LEA 알고리즘을 이용한 MQTT 프로토콜 보안

락스모노 아구스 마하르디카 아리 [1], 이크발 무함마드 [1], 프라타마 데리 [2], 김호원 [3]
[1] 부산대학교 정보융합공학과 석사과정
[2] 부산대학교 정보융합공학과 박사과정
[3] 부산대학교 정보융합공학과 교수

agusmahardika@pusan.ac.kr, iqbal@islab.re.kr, derryprata@gmail.com, howonkim@pusan.ac.kr

Securing the MQTT Protocol using the LEA Algorithm

Laksmono Agus Mahardika Ari, Iqbal Muhammad, Pratama Derry, Howon Kim
Dept. of Computer Information Convergence Engineering, Pusan National University

Abstract

IoT is becoming more and more popular, along with the massive availability of cheap and easy-to-use IoT devices. One protocol that is often used in IoT devices is the Message Queuing Telemetry Transport (MQTT) protocol. By default, the MQTT protocol does not activate encrypted data security features. This MQTT default feature makes the transmitted and received message data vulnerable to attacks, such as eavesdropping. Therefore, this paper will design and implement encrypted data security using the lightweight cryptography algorithm. The focus of this paper will be on securing MQTT message data at the application layer. We propose a method for encrypting specific MQTT message fields while maintaining compatibility with the protocol's functionalities. The paper then analyzes the timing performance of the MQTT-LEA implementation on the Raspberry Pi 3+. Our findings demonstrate the feasibility of using LEA at the application layer to secure MQTT message communication on resource-constrained devices.

1. Introduction

In the age of the internet, interconnected Internet of Things (IoT) devices are networked together for the purpose of collecting and exchanging information. These devices, especially IoT based smart homes device, are highly vulnerable to cyber and physical security attacks [1]. This is because by default the protocols used on IoT devices do not enable encryption features, one of which is the Message Queuing Telemetry Transport (MQTT) protocol. MQTT is a popular choice for communication in IoT applications. MQTT has been widely implemented in various industrial sectors, such as manufacturing, automotive, sports, energy, military, telecommunications and healthcare [2]. Its lightweight design and publish-subscribe architecture make it efficient for resource-constrained devices to exchange data. However, the lack of the standard MQTT protocol does not activate encrypted data security features [3]. If we use the standard MQTT protocol, the data transmitted between devices will be vulnerable to eavesdropping, tampering, and unauthorized access. This makes message content to potential security breaches during transmission, risking sensitive data's confidentiality and integrity. Attackers get attracted to this and want to obtain user data.

To address this security gap, this paper proposes an implementation for securing MQTT messages using the Lightweight Encryption Algorithm (LEA). The LEA is a lightweight block cipher specifically designed for resource-constrained environments, making it suitable for implementation on devices with limited processing power. Our approach focuses on encrypting specific MQTT message fields while ensuring compatibility with the core functionalities of the protocol. The initial communication process will not be modified for concern that it will disturb the handshaking process between the publisher/subscriber and the broker server. Meanwhile, the decryption of the message on the subscriber side will be done after the subscriber connects to the broker with a certain topic and waits for the publisher's publication message. For the analysis, we will present timing performance of our implementation versus conventional MQTT message delivery times. We also evaluate the impact of encryption on message transmitting.

2. Theoretical Background

2.1 MQTT

Message Queuing Telemetry Transport (MQTT) is an internet of things (IoT) protocol standard issued by OASIS. MQTT is a client server publish/subscribe messaging transport protocol. The server in MQTT is called a broker, and the MQTT client has two rules, as a publisher or as a subscriber. It is light weight, open, simple, and designed to be easy to implement. These characteristics make it ideal for use in many situations, including constrained environments such as for communication in Machine to Machine (M2M) and Internet of Things (IoT) contexts where a small code footprint is required and/or network bandwidth is at a premium [4]. Three qualities of service (QoS) for message delivery:

a. Q0: "At most once", where messages are delivered according to the best efforts of the operating environment. Message loss can occur. This level could be used, for example, with ambient sensor data where it does not matter if an individual reading is lost as the next one will be published soon after.
b. Q1: "At least once", where messages are assured to arrive but duplicates can occur.
c. Q2: "Exactly once", where messages are assured to arrive exactly once. This level could be used, for example, with billing systems where duplicate or lost messages could lead to incorrect charges being applied.

