personal data protection is part of the goal of information security. Information security is a long and continuous process, not an instant result [8]. The fulfillment of information security services is a complex process that must be carried out carefully, starting from the establishment, implementation, and evaluation of information security policy compliance [9]. Effective information security must ensure a rational balance between requirements and controls provided to operate optimally and adequately [10]. Information security operates dynamically following technological

developments, so it is necessary to find new methods through research channels, both offensive and defensive methods, in accordance with applicable regulations [8]. This information security need must be met in the IoT communication protocol.

The communication protocol that is widely used in IoT implementations is MQTT [11]. By default, this protocol is not equipped with a data security system, so a security mechanism is needed for the protocol [12]. MQTT security can be achieved in the network [13], transport [14], and application [15]. Security at the user data layer can be achieved by encrypting the payload. In this research, a security design for the MQTT protocol is carried out using a Secure End-to-End Encryption MQTT with Block Cipher-Based Lightweight Cryptography. The use of end-to-end encryption is expected to be a data security solution at the application layer even though it uses an untrusted infrastructure.

Research related to MQTT security includes MQTT security at the application layer with the LBLOK, AES-128, PRESENT, Trivium, and Grain algorithms [16]. In addition, there is research on MQTT/MQTT-SN security with the ChaCha20-Poly1305 algorithm with AEAD mode [17] and MQTT security with a key delegation system for end-to-end encryption communication [18]. Each protocol is an MQTT security solution at the application layer.

In this study, Secure End-to-End Encryption on the MQTT protocol with Block Cipher-based Lightweight Cryptography was designed and implemented using the Galantucci secret sharing mechanism [19]. This secret sharing scheme is designed using multiple XOR operations, which are suitable for low-power IoT devices. The Secure End-to-End Encryption MQTT protocol design uses a lightweight cryptographic algorithm based on a block cipher, namely GIFT-COFB [20], Romulus [21,22], and Tiny JAMBU [23]. This scheme has several advantages:

1. Secure End-to-End Encryption on the MQTT protocol with Block Cipher-based Light-weight Cryptography can be used for low power devices;
2. This scheme provides convenience in key communication management by using the advantages of Galantucci secret sharing;
3. It can be implemented for brokers who do not support the use of SSL/TLS or untrusted brokers.

The paper is structured as follows: Section 1 describes the introduction; Section 2 highlights related work; and Section 3 presents some of the literature reviews. Section 4 explains some experimental methods. Section 5 proposes the Secure End-to-End Encryption Scheme on the MQTT protocol with Block Cipher-based Lightweight Cryptography. Section 6 shows experimental results and discussion. Section 7 discusses our conclusions.

## 2. Related Works

The IoT has grown significantly in the industrial era 4.0. The utilization of the IoT has tremendous market potential, with an economic value of 5.5 trillion to 12.6 trillion dollars in 2030 [24]. In addition, IoT is also widely used for lifestyle fulfillment, process efficiency, and health. Privacy of data transmitted via IoT is of particular concern and handling [25]. Information security in transmission is an important aspect that must be considered.

MQTT is an IoT communication protocol that plays an important role in sending messages from publishers to subscribers. MQTT security can be done at the network, transport, and application levels. Research that has been done related to data security in the MQTT protocol at the application layer, among others, is the study of Peniak et al. [16] about secure communication models for IoT and MQTT. This research uses a symmetric key algorithm, namely PRESENT, LBLOK, AES-128, Trivium, and GRAIN. The use of symmetric keys, especially LWC in this study, is due to the consideration of limited memory usage on embedded devices [16].

A similar study was conducted by Sadio [17], regarding the end-to-end security scheme of the MQTT protocol using the ChaCha20-Poly1305 algorithm with AEAD

mode. The choice of this algorithm is due to the limitations of devices that do not support the use of TLS at the transport layer. ChaCha20-Poly1305 is a popular lightweight stream cipher for devices with limited memory [17].

Research related to end-to-end encryption on MQTT was also carried out by Kumar et al. [18], who proposed an IoT security design in the form of **Joining Encryption and Delegation for IoT (JEDI)**. This scheme allows for end-to-end encryption with a key delegation system on each principal. Another feature that JEDI has is that it allows the encryption to be implemented in a decentralized key system, with decoupled communication and a tiered encryption system using the identity-based encryption algorithm with wildcard key derivation (WKD-IBE) [18].

Our research proposes a secure end-to-end encryption design for IoT communication protocols, especially MQTT, using a secret sharing scheme and a block cipher-based lightweight cryptography algorithm, i.e., AES-128 GCM mode, GIFT-COFB, ROMULUS N, and Tiny JAMBU. The secret sharing scheme used is secret sharing with the XOR model proposed by Galantucci et al. [19]. Table 1 describes the comparison of our research with previous research.

**Table 1.** Comparison with Previous Researchs.

No	Research	Author	Algorithm
1	Extended Model of Secure Communication for Embedded Systems with IoT and MQTT	Peter Peniak et al.	LBLOK, AES-128, PRESENT, Trivium dan Grain
2	Lightweight Security Scheme for MQTT/MQTT-SN Protocol	Ousmane Sadio et al.	ChaCha20-Poly1305
3	JEDI: Many-to-Many End-to-End Encryption and Key Delegation for IoT	Sam Kumar et al.	WKD-IBE
4	Secure End-to-End Encryption on the MQTT protocol with Block Cipher-based Lightweight Cryptography	Our Research	Secret Sharing, AES-128 GIFT-COFB, Romulus, dan Tiny JAMBU

### 3. Background Theory

Section 3 will relay the background theory related to MQTT, Eclipse Mosquitto, Advanced Encryption Standard (AES), Lightweight Cryptography, BAN logic, and Secret Sharing.

#### 3.1. Message Queuing Telemetry Transport (MQTT)

Message Queuing Telemetry Transport (MQTT) is an Internet of Things (IoT) protocol standard issued by OASIS [26]. MQTT is designed by carrying a publish/subscribe protocol that is lightweight and requires small bandwidth [27]. MQTT has been widely implemented in various industrial sectors, such as manufacturing, automotive, sports, energy, military, telecommunications, and health [28].

#### 3.2. Eclipse Mosquitto Sebagai MQTT Broker

Eclipse Mosquitto is an open-source project that aims to provide client and server implementation features according to the MQTT standard. Mosquitto was developed by Roger A. Light with the aim of implementing IoT with the needs of light messaging, low-power devices, and limited network resources based on the MQTT protocol [29]. The development of Mosquitto is under the Eclipse Foundation.

Eclipse Mosquitto has been widely implemented in the academic world and in real life. Mosquitto can be used as a means of evaluating the MQTT protocol on railway bridge

monitoring systems [30], data convergence in the electricity industry [31], and evaluating the performance of the publish-subscribe protocol on the IoT [32].

### 3.3. Advanced Encryption Standard (AES)

AES is a standard algorithm issued by NIST as a replacement for the Data Encryption Standard (DES). AES is designed as a symmetric key algorithm that manages messages in the form of blocks, often called block ciphers. NIST began announcing AES as a new standard starting from 26 November 2001 [33]. The algorithm chosen for AES is Rijndael, which was designed by Joan Daemen and Vincent Rijmen [34]. Rijndael was chosen as the winner of a contest for the encryption standard algorithm held by NIST for 5 years. AES is designed to process data in blocks of 128 bits. AES has three key variations that can be used, namely AES-128, AES-192, and AES-256 [35].

Structurally, AES has a simple structure so that the encryption and decryption process can use the same structure. The AES process can be divided into three phases, namely the initial round, the main round, and the final round. In the initial round phase, the algorithm processes messages with the add round key function, namely the process of XORing messages with keys. In the next phase, AES process messages with the functions of sub bytes, shift rows, mix columns, and add round key. Meanwhile, the final round performs the same function as the main round but eliminates the mix column function [36]. The AES algorithm above has four main components, namely sub bytes, shift rows, mix columns, and add round key [37].

