UNIVERSAL SECURITY SOLUTION FOR IOT DATA COLLECTION SYSTEMS

Petr Šťovíček
Bachelor Degree Programme FEEC BUT
E-mail: xstovi04@stud.feec.vutbr.cz

Supervised by: Petr Dzurenda
E-mail: dzurenda@vutbr.cz

Abstract: The article presents our data collection solution based on RIOT operating system and Message Queuing Telemetry Transport (MQTT) technology. Our solution is secure, universal, and easy to integrate to already existing and often insecure IoT networks. Furthermore, we integrated our solution into the temperature and humidity monitoring system. This system detects the potentially dangerous level of these physical quantities and warns users by smartphone notification about relevant threats such as the impact on the infectivity of COVID-19.

Keywords: IoT, Internet of Things, Cryptography, Constrained Devices, Microcontrollers, MQTT, RIOT, OpenHAB, COVID-19

1 INTRODUCTION

The current growth of Internet of Things (IoT) systems and Industry 4.0 is closely associated with connecting a large amount of computationally and memory-constrained devices to the Internet. These devices usually consist of microcontrollers with very different hardware and software resources. Common cryptographic schemes are often impossible to implement because of their computational and memory requirements. This leaves IoT systems susceptible to all kinds of attacks.

One of many application scenarios for these devices is data collection systems such as air quality evaluation systems. Recent research [1] on air quality suggests that temperature and humidity have a direct impact on the viability and transmissibility of COVID-19. Specifically cold and dry environment increases spread of this virus. Therefore, monitoring and maintaining the optimum temperature and humidity in the room is desirable, thus slow down the spread of viruses.

In this article, we proposed a system that allows a user to monitor and react to the current temperature and humidity in the rooms where transmission may occur. Collected data is then analyzed, evaluated, and interpreted to the user. The proposed system is based on MQTT [2] IoT messaging protocol and RIOT [3] operating system. RIOT allows the creation of a program that can be executed on devices from various manufacturers and architectures hence providing hardware independence. That can lead to significant cost saving during project creation.

2 SYSTEM ARCHITECTURE

The proposed data collection system works in a client-server model, where all communication takes place only between the client and the server. Clients do not communicate with each other but only with the server. Clients are represented as various microcontrollers combined with sensors. The data collection server is in the form of a microcomputer. Widely spread WiFi technology is used for data transfer between system elements, this enables system deployment even in places where routing cables is not possible.

The MQTT protocol ensures the transmission of messages between clients and the server. This protocol is simple and lightweight. The MQTT server is called `broker`, its primary role is to route messages between clients, but it can also provide client authentication and access control. There are two types of clients. A `publisher` is a client that sends data to the broker. A `subscriber` is a client that receives data from the broker. MQTT messages are sent to so-called topics. The publisher sends data to the topic via message `publish`. If the subscriber is subscribed to this topic, the message will be forwarded to him by the broker. A Client can be a subscriber and a publisher to many topics at the same time. MQTT alone does not offer a sufficient level of security. For this reason integrity, authenticity, and confidentiality of the transmitted data are additionally ensured with available cryptographic mechanisms.

Security of transmitted data is ensured by the Advanced Encryption Standard (AES) cipher in the authenticated mode Counter with Cipher block chaining Message authentication code (CCM). Each topic has its own encryption key which is assigned to the client by the central trusted authority Key Management Service (KMS) which provides authentication and authorization of clients. KMS maintains a database of clients and their rights to the topics.

Figure 1 depicts a sketch of the proposed system architecture. In step one, the client sends data over an insecure channel to the broker. Data in the message are encrypted with the topic key. In step two, the broker forwards the message to the MKS. KMS performs client authentication and authorization. If the process is successful, KMS decrypts data and inserts it into the new message, which is sent in step three to the broker. The broker distinguishes between two types of traffic. If the message comes from an insecure channel, it is considered as untrustworthy. If the message comes from KMS, it is considered as trustworthy. Only trustworthy messages can be written to the subtopics of KMS (e.g. `kms/topic`). In step four, the message is forwarded to the OpenHAB for storage. OpenHAB is an open-source automation tool that allows to store and further work with measured data [4]. Step five shows data presentation to the user which can take place via mobile application or web interface.

