IoT Smart Door Lock with Wireless Key Sharing for Tourism Applications

Bernardo Marques
bernardocmarques@tecnico.ulisboa.pt
Instituto Superior Técnico - University of Lisbon
Lisbon, Portugal

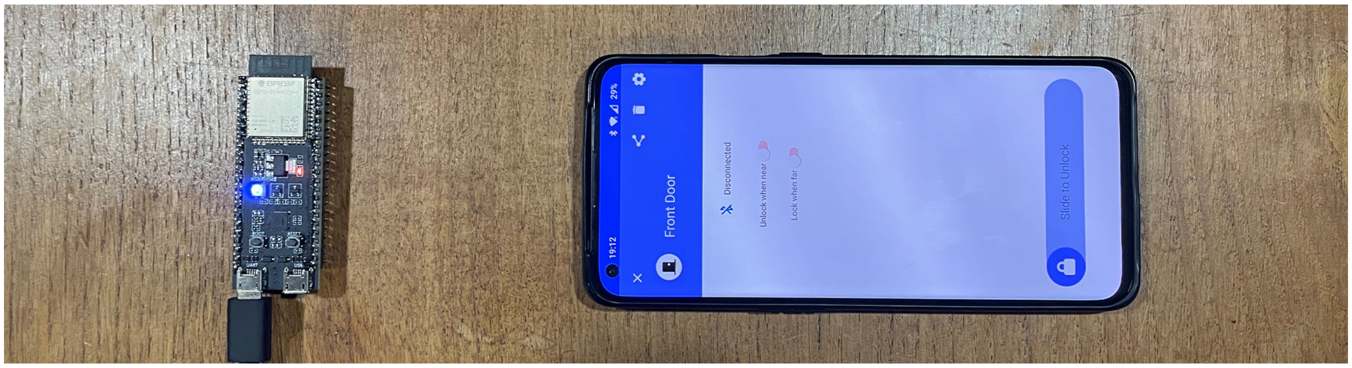


Figure 1. ESP32-S3 and a smartphone running the SDL system Mobile App

Abstract

IoT technology has rapidly grown in recent years, both in terms of research and product development. The usage of IoT technology is growing in a variety of Smart Home applications, such as Smart Home Security, as a result of the development of technologies like 5G network and BLE as well as the appearance of faster and more compact embedded systems. In this project, we implemented a Smart Door Lock capable of being controlled by the user's smartphone in a user-friendly and secure way, using BLE and IoT technology. This system focuses on capabilities that let the user remotely grant access to his Smart Door Lock, which makes it a great resource for rental housing environments. The final developed system could be given a level 4 classification on the Technology Readiness Level scale, which is equivalent to a technology validated in a lab, and it could easily be improved to a level 5 classification, which corresponds to a technology validated in a relevant environment.

Keywords: Smart Home; Smart Door Lock; BLE; Internet of Things; Home Security; Embedded Systems

1 Introduction

The field of Internet of Things (IoT) technology is quickly growing, both in terms of research and commercial development. The use of IoT technology is expanding across a variety of applications, especially in Smart Home applications, thanks to the development of faster and more compact embedded systems as well as developing technologies like the 5G network and Bluetooth Low Energy (BLE).

Security-related technology, such a Smart Door Lock System, is a crucial component of the Smart Home. IoT technologies are currently being used in a variety of smart lock systems, including password locks, fingerprint locks, and even smart locks that operate with a secret code stored on a magnetic band card, Near Field Communication (NFC) card, or smartphone. However, the majority of currently in use systems have certain shortcomings or related issues, such as security flaws, poor power efficiency, or a lack of crucial features.

The goal of this project is to develop a system with a ton of user-friendly features, including "Proximity control," "Unlock on-demand mode," "Remote Control," "Guest access and User roles," and others, while maintaining security and low power consumption in a small, tightly integrated, and reasonably priced product. A feature that enable users to remotely give access to his locks to another user without physically meeting them is the main emphasis of this project.

The final solution will be open-source, enabling community collaboration and the development of a more secure and reliable system. By doing so, users may verify that the system is secure rather than relying on the promises of a black-boxed commercial solution, and they can even help develop a more robust and feature-rich solution.

2 Background and Related Work

This section examines some of the existing IoT Smart Door Locks and looks at the current state of the art. It also explains some of the conventional ways to exchange keys. Finally, we will look at some of the hardware platforms proposed for this project.

2.1 Existing Smart Lock solutions

Smart Locks with Keypad - . This type of smart lock, which is perhaps the oldest of all [5], is frequently used to secure shared spaces. To open the door with one of these locks, the user needs a passcode. If the passcode is entered correctly into a keypad, the door will unlock. It is possible to have numerous passcodes for various users in some locks. On that basis, it is possible to track door usage or even prevent the use of specific codes on particular days.