### 3.4. Lightweight Cryptography Final Candidate

Lightweight cryptography is an algorithm designed to meet the needs of cryptography in limited conditions that are not supported by current cryptography. In August 2018, NIST held a competition for the creation of lightweight cryptography standards. In the first round of the competition, there were 56 algorithm proposals submitted by researchers and practitioners. The algorithms were selected by NIST and became 32 algorithms in the second round and became 10 algorithms in the final round, which was announced in March 2021. The finalist candidates for lightweight cryptography included ASCON, Elephant, GIFT-COFB, Grain128-AEAD, ISAP, Photon-Beetle, Romulus, Sparkle, Tiny JAMBU, and Xoodyak. Of the 10 algorithms, there are three algorithms based on block ciphers, i.e., GIFT-COFB, Romulus, and Tiny JAMBU.

#### 3.4.1. GIFT-COFB

The GIFT-COFB algorithm is a lightweight cryptography algorithm designed based on the GIFT block cipher algorithm with COMBined FeedBack (COFB) mode. The GIFT algorithm is an algorithm of the Substitution Permutation Network (SPN) type, which has two variants, namely GIFT 64-128 and GIFT-128-128. The two variants have differences in the length of the message block that is processed and the number of rounds. GIFT 64-128 can process 64-bit message blocks with a 128-bit key with 28 block iterations. Meanwhile, GIFT-128-128 can process 128-bit message blocks with 128-bit keys with 40 block iterations. In the GIFT-COFB algorithm, the variant used is GIFT-128-128 [20].

The COFB mode in the lightweight cryptography algorithm was designed by Chakraborti et al. with the aim of providing an authenticated encryption scheme. The COFB algorithm is an improvement algorithm of iCOFB to make it more efficient [38].

#### 3.4.2. Tiny JAMBU

Tiny JAMBU was one of the finalists in the lightweight cryptography algorithm selection contest organized by NIST. Tiny JAMBU is designed based on the JAMBU block cipher by providing data authentication features. Tiny JAMBU has three key variants, namely 128-bit (Tiny JAMBU-128), 192-bit (Tiny JAMBU-192), and 256-bit (Tiny JAMBU-

256). Tiny Jambu was designed by Hongjun Wu and Tao Huang by adding security parameters in the form of a 64-bit tag and a 96-bit nonce [23].

### 3.4.3. Romulus

Romulus is a lightweight cryptography mode AEAD algorithm based on Skinny 128/384+ Tweakable Block Ciphers. Romulus is designed to be optimally implemented on hardware with a security claim of 128-bit provable security [21,22]. Romulus has four variants, namely Romulus-N, Romulus-M, Romulus-T, and Romulus-H for hash functions.

### 3.5. BAN Logic

Burrows–Eternal–Needham (BAN) Logic is a formal testing method used to test protocols so that users can determine the security and trustworthiness of the information exchanged. This makes BAN logic often referred to as the logic of belief. BAN logic testing is carried out with the assumption that every exchange is carried out on an insecure medium and is susceptible to eavesdropping [39,40].

### 3.6. Secret Sharing

In this study, the use of secret sharing is aimed at supporting key management in the secure MQTT protocol. Secret sharing is a protocol scheme that supports multi-user communication by sharing sensitive information or a secret key into several parts. Such information can be recovered by a combination of part or all of the entity with a specified mechanism [41]. Secret sharing was first introduced by Adi Shamir with the aim of providing a solution for sharing secret keys in cryptographic communication [42].

Secret sharing protocols are widely used in our daily lives, such as in crypto currency wallets, for key management in vault by Hashicorp, and on Amazon Cloud HSM.

Secret sharing research has developed both in terms of method and efficiency. In 2012, Yi Sun et al. designed a secret sharing to simplify the reconstruction process without involving other entities [43]. In 2017, Kenko and Iwamura conducted a study to increase the efficiency of proactive secret sharing schemes in reducing communication traffic [44]. Another research project related to secret sharing was also carried out by Galantucci et al. regarding secret sharing with the XOR model, which is intended for distributed systems with many users [19]. This secret sharing scheme proposes a formal method, multiple XOR-ED secret sharing approach, and One Time User Key.

## 4. Method

The research flow used in the research includes problem identification, literature study, design and implementation, testing and analysis, and conclusion. Our research flow can be seen in Figure 1.

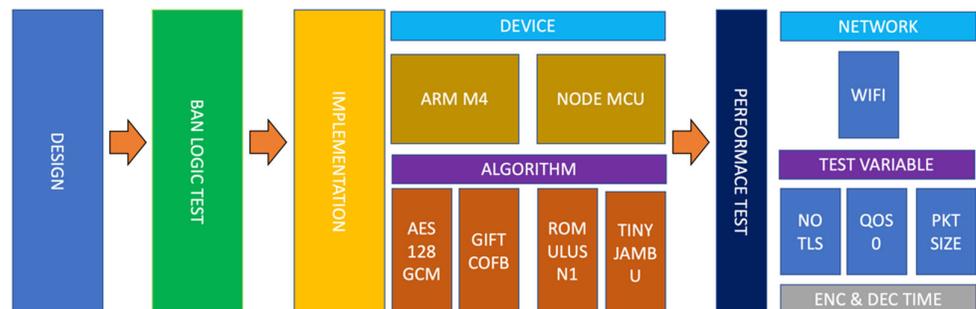


Figure 1. Research Flow.

We designed a secure end-to-end encryption for IoT communications by leveraging Galatucci’s secret sharing as a key management scheme for generation and rekeying. The algorithm used in the design is a final candidate for lightweight cryptography based on

block ciphers. Implementation is carried out at the communication stage on ARM M4 and NodeMCU.

The lightweight algorithm performance test was carried out on WiFi networks. Performance testing parameters include encryption, decryption, and publication of messages to the MQTT broker.

### 5. Secure End-to-End Encryption for Iot Communication Scheme

The design of the secure MQTT protocol aims to provide information security guarantees in the form of data confidentiality and authentication. This design can be applied to hardware devices that do not support SSL/TLS so that they can communicate securely even through the internet. The general design of the protocol can be seen in Figure 2.

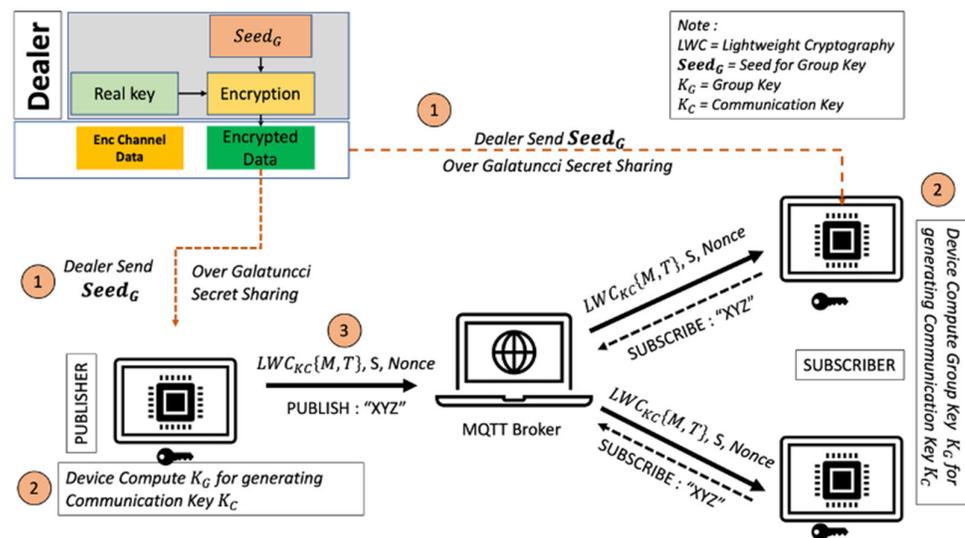


Figure 2. Secure End-to-End Design on MQTT Protocol.

Figure 2 provides an explanation of the working system of the Secure End-to-End Scheme on the MQTT Protocol. Here is the explanation:

1. The dealer sends a  $seed_G$  via Galatuncci’s secret sharing as a component in calculating the group key  $K_G$ ;
2. Each device that receives the  $seed_G$  can compute the group key  $K_G$ . Each device that is registered and has the same license will have the same group key  $K_G$ ;
3. Each device that has the same  $K_G$  group key can communicate with the session key  $K_C$  derived from the group key  $K_G$ . The cryptographic algorithm used in communication is lightweight cryptography (LWC), which provides data confidentiality and authentication features.

