Development of a Secure Medical IoT Infrastructure

J. Bermejo Torres

Higher Polytechnic School, San Pablo CEU University, Madrid, Spain, j.bermejo1@usp.ceu.es

Abstract

The aim of the present project is the development of a secure IoT infrastructure with medical purposes, being configurable to offer a close user-friendly experience. The system works by collecting the data from the user in an IoT device, encrypting it with AES and sending it to a remote server, all of this with low consumption. Once data arrive, they are decrypted and shared with other legitimate devices through MQTT using an MQTT broker server with different purposes. The data from the MQTT broker is presented in a dashboard as an example of these purposes.

1. Introduction

The Internet of Things (IoT) is the concept of connecting any device to the Internet and to other connected devices [1]. The IoT world is mostly made up of systems with low computing power, power consumption, memory and disk space that are deployed in insecure networking environments [2]. For that, IoT devices and their infrastructure can be considered vulnerable assets which must be protected using different controls and cybersecurity policies in order to avoid failures. Especially in the health sector these must be taken into account, due to, the consequences of IoT security failures might cause a direct loss of life or a sensitive data loss [2].

An IoT ecosystem is typically composed of embedded devices and sensors, mobile applications, cloud infrastructure, and network communication protocols [2]. In each one of these components must be present the CIA triad (Confidentiality, Integrity and Availability) considered the three most crucial components of security.

A solution to fulfil these requirements is the encryption of the communication between the elements that compose the IoT infrastructure. Symmetric or asymmetric encryption can be applied. The difference is the trust between the data intermediaries, who have to encrypt and decrypt data. In this work, symmetric will be enough because there is trust in the IoT ecosystem developed. The most famous, used and secure symmetric encryption algorithm is AES (Advanced Encryption Standard). It is based on a design principle known as a substitution–permutation network and is efficient in both software and hardware. AES processes blocks of 128 bits using a secret key of 128, 192 or 256 bits.

The secure IoT infrastructure challenge is well known by the whole cybersecurity community. K. Siva Kumar Swamy et al. [3] developed a secure IoT infrastructure using Esp32 board as a device and client, MQTT (Message Queuing Telemetry Transport) bridge and broker (Mosquitto) as intermediaries, MQTT protocol with AES over Wi-Fi as communication protocols, and services as final receivers. MQTT is a Machine-Machine connectivity protocol. MQTT works on top of the TCP/IP stack but is extremely lightweight because it minimizes messaging using a publish-subscribe architecture [2]. One example of how MQTT works is shown in Figure 1 with temperature sensors, where the numbers indicate the order of operation.

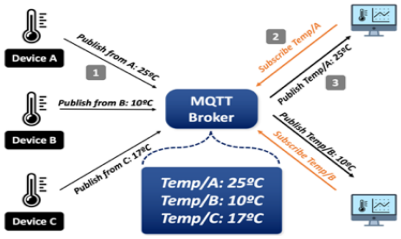

Figure 1: MQTT workflow diagram example.

One of the main problems with MQTT is that authentication is optional, and even if it is used, it is unencrypted by default. When credentials are transmitted in cleartext, attackers with a man-in-the-middle position on the network can steal them and even brokers don't typically limit the number of authentication attempts per client [2]. So before using MQTT, these problems should be solved or be used in a controlled environment. A system very similar to the previous work mentioned is shown by Mohammad Al-Mashhadani et al. [4], using also AES, but avoiding MQTT. The Esp32 is also used, but the device has high consumption and size.

One disadvantage of encryption is the increase of consumption, complexity, and delays of the system, in a way directly proportional to the size of data to be encrypted. Some cryptographers try to improve the AES algorithm to be applied in a light mode to IoT devices in order to reduce the consumption and times [5]. However, AES is standardized and tested, and applying a new algorithm (or only a little modification) that has not been sufficiently tested can be a risky action, only to reduce a little unit of consumption or time.

