Advanced Human Computer Interaction

Nikolas Banea (79359), Marcus Debera (76834)

Ubiquitous Computing - ESP32

Winter semester 2024/2025

Aalen University
Beethovenstr. 1
73430 Aalen
Germany

Authors:	Nikolas Banea, Marcus Debera
Title:	Ubiquitous Computing - ESP32
Version:	Final
Lecture:	Advanced Human Computer Interaction

Address: Aalen University
Beethovenstr. 1
73430 Aalen
Germany

1 Abstract

Goal of this project is the creation of an ubiquitous computing smart home example, where the user's presence is recognized automatically and an action is executed. To realize this, we have used three ESP32C6 devices that each implement a separate role. The focus of this project lies on extensibility, privacy and security with the assistance of SSL certificates and Mosquitto MQTT broker.

2 Statement of problem

The issues with Smart Home in relation to Ubiquitous Computing tend to be privacy, costs and security. Privacy tends to be neglected, as the devices log data that is hosted on third-party providers, which can be breached or the usage of said data is a gray zone. Additionally, costs tend to be an issue as Smart Home devices tend to be pricey. Finally, as recently seen with the Eufy scandal [1] , even commercial providers tend to have security issues and allow for unknown parties to gain access to sensitive data.

3 Suggested solution

Our suggested solution addresses all issues mentioned above. Privacy can be enhanced, as the code and the communication protocol are open-source and available for self-hosting. For the communication protocol, we have opted for MQTT with Mosquitto due to its versatility, ease of use and open-source nature. Costs for operation are quite low as ESP32C6 can be purchased for approx. 10 Euro per device, while the MQTT broker can be hosted on a Raspberry Pi or use the publicly available Mosquitto MQTT broker offered by Eclipse. Finally, security can be guaranteed with the usage of self-signed SSL certificates and the usage of an access list along with the requirement of a username and password.

4 ESP32 BLE Scanner

The first implemented device with the ESP32C6 is the BLE scanner. This device scans for a defined BLE beacon periodically. Once the defined beacon is within a specified range, a command is sent to an MQTT-based lamp via a defined MQTT topic (publish). Adaptability of the code is rather easy, as MQTT, WiFi and BLE parameters can be adapted accordingly. For the MQTT connectivity, the SSL certificates generated by the MQTT server are required and are defined within the code base. To further enhance security, it is also required to provide user credentials (username and password). The BLE scanner has the ability to turn its built-in LED on whenever a defined beacon is within range for debugging and general awareness purposes. The device connects to a local WiFi network. The network must support 2.4 Ghz frequency. If the MQTT broker is hosted locally, the WiFi network doesn't require internet connection. In our case, the MQTT broker was hosted on the internet and therefore an internet connection is required.

5 ESP32 Lamp

A separate ESP32C6 device implements a MQTT-based smart home lamp. This device also connects to the MQTT broker with the specified SSL certificates and user credentials. The device then subscribes to the defined MQTT topic and waits for messages. The device will reconnect to the MQTT server in case a disconnection occurred. If the message "on" has been received, the built-in LED will be turned on (default colour: white) and in case of "off", the LED will be turned off. The requirement for a WiFi network (with internet connection) applies for this device as well.

6 BLE Beacon

This device implements a BLE server with a specified name and UUID. The advertisements occur on intervals of 2000 milliseconds. The UUID in our case is Version 4, which is a combination of random numbers due to simplicity reasons. The version of the UUID could be adapted to other versions if required. The UUID used in our example has been generated on UUID generator. The name and UUID is used by the BLE scanner to verify authentificity, although this can be spoofed.

7 MQTT - Mosquitto

MQTT is being used for the communication between devices with the implementation of Eclipse Mosquitto. Mosquitto is simple to use, offers strong access control, wide extensibility, is open-source and free to use. It is also lightweight to host and widely available for many Linux distributions. For using SSL certificates, user credentials and access list, the following configuration file (mosquitto.conf) has been used:

listener 8883
allow_anonymous false
cafile path/to/ca.crt
keyfile path/to/mosquitto.key
certfile path/to/mosquitto.crt
require_certificate true
password_file path/to/passwordfile
acl_file path/to/aclfile

Please note that the require_certificate set to true requires valid client certificate and private key (keyfile, certfile). Please note that this is optional but enhances security even further. For defining the access list, please take a look at the official Mosquitto documentation. (https://mosquitto.org/man/mosquitto-conf-5.html).

8 Security

8.1 SSL certificates

OpenSSL has been used to create the certificates. These certificates are self-signed and require no third-party CA for validation, although a third-party CA can be defined if

wanted. To generate the CA certificate, which is required for encrypted communication, following commands are required:

openssl req -new -x509 -days 365 -extensions v3_ca -keyout ca.key -out ca.crt -subj '/CN=localhost' openssl genrsa -out mosquitto.key 2048 For the following command it is required to set the /CN parameter to the address of the Mosquitto broker: openssl req -out mosquitto.csr -key mosquitto.key -new -subj '/CN=Mosquitto_broker_address' Finally, create the CA certificate: openssl x509 -req -in mosquitto.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out mosquitto.crt -days 365

The following steps are only required if the Mosquitto broker requires client authentification with certificates (require_certificates in mosquitto.conf set to true): openssl genrsa -out esp32.key 2048 openssl req -out esp32.csr -key esp32.key -new -subj '/CN=localhost' openssl x509 -req -in esp32.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out esp32.crt -days 365

The resulting ca.crt, esp32.crt and esp32.key will need to be copied and applied in the corresponding parts within the BLE scanner and MQTT lamp code base. The ca.crt, mosquitto.key and mosquitto.crt files need to be in a location that is readable by the user mosquitto (/etc/mosquitto/ca_certificates or /etc/mosquitto/certs/). A manual permission override might be required.

8.2 Users

With the mosquitto_passwd utility, users can be defined with a username and a hashed password. For further details, please look at https://mosquitto.org/documentation/authentication-methods/.

9 Restrictions

9.1 Range

As the ESP32C6 is used both WiFi and BLE at the same time, the range of the BLE beacon can be inaccurate.

9.2 Latency

As in our example, we used a externally hosted MQTT broker, network latency could be a limiting factor in terms of performance.

10 Summary and outlook

This project has proven the possibility of implementing a simple smart home environment locally while still maintaining a high level of security. It also can be extended further with the usage of Mosquitto plugins, for example logging messages to a database.

List of references

[1] heiße, Nico Ernst. *Eufys Kameras funken ungefragt in die Cloud und sind per Web zugänglich*. Dec. 2022. URL: https://www.heise.de/news/Eufys-Kameras-funken-ungefragt-in-die-Cloud-und-sind-per-Web-zugaenglich-7358310.html.