The main benefit of this kind of lock is how simple it is to share keys with another user, only a passcode needs to be exchanged. Unfortunately, this convenience came at the expense of security.

Smart Locks with RFID and Magnetic cards - . This type of lock is widely used in hotels and rental rooms. In this system, the user has a keycard or RFID tag with the credentials to unlock the door. [4] The user needs to swipe the card in the lock to open it. These cards can use RFID technologies or magnetic bands. With these locks, it is also possible to log and manage the use of the door because each user can have a different keycard.

This solution can be very secure, since it is possible to use more advanced encryption algorithms. The main disadvantage of this solution is the difficulty to share keys with other users. If the user wants to share a key it is necessary to create a new keycard for the new user, and then physically give it to the user.

Smart Locks with RFID and Magnetic cards - . This type of lock is widely used in hotels and rental rooms. These locks require a keycard or RFID card that has the credentials to unlock the door. Magnetic bands or RFID technology can be used in these cards. Because each user can have a unique keycard, it is also possible to track and monitor door usage with these locks.

This solution can be very secure, since it is possible to use more advanced encryption algorithms. The main disadvantage of this solution is the difficulty to share keys with other users. If the user wants to share a key it is necessary to create a new keycard for the new user, and then physically give it to the user.

Smart Locks with Biometric sensors - . Biometric sensor locks operate by verifying a distinctive physical characteristic of the user, such as fingerprints [9], face recognition [8], voice recognition [7], iris scanning, or palm recognition. This type of lock is normally used in high-security facilities. They work by scanning a physical characteristic of the user and converting it to a numerical template. The device then compares the user's biometric characteristic to the saved template when they attempt to unlock the door.

This solution provides a highly secure way to authenticate a user because these biometric features are unique to a given user. The main disadvantage of these systems is the difficulty to share access to the door because the new user needs to register his biometric feature in loco.

Smart Locks with wireless technologies - . The most recent innovation in smart locks is related to the use of wireless technologies to automatically send credentials over the air. These technologies include 5G, Wi-Fi [3], Bluetooth [6], ZigBee, Z-Waze, BLE and many others. The basic concept is to use a mobile device (i.e. smartphone) to communicate with the lock sending the necessary credentials over the air and unlocking the door. This kind of system allows the use of more complex encryption algorithms and, therefore, creates more secure systems. All these can be done automatically and without any input from the user.

These systems have many advantages, like the possibility to create secure systems based on modern cryptography algorithms and the ease of use for the end-user. Another advantage of these systems is the possibility to generate keys and share them with other users remotely.

Comparison between Smart Locks - . All these different types of locks have advantages and disadvantages. Keypad locks are a good choice if it is necessary to share access with multiple users, but they lack security and they are hard to maintain. Both keycard technologies and biometrics, are good candidates for a secure lock, but it is hard to share access without coming into close contact. Finally, we have locks that use wireless technologies, these locks, if used correctly, can provide a high level of security while being easy to share between users, being easy to use, requiring low maintenance and having a good power efficiency.

Table 1. Table comparing the different types of smart lock

Smart Lock Types	Security	Share access	Ease of use	Maintenance	Energy efficiency	Avg. Cost
Keypad	Poor	Excellent	Good	Poor	Good	$150
RFID or Magnetic Keycards	Excellent	Poor	Good	Good	Good	$200
Biometric	Excellent	Poor	Good	Poor	Good	$350
Wireless	Excellent	Excellent	Excellent	Excellent	Good	$250

2.2 Traditional key sharing mechanisms

One of the main features intended for our system is its use in rental houses to facilitate key sharing between the owner and the tourist, thus, it is important to analyse the conventional ways used to share keys in rental houses situations.

- **In-person delivery** - The most basic way to share keys is to meet the guests in person and give them the key when first arriving at the house. This method has some obvious disadvantages, just as time coordination and unnecessary dislocations.

- **Rely on a third person** - Another usual method is to rely on a third person, like a neighbour, a cleaner or even a local coffee shop, to deliver the key to the guests. This is slightly better than the previous method, the owner can live further from the rental house, however, it shares most of its disadvantages. The third person is also time-constrained, and he is not available 24/7 to deliver the key.
- **A Lockbox** - One of the most used solutions is to have a lockbox, a small and secure container to store the house keys inside the property. With a lockbox, the owner communicates the correct lockbox combination, and the guest can unlock it and retrieve the house key. This solves all the time constraints given that the key is accessible 24/7, without relying on an in-person meeting. However, this method has some disadvantages of its own, just as, the need to trust the guests to store the key in the lockbox when leaving, without stealing the key.

These are the most usual solutions to share keys in a rental house, and as we can see, we have some disadvantages to all of them. We believe that our Smart Door Lock (SDL) system solves most of these problems, giving the easiness and comfort of the lockbox, without the need to worry if the guest will return the key since the system will automatically revoke access when the rental period ends.