In a previous project an IoT device was made [6]. The main goal of this project is to improve the previous one , giving it security, new features and services to enhance the system and infrastructure for a close user-friendly and professional experience.

2. Technical Proposal

2.1 Requirements

The main purpose is to develop a platform that reaches the following objectives:

- First, to keep the features and objectives reached in the previous project [6], explained in the next section, (those were a multisensor, wearable, wireless, low-cost, scalable, and small device system) without interfering with the new objectives here proposed.

- Second, to apply configurability: some parameters can change in the IoT environment like the Server IP and port, or the Wi-Fi network and its password. For a more user-friendly experience, the device can be put in a Configuration Mode and modify its configuration via web browser with a graphical user interface.

- Third, to use communication encryption and authentication: using AES 128-bit key for both purposes. The encryption is reasonable with AES, and authentication is a consequence of it because only those who have the key can encrypt and decrypt the data, only two parts know the key, so each one can identify the other.

- Fourth, to be a multiclient service: The server can manipulate multiple TCP/IP connections that arrive and manage their data flows.

- Fifth, to use an MQTT Broker and MQTT in a secure environment as a scalable system to share the data collected from each device and share it with authenticated applications or other devices.

2.2 Device

The device consists of a Raspberry Pi Pico connected to an ESP-8266 Wi-Fi Module and 3 module sensors (pulse, temperature and movement) connected to the Pico (Figure 2). This device was developed in the previous project [6]. It has two slide buttons one for switching on-off and the other to change work mode. The device can work in a Configuration Mode or in a Data Collector Mode.

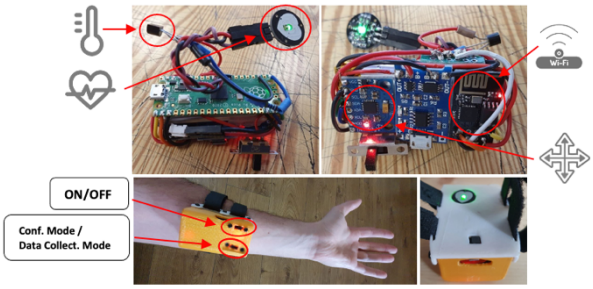

Figure 2: IoT Device used in the project

3. Architecture

3.1 Configuration Mode

If the device has never been configured, it must be put in configuration mode with a slide button that is in the device.

Next, the device works as an access point enabling a Wi-Fi network called "Med_IoT" without a password but allowing only one connection as a cybersecurity policy. Then, any user can access with a laptop or smartphone. Immediately after the access point is enabled, a web server is deployed in port 80, so the device is working as a server, and the laptop or smartphone, which is used to communicate with it, as a client. It offers a web page based on HTML and JavaScript in the IP address 192.168.4.1 (the IP address of the device in the network created by itself) called "init_page.html". It's shown in the figure 3.

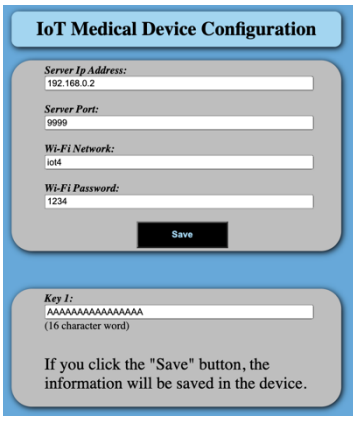

Figure 3. Configuration page shown in the browser

The user can register the IP address and port of the server, the Wi-Fi where the device will be connected and its password. Also, there is a key that must be entered to cipher the data which will be sent to the device. Once everything is completed the user clicks on the save button and this information is saved into the device through a encrypted communication. A configuration file is used to save the information inside the device.

All the communication process is carried over HTTP/TCP. Figure 4 shows the whole process.

