Securing Data Communication Through MQTT Protocol with AES-256 Encryption Algorithm CBC Mode on ESP32-Based Smart Homes

Conference Paper · October 2021

DOI: 10.1109/COSITE52651.2021.9649577

2 authors:

Fauzan Budi Setiawan

4 PUBLICATIONS **17** CITATIONS

SEE PROFILE

Magfirawaty ..

Politeknik Siber dan Sandi Negara

18 PUBLICATIONS **51** CITATIONS

SEE PROFILE

Securing Data Communication Through MQTT Protocol with AES-256 Encryption Algorithm CBC Mode on ESP32-Based Smart Homes

1st Fauzan Budi Setiawan
Dept. Cryptographic Hardware Engineering
Politeknik Siber dan Sandi Negara
Bogor, Indonesia
fauzan.budi@student.poltekssn.ac.id

2nd Magfirawaty*
Dept. Cryptographic Hardware Engineering
Politeknik Siber dan Sandi Negara
Bogor, Indonesia
magfirawaty@poltekssn.ac.id

***Abstract*—The Internet of Things (IoT) is a technology that allows connection between devices using the internet to collect and exchange data with each other. Privacy and security have become the most pressing issues in the IoT network, especially in the smart home. Nevertheless, there are still many smart home devices that have not implemented security and privacy policies. This study proposes a remote sensor control system built on ESP32 to implement a smart home through the Message Queuing Telemetry Transport(MQTT) protocol by applying the Advanced Encryption Standard (AES) algorithm with a 256-bit key. It addresses security issues in the smart home by encrypting messages sent from users to sensors. Besides ESP32, the system implementation also uses Raspberry Pi and smartphone with Android applications. The network was analyzed using Wireshark, and it showed that the message sent was encrypted. This implementation could prevent brute force attacks, with the result that it could guarantee the confidentiality of a message. Meanwhile, from several experiments conducted in this study, the difference in the average time of sending encrypted and unencrypted messages was not too significant, i.e., 20 ms.**

***Index Terms*—smart home, MQTT protocol, ESP32, AES**

I. Introduction

The Internet of Things (IoT) is a technological revolution that aims to interconnect and exchange data with other devices and systems through the internet. This revolution has had a direct impact on the development of digital transformation. In addition, IoT is a concept where an object can transfer data over a network without human-to-human or human-to-computer interaction [1]. The smart home is one of the implementations of IoT development. It combines various technologies and provides solutions for a better life. Furthermore, a smart home provides security, efficiency, and comfort solutions for the residents. The smart home system usually consists of monitoring devices, control devices, and several automation devices that could be accessed using a computer [2]. This system allows communication between sensors, devices, and other equipment with the user [3]. Based on the architecture, the smart home provides three main components [2]:

1) Communication networks: wired, wireless
2) Intelligent control system: a gateway to manage system features, and
3) Home automation: home services that are connected to remote systems.

IoT utilizes lightweight protocols when transmitting data, one of which is Message Queuing Telemetry Transport (MQTT) [4]. OASIS introduced MQTT in 2013, then became a client-server standard version 3.1.1 in 2014. MQTT applies a publish/subscribe messaging protocol to deliver messages over low bandwidth connections in real-time with small headers [4]. Information sent through the protocol is known as application message; publisher and subscriber are terms for MQTT clients who send and receive messages, while brokers are parties that allow multiple clients to connect [5].

Smart home systems are closely related to a person's daily life. Therefore, the implementation of IoT in the smart home increases many cyber incidents such as trespasses, monitoring and leakage of personal information, Dos/DDoS, and forgery [6]. In the process of sending messages, an intruder can capture signals or can damage information on the wireless network, with the result that, if the intruder gains access to the system, it will obtain confidential information about home users and can manipulate data [2], [3], [6], [7]. Advanced Encryption Standard (AES) is a symmetric encryption algorithm that is used to secure messages. This algorithm provides high security and low power consumption [8]. The AES algorithm can be applied to the MQTT protocol to improve security, efficiency and prevent brute force attacks [8]- [11].