2.3 Hardware Platforms

Given the objectives of this project and the security and power consumption requirements, multiple boards were considered to build the SDL system, some of which were used in the work reviewed. We considered using the Atmel ATmega32U4, the PSoC 4 BLE and ESP32-S2/S3.

ESP32-S2/S3: ESP32-S2 [1] and ESP32-S3 are highly integrated, low-power, single-core Wi-Fi Microcontrollers SoC, designed to be secure and cost-effective, with high performance and a rich set of IO capabilities.

Ultimately the ESP32-S family (S2 and S3) was chosen for this project because of the low-power features and hardware-accelerated security features.

3 Proposed IoT Smart Lock System

This section will describe the proposed SDL system, go over the system requirements, take a look at the system as a whole, and then go into more detail about each component.

3.1 System Requirements

The SDL system developed has some features already present in the existing systems, the main difference and novelty, beyond the open source nature of this system, is the focus on the key sharing aspect of the system, which allows users to share access to the smart lock with multiple users with different levels of access control. This feature will be very useful in home renting for tourism scenarios.

The main features of this smart lock system are the following:

- **Proximity control:** Lock or unlock the smart lock if the user enters or exits a given range. This feature allows the smart lock to detect a mobile phone with permission to control it and automatically unlock the smart lock when the user enters a close range of the smart lock and then automatically locks it when the user leaves that range.
- **Unlock on-demand mode:** This feature allows the user to lock or unlock the smart lock on-demand through the mobile app. With this feature, a user with permission can control the smart lock without relying on the proximity control feature.
- **Unlock on-demand mode remote:** This feature allows the user to remotely lock or unlock the smart lock on-demand through the mobile app. With this feature, a user with permission can remotely control the smart lock from any place in the world.
- **Guest access and User roles:** This is the main focus of this system, and it gives the possibility to share access to a given lock. The user can choose between multiple user roles which can control the smart lock, but with different levels of permissions.

3.2 System Overview

The developed SDL system have three main components: the **IoT to control the smart lock**, a **central service server** and a **mobile application**. The user can control the smart lock with the mobile app, lock and unlock it when in range and lock and unlock it on-demand. The mobile application also allows the owner or tenant to manage his smart locks and their users.

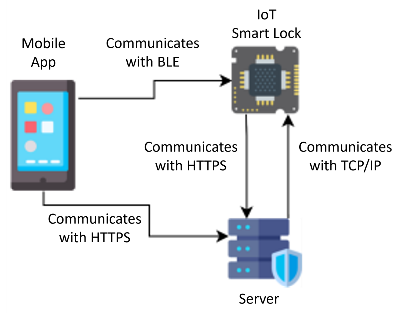


Figure 2. Multiple components of the system and how they interact.

The system uses many communication protocols and technologies to communicate between each component, as seen in figure 2. The central service server is running an HTTPS server and so, all the components that communicate with the central service server use HTTPS. The components use two distinct technologies to communicate with the IoT: the Mobile App only uses BLE for direct communication with the IoT, while the Service Server uses a TCP/IP connection to transmit commands to the IoT.

3.3 Secure Communication

All communications with the central service server is already secured using SSL/TLS, however, both communication channels with the IoT are not secure by default. For that reason, it was necessary to implement an extra layer of security on top of the communication channels. In this secure channel, all the messages exchanged are encrypted using AES-256 in CRC mode.

In figure 3, it is possible to see how the creation of this secure channel works. First, the mobile phone tries to connect to the smart lock (IoT). When the connection is accepted, the phone then creates an AES-256 session key and sends it to the smart lock, encrypted with the smart lock public key. Only the smart lock can decrypt this message and therefore retrieve the session key.

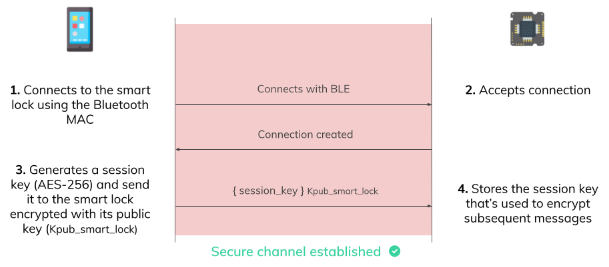

1. Connects to the smart lock using the Bluetooth MAC
Connects with BLE
2. Accepts connection
Connection created
3. Generates a session key (AES-256) and send it to the smart lock encrypted with its public key (Kpub_smart_lock)
{ session_key } Kpub_smart_lock
4. Stores the session key that's used to encrypt subsequent messages
Secure channel established ✓

Figure 3. Creation of a secure channel between the mobile app and the smart lock

After this initial session key exchange, all the following messages are encrypted with this session key. The messages also include timestamps and a nonce to prevent replay attacks.