Secure end-to-end MQTT with lightweight cryptography designed based on Galatuncci’s secret sharing scheme [19] and lightweight cryptography. The secure end-to-end MQTT protocol was designed with the aim of providing encryption for message payloads sent over the MQTT protocol. The secret sharing scheme used is intended to make it easier for dealers to generate group keys, device management, and key updates. Updates are made if there is an addition or deletion of devices.

The secure end-to-end MQTT design is divided into three stages, i.e., registration and key generation, communication, and rekeying.

### 5.1. Registration and Key Generation

The registration process for the secure end-to-end MQTT protocol is the initial phase to register devices that communicate in the MQTT protocol. This phase is controlled by a factory key ( $K_F$ ) and a subscription key ( $K_S$ ). The factory key in the protocol is used for communication between dealers or third parties with IoT devices in a secret sharing scheme to generate group key seeds. Meanwhile, the subscription key functions as the master key in generating the group key in combination with the group key seed  $seed_G$ .

The registration phase of the secure end-to-end MQTT protocol is designed based on the secret sharing protocol developed by Galantucci et al. [19]. This protocol was chosen because it supports multi-user communication using light operations because it uses XOR operations. This scheme also has advantages in several key management schemes. Implementation of Galantucci's secret sharing scheme can be intended for several uses, including one time keys, groups of valid keys, and dealer key management.

The pre-registration phase aims to encrypt the channel before the registration as shown in the following process:

1. Set-up environment:
  - a. Dealers store information on salt values, factory key hashes, and device subscription keys.
  - b. Each device has a factory key and a subscription key and can access salt information.
  - c. The dealer and each device have the functions,  $\Omega$  and  $\omega$ , which satisfy Galantucci's secret sharing rule [19].
2. Encrypted Channel Creation Process
  - a. The dealer creates a one time user key for the IoT device and himself. One time user keys (OTUK) of IoT devices can be generated from the hash value of the factory key and device salt stored at the dealer.
  - b. Dealers generate communication keys and encrypted channels from dealer OTUK and OTUK IoT devices.
3. Process on IoT Device
  - a. The IoT device creates a device OTUK from the factory key and salt provided by the dealer.
  - b. IoT devices can communicate securely with encrypted channels created by dealers.
  - c. Through this encrypted channel, the device can exchange information or key seeds securely with the dealer.

The proposed registration scheme for Galantucci's secret sharing is as a secure end-to-end MQTT registration scheme as shown in Figure 3.

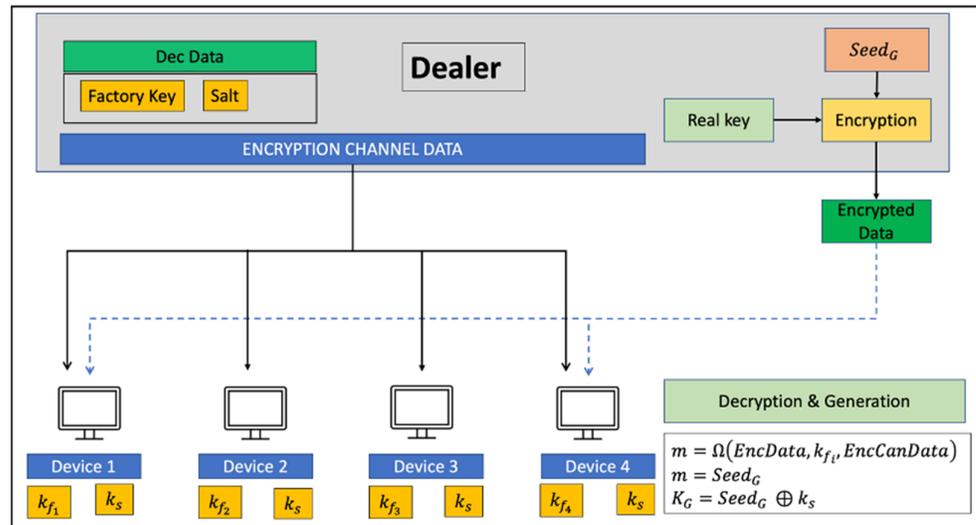


Figure 3. Group Key Generation Phase.

At this registration stage, each IoT device group key ( $K_G$ ) is the same to communicate in the MQTT protocol.

### 5.2. Communication

This phase is an important part of the protocol in the communication process between devices in the MQTT network. In this phase, all IoT devices that have the same group key ( $K_G$ ) can communicate and exchange information securely. The following is the proposed communication protocol:

1. Set-up environment:
  - a. Device A and Device B have Key Group  $K_G$
  - b. Device A generates a Nonce to generate a session key  $K_C$  as communication key
2. Encrypted Channel Creation Process
  - a. Calculates the session key ( $K_C$ ) used for communication.
 
$$K_C = K_G \oplus Nonce$$
  - b. Calculating the S signature to check the validity of the communication key  $K_C$ :
 
$$S = hash(Nonce || hash(KG)^{128})$$
  - c. Device A sends an encrypted message C to B via MQTT Broker:
 
$$E_{K_C}\{M, t_A\}, S, Nonce$$
3. Device B receives encrypted messages from Device A via the MQTT broker. Device B does calculate the following:
  - a. Calculating the Signature  $S'$  with the nonce information sent and  $K_G$  held:
 
$$S' = hash(Nonce || hash(KG)^{128})$$
  - b. If  $S' \neq S$ , the message is not processed. However, if  $S' = S$ , then go to step c.
  - c. Performs session key calculations for communication with received nonce information.
 
$$K_C = K_G \oplus Nonce$$
  - d. Decrypt the message with  $K_C$  calculated to get the message.
 
$$D_{K_C}\{E_{K_C}\{M, t_A\}\}$$
  - e. Device B gets messages M and  $t_A$ , the information  $t_A$  is used to check the freshness of the received message.

The protocol in the communication phase above can be visualized with a communication scheme as shown in Figure 4.

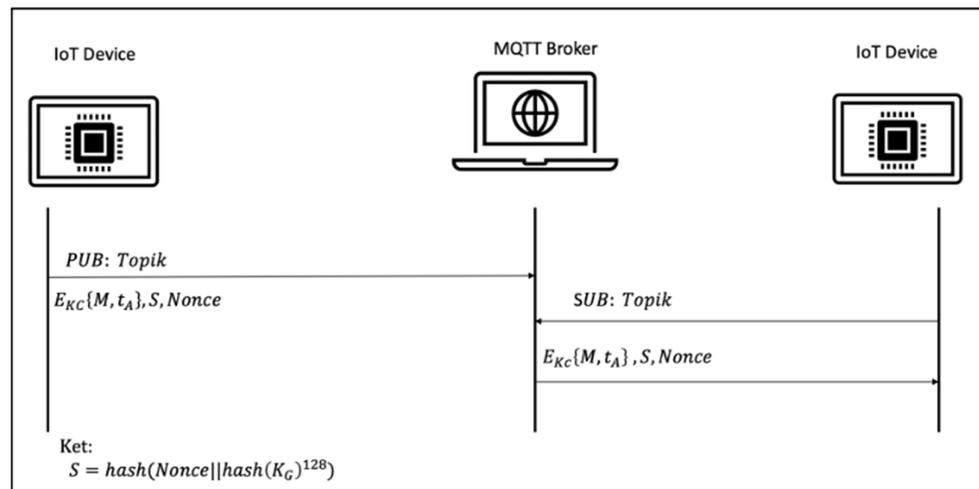


Figure 4. Communication Phase.

### 5.3. Rekeying

The key update process is a control in key management on the protocol in the use and renewal of group keys (group key). This phase makes it easy for the system to update the key when there is an addition or deletion of devices connected to the communication scheme. The key update scheme in this protocol takes advantage of the easy key management of the secret sharing scheme designed by Galantucci et al. [19]. The key update scheme for secure end-to-end on the MQTT protocol is as follows:

- The system is experiencing an increase or decrease in the number of IoT devices.
- Dealers update legitimate devices connected to the network
- The dealer updates the encrypted channel data and seed to generate the group key ( $K_G$ ) with the mechanism in Section 5.1.
- The dealer sends the seed to the legitimate device.
- Each legitimate device receives the seed and recalculates  $K_G$  as the group key to communicate.