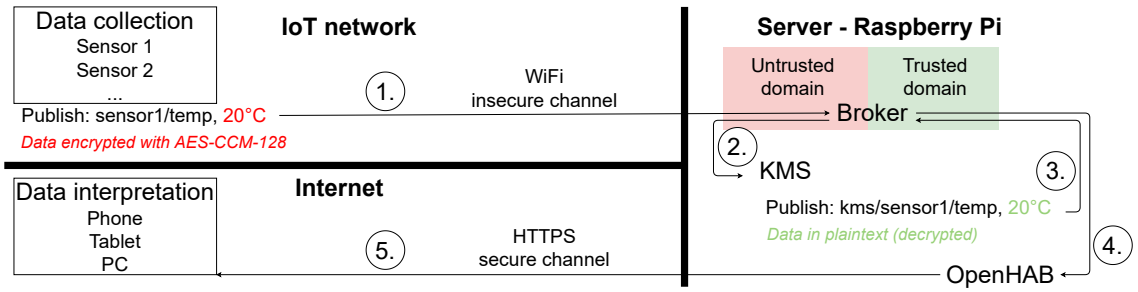

Figure 1: Simplified system architecture.

Figure 1 does not specify a key distribution. Before the client can encrypt any data he must first obtain an encryption key for a given topic (`topic_key`) by requesting it from KMS. Key requests are encrypted with session key (`temp_key`). Negotiation of the session key is achieved by using the Schnorr signature scheme combined with Diffie–Hellman (DH) protocol.

Whole process of session key (`temp_key`) creation, negotiation of the topic key (`topic_key`) and sending data to the topic is depicted in Figure 2. The process begins with a random value `nonce`, which is requested in step one and sent to the client in step two. The client uses this value to compute the session key that is then used to encrypt the name of the topic requested by the client. The ciphertext, signature (`e, z`) and client identifier `ClientID` are sent to the KMS in step three. KMS then verifies the validity of the signature, authenticates the client, and calculates the session key according to Equation 1.

$$temp\_key = A^{SK_{KMS}} = (g^r)^{SK_{KMS}} = g^{r \cdot SK_{KMS}} = (g^{SK_{KMS}})^r = PK_{KMS}^r \tag{1}$$

In step four, KMS sends the topic key to the client along with the `counter_max` value, which indicates how many times the key can be used for encryption. This message is then decrypted by the client with the session key. Topic key negotiation is now complete, the client can use the obtained key to encrypt data in step five. KMS authenticates and decrypts received data and in step six it redirects it in plaintext to the trustworthy topic, which is OpenHAB subscribed to.