3.4 Smart Lock Hardware - ESP32-S2 IoT

The smart lock is one of the most crucial components in this system. The idea was to use an IoT, the ESP32-S2 and ESP32-S3, to control an electronic lock mechanism or a latch with a servo motor. However, in this thesis, the final solution is a proof of concept that makes use of an RGB led to simulate the lock status.

3.4.1 Hardware.

The SDL system has 2 versions, they are very similar between them, the main difference being the hardware it was used.

For the first version, the hardware consists of the ESP32-S2 IoT and a BLE module, and in the second version, the hardware consists of only the ESP32-S3 IoT, which already contains an internal BLE module.

For this project, the ESP32-S2/S3 will be powered by a USB adapter, and the status of the smart lock will be simulated with an RGB led, according to table 2. Considering a scenario for a real implementation for the SDL system, the ESP32-S2 could be powered by batteries (for instance 18650 lithium-ion cell batteries). To unlock and lock the door, we could send pulses through some of the available GPIOs in the ESP32-S2/S3.

Table 2. Table with the main status of the smart lock and its corresponding colour

Colour	Status
Yellow	Idle - Waiting Connection
Blue	Connected (Smart Lock Locked)
Green	Smart Lock Unlocked
Yellow (flashing)	Smart Lock Locking
Red (flashing)	Invalid Authentication

3.4.2 BLE Server.

The BLE Server handles the communication between the Moblie App client and the smart lock. This BLE Server allows the smart lock to be controlled directly by the user at close range, using the Mobile App or its proximity features.

A BLE Server does not work exactly like a TCP server or even a Classic Bluetooth server. The BLE server is called a Generic Attribute Profile (GATT) server, and it contains *services* and *characteristics* [11]. Services are a container that conceptually groups related attributes, while characteristics are the attributes included in a service, and each of them is used to communicate a specific type of data.

The characteristics hold the data that can be accessed by the client. Characteristics can allow different types of operations: **broadcast**, **read**, textbfwrite, textbfnotify . With these operations, the client can read and write data in the server, and therefore communicate with the server.

3.4.3 TCP Server.

The TCP Server handles the communication between the central service server and the smart lock. This TCP Server allows the smart lock to be controlled with requests from the central service server. The user can send remote requests to the central service server, and these requests are forwarded to the smart lock.

3.4.4 Storage Database.

Although the Service Serve is where the majority of information and data are stored, each smart lock also temporarily stores certain essential data, working like a cache. This data is stored in the ESP32-S2/S3 Non-Volatile Storage (NVS) memory.

3.4.5 Message protocol and Operations Examples.

Both the BLE and TCP servers use a simple message protocol that consists of the commands presented in table 3:

Table 3. Message protocol commands

Protocol Commands	Request Result
SSC *session_key*	Send Session Credentials
RAC *seed*	Request Authorization Credentials
SAC *user_id, auth_code*	Send Authorization Credentials
RLD	Request Lock Door
RUD	Request Unlock Door
RDS	Request Door Status
SDS *status*	Send Door Status
RNI	Request New Invite
SNI	Send New Invite
RFI	Request First Invite
SFI	Send First Invite
RUI	Request User Invite
ACK	Acknowledge
NAK	Not Acknowledge

3.5 Smart Lock Service Server

The Service Server consists of a RESTful API in a python web server built with Flask and hosted in a Linux machine provided by Amazon Web Services (AWS). The server also interacts with the Authentication Service provided by Google Firebase with specific libraries that use HTTP requests.

The Service Server have two main objectives in this system:

3.5.1 Gateway for Remote Control. The Service Server is used as a middleman between the smart lock and the client's Mobile App. This communication channel is used when the Mobile App want to remotely connect to the Smart Lock, via the Internet. Instead of connecting directly to the Smart Lock TCP server, the Mobile App will send the messages to the Service Server using the RESTful API, and the Service Server will then redirect these messages to the Smart Lock via TCP/IP, as demonstrated in figure 4. By allowing only requests from the Service Server and blocking unauthorized access to the Smart Lock TCP Server directly, this middleman can increase the security of the Smart Lock.

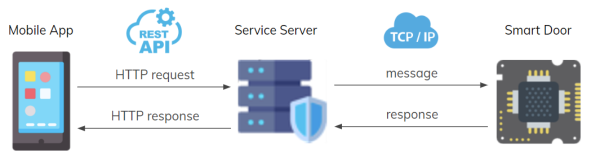


Figure 4. Example of communications using the Server as a gateway

3.5.2 Database. Firebase Realtime Database [1] is used to manage the system's database. Firebase Realtime Database is a cloud-hosted NoSQL database provided by Google that is used to store and sync data in real-time. The Service Server (through the REST API) is used as an interface to communicate with the Firebase Realtime Database.