2.2 LEA Algorithm

Block cipher LEA is an algorithm that encrypts 128-bit data blocks. It can use 128-bit, 192-bit, and 256-bit secret keys, and its use can be classified according to the required safety standards [5]. The round function of LEA consists only of ARX (Addition, Rotation, XOR) operations in 32-bit units, so these operations. It operates at high speed on a general-purpose 32-bit software platform that supports. Also, the round function of the internal ARX (Addition, Rotation, XOR) operation arrangement ensures sufficient safety and at the same time a lightweight implementation is possible by excluding the use of S-box. The structure of LEA has the following features:

a. LEA consists of only ARX (modular Addition, bitwise Rotation, and bitwise XOR) operations for 32-bit words.
b. The last round function of LEA is the same as other round functions.
c. The key schedule of LEA has a simple structure without any interleaving between 32-bit key words.

3. Proposed Design

To implement the LEA algorithm on the MQTT protocol, we must recognize the mechanism and syntax of the MQTT protocol. The focus of the LEA encryption implementation is to secure message packets at the application layer. Therefore, the workflow of implementing the LEA algorithm on the publisher side must consider the message parameters to be encrypted. If the implementation is incorrect, it will interfere with the process and workflow of MQTT. The proposed algorithm flow on the publisher side can be seen in the following step:

Algorithm 1. MQTT-LEA Publisher Algorithm Flow

```
Input : MQTT Syntax (S), Message (M)
Output : Encrypted Message (EncM)
1. MQTT_Start(S,M);
   Begin
2. state ← M
3. EncM ← LEA_Encrypt_Key (M)
4. MQTT_Publish(EncM, topic, QoS)
   End
5. MQTT_End();
```

From the flow description above, the publisher inputs are syntax and message. The syntax contains programs from the publisher that manage parameters such as the broker host address, topic, and QoS level of the message. Meanwhile, the message in the publisher input is a packet of information delivered by the user/IoT sensor device to other entities that act as subscribers. The message will be encrypted using a 128-bit LEA module with a secret key of 128 bits in length. The secret key has predetermined when the handshaking phase with the broker has not been executed. In this research, the broker server is installed on an Oracle cloud server that has a public IP address. For the QoS level, if the publisher does not specify which options are used in the syntax, the MQTT publisher application will automatically use the default parameter, which is type Q0.

Algorithm diagram 2 shows the stages that occur on the client side of the subscriber. After running the subscriber application with the parameters set by the syntax command, the subscriber will continue to stand by, waiting for messages sent from the publisher according to the predetermined topic. If there is an incoming message with the corresponding topic, the subscriber will decrypt the message using the 128-bit LEA module and the symmetric key associated with the encryption process. The results of the decrypted message will then be displayed in the subscriber application interface.

Algorithm 2. MQTT-LEA Subscriber Algorithm Flow

```
**Input** : MQTT Syntax (S), Encrypted Message
            (EncM)
**Output** : Message (M)
1. MQTT_Start(S, EncM);
   **Begin**
2. MQTT_Subscriber(EncM, topic, QoS)
3. state ← LEA_Decrypt_Key (EncM)
4. M ← state
   **End**
5. MQTT_End();
```

4. Result and Analysis

The proposed design simulation uses a broker server installed on a public network. The purpose of this simulation design is so that the research can be tested in accordance with real infrastructure conditions. Figure 1 is the simulation architecture design of the research on the implementation of the LEA algorithm in the MQTT protocol. The broker server used is the Mosquitto Broker application [6] which is installed in the Canonical Ubuntu 22.04 Linux operating system on the Oracle cloud. The broker server host address also uses the public IP address provided by the Oracle cloud with port 1883. The IoT device used is a Raspberry Pi 3b+ device operated using the Linux Raspbian 6.1.21-v7 operating system.