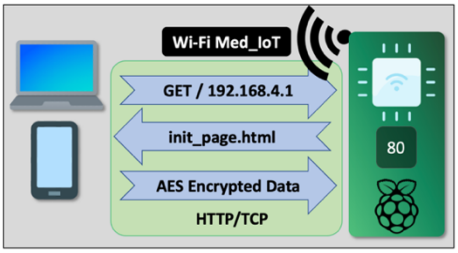

Figure 4. The Configuration Mode process diagram.

3.2. Data Collector Mode

Once the data is configured the device starts to operate in Data Collector mode or also if the slide button is in Data Collector mode state.

The device takes parameters that needs to establish the TCP connection with the server (IP address and port 9998; and the Wi-Fi network and password as network gateway) and establish it. Then, it collects the data from its sensors and formats it in a JSON. Next, it encrypts using AES and sends the encrypted data through the connection established. After all, it starts again the loop since the part of collecting sensor data and so on as Figure 5 shows.

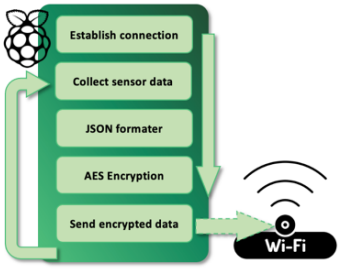

Figure 5. Pico intern process diagram.

Then the data travel secure, by the encryption, through a network, that can be a local network or the Internet (Step 1 in Figure 6), until arriving at the destination, a server listening in the 9998 port (Step 2 in Figure 6).

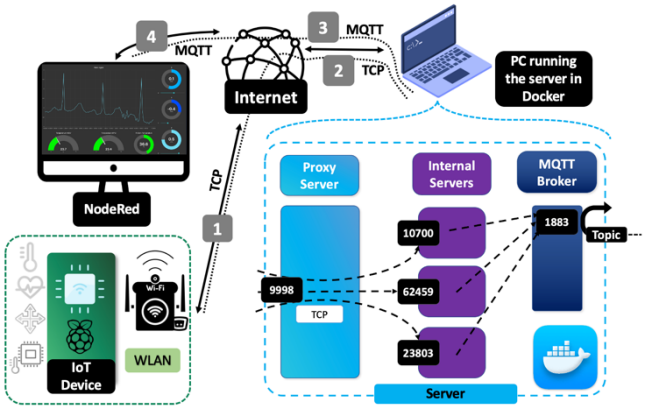

Figure 6. Collector Mode workflow diagram architecture

The server can be explained in three parts: proxy server, internal server and MQTT broker. The proxy server listens in the 9998 port, when a connection arrives it manages the connection, deploys an internal server listens in a specific port (random free port), establishes a TCP connection with that port, and bypass all the traffic that arrives from the initial connection to the TCP connection that connects with the internal server. If it receives another connection in port 9998, the same process is repeated, and so on but changing the port of the internal server. When the data is received by the internal server deployed it has the following tasks (Figure 7):

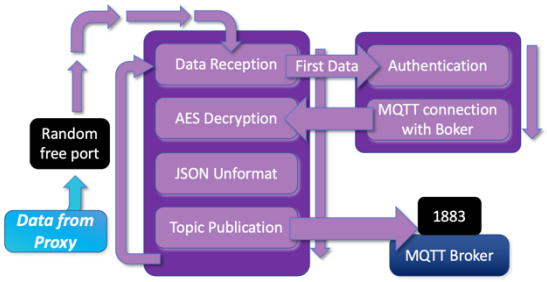

Figure 7. Internal server workflow diagram.

- Authentication: When data flow arrives at the server, it is ciphered and is unreadable, so the internal server must identify to whom it belongs to load the respective key. For that, it has a file with all the keys for each device, and it tries to decrypt the message with each one, when it finds something that it expects with a determined format, it assumes that it has identified the user, so it loads the key and the initialization vector. This process only occurs once, then the key will be associated with the connection.

- MQTT Connection with Broker: When the authentication is successful, the internal server connects with the MQTT Broker. The Broker is listening on port 1883, so the connection is established with that port.