[1]https://firebase.google.com/products/realtime-database

3.6 Smart Lock Mobile App - Client Key

The mobile app is the primary interface for the smart lock and serves as its "key". The user is able to access and control the multiple smart locks registered in the app. It has features to remotely lock or unlock the smart lock and options to turn on or off the proximity lock or unlock. The user can also check the smart lock's state, and have the features to add, edit or remove users for their smart locks.

3.6.1 Main features. The mobile app has two different sets of features. Features to control a smart lock, meant for every user to access a given smart lock. And features to manage a smart lock and its users meant for the Admin, Owner or Tenant of a given smart lock.

Control features (all users):

- **List of smart locks** - The user has a list of smart locks that he can control, and he is able to select one to use it.
- **Toggle Proximity Unlock/Lock** - The user has the option to enable or disable the feature that automatically unlocks/lock the smart lock by proximity to the smart lock.
- **Toggle lock or unlock** - The user has a button where he can lock or unlock the smart lock, both near it or remotely.
- **Add a new smart lock** - The user has the option to add new smart locks, by using an invite code created by an Admin, Owner or Tenant.

Manage features (Admin, Owner and Tenant):

- **Invite new users** - The Admin, Owner and Tenant can use this option to create an invite for a new user to control the smart lock.
- **Remove user** - The Admin, Owner and Tenant can use this option to remove a user from the authorized users to control a smart lock.
- **Edit user** - The Admin, Owner and Tenant can use this option to edit a given user, changing its user type for instance.

3.6.2 Mobile App User Interface. The mobile app was developed using Java in the Android Studio IDE[2]. For the User Interface (UI), we followed the Material Design[3] guidelines as much as possible. The application is divided into multiple pages, called Activities. Each activity represents a screen or page with a visual UI.

The main activities that we have in the application are:

[2]https://developer.android.com/studio
[3]https://material.io/design

Table 4. Mobile App activities

Sign-In	Activity where the user can log in to the app.
Sign-Up	Activity where the user can create a new account.
Smart Locks List	This activity contains a list of the user's smart locks and the user can select a smart lock to interact with it.
Smart Lock Control Page	Activity where the user can control a selected smart lock.
Add a new smart lock	In this activity, the user can set up a new smart lock and add it to his account.
Create a new invite	In this activity, the user can create a new invite to add a new user to a smart lock.
User Managing	Activity where the owner of a smart lock can manage his the smart lock.

4 Smart Door Lock Implementation

This section will examine how the proposed SDL system was implemented. We will describe the development methodology and the development environment. We will also explain in more detail how each component was implemented.

4.1 Development Methodology

The development process of this project consisted on the implementation of an initial proof of concept to validate our design before developing the whole system, which was done on a per component basis. At the end of the development, the whole system was tested for correct functioning and validation of the implementation of the initial requirements

Although all the different components were developed simultaneously, in the following sections, we decided to split the explanation of each component.

4.2 Development Environment

This project was developed mainly on a Windows-powered computer, using multiple different tools and frameworks for each parts of the SDL system.

Smart Lock Hardware:

- **ESP-IDF v4.4.1** - ESP-IDF is development framework for Espressif SoCs. It was used to compile and flash all the code related to the ESP32-S2/S3.
- **CLion IDE** - CLion is a IDE for C and C++ developed by JetBrains. It was used as the main IDE while coding the ESP32-S2/S3 part of the system since it is coded in C.

Smart Lock Service Server:

- **Flask** - Flask is a micro web framework written in Python. Flask was used to develop the RESTful API interface that is used to communicate with the Central Server.
- **Firebase Realtime Database** - Firebase Realtime Database is a cloud-hosted NoSQL database that lets you store and sync data in realtime. This Google-provided product was used as a database for our system.
- **Firebase Authentication** - Firebase Authentication is a secure and easy user authentication solution provided by Google. This product was used to authenticate the user in the system.
- **PyCharm IDE** - PyCharm is a IDE for Python developed by JetBrains. This was used as the main IDE while coding the Flask RESTful API in the Service Server.

Smart Lock Mobile App:

- **Android Studio** - Android Studio is the official IDE used to build Android apps. This IDE was used for the development of our mobile app client for Android.
- **Figma** - Figma is a collaborative browser-based interface design tool. This tool was used to create the initial design for the Android mobile app.

4.3 Smart Lock Hardware

This section explains the development process of the Smart Lock Hardware, the ESP32-S2 and S3. It explains with some level of detail each feature, how it was done and which technologies were used. It also explains the optimizations done, and solutions found for any problem encountered.