This study implements a secure smart home on an android application to a sensor connected to ESP32. Before sending, the data is encrypted using the AES algorithm with a 256-bit key length. The resulting ciphertext is sent over the internet using the MQTT protocol with a broker on the Raspberry Pi. MQTT has a low-security level using topics such as a data filter and a password for authentication when connected to an MQTT broker. Furthermore, to determine the application's performance, we used Wireshark to analyze network performance. Wireshark could capture messages in the communication network between the smartphone and the ESP32. This study is divided into several sections. Section II presents the methods used, the design and implementation

978-1-6654-2509-4/21/$31.00 ©2021 IEEE

results are shown in section III, Section IV presents the analysis of the implementation results, while the conclusions are obtained in Section V.

II. Research Method

The design of the smart home application system consists of hardware and software designs. This study utilizes ESP32 as the primary device, Raspberry Pi 3 as MQTT broker with Eclipse Mosquitto, Android application, and several sensors. Thonny IDE and Android Studio are utilized to develop Python - Micropython code and Android applications. Data exchange between android application and ESP32 using MQTT protocol. Fig. 1 shows how the MQTT protocol works. The testing mechanism is executed by analyzing the data sent between the android application and ESP32. The analysis was implemented by comparing the results of the Wireshark capture of sending plaintext and ciphertext using the AES algorithm on the MQTT protocol. The scheme of system design is presented in Fig. 2.

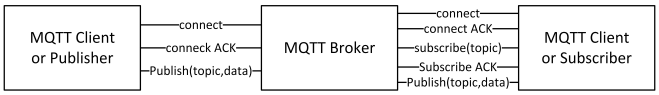

Fig. 1. MQTT Protocol [4]

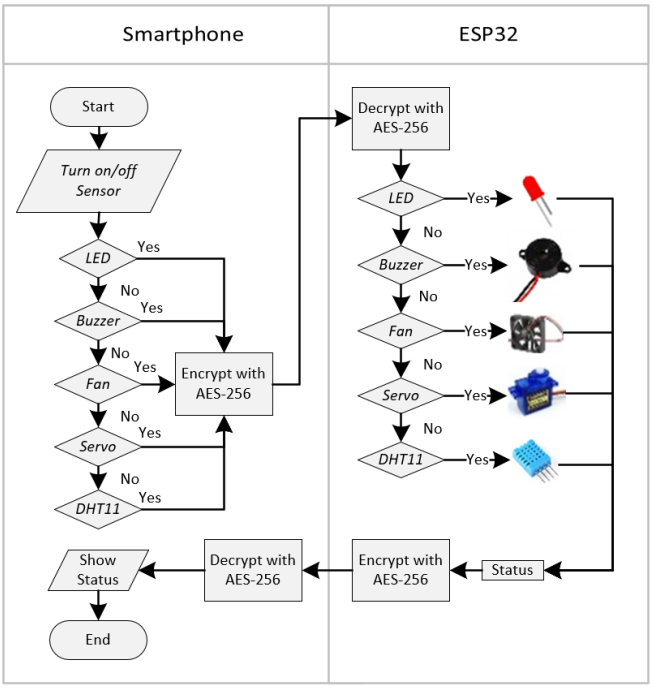

Fig. 2. Flowchart System

III. Design and Implementation

An overview of the connectivity of the hardware used is shown in Fig. 3. Meanwhile, the system design of this research is shown in Fig. 3. The architecture system involves four primary devices, they are :

- A smartphone is used to implement an android application to control and monitor sensor data.
- ESP32 functions as a system that executes and monitors the sensor work process.
- Raspberry Pi: is the gateway between the smartphone and the ESP32.
- Sensors: providing control information of a system.

This study uses several sensors such as a buzzer, dc fan, servo, temperature sensor, and LED. All of these sensors are connected to the ESP 32. As provided in Fig. 2, the communication between the android application and ESP32 uses the MQTT protocol. While the MQTT Broker used is Eclipse Mosquitto which is installed on the Raspberry Pi. Afterward, the implementation of the system on the device could be seen in Fig. 4.

The encryption algorithm utilized in this research is AES-256 with CBC (Cipher Block Chaining) mode. The encryption and decryption process is executed on Android applications and ESP32 with a 256-bit key length. Furthermore, encrypted messages are sent using the MQTT protocol from the smartphone to the ESP32 or otherwise.