## 6. Results

This section may be divided by subheadings. It should provide a concise and precise description of the experimental results, their interpretation, as well as the experimental conclusions that can be drawn.

### 6.1. BAN Logic Test

Our testing for the design of the secure MQTT protocol was carried out using the BAN logic method. The test is carried out based on the rules that apply in BAN logic, namely the message meaning rule, the non-verification rule, and the jurisdiction rule. The stages carried out in the BAN logic test consist of four steps, namely:

- Changing the protocol in its ideal form;
- Making assumptions from the ideal protocol;
- Explain each step in the ideal protocol;
- Carry out protocol evaluation with BAN logic rules.

The communication protocol for secure end-to-end encryption on the MQTT protocol has the following form:

$$A \rightarrow B : E_{KC}\{M, Nonce_A\}, S, Nonce \tag{1}$$

The steps that must be taken in testing the BAN logic in the secure MQTT design are:

1. Changing the protocol in the ideal form:

$$M1 A \rightarrow B : \left\{ \left( A \stackrel{K_{ab}}{\leftrightarrow} B \right), N_A \right\}_{K_{ab}}, \# \left\{ A \stackrel{K_{ab}}{\leftrightarrow} B \right\}_{K_{ab}} \quad (2)$$

2. Making assumptions from the ideal protocol:

$$P1 A \models A \stackrel{K_{ab}}{\leftrightarrow} B \quad (3)$$

$$P2 B \models A \stackrel{K_{ab}}{\leftrightarrow} B \quad (4)$$

$$P3 B \models A \Rightarrow A \stackrel{K_{ab}}{\leftrightarrow} B \quad (5)$$

$$P4 B \models \left( A \Rightarrow \# \left( A \stackrel{K_{ab}}{\leftrightarrow} B \right) \right) \quad (6)$$

$$P5 A \models \#(N_A) \quad (7)$$

3. Explain each step in the ideal protocol:

$$P6 B \triangleleft \{N_A\}_{K_{ab}} \quad (8)$$

4. Carry out protocol evaluation with BAN logic rules.

After M1

This test aims to prove that party *B* believes in the key to the communication between *A* and *B*, as well as the novelty of the key used.

- Jurisdiction rule

$$\frac{B \models A \Rightarrow (A \stackrel{K_{ab}}{\leftrightarrow} B), B \models A \models (A \stackrel{K_{ab}}{\leftrightarrow} B)}{B \models (A \stackrel{K_{ab}}{\leftrightarrow} B)} \quad (9)$$

$$\frac{B \models \left( A \Rightarrow \#(A \stackrel{K_{ab}}{\leftrightarrow} B) \right), B \models A \left( A \models \#(A \stackrel{K_{ab}}{\leftrightarrow} B) \right)}{B \models \#(A \stackrel{K_{ab}}{\leftrightarrow} B)} \quad (10)$$

After *B* trusts the communication key between *A* and *B*, BAN logic proves that the message sent is a new message through the trust nonce sent by *A*.

- Message meaning rule

$$\frac{B \models A \Rightarrow A \stackrel{K_{ab}}{\leftrightarrow} B, B \triangleleft \{N_A\}_{K_{ab}}}{B \models A \mid \sim N_A} \quad (11)$$

- Nonce Verification rule

$$\frac{A \Rightarrow \#(N_A), B \models A \mid \sim N_A}{B \models A \models N_A} \quad (12)$$

- Jurisdiction rule

$$\frac{B \models A \Rightarrow N_A, B \models A \models N_A}{B \models N_A} \quad (13)$$

In the BAN logic test, the two entities can trust each other. Party *B* believes that party *A* has the same access rights and group key. Party *B* believes that the message sent by *B* is a fresh message.

### 6.2. Performance On ESP 8266 and ARM M4

The process of implementing secure end-to-end encryption on the MQTT protocol is carried out on two hardware devices, namely STM32L4 Discovery (ARM M4) and NodeMCU (esp 8266). Both hardware are low-power microcontrollers that are suitable for

IoT applications. In the implementation of the protocol, it is only carried out at the communication stage.

At the design stage, the package sent from publisher to subscriber is  $E_{KC}\{M, Nonce_A\}, S, Nonce$ . At the implementation stage,  $S$  and  $Nonce$  are ignored to check the performance of the implemented cryptographic algorithms. The implemented communication design can be seen in Figure 5.

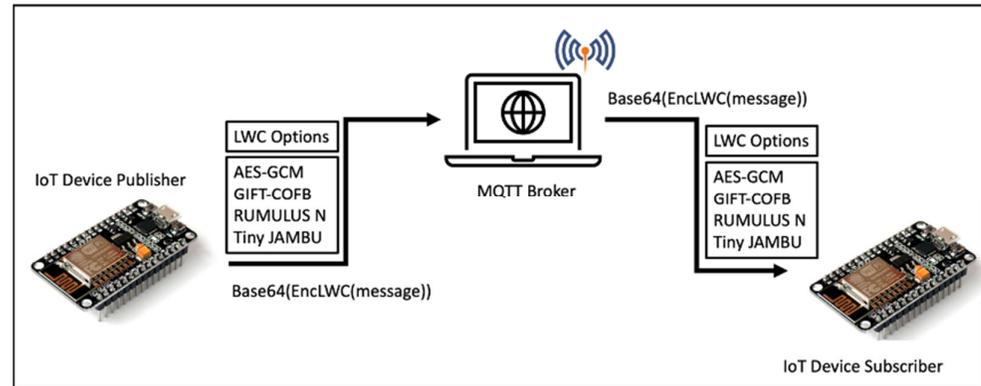


Figure 5. Secure MQTT Implementation Scheme with LWC.

In the process of implementing secure end-to-end encryption, the MQTT protocol utilizes a lightweight cryptographic algorithm that has features that can guarantee the confidentiality and authenticity of data. Lightweight cryptographic algorithms implemented in this study include AES-128 GCM mode, GIFT COFB, ROMULUS N, and Tiny JAMBU. These four algorithms are the final candidates in the lightweight cryptographic algorithm competition organized by NIST.

Testing the performance of the lightweight cryptography algorithm on STM32L4 and ESP8266 is done by setting several test parameters, namely encryption time, decryption time, and the time required for packet delivery. The test is carried out on a WiFi network with specifications as shown in Table 2:

Table 2. Test Parameters.

No	Parameter	Description	Value
1	WiFi	Download Speed	43.49 Mbps
		Upload Speed	46.73 Mbps
2	Messages	80 bit (10 Char)	hello eqmx
		120 bit (10 Char)	hello eqmx dear
		160 bit (10 Char)	hello eqmx dear agus

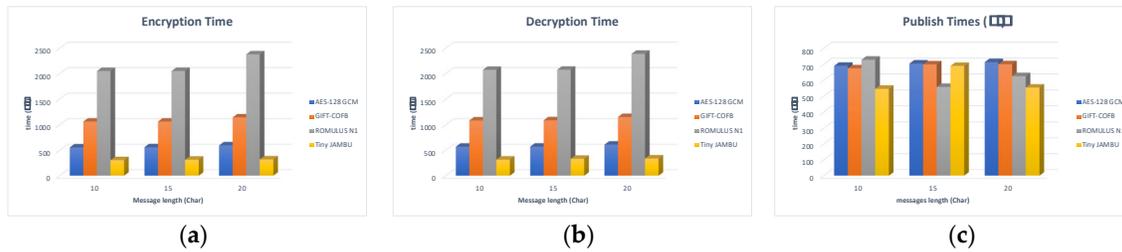
Based on Figure 5, encrypted messages are sent with base64 and encoded so that the encrypted messages are not changed during transmission from publisher to subscriber. Sending encrypted messages without base64 has the potential that the message cannot be decrypted because the encrypted message can be in the form of special characters that can affect the transmission process, such as Null, Delete, or Enter characters. The MQTT server used in this simulation is Eclipse Mosquitto version 2. This MQTT server supports MQTT version 5. In this study, the QOS used by the publisher is QOS0. The results of testing the secure MQTT protocol are presented in Sections 6.2.1 and 6.2.2.