Creating an Initial Proof of Concept. The first step in the development of the Smart Lock Hardware was to learn how to use the ESP-IDF and how to write and flash code for the ESP32-S2. After understating the basics of the ESP-IDF we created a simple proof of concept with a subset of the planned features. This simple Proof of Concept consisted of a TCP server with the feature to unlock on-demand and the necessary functions to perform the exchanges of credentials for user authorization and create a secure channel. This initial proof of concept was used for a simple evaluation to validate the feasibility of the system.

ESP-Touch to configure Wi-Fi and HTTP Client Test. After the initial proof of concept, we needed to implement a way to set up a Wi-Fi connection on the first configuration of the ESP32-S2/S3 without hard-coded Wi-Fi credentials. To do so we used the ESP-Touch[4] protocol, created by Espressif Systems used to seamlessly configure Wi-Fi devices connecting to a router.

We tested the internet connection on the ESP32-S2/S3 using a simple HTTP client that communicated with a free notification web service called PushingBox.

BLE server for for Close Range Communication. The next development step was to improve the communication capabilities of our system and implement the BLE server for short-range communications. We used the JDY-23 Ultra Low

[4]https://www.espressif.com/en/products/software/esp-touch/overview

Power Bluetooth BLE Module [10] because the ESP32-S2 doesn't have an internal BLE module. The JDY-23 automatically creates a GATT server and communicates to the ESP32-S2 using the Universal Asynchronous Receiver/Transmitter (UART) protocol.

We also implemented a different BLE server for the ESP32-S3, which has an internal BLE 5.0 module. This server follows the same principles as the server used by the ESP32-S2. It is also a GATT server but we coded it directly, instead of using UART to communicate with a module which had an already working GATT server. The new BLE 5.0 server implemented works just like the previous BLE 4.2 server, so both could be used by the same client.

HTTPS Client to communicate with the server. We also implement a more robust and secure HTTPS client, that supports both GET and POST requests and verifies Secure Sockets Layer (SSL) certificates to securely communicate with the Central Service Server. We used the provided "esp_http_client.h" library to create the HTTP client and the MIT Licensed C library "cJSON" [5] to manage the JSON object used in the POST data and requests response.

Invites and Share Access Mechanisms. After developing the TCP and BLE server and securing both communication channels, we implemented the invite system, the feature to create invites and share door access. We end up developing three different types of invites, each with its purpose.

- **User created invites:** These invites are the main type of invite and they are created by a user who wants to share his lock with another user.
- **First user creation:** These invites are used to create the first user of each lock.
- **Multiple phones per user:** This kind of invite is used to automatically create a new authorization for the pair {phone, smart_lock} using an invite when the user uses a new phone.

Creation and exchange of RSA keys. We implemented a method to dynamically configure the keys in the ESP32-S2/S3 (instead of hard-coding it) and then safely share the public key, in a way that it could be trusted.

To do so we created a private Certificate Authority (CA) with a self-sign certificate. This CA was used to sign the RSA public keys used by each smart lock, creating an X.509 certificate which is then stored in the ESP32-S2/S3 using the SPIFFS Filesystem [6].

4.4 Smart Lock Service Server

RESTful flask API. The initial step in the server implementation was to implement a simple RESTful API, with only the necessary endpoints to test GET and POST requests, using the Flask framework [7] and Python 3.10 [8]. We also implemented the necessary functions to help us with the cryptographic operations, using the pycryptodome [9] python library, and function to communicate with the Firebase Realtime Database database using the official library firebase-admin [10].

Developing the basic endpoints. Most of the necessary endpoints and functions in the server are related to operations in the database, there are the following:

Endpoints related with the user account database:

- **GET** */get-user-locks?id_token={id_token}*
- **POST** */set-user-locks*
- **POST** */delete-user-locks*
- **POST** */register-phone-id*

Endpoints related with smart locks database:

- **POST** */register-door-lock*
- **GET** */check-lock-registration-status?MAC={MAC}*
- **GET** */get-door-certificate?id_token={id_token}&MAC={MAC}*
- **POST** */request-authorization*

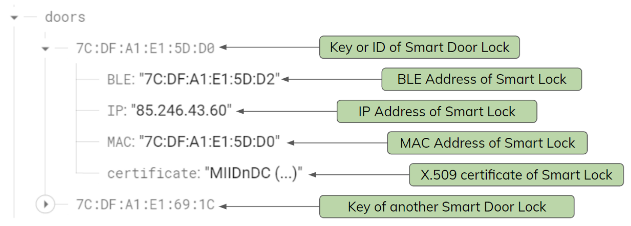
Create and redeem invites. The server is responsible for creating and redeeming the invites used to share lock access, as explained in section 4.3. The Smart Lock can request a invite creation using a **POST** request to */register-invite*. To redeem the invite, the mobile app can send a **POST** request to */redeem-invite* endpoint.

The server is also responsible for saving and redeeming the User Invites, with the **POST** */save-user-invite* and **POST** */redeem-user-invite* endpoint.