- AES Decryption: Once the key is loaded it decrypts the data with it.

- Unformat JSON into variables: after decryption, the server finds a string with a JSON format, so it unformats the JSON into different variables, which then will be used in the topic publication.

- Topics Publication in Broker: variables are related to an MQTT topic, which will be sent together to the broker.

The broker is at the heart of any publish/subscribe protocol. The broker is responsible for receiving all messages, filtering the messages, determining who is subscribed to each message, and sending the message to these subscribed clients (Steps 3 and 4 in Figure 6). The broker also holds the session data of all clients that have persistent sessions, including subscriptions and missed messages. Another responsibility of the broker is the authentication and authorization of clients.

All the server infrastructure can be dockerized, giving the infrastructure the advantage that offers containerization, like portability, lightweighty and security.

4. Implementation

4.1 Cybersecurity

All the communications susceptible to Man in the Middle attacks are encrypted using AES. Specifically, AES CBC (Cipher Block Chaining) operation mode, using a 128-bit key and a 128-bit initialization vector. In cryptography, a block cipher mode of operation is an algorithm that uses a block cipher to provide information security such as confidentiality or authenticity. In CBC mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. This way, each ciphertext block depends on all plaintext blocks processed up to that point. To make each message unique, an initialization vector must be used in the first block. The initialization vector can be generated randomly.

Other cybersecurity policies applied to the infrastructure can be the unique connection to the wi-fi network generated by the device in the Configuration Mode, the lack of the Internet in that phase, the avoidance of MQTT protocol in the device and its threats, or the dockerization of the server isolating it.

4.2 Configurability

The device is configurated thanks to a user graphical interface developed with HTML, for the structure, JavaScript, for the functionality, and some CSS for the style. All are summarized into a file called "init_page.html", which the web server running on the device sends to the client when the user searches in the browser "192.168.4.1". JavaScript is used to check user inputs and cipher the text inputs when being sent to the device, for that last one a pure JavaScript AES implementation library, called "aes-js", is employed. Something to emphasize is that the system doesn't have Internet access, which is good to avoid external hacking attacks, but it's a disadvantage when using JavaScript because it works with libraries referenced using the Internet. But it can be solved by writing in the library into the HTML code. It is 62 KB additional to a page of 3 KB, with a wi-fi board rate bit of 6,66 KBps it takes about 9 seconds to load the page. By becoming a better and more secure system the price to pay is in time. Making a system

configurable could open vulnerabilities that previously doesn't exist, for that reason the configuration must be implemented bearing in mind all the possible leaking features which can be exploited. One is that the board doesn't support HTTPS, for that an HTTP combined with AES cipher is used.

4.3 MQTT & Broker

MQTT is implemented from the internal server towards the MQTT Broker and beyond. For its implementation, the "paho-mqtt" python library is employed in the server. The topics are defined by the internal server, the MQTT broker publishes them, and then each device or service must know the topic which it wants to access for each information. The topic has a directory string structure so it has an intuitive format. The internal server shares the following topics (where "iot_dev_01" refers to the identified IoT device):

```
Pico/iot_dev_01/Physiological_Data/Temperature
Pico/iot_dev_01/Physiological_Data/Pulse_Signal
Pico/iot_dev_01/Physiological_Data/Accelerometer/x
Pico/iot_dev_01/Physiological_Data/Accelerometer/y
Pico/iot_dev_01/Physiological_Data/Accelerometer/z
Pico/iot_dev_01//Internal_Device_Data/Temperature/MCU
Pico/iot_dev_01//Internal_Device_Data/Temperature/MPU
```

The Broker is implemented with the most powerful, open-source and industry employed nowadays, called EMQX.

4.4 External Applications using the data

The broker works as an API, where services request a directory (topic) and it sends the information stored there. With correct MQTT words, services subscribe to that topic and receive periodically the data that contain that topic. It can be employed by external applications like a dashboard. To show how easy is and how it works a dashboard application using Node-Red has been deployed as shown in Figure 8.