Fig. 5 presents an Android application designed using the publish and subscribe mechanism in MQTT. This application is used to control and monitor sensors, which are connected to the ESP32. The first process of the system begins when the Android application publishes an encrypted instructions message through the MQTT broker on the Raspberry Pi to ESP32. The Android application and ESP32 send requests to the MQTT broker to connect using the same topic. After receiving the message, ESP32 decrypts the message and sends the encrypted response data to the Android application. The process of messages encryption and decryption using AES-256 algorithm

IV. Result and Analysis

As explained in the previous section, the designed application consists of three main components, i.e., Android, Raspberry Pi, and ESP32, that are connected to sensors. Android applications and ESP32 are connected to the internet using Wi-Fi and communicate using the MQTT protocol. The message from the android application contains several instructions to the ESP32, which are encrypted first and then send to the ESP32 device through the Raspberry Pi. Fig. 6 and Fig. 7 show some plaintext and ciphertext on ESP32 and android application.

The implementation of the encryption algorithm has been verified using test vectors. It was implemented by comparing the output generated by the AES-256 algorithm CBC mode on the Android application and ESP32 with test vector in NIST SP 800-38A document [12]. The test was performed using the same data (keywords, initial value (IV), and plain text). Table I shows the results of the test vectors that have been performed. It presents the results of the encryption process on the Android