Firebase Realtime Database and Authentication. The Firebase Realtime Database is used to store all the necessary information in the system, this information is accessible by the Smart Door Locks and the Clients through the Central Service Service. In Realtime Database the information is stored in a JSON tree structure and the data is stored as JSON objects. In figure 5 we can see the structure of the information stored in the database about each Smart Door Lock.

Gateway to communicate remotely with a Smart Door Lock. The server is used as a middleman in the communication between the mobile app client and the smart door lock, as already explained in section 3.5.1. The client can do a **POST** request to "*/remote-connection*" with the message or command to send to the Service Server, and this command will be forward to the respective Smart Door Lock, using a TCP connection between the Service Server and the Smart Lock. To implement this feature we developed a simple python TCP client in the Service Server. This client can

[5]https://github.com/DaveGamble/cJSON
[6]https://docs.espressif.com/projects/esp-idf/en/latest/esp32s3/api-reference/storage/spiffs.html
[7]https://flask.palletsprojects.com/en/2.2.x/Flask framework
[8]https://www.python.org/downloads/release/python-3100/
[9]https://pypi.org/project/cryptography/
[10]https://pypi.org/project/firebase-admin/



```
doors
  7C:DF:A1:E1:5D:D0
    BLE: "7C:DF:A1:E1:5D:D2"
    IP: "85.246.43.60"
    MAC: "7C:DF:A1:E1:5D:D0"
    certificate: "MIIDnDC (...)"
  7C:DF:A1:E1:69:1C
```

Key or ID of Smart Door Lock
BLE Address of Smart Lock
IP Address of Smart Lock
MAC Address of Smart Lock
X.509 certificate of Smart Lock
Key of another Smart Door Lock

Figure 5. Example of information about the each Smart Door Lock

establish a connection to a given smart lock and then send commands and receive responses.

Central Service Server Deployment in AWS. This Service Server consists of a python application running a Flask RESTful API. This application was deployed in a Ubuntu Linux Machine in AWS. The AWS EC2 machine used was a t2.micro running Ubuntu 22.04 LTS. To deploy this server we loosely followed a guide untitled "How To Serve Flask Applications with Gunicorn and Nginx on Ubuntu 22.04" [11] and used the following tools - Gunicorn to run the server and Nginx to act as a front-end reverse proxy. We also used Let's Encrypt Certificate Authority and the tool Certbot to create a SSL/TLS certificate to secure the communication to this server.

4.5 Smart Lock Mobile App

Android Mobile App - Initial Steps. The first step in the Android Client App development was to create a simple version with features like the ability to use the ESP-Touch protocol to configure the Wi-Fi credentials in the ESP32-S2/3 and the functionality to send commands through BLE to the Smart Locks.

The implementation of the ESP-Touch protocol was based on the ESP-Touch Android app [12]. With this feature, the application automatically obtains the SSID and BSSID of the connected Wi-Fi network. Then the user just needs to provide the password for that Wi-Fi network and press connect. The mobile app sends this Wi-Fi credential to the Smart Lock using the ESP-Touch protocol.

BLE communication. We developed a BLE client to establish a BLE communication with the Smart Locks. To do so, we studied the Android BLE communications documentation [2], and the code from multiple BLE apps, like the BLE Gatt Client [13], provided in the android GitHub.

We also created multiple Util files, to help us with the AES and RSA encryption and decryption and even a BLE Message class to parse and manage BLE messages and automatically validate nonces and timestamps sent in the messages.

Login and Account Creation - Firebase Authentication. To manage the user creation and logins in our Mobile App we use the Firebase Authentication service. This service allows us to create accounts using both email and password credentials, and Sign-In with Google. This feature was developed using the official Firebase Authentication library for Android [14].

Integration with Service Server and Database. To access the database, the mobile app needs to connect and communicate with our Central Service Server through the RESTful API, it implemented a simple HTTP Client with multiple helper functions to create an abstraction layer in the communication with the server endpoints and access the database information.

Control the Smart Lock. One of the main purposes of the Mobile App Client is the control the Smart Door Lock, unlocking, locking and changing settings. The Mobile App can control the Smart Lock in two different ways, it can be controlled On Demand, both closely via BLE and remotely via an internet connection, and also using an advanced proximity control feature.

To control the Smart Lock on demand, the Mobile App just needs to send the necessary commands according to the protocol explained in section 3.4.5, using the BLE connection or the HTTP client, as explained in sections 4.5 and 4.5.

The Proximity Control feature takes advantage of the GPS localization of the user's phone and BLE of the Smart Lock to automatically unlock the Smart Lock when the user is near it and lock it when the user distances himself from the Smart Lock.