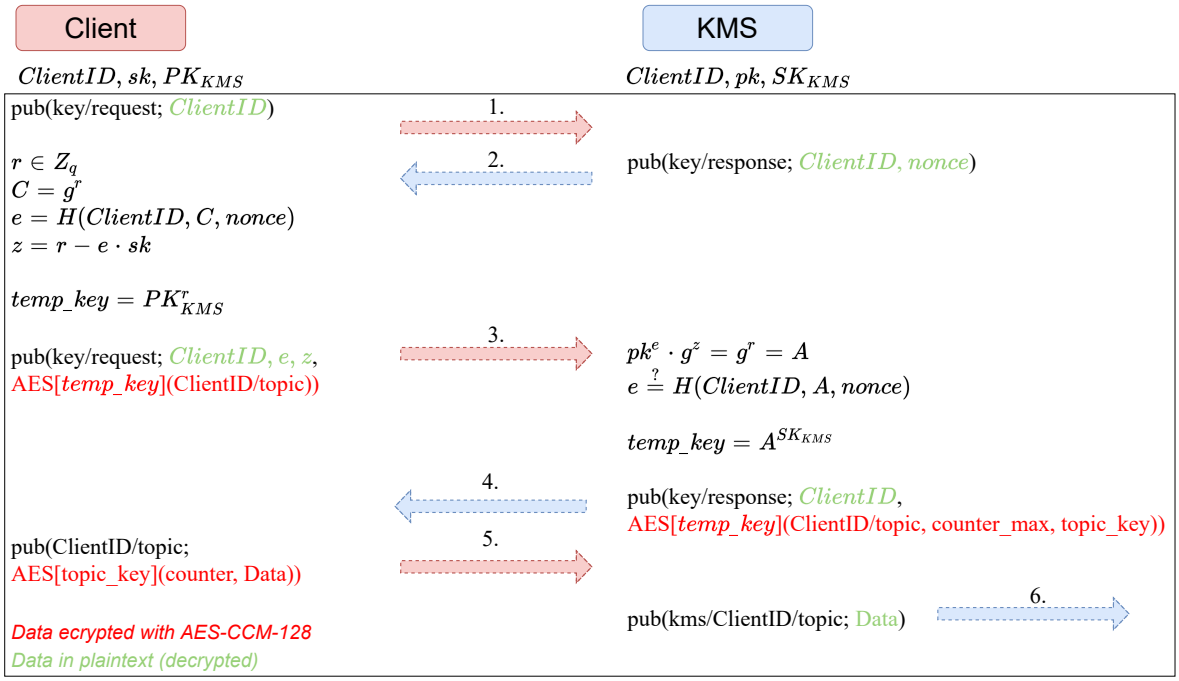

Figure 2: The cryptographic core of the proposed system.

3 SYSTEM IMPLEMENTATION

The proposed system utilizes Raspberry Pi 4 Model B as a server, which hosts MQTT broker Mosquitto, KMS (python script), and OpenHAB. ESP32 and ESP8266 microcontrollers paired with BME280 temperature and humidity sensors represent clients. Clients run RIOT OS, their code is written in C programming language. The cryptographic core is implemented over elliptic curves. Elliptic curve operations and modular arithmetic are provided by the micro-ecc [5] library, while the AES-CCM-128 cipher is provided by the crypto [6] library. Figure 3 shows the performance of both mentioned RIOT compatible libraries for the most computationally demanding operations. The micro-ecc library was selected over the Relic library due to smaller memory requirements.

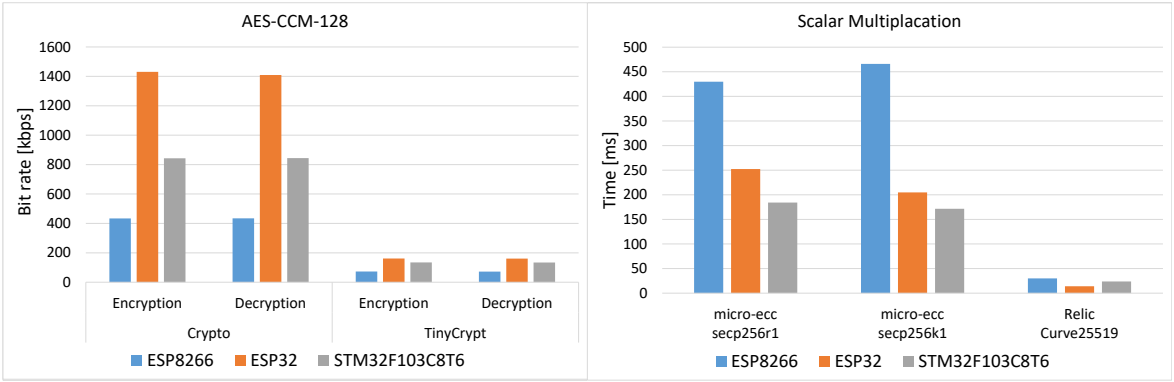

Figure 3: Performance tests: AES-CCM-128 (left), Scalar Multiplication (right).

Figure 4 shows the mobile application interface that is available for iOS and Android. The first from the left is the home screen, which shows an evaluation of the current states in the monitored rooms. When the set limit is exceeded, the background color changes from green to red, the warning message is displayed and the notification is sent. The remaining three images show graphs that appear when the user clicks on the current humidity/temperature on the home screen. Graphs are customizable (display min/max, change the length of the monitored period, combine graphs, etc.).