### 6.2.1. Node MCU

The performance test of the lightweight encryption algorithm using anonymous and registered accounts on the MQTT server was carried out on the WiFi network. The registered account used for testing is “winarnolab” with the password “loop1206”. The test was carried out with three different message lengths, namely 10, 15, and 20 characters.

- Anonymous Account

The results of the tests carried out on the WiFi network as anonymous users can be seen in Tables 3–5 and Figure 6. Testing is done by sending messages with a length of 80 bits (10 characters), 120 bits (15 characters), and 160 bits (20 characters).



**Figure 6.** Lightweight Cryptographic Performance on NodeMCU on WiFi Network Used Anonymous Account. (a) Encryption Time; (b) Decryption Time; (c) Message Publication Time to MQTT Server.

**Table 3.** Lightweight Cryptographic Encryption and Decryption Performance on NodeMCU with 10 Characters Messages used Anonymous Account.

No	LWC Algorithm	Encryption Time (µs)	Decryption Time (µs)	Publication Time (µs)
1	AES-128 GCM	558	570	690
2	GIFT-COFB	1066	1087	674
3	Tiny JAMBU	306	317	545
4	Romulus-N1	2052	2075	728

**Table 4.** Lightweight Cryptographic Encryption and Decryption Performance on NodeMCU with 15 Characters Messages used Anonymous Account.

No	LWC Algorithm	Encryption Time (µs)	Decryption Time (µs)	Publication Time (µs)
1	AES-128 GCM	560	571	704
2	GIFT-COFB	1066	1090	699
3	Tiny JAMBU	315	332	557
4	Romulus-N1	2053	2077	689

**Table 5.** Lightweight Cryptographic Encryption and Decryption Performance on NodeMCU with 20 Characters Messages used Anonymous Account.

No	LWC Algorithm	Encryption Time (µs)	Decryption Time (µs)	Publication Time (µs)
1	AES-128 GCM	596	612	713
2	GIFT-COFB	1146	1156	700
3	Tiny JAMBU	321	338	553
4	Romulus-N1	2381	2391	625

From Table 3, it can be seen that Tiny JAMBU can encrypt messages with a time of 306 µs and decryption of 317 µs, for messages with a length of 10 characters. AES-128

GCM mode can perform encryption and decryption of 558  $\mu$ s and 570  $\mu$ s, respectively. GIFT-COFB takes 1066  $\mu$ s and 1087  $\mu$ s for encryption and decryption. Meanwhile, Romulus N1 is capable of encrypting 10-character messages for 2052  $\mu$ s and 2075  $\mu$ s and message publication to the MQTT server takes 728  $\mu$ s.

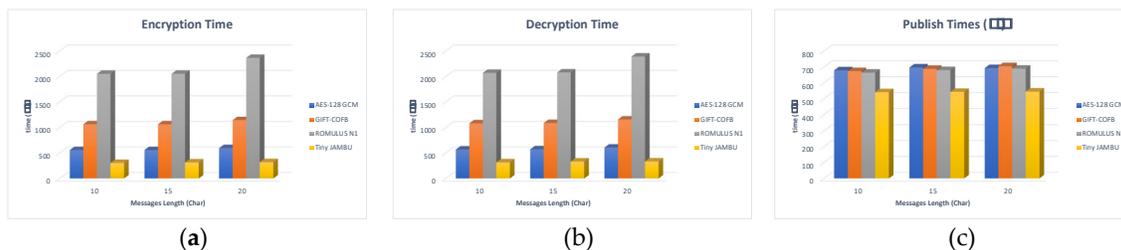
Table 4 shows the results of testing the secure end-to-end encryption on the MQTT protocol by sending a 15-character message on a WiFi network as an anonymous user on the MQTT server. Tiny JAMBU produces an excellent lightweight cryptographic algorithm with an encryption time of 315  $\mu$ s, a decryption time of 332  $\mu$ s, and a message publication time of 557  $\mu$ s. Meanwhile, the AES-128 GCM is still tight with an encryption time of 560  $\mu$ s and a decryption time of 571  $\mu$ s and a publication time of 704  $\mu$ s. This is followed by GIFT-COFB with an encryption time of 1066  $\mu$ s and a decryption time of 1090  $\mu$ s and Romulus N1 with an encryption time of 2053  $\mu$ s and a decryption time of 2077  $\mu$ s with a message publication time of 689  $\mu$ s.

Table 5 shows the results of testing secure end-to-end encryption on the MQTT protocol by sending messages of 20 characters long on the WiFi network as an anonymous user on the MQTT server. In the results of this test, the highest performance is still produced by Tiny JAMBU, with an encryption time of 321  $\mu$ s and a decryption time of 338  $\mu$ s.

Based on the results of testing secure end-to-end encryption on the MQTT protocol on the WiFi network as an anonymous user of the MQTT server, it was discovered that sending messages of 10 to 20 characters can be done in less than 1 ms and the fastest performance is produced by Tiny JAMBU, as can be seen in Figure 6.

- Registered account

Testing of secure end-to-end encryption on the MQTT protocol on a WiFi network as a registered user of the MQTT server can be seen in Tables 6–8 and Figure 7. Testing is done by sending messages with a length of 80 bits (10 characters), 120 bits (15 characters) and 160 bits (20 characters).



**Figure 7.** Lightweight Cryptographic Performance on NodeMCU on WiFi Network Used Registered Account. (a) Encryption Time; (b) Decryption Time; (c) Message Publication Time to MQTT Server.

**Table 6.** Lightweight Cryptographic Encryption and Decryption Performance on NodeMCU with 10 Characters Messages used Registered Account.

No	LWC Algorithm	Encryption Time ( $\mu$ s)	Decryption Time ( $\mu$ s)	Publication Time ( $\mu$ s)
1	AES-128 GCM	559	569	679
2	GIFT-COFB	1066	1084	674
3	Tiny JAMBU	305	317	541
4	Romulus-N1	2051	2071	664

**Table 7.** Lightweight Cryptographic Encryption and Decryption Performance on NodeMCU with 15 Characters Messages used Registered Account.

No	LWC Algorithm	Encryption Time ( $\mu$ s)	Decryption Time ( $\mu$ s)	Publication Time ( $\mu$ s)
1	AES-128 GCM	559	572	697
2	GIFT-COFB	1066	1090	688
3	Tiny JAMBU	315	332	543
4	Romulus-N1	2052	2078	680

**Table 8.** Lightweight Cryptographic Encryption and Decryption Performance on NodeMCU with 20 Characters Messages used Registered Account.

No	LWC Algorithm	Encryption Time ( $\mu$ s)	Decryption Time ( $\mu$ s)	Publication Time ( $\mu$ s)
1	AES-128 GCM	596	608	693
2	GIFT-COFB	1146	1160	705
3	Tiny JAMBU	321	334	545
4	Romulus-N1	2365	2391	689

From Table 6, it can be seen that Tiny JAMBU is able to encrypt messages with a time of 305  $\mu$ s and decryption of 317  $\mu$ s for messages with a length of 10 characters. GIFT-COFB takes 1066  $\mu$ s and 1084  $\mu$ s for encryption and decryption. AES-128 GCM mode can perform encryption and decryption with a time of 559  $\mu$ s and 569  $\mu$ s. Meanwhile, Romulus N1 can encrypt 10-character messages for 2051  $\mu$ s and 2071  $\mu$ s.

Table 7 provides an overview of the performance of lightweight cryptographic algorithms on secure MQTT on a WiFi network. The test results show tiny JAMBU has the fastest performance, with encryption and decryption times of 315  $\mu$ s and 332  $\mu$ s, respectively. Meanwhile, the lowest performance was produced by Romulus-N1, 2053  $\mu$ s for encryption and 2078  $\mu$ s for decryption.