Using the architectural design as shown in Figure 1, three IoT devices are used to simulate the results of applying the LEA algorithm module to the MQTT protocol. Two devices installed the MQTT Client application using the Paho MQTT Client source code [7] equipped with the LEA module. Meanwhile, one device was installed using the standard Paho Client application. The device without the LEA module is assumed to be an eavesdropper trying to obtain sensitive data from the MQTT communication.

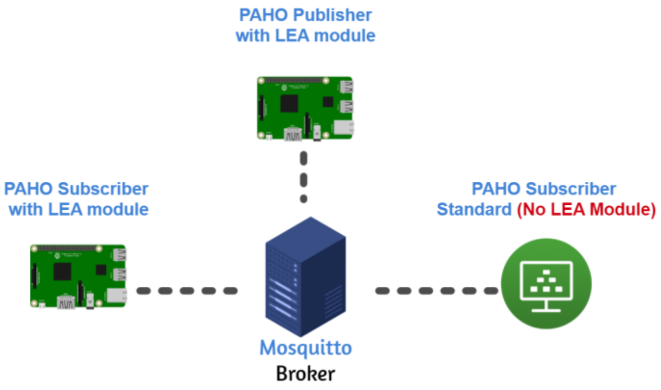
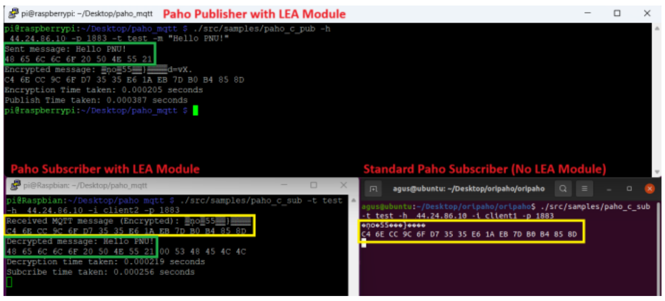


Figure 1. Simulation architecture of LEA implementation in MQTT

Figure 2 showcases an application demonstration where the encryption and decryption process involving two LEA-equipped IoT devices successfully secures the MQTT message exchange. The message content displayed on a standard MQTT Paho client device, acting as an eavesdropping simulation, also show that unauthorized parties will not be able to know the sensitive data encrypted. This application demonstrates the success of the proposed application-layer MQTT security approach using the LEA algorithm. In addition, testing for message delivery further confirms that communications encrypted with LEA do not need to be re-encoded using modules such as base64. This is so that effective message transmission via the MQTT protocol can be achieved without the need for additional encoding stages, as the LEA encryption process itself produces an output that is string-formatted.


pi@raspberrypi: ~/Desktop/paho_mqtt
Paho Publisher with LEA Module
Paho Subscriber with LEA Module
pi@Raspbian: ~/Desktop/paho_mqtt
Standard Paho Subscriber (No LEA Module)
agus@ubuntu: ~/Desktop/oripaho/oripaho

```
pi@raspberrypi:~/Desktop/paho_mqtt $ ./src/samples/paho_c_pub -h
 44.24.86.10 -p 1883 -t test -m "Hello PNU!"
Sent message: Hello PNU!
48 65 6C 6C 6F 20 50 4E 55 21
Encrypted message: ▒ŋo▒55▒▒}▒▒▒▒d=vX.
C4 6E CC 9C 6F D7 35 35 E6 1A EB 7D B0 B4 85 8D
Encryption Time taken: 0.000205 seconds
Publish Time taken: 0.000387 seconds
pi@raspberrypi:~/Desktop/paho_mqtt $
```

```
pi@Raspbian:~/Desktop/paho mqtt $ ./src/samples/paho_c_sub -t test
-h  44.24.86.10 -i client2 -p 1883
Received MQTT message (Encrypted): ▒ŋo▒55▒▒}▒▒▒▒
C4 6E CC 9C 6F D7 35 35 E6 1A EB 7D B0 B4 85 8D
Decrypted message: Hello PNU!
48 65 6C 6C 6F 20 50 4E 55 21 00 53 48 45 4C 4C
Decryption time taken: 0.000219 seconds
Subcribe time taken: 0.000256 seconds
```

```
agus@ubuntu:~/Desktop/oripaho/oripaho$ ./src/samples/paho_c_sub
-t test -h  44.24.86.10 -i client1 -p 1883
◆ŋo◆55◆◆◆}◆◆◆◆
C4 6E CC 9C 6F D7 35 35 E6 1A EB 7D B0 B4 85 8D
```