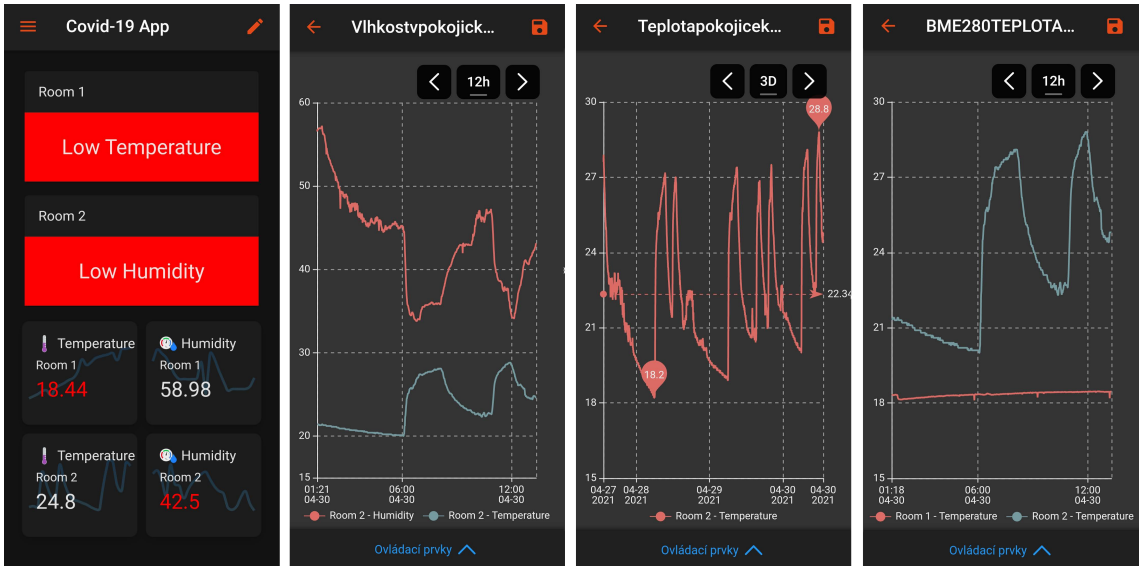

Figure 4: Screenshots from the mobile application.

4 CONCLUSION

In this article, we introduced a data collection system that ensures confidentiality, authenticity, and integrity of transmitted data. We use the WiFi technology and MQTT protocol to exchange application messages. Code is RIOT OS compatible, so it is portable to other supported devices. Measured data are stored and interpreted by OpenHAB service. That allows the creation of graphical interfaces, graphs, setting limits, sending notifications, etc. This particular implementation of the proposed system allows the user to monitor temperature and humidity in rooms. The system then warns the user when the limits, at which the spread of the disease can accelerate, are exceeded. The future goal of the project is the incorporation other devices into the system such as a humidifier or heating control.

ACKNOWLEDGEMENT

This paper is supported by the Czech Ministry of Industry and Trade grant # FV40340.

REFERENCES

[1] MECENAS, BASTOS, VALLINOTO a NORMANDO. Effects of temperature and humidity on the spread of COVID-19: A systematic review [online]. 2020 [cit. 2021-03-07]. Dostupné z: https://doi.org/10.1371/journal.pone.0238339

[2] MQTT: The Standard for IoT Messaging [online]. [cit. 2021-03-23]. Dostupné z: https://mqtt.org/

[3] RIOT OS [online]. Berlin, 2008 [cit. 2021-03-07]. Dostupné z: https://www.riot-os.org/

[4] OpenHAB [online]. [cit. 2021-03-23]. Dostupné z: https://www.openhab.org/

[5] Kmackay / micro-ecc [online]. 2017 [cit. 2021-03-07]. Dostupné z: https://github.com/kmackay/micro-ecc

[6] Crypto [online]. [cit. 2021-03-07]. Dostupné z: http://api.riot-os.org/group__sys__crypto.html