Table 8 shows that tiny JAMBU and AES-128 GCM Mode can perform encryption and decryption in less than 1 ms. Tiny JAMBU can encrypt for 315  $\mu$ s and decrypt for 332  $\mu$ s. AES-128 GCM mode requires 596  $\mu$ s for encryption and 612  $\mu$ s for decryption. Meanwhile, GIFT-COFB and Romulus-N1 can perform encryption and decryption in 1 ms and 2 ms, respectively.

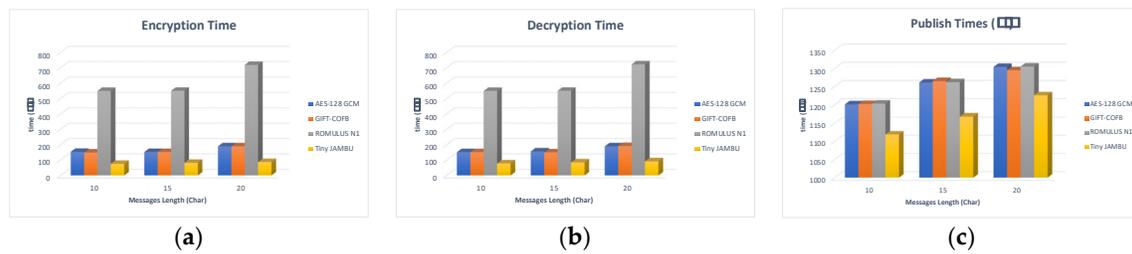
Figure 7 provides information that sending messages of 10 to 20 characters in the secure MQTT test on a WiFi network as a registered user of the MQTT server can be carried out in less than 1 ms and the fastest performance is produced by Tiny JAMBU with a time of 545  $\mu$ s.

### 6.2.2. STM32L4 Discovery

The performance test of the lightweight encryption algorithm using an anonymous and a registered account on the MQTT server was carried out on WiFi network. The registered account used for testing is “winarnolab” with the password “loop1206”. The test is carried out with three different message lengths, namely 10, 15, and 20 characters.

- Anonymous Account

The results of the tests carried out on the WiFi network as anonymous users can be seen in Tables 9–11, and Figure 8.



**Figure 8.** Lightweight Cryptographic Encryption and Decryption Performance on STM32L4 on WiFi Network Used Anonymous Account. (a) Encryption Time; (b) Decryption Time; (c) Message Publication Time to MQTT Server.

**Table 9.** Lightweight Cryptographic Encryption and Decryption Performance on STM32L4 with 10 Characters Messages Used Anonymous Account.

No	LWC Algorithm	Encryption Time (µs)	Decryption Time (µs)	Publication Time (µs)
1	AES-128 GCM	153	151	1201
2	GIFT-COFB	149	151	1202
3	Tiny JAMBU	75	78	1119
4	Romulus-N1	549	549	1203

**Table 10.** Lightweight Cryptographic Encryption and Decryption Performance on STM32L4 with 15 Characters Messages Used Anonymous Account.

No	LWC Algorithm	Encryption Time (µs)	Decryption Time (µs)	Publication Time (µs)
1	AES-128 GCM	152	156	1261
2	GIFT-COFB	152	150	1265
3	Tiny JAMBU	82	85	1168
4	Romulus-N1	550	550	1262

**Table 11.** Lightweight Cryptographic Encryption and Decryption Performance on STM32L4 with 20 Characters Messages Used Anonymous Account.

No	LWC Algorithm	Encryption Time (µs)	Decryption Time (µs)	Publication Time (µs)
1	AES-128 GCM	189	189	1304
2	GIFT-COFB	189	191	1295
3	Tiny JAMBU	87	92	1226
4	Romulus-N1	718	722	1305

From Table 9, it can be seen that Tiny JAMBU can encrypt messages with a time of 75 µs and decryption of 78 µs, for messages with a length of 10 characters. AES-128 GCM can perform encryption and decryption with 153 µs, and 151 µs. GIFT-COFB takes 149 µs and 151 µs for encryption and decryption. Meanwhile, Romulus N1 can encrypt 10-character messages for 549 µs and 549 µs and message publication to the MQTT server takes 1203 µs.

Table 10 shows the results of testing the secure end-to-end encryption on the MQTT protocol by sending a 15-character message on a WiFi network as an anonymous user on the MQTT server. Tiny JAMBU produces an excellent lightweight cryptographic algorithm with an encryption time of 82 µs, a decryption time of 85 µs, and a message publication time of 1168 µs.

Meanwhile, the AES-128 GCM is still tight with an encryption time of 152 µs and a decryption time of 156 µs and a publication time of 1261 µs. This is followed by GIFT-

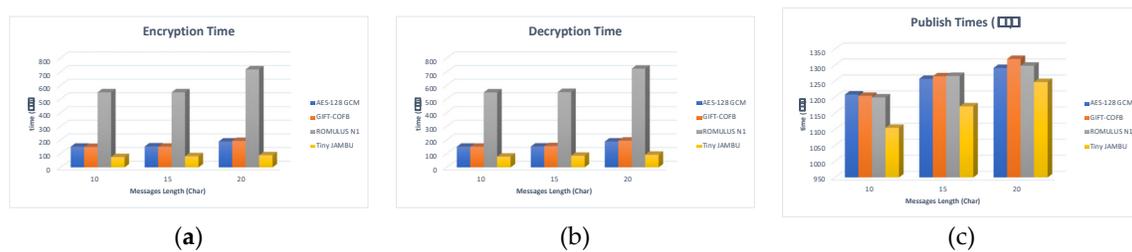
COFB with an encryption time of 152  $\mu$ s and a decryption time of 150  $\mu$ s and Romulus N1 with an encryption time of 550  $\mu$ s and a decryption time of 550  $\mu$ s with a message publication time of 1262  $\mu$ s.

Table 11 shows the results of testing secure end-to-end encryption on the MQTT protocol by sending messages of 20 characters long on the WiFi network as an anonymous user on the MQTT server. In the results of this test, the highest performance is produced by Tiny JAMBU with an encryption time of 87  $\mu$ s and a decryption time of 92  $\mu$ s.

Based on the results of testing secure end-to-end encryption on the MQTT protocol on the WiFi network as an anonymous user of the MQTT server, it was discovered that sending messages of 10 to 20 characters can be done in less than 1.4 ms and the fastest performance is produced by Tiny JAMBU, as can be seen in Figure 8.

- Registered Account

Testing of secure end-to-end encryption on the MQTT protocol on a WiFi network as a registered user of the MQTT server can be seen in Tables 12–14 and Figure 9. Testing is done by sending messages with a length of 80 bits (10 characters), 120 bits (15 characters), and 160 bits (20 characters).



**Figure 9.** Lightweight Cryptographic Encryption and Decryption Performance on STM32L4 on WiFi Network Used Registered Account. (a) Encryption Time; (b) Decryption Time; (c) Message Publication Time to MQTT Server.

**Table 12.** Lightweight Cryptographic Encryption and Decryption Performance on STM32L4 with 10 Characters Messages Used Registered Account.

No	LWC Algorithm	Encryption Time (µs)	Decryption Time (µs)	Publication Time (µs)
1	AES-128 GCM	151	151	1208
2	GIFT-COFB	149	151	1204
3	Tiny JAMBU	75	78	1105
4	Romulus-N1	549	548	1199

**Table 13.** Lightweight Cryptographic Encryption and Decryption Performance on STM32L4 with 15 Characters Messages Used Registered Account.

No	LWC Algorithm	Encryption Time (µs)	Decryption Time (µs)	Publication Time (µs)
1	AES-128 GCM	153	152	1257
2	GIFT-COFB	151	155	1265
3	Tiny JAMBU	81	85	1171
4	Romulus-N1	549	551	1266

**Table 14.** Lightweight Cryptographic Encryption and Decryption Performance on STM32L4 with 20 Characters Messages Used Registered Account.