Figure 2. Simulation of 128-bit LEA implementation in MQTT

This research compares the usage of processing time between publisher clients for standard MQTT and MQTT-LEA. This processing time is recorded from the publisher successfully connecting with the broker until the publisher has finished sending the message. The time taken by the LEA module to encrypt and decrypt messages was also recorded. Two message lengths were used for testing: 10 characters (10 bytes) with the message "Hello PNU!" and 16 characters (16 bytes) with the message "Hello ISLAB-PNU!". Each message length was sent 20 times, and the average processing time was calculated. For additional information, the internet speed used in this simulation is 5.19 MBps for upload speed and 38.05 MBps for download speed. The result of research time can be seen in Table 1 and Table 2.

Table 1. Research time data for publish and subscribe process

Module	Publish Time (μs)		Subscribe Time (μs)	
	10 bytes	16 bytes	10 bytes	16 bytes
MQTT Standard	134.2	137.6	54.8	44.8
MQTT-LEA	269.45	283.95	112.7	167.9
Ratio	2.008	2.064	2.057	3.748

Table 2. Research time data for encryption and decryption process

Module	Encryption Time (μs)		Decryption Time (μs)	
	10 bytes	16 bytes	10 bytes	16 bytes
MQTT-LEA	120.5	125.75	98.3	152.4

Table 1 reveals that the publisher time of MQTT with 128-bit LEA has twice as much time compared to conventional MQTT. This is due to the fact that the MQTT-LEA module requires additional time to calculate the message encryption process. As for the subscriber processing time, the simulation using the 16-byte character length takes the most additional time-three and a half times longer than the standard MQTT subscriber time. Meanwhile, table 2 shows that the average processing time increases according to the message length transmitted. It takes longer to send a 16-bit message than a 10-bit message.

5. Conclusion

This research demonstrates that applying LEA at the application layer effectively secures MQTT messages on resource-constrained devices. The public network simulation with multiple IoT devices confirms ability of LEA to encrypt messages, rendering them unreadable for eavesdroppers.

6. Future Work

Research can be continued by improving the performance of the LEA-based MQTT approach in larger scale IoT development with more devices and message variations. Comparing results with other studies that use different cryptographic techniques to provide MQTT security might also help develop research. Other potential areas of research include the dissemination of key management or the application of dynamic key management to IoT devices that are connected.

7. Acknowledgement

This work is financially supported by Korea Ministry of Land, Infrastructure and Transport (MOLIT) as 「Innovative Talent Education Program for Smart City」.

Reference

[1] Bako Ali and Ali Ismail Awad, "Cyber and Physical Security Vulnerability Assessment for IoT-Based Smart Homes", in Sensors 18, no. 3: 817, 2018.

[2] B. Mishra and A. Kertesz, "The Use of MQTT in M2M and IoT Systems: A Survey," in IEEE Access vol. 8, 2020.

[3] Ahmed J. Hintaw, Selvakumar Manickam, Mohammed Faiz Aboalmaaly, and Shankar Karuppayah, "MQTT vulnerabilities, attack vectors and solutions in the internet of things (IoT)", IETE Journal of Research 69 No. 6, pp: 3368-3397, 2023.

[4] OASIS. MQTT Version 5.0 Committee Specification 02. 15 May 2018.

[5] Deukjo Hong, Jung-Keun Lee, Dong-Chan Kim, Daesung Kwon, Kwon Ho Ryu, and Dong-Geon Lee. “LEA: A 128-Bit Block Cipher for Fast Encryption on Common Processors”. 14th International Workshop on Information Security Applications - Volume 8267 (WISA 2013). Pages 3–27.

[6] Eclipse Mosquitto™, An open source MQTT Broker. Available online: https://mosquitto.org/ (accessd on 8 April 2024)

[7] Eclipse Paho C Client Library for the MQTT Protocol. Available online: https://github.com/eclipse/paho.mqtt.c (accessd on 8 April 2024)