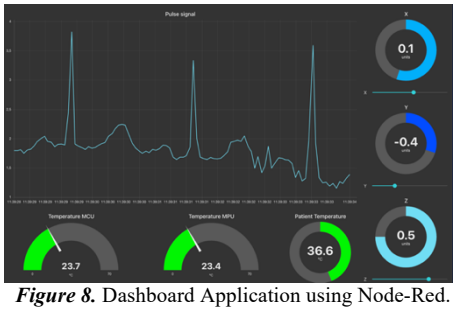

Figure 8. Dashboard Application using Node-Red.

4.5. Docker

Docker offers standardization, being portable anywhere; lightweight, it only installs and uses resources that it will need; and security, isolating the programs and services running from the host machine. First, a dockerfile is used to configure specific dependencies that the containers need, like libraries, but based in an image. Then a docker-compose is used to run the containers. Three containers are running with different ports opened depending on the service offered: Python server: 9998; EMQX: 18083 and 1883; and Node-Red: 1880.

5. Results

To test all the infrastructure a python program has been written to simulate the behavior of the IoT medical device, calling it "pseudo_pico_client.py". It is implemented to avoid the costs of developing many IoT devices and test the multiclient infrastructure. Simultaneous clients are running at the same time using the infrastructure and it responds properly.

Regarding the power consumption of the device in the previous work [6] it was 110 mAh and now in the Configuration Mode is 100 mAh and in Data Collector mode with the encryption is the same 110 mAh as without encryption as in the previous work, 70 mAh of that comes from Wi-Fi-module.

6. Conclusion

A secure IoT infrastructure has been developed fulfilling the initial objectives by keeping the previous project ones, by applying configurability and cybersecurity, using AES encryption, apart from introducing MQTT and multiclient service on the server side.

Regarding the consumption, it is shown that the encryption doesn't affect to the consumption in our case (being compared with the previous work [6]), and the majority of consumption comes from the wireless communication (≈ 70%, depending on the Mode).

This project could help in the future to secure and deploy IoT multiclient infrastructures in an easy mode in medical environments where a failure or a cyberattack can cause a direct loss of life.

As future improvements could be to change Wi-Fi by 5G to permit a high range of operation, the incorporation of location into the device, a final application to exploit all the device possibilities or the adaptability of the infrastructure to be used by other different devices.

The project is uploaded in Github as open-source [7].

References

[1] IBM IoT Definition: https://www.ibm.com/blogs/internet-of-things/what-is-the-iot/

[2] F. Chantzis, I. Stais, P. Calderon, E. Deirmentzoglou, B. Woods. Practical IoT Hacking, The Definitive Guide to Attacking the Internet of Things. no starch press, 2021 (ISBN: 978-1718500907).

[3] K.Siva Kumar Swamy, G.Sony, Ch.Jagadeesh Ram, B.Navven, J.Harshitha. Secure IoT Devices using AES Encryption. Journal of Engineering Sciences. Vol 11, issue 4, 2020 (ISSN: 0377-9354)

[4] M. Al-Mashhadani, M. Shujaa. IoT Security Using AES Encryption Technology based ESP32 Platform. The International Arab Journal of Information Technology, vol 19, no 2, 2022 (ISSN: 1683-3198)

[5] M. Saad Fadhil, A. Kadhim Farhan, M. Natiq Fadhil. A lightweight AES Algorithm Implementation for Secure IoT Environment. Iraqi Journal of Science, vol 62, no 8, 2021, pp: 2759-2770 (ISSN: 0067-2904)

[6] J.Bermejo Torres, Development of a medical monitoring system based on the Internet of Things, Madrid, May 2022.

[7] Project uploaded in Github: https://github.com/bermejo4/Secure_IoT_Medical_Infrastructure