No	LWC Algorithm	Encryption Time (µs)	Decryption Time (µs)	Publication Time (µs)
1	AES-128 GCM	189	189	1291

2	GIFT-COFB	193	195	1319
3	Tiny JAMBU	88	92	1247
4	Romulus-N1	717	722	1298

In Table 12, it can be seen that Tiny JAMBU is able to encrypt messages with a time of 75  $\mu$ s and decryption of 78  $\mu$ s for messages with a length of 10 characters. GIFT-COFB takes 149  $\mu$ s and 151  $\mu$ s for encryption and decryption. AES-128 GCM mode can perform encryption and decryption with a time of 151  $\mu$ s and 151  $\mu$ s. Meanwhile, Romulus N1 can encrypt 10-character messages for 549  $\mu$ s. and 548  $\mu$ s.

Table 13 provides an overview of the performance of lightweight cryptographic algorithms on secure MQTT on a WiFi network. The test results show tiny JAMBU has the fastest performance with encryption and decryption time of 81  $\mu$ s and 85  $\mu$ s, respectively. Meanwhile, the lowest performance was produced by Romulus-N1, 549  $\mu$ s for encryption and 551  $\mu$ s for decryption.

Table 14 shows that tiny JAMBU can perform encryption and decryption in less than 0.1 ms. Tiny JAMBU can encrypt for 88  $\mu$ s and decrypt for 92  $\mu$ s. AES-128 GCM and GIFT-COFB can perform encryption and decryption in less than 0.2 ms. AES-128 GCM requires 189  $\mu$ s for encryption and 189  $\mu$ s for decryption. GIFT-COFB requires 193  $\mu$ s for encryption and 195  $\mu$ s for decryption. Meanwhile, Romulus-N1 can perform encryption and decryption in 717  $\mu$ s and 722  $\mu$ s.

Based on the results of testing secure end-to-end encryption on the MQTT protocol on a WiFi network as a registered user of the MQTT server, information is obtained that sending messages of 10 to 20 characters can be done from 1.1 ms to 1.3 ms and the fastest performance is produced by tiny JAMBU with a publication time of 1105  $\mu$ s to 1247  $\mu$ s, as can be seen in Figure 9.

## 7. Conclusions

MQTT Secure End-to-End Encryption Design with Lightweight Cryptography Based on Block Cipher is designed by utilizing Galantucci's secret sharing, which makes it easy to perform key and user management and uses multiple XOR operations that are compatible with low-power IoT devices.

The performance of Secure End-to-End Encryption MQTT with Block Cipher-Based Lightweight Cryptography is performed on two development boards, i.e., NodeMCU and STM32L4 Discovery. The result of our testing in NodeMCU, the Tiny JAMBU algorithm can perform encryption for 313  $\mu$ s and decryption for 327  $\mu$ s. AES-128 GCM mode can encrypt for 572  $\mu$ s and decrypt for 584  $\mu$ s, while GIFT-COFB can encrypt for 1094  $\mu$ s and decrypt for 1110  $\mu$ s. Meanwhile, Romulus N1 has an encryption time of 2157  $\mu$ s and a decryption time of 2180  $\mu$ s. In STM32L4 Discovery, the Tiny JAMBU algorithm can perform encryption for 82  $\mu$ s and decryption for 85  $\mu$ s. AES-128 GCM mode encrypts for 163  $\mu$ s and decrypts for 164  $\mu$ s. GIFT-COFB can encrypt for 164  $\mu$ s and decrypt for 165  $\mu$ s. Romulus N1 encrypts for 605  $\mu$ s and decrypts for 607  $\mu$ s.

**Author Contributions:** A.W. and R.F.S., overall design scheme; A.W., implemented the communication phase in STM32L4 and NodeMCU; A.W. and R.F.S., conducted tests and data analysis; A.W. and R.F.S., wrote the paper. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work is supported by Universitas Indonesia under PUTI Q2 Grant with contract number NKB-710/UN2.RST/HKP.05.00/2022.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The data presented in this study are available from the corresponding author upon request.

**Acknowledgments:** This work is supported by Universitas Indonesia.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. IoT Connections Outlook. Available online: <https://www.ericsson.com/en/reports-and-papers/mobility-report/dataforecasts/iot-connections-outlook> (accessed on 6 June 2022).
2. Manyika, J.; Chui, M.; Bughin, J.; Dobbs, R.; Bisson, P.; Marrs, A. Disruptive Technologies: Advances That Will Transform Life, Business, and the Global Economy. Available online: <https://www.mckinsey.com/business-functions/mckinsey-digital/our-insights/disruptive-technologies> (accessed on 6 June 2022).
3. Chen, M.-C.; Ho, P.H. Exploring Technology Opportunities and Evolution of IoT-Related Logistics Services with Text Mining. *Complex Intell. Syst.* **2021**, *7*, 2577–2595. <https://doi.org/10.1007/s40747-021-00453-3>.
4. Research and Markets Global 5G in IoT Market by Connectivity, Infrastructure, Sensors, Devices (Type, Sector, Verticals), and Things 2022–2027. Available online: <https://www.researchandmarkets.com/reports/5387753/global-5g-in-iot-market-by-connectivity> (accessed on 6 June 2022).
5. Lee, G.Y.; Cha, K.J.; Kim, H.J. Designing the GDPR Compliant Consent Procedure for Personal Information Collection in the IoT Environment. In Proceedings of the 2019 IEEE International Congress on Internet of Things (ICIOT), Milan, Italy, 8–13 July 2019; pp. 79–81.
6. Rizou, S.; Alexandropoulou-Egyptiadou, E.; Ishibashi, Y.; Psannis, K.E. Protecting Minors' Personal Data in IoT-Based Smart Homes According to GDPR. In Proceedings of the 2021 IEEE 9th International Conference on Information, Communication and Networks (ICICN), Xi'an, China, 25–28 November 2021; pp. 269–272.
7. Oh, H.; Park, S.; Lee, G.M.; Heo, H.; Choi, J.K. Personal Data Trading Scheme for Data Brokers in IoT Data Marketplaces. *IEEE Access* **2019**, *7*, 40120–40132. <https://doi.org/10.1109/ACCESS.2019.2904248>.
8. van Daalen, O. In Defense of Offense: Information Security Research under the Right to Science. *Comput. Law Secur. Rev.* **2022**, *46*, 105706. <https://doi.org/10.1016/j.clsr.2022.105706>.
9. Karlsson, F.; Kolkowska, E.; Petersson, J. Information Security Policy Compliance-Eliciting Requirements for a Computerized Software to Support Value-Based Compliance Analysis. *Comput. Secur.* **2022**, *114*, 102578. <https://doi.org/10.1016/j.cose.2021.102578>.
10. Al-Kasasbeh, B. Model of the Information Security Protection Subsystem Operation and Method of Optimization of Its Composition. *Egypt. Inform. J.* **2022**, *23*, 1110–8665. <https://doi.org/10.1016/j.eij.2022.05.003>.
11. Yassein, M.B.; Shatnawi, M.Q.; Aljwarneh, S.; Al-Hatmi, R. Internet of Things: Survey and Open Issues of MQTT Protocol. In Proceedings of the 2017 International Conference on Engineering & MIS (ICEMIS), Monastir, Tunisia, 8–10 May 2017; pp. 1–6.
12. Alkhafajee, A.R.; Al-Muqarm, A.M.A.; Alwan, A.H.; Mohammed, Z.R. Security and Performance Analysis of MQTT Protocol with TLS in IoT Networks. In Proceedings of the 2021 4th International Iraqi Conference on Engineering Technology and Their Applications (ICETA), Najaf, Iraq, 21–22 September 2021; pp. 206–211.
13. Sushma, P.; Gopal, V.; Murthy, M. Secure Smart Home Using MQTT Protocol An Internet of Things Application. *Int. J. Adv. Res. Sci. Eng.* **2021**, *6*, 2319–8346. Available online: [http://www.ijarse.com/images/fullpdf/1504695802\\_BVCNCS17020\\_P\\_Sushma.pdf](http://www.ijarse.com/images/fullpdf/1504695802_BVCNCS17020_P_Sushma.pdf) (accessed on 27 July 2022).
14. V, S.; A, V.; Pattar, S. MQTT Based Secure Transport Layer Communication for Mutual Authentication in IoT Network. *Glob. Transit. Proc.* **2022**, *3*, 60–66. <https://doi.org/10.1016/j.gltip.2022.04.015>.
15. Varma, A.; UniKrishnan, S. Effect of Payload Security in MQTT Protocol over Transport and Application Layer. *IOP Conf. Ser. Mater. Sci. Eng.* **2021**, *1166*, 012019. <https://doi.org/10.1088/1757-899X/1166/1/012019>.
16. Peniak, P.; Franekova, M. Extended Model of Secure Communication for Embedded Systems with IoT and MQTT. In Proceedings of the 2018 International Conference on Applied Electronics (AE), Pilsen, Czech Republic, 11–12 September 2018; pp. 1–4.
17. Sadio, O.; Ngom, I.; Lishou, C. Lightweight Security Scheme for MQTT/MQTT-SN Protocol. In Proceedings of the 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS), Granada, Spain, 22–25 October 2019; pp. 119–123.
18. Kumar, S.; Hu, Y.; Andersen, M.P.; Popa, R.A.; Culler, D.E. JEDI: Many-to-Many End-to-End Encryption and Key Delegation for IoT. In Proceedings of the Proceedings of the 28th USENIX Conference on Security Symposium, Berkeley, CA, USA, 14–16 August 2019; USENIX Association: USA, 2019; pp. 1519–1536.
19. Galantucci, S.; Impedovo, D.; Pirlo, G. One Time User Key: A User-Based Secret Sharing XOR-Ed Model for Multiple User Cryptography in Distributed Systems. *IEEE Access* **2021**, *9*, 148521–148534. <https://doi.org/10.1109/ACCESS.2021.3124637>.
20. Banik, S.; Chakraborti, A.; Inoue, A.; Iwata, T.; Minematsu, K.; Nandi, M.; Peyrin, T.; Sasaki, Y.; Sim, S.M.; Todo, Y. GIFT-COFB. *IACR Cryptol. ePrint Arch.* **2020**, *2020*, 738.
21. Iwata, T.; Khairallah, M.; Minematsu, K.; Peyrin, T. New Results on Romulus. In *NIST Lightweight Cryptography Workshop, Gaithersburg, MD, USA, 19–21 October 2020*; NIST CSRC: Gaithersburg, MD, USA, 2020; pp. 1–17.
22. Iwata, T.; Khairallah, M.; Minematsu, K.; Peyrin, T. Duel of the Titans: The Romulus and Remus Families of Lightweight AEAD Algorithms. *IACR Trans. Symmetric Cryptol.* **2020**, *2020*, 43–120. <https://doi.org/10.13154/tosc.v2020.i1.43-120>.