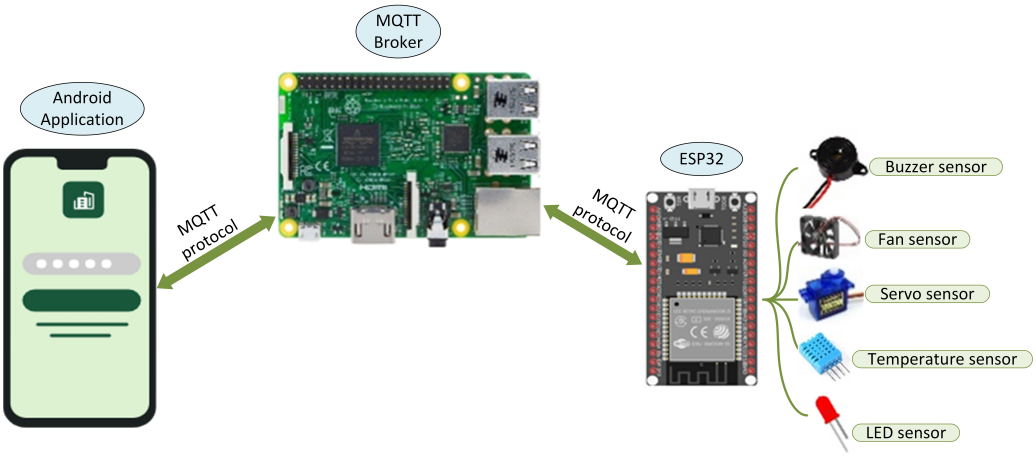

Fig. 3. System Design

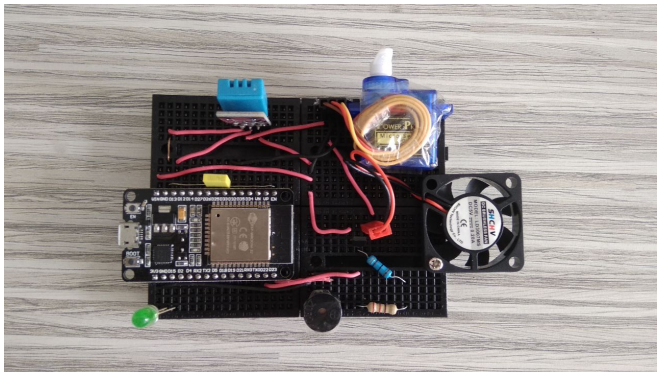

Fig. 4. Hardware implementation

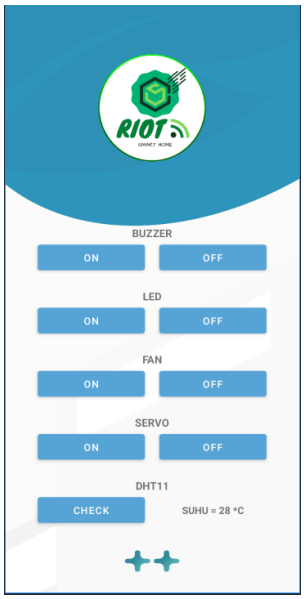

Fig. 5. Smart home application on Android

```
>>> %Run -c $EDITOR_CONTENT

  Plain Text   :  TURN LED = ON
  Cipher Text  :  911bf3bf311028941aa3046459ac540e
  Plain Text   :  TURN LED = OFF
  Cipher Text  :  b2e8302292640f7a7719a75eef020ad1
  Plain Text   :  TURN BUZZER = ON
  Cipher Text  :  e9930e98ffc95c948ffcf75157e6d3275db26975e5912ffa3bf343ee31b7ef73
  Plain Text   :  TURN BUZZER = OFF
  Cipher Text  :  d653814ac5643f748625611bd9b32333684a8f53734468297db4ed5d021bbf7b
```

Fig. 6. Plaintext and ciphertext on ESP32

```
> Task :javacodelib:Algoritma.main()
Plain Text      = TURN LED = ON
Cipher Text     = 911bf3bf311028941aa3046459ac540e
Plain Text      = TURN LED = OFF
Cipher Text     = b2e8302292640f7a7719a75eef020ad1
Plain Text      = TURN BUZZER = ON
Cipher Text     = e9930e98ffc95c948ffcf75157e6d3275db26975e5912ffa3bf343ee31b7ef73
Plain Text      = TURN BUZZER = OFF
Cipher Text     = d653814ac5643f748625611bd9b32333684a8f53734468297db4ed5d021bbf7b
```

Fig. 7. Plaintext and ciphertext on Android

application and ESP32 that generate the same output and are compliant with NIST SP 800-38A.

Packet capture analysis using the Wireshark Sniffer application was performed on the system that had been implemented. The capture results show the message packets sent through the MQTT protocol between the Android application and ESP32. Fig. 8 represented a plain text message packet sent from the android application and addressed to ESP32. In this study, the message to be sent must be encrypted first. If the message sent by the smartphone to ESP32 is " TURN BUZZER ON," it is encrypted first using the AES 256 algorithm. Furthermore, the message shown in the blue box in Fig. 9 is the result of message encryption.

The analysis results show that confidentiality and security of information between the sender and the recipient have been maintained. Intruders perform several attacks on the ciphertext. One of the popular attacks is brute force, which is a trial and error method to get passwords. AES is computationally secure against brute force attacks because it is practically impossible