Invites and Shared Access. The Mobile App can request the creation of the invite, with the **"RNI"** command, using the BLE connection or the HTTP client. After the creation of the invite is done the Mobile App receives the invite code, which is then converted by the app into a QR code and a link, for easy sharing.

Redeem invites and create new Smart Lock access. To redeem an invite the user needs to create a new account or login into an existing one. Then he can use the invite code, the QR code, or the link to redeem the invite and create a new Smart Lock, sending a **POST** request to the RESTfull API */redeem-invite* endpoint.

CA and Validation of the RSA Keys. The Mobile App uses X.509 Certificates signed by our CA to validate and obtain the RSA public key of the Smart Locks. To read and validate the X.509 Certificates it was necessary to configure our CA as a trusted source in the Mobile App. This CA is then

[11] https://www.digitalocean.com/community/tutorials/how-to-serve-flask-applications-with-gunicorn-and-nginx-on-ubuntu-22-04

[12] https://github.com/EspressifApp/EsptouchForAndroid/tree/master/esptouch-v2

[13] https://github.com/android/connectivity-samples/tree/main/BluetoothLeGatt

[14] https://firebase.google.com/docs/auth/android/start

used by the Mobile App to validate the X.509 Certificates stored in the Service Server database.

5 Evaluation

The SDL system was evaluated to verify if it meets the requirements and features previously described. This evaluation was performed in five different levels, 1) an evaluation of the **Hardware Platform and Peripherals**, 2) followed by the evaluation of the **Service Server** and 3) the **Mobile App**. Finally, 4) we evaluated the performance of the **Communication Between Components** and 5) did a final **System Demonstration**.

5.1 Hardware Platform and Peripherals

The ESP32-S2/3 is the main hardware in our SDL system, all the peripherals necessary to control the Smart Lock connect to the ESP32-S2/3. The ESP32-S2/3 is responsible for running the firmware that controls the Smart Lock and communicates with the other components. To develop the firmware for the ESP32-S2/3 we used the ESP-IDF v4.4. This IDF offers us a set of tools used to: a) build and flash code for the ESP32-S2/3; b) and tools to monitor the executions of the firmware. The ESP-IDF communicates with the ESP32-S2/3 using a USB to UART protocol, via the COM ports in Windows.

5.2 Power Optimization

To evaluate the power saved by this approach we measured the power consumption of the ESP32-S3 with the UM24C USB power meter and the Keithley Model 2001 Multimeter[15]. We measured the instant power consumption in multiple scenarios, a) in initialization, b) running (waiting for communication), c) deep-sleep and d) while receiving/decryption communication.

Table 5. Power consumption of the ESP32-S3 in different states of execution

Status	Power Consumption (mA)
Initialization	≈ 75
Running	≈ 100
Deep-sleep	≈ 0.87
Communication	≈ 120

5.3 Service Server

The Service Server is responsible for all the communication with the Firebase database, and also for the remote communication with the Smart Door Lock. For this reason, it is important to test and verify the correctness of its code.

To do this evaluation it was implemented a suite of unit tests that checks the validity of the server code. These unit tests were made using the python library "unittest", and tested each function both with correct and incorrect inputs. The files tested were the a) "rsa_util.py"; the b) "firebase_util.py"; and the main file with the RESTful API c) "app.py".

RSA Util file: The main purpose of this file is to help in the RSA encryption and decryption of messages and also verify the signatures made by the Smart Lock with the private keys. The tests for this file were simple, testing the initialization function, the encrypt and decrypt functions and the sign and verify the signature.

Firebase Util file: The objective of this file is to help with the access and manipulation of the Firebase Database, we tested mainly the functions used to read and write in the database.

Main RESTful API file: The most important file in the python Server is the app.py, which contains the endpoints and the main code for the RESTful API. This file used functions from both the util files mentioned above and given its importance, this file was tested more intensively. We tested all the endpoints available to the client, with multiple tests for each function, inputting both valid and invalid parameters to test all the possible scenarios.

5.4 Mobile App

The Mobile App Client is the user interface to interact with the SDL system. Through the Mobile App, the user can control the Smart Lock and change its settings, he can also create invites to share access to his Smart Locks. To develop the Android App we used the Android Studio IDE. The Android Studio offers us a set of tools used to: a) Build and Run the Mobile App and b) monitor and debug the Mobile App code using the **Logcat** window. The Android Studio communicates with the Android Device for debugging using the Android Debug Bridge (ADB) programming tool, through USB or TCP.

5.5 Communication Performance

Measuring the time it takes to open the Smart Lock is very important to validate if the system is useful for the user, no one will use a SDL system if it took too long to open or close the door.

To evaluate the time performance of these communications we perform 4 different tests, we measured the Round Trip Time (RTT) of one message sent both through BLE and remotely, and we measure the time it took to perform the RUD operation both through BLE and remotely.

We performed each of the aforementioned tests 2000 times