23. Wu, H.; Huang, T. TinyJAMBU: A Family of Lightweight Authenticated Encryption Algorithms. In *Submission to the NIST Lightweight Cryptography Standardization Process*; NIST CSRC: Gaithersburg, MD, USA, 2019.
24. Chui, M.; Collins, M.; Patel, M. *The Internet of Things: Catching up to an Accelerating Opportunity*; McKinsey & Company: New York, NY, USA, 2021.
25. Abuladel, A.; Bamasag, O. Data and Location Privacy Issues in IoT Applications. In Proceedings of the 2020 3rd International Conference on Computer Applications & Information Security (ICCAIS), Riyadh, Saudi Arabia, 19–21 March 2020; pp. 1–6.
26. MQTT Version 5.0. Available online: <https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html> (accessed on 6 June 2022).
27. HariPriya, A.P.; Kulothungan, K. Secure-MQTT: An Efficient Fuzzy Logic-Based Approach to Detect DoS Attack in MQTT Protocol for Internet of Things. *EURASIP J. Wirel. Commun. Netw.* **2019**, *2019*, 1–15.
28. Mishra, B.; Kertesz, A. The Use of MQTT in M2M and IoT Systems: A Survey. *IEEE Access* **2020**, *8*, 201071–201086. <https://doi.org/10.1109/ACCESS.2020.3035849>.
29. Light, R. Mosquitto: Server and Client Implementation of the MQTT Protocol. *J. Open Source Softw.* **2017**, *2*, <https://doi.org/10.21105/joss.00265>.
30. Hoang, T.; Fu, Y.; Mechitov, K.; Sánchez, F.G.; Kim, J.R.; Zhang, D.; Spencer, B.F. Autonomous End-to-End Wireless Monitoring System for Railroad Bridges. *Adv. Bridge Eng.* **2020**, *1*, 17. <https://doi.org/10.1186/s43251-020-00014-7>.
31. Gil, S.; Zapata-Madrigal, G.D.; García-Sierra, R.; Cruz Salazar, L.A. Converging IoT Protocols for the Data Integration of Automation Systems in the Electrical Industry. *J. Electr. Syst. Inf. Technol.* **2022**, *9*, 1–21. <https://doi.org/10.1186/s43067-022-00043-4>.
32. Junior, N.; Silva, A.; Guelfi, A.; Kofuji, S. Performance Evaluation of Publish-Subscribe Systems in IoT Using Energy-Efficient and Context-Aware Secure Messages. *J. Cloud Comput.* **2022**, *11*, 1–17. <https://doi.org/10.1186/s13677-022-00278-6>.
33. Bedoui, M.; Mestiri, H.; Bouallegue, B.; Hamdi, B.; Machhout, M. An Improvement of Both Security and Reliability for AES Implementations. *J. King Saud Univ. -Comput. Inf. Sci.* **2022**, 1319–1578. <https://doi.org/10.1016/j.jksuci.2021.12.012>.
34. Daemen, J.; Rijmen, V. *The Design of Rijndael: AES—The Advanced Encryption Standard*; Springer: Berlin/Heidelberg, Germany, 2001.
35. Daemen, J.; Rijmen, V. The Advanced Encryption Standard Process. In *The Design of Rijndael: AES—The Advanced Encryption Standard*; Springer: Berlin/Heidelberg, Germany, 2002; pp. 1–8, ISBN 978-3-662-04722-4.
36. William, S. *Cryptography and Network Security: Principles and Practice*, 6th ed.; Person Education Inc.: Loudon, UK, 2014.
37. Ajish, S.; AnilKumar, K.S. Secure Mobile Internet Voting System Using Biometric Authentication and Wavelet Based AES. *J. Inf. Secur. Appl.* **2021**, *61*, 102908. <https://doi.org/10.1016/j.jisa.2021.102908>.
38. Chakraborti, A.; Iwata, T.; Minematsu, K.; Nandi, M. Blockcipher-Based Authenticated Encryption: How Small Can We Go? *J. Cryptol.* **2020**, *33*, 703–741. <https://doi.org/10.1007/s00145-019-09325-z>.
39. Sierra, J.M.; Hernández, J.C.; Alcaide, A.; Torres, J. Validating the Use of BAN LOGIC. In Proceedings of International Conference Computational Science and Its Applications – ICCSA 2004, Assisi, Italy, 14–17 May 2004; Springer Berlin Heidelberg: Berlin/Heidelberg, Germany, 2004; pp. 851–858.
40. Boyd, C.; Mao, W. On a Limitation of BAN Logic. In *Advances in Cryptology—EUROCRYPT 93*; Springer Berlin Heidelberg: Berlin/Heidelberg, Germany, 1993; pp. 240–247.
41. Menezes, A.J.; van Oorschot, P.C.; Vanstone, S.A. *Handbook of Applied Cryptography*; CRC press: Boca Raton, FL, USA, 2018.
42. Shamir, A. How to Share a Secret. *Commun. ACM* **1979**, *22*, 612–613. <https://doi.org/10.1145/359168.359176>.
43. Sun, Y.; Wen, Q.; Sun, H.; Li, W.; Jin, Z.; Zhang, H. An Authenticated Group Key Transfer Protocol Based on Secret Sharing. *Procedia Eng.* **2012**, *29*, 403–408. <https://doi.org/10.1016/j.proeng.2011.12.731>.
44. Kaneko, N.; Iwamura, K. Improvement of Communication Traffic and Security of Proactive Secret Sharing Schemes and Combination Proactive Secret Sharing Scheme with an Asymmetric Secret Sharing Scheme. In Proceedings of the 2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA), Taipei, Taiwan, 27–29 March 2017; pp. 13–18.