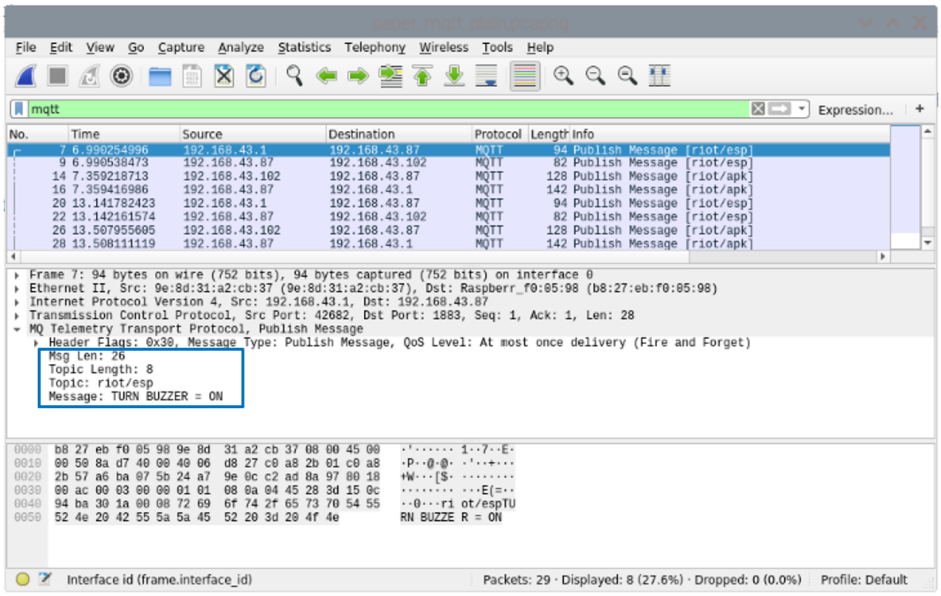

Fig. 8. MQTT packet capture without AES encryption

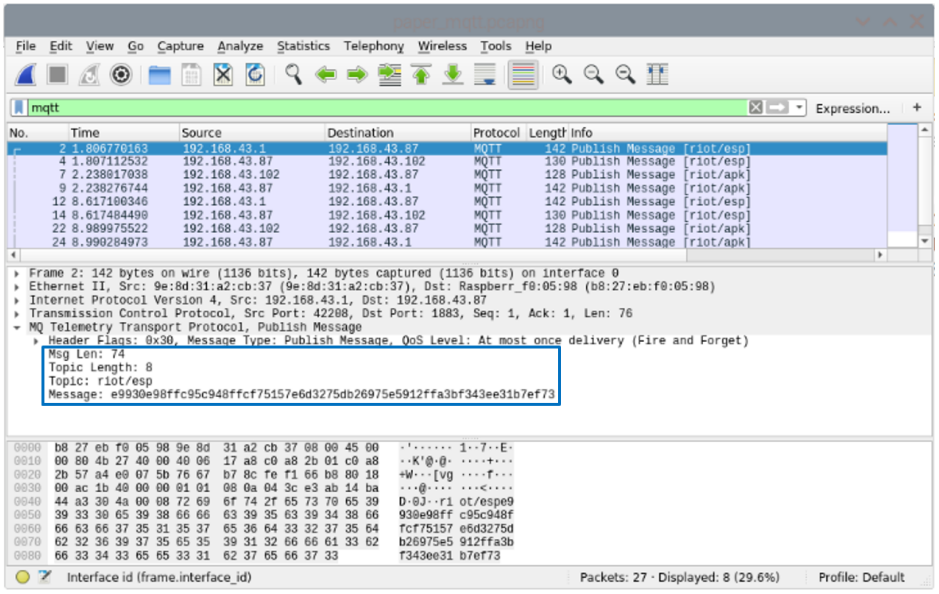

Fig. 9. MQTT packet capture with AES encryption

to obtain a 256 bit key in short time.

Performance testing is performed by calculating the total time required for sending messages between devices. The time calculation begins when the user presses the sensor control button on the Android until it gets a status response from the sensor. We calculated the time to send messages over the MQTT protocol with and without the encryption process. The comparison of the time required is shown in Fig. 10. We have done a total of 50 messaging attempts, most of the messages sent without using the encryption process take less time than encrypted messages. Several factors influence it, i.e., the internet connection and changes in the communication stages of each device. However, the time difference between sending encrypted and unencrypted messages is not very significant. In this experiment, the difference in the average delivery time of encrypted and unencrypted messages is 20 ms

V. Conclusion

The smart home represents a potential area of research with significant development due to the growing industry demand. Secure data transmission between devices in a smart home is a challenge. There are several algorithms that provide secure data transmission from it, one of which is AES. This study

TABLE I
AES-256 CBC ALGORITHM VECTOR TEST

Hardware	Output
Key: ”603deb1015ca71be2b73aef0857d77811f352c073b6108d72d9810a30 914dff4” IV: ”000102030405060708090a0b0c0d0e0f” Plaintext: ”6bc1bee22e409f96e93d7e117393172aae2d8a571e03ac9c9eb76f ac45af8e5130c81c46a35ce411e5fbc1191a0a52eff69f2445df4f9b1 7ad2b417be66c3710”	
NIST Special Publication 800-38A	f58c4c04d6e5f1ba779eabfb5f7bfbd 69cfc4e967edb808d679f777bc6702 c7d39f23369a9d9bacfa530e263042 31461b2eb05e2c39be9fcda6c19078 c6a9d1b
Android	f58c4c04d6e5f1ba779eabfb5f7bfbd 69cfc4e967edb808d679f777bc6702 c7d39f23369a9d9bacfa530e263042 31461b2eb05e2c39be9fcda6c19078 c6a9d1b
ESP32	f58c4c04d6e5f1ba779eabfb5f7bfbd 69cfc4e967edb808d679f777bc6702 c7d39f23369a9d9bacfa530e263042 31461b2eb05e2c39be9fcda6c19078 c6a9d1b

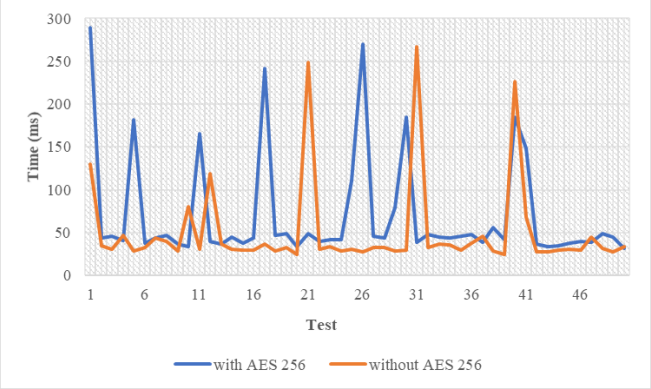

Fig. 10. Time Requirement of The System

focuses on implementing the MQTT protocol and the AES-256 algorithm on ESP32, Raspberry Pi, and Android application. This study indicates that messages sent by Android application to ESP32 and otherwise are more resistant to brute force attacks. Therefore, the system applied in this study has a more practical level of security than the existing smart home system.

REFERENCES

[1] Y. Guaman, G. Ninahualpa, G. Salazar, and T. Guarda, “Comparative Performance Analysis between MQTT and CoAP Protocols for IoT with Raspberry PI 3 in IEEE 802.11 Environments,” Iber. Conf. Inf. Syst. Technol. Cist., vol. 2020-June, no. June, pp. 24–27, 2020, doi: 10.23919/CISTI49556.2020.9140905.

[2] M. R. Alam, M. B. I. Reaz, and M. A. M. Ali, “A review of smart homes - Past, present, and future,” IEEE Trans. Syst. Man Cybern. Part C Appl. Rev., vol. 42, no. 6, pp. 1190–1203, 2012, doi: 10.1109/TSMCC.2012.2189204.

[3] S. Ul Rehman and S. Manickam, “A Study of Smart Home Environment and its Security Threats,” Int. J. Reliab. Qual. Saf. Eng., vol. 23, no. 3, 2016, doi: 10.1142/S0218539316400052.

[4] M. Kashyap, V. Sharma, and N. Gupta, “Taking MQTT and NodeMcu to IOT: Communication in Internet of Things,” Procedia Comput. Sci., vol. 132, no. Iccids, pp. 1611–1618, 2018, doi: 10.1016/j.procs.2018.05.126.

[5] B. Mishra and A. Kertesz, “The Use of MQTT in M2M and IoT Systems: A Survey,” IEEE Access, vol. 8, pp. 201071–201086, 2020, doi: 10.1109/access.2020.3035849.

[6] S. Yoon, H. Park, and H. S. Yoo, “Security issues on smarthome in IoT environment,” Lect. Notes Electr. Eng., vol. 330, pp. 691–696, 2015.

[7] U. Saxena, J. S. Sodhi, and Y. Singh, “Analysis of security attacks in a smart home networks,” Proc. 7th Int. Conf. Conflu. 2017 Cloud Comput. Data Sci. Eng., pp. 431–436, 2017, doi: 10.1109/CONFLUENCE.2017.7943189.

[8] P. Dabas, “Secure Data Transmission using AES in IoT,” Int. J. Appl. or Innov. Eng. Manag., vol. 6, no. 6, pp. 283–289, 2017.

[9] B. K. S. Rajaram and N. Krishna Prakash, “Secure mqtt using aes for smart homes in iot network,” Int. J. Innov. Technol. Explor. Eng., vol. 8, no. 5s, pp. 483–485, 2019.

[10] Y. Javed, A. S. Khan, A. Qahar, and J. Abdullah, “Preventing DoS attacks in IoT using AES,” J. Telecommun. Electron. Comput. Eng., vol. 9, no. 3–11, pp. 55–60, 2017.

[11] J. Ahamed, M. Zahid, M. Omar, and K. Ahmad, “AES and MQTT based security system in the internet of things,” J. Discret. Math. Sci. Cryptogr., vol. 22, no. 8, pp. 1589–1598, 2019, doi: 10.1080/09720529.2019.1696553.

[12] M. Dworkin, “Recommendation for Block Cipher Modes of Operation,” Natl. Inst. Stand. Technol. Spec. Publ. 800-38A 2001 ED, vol. X, no. December, pp. 1–23, 2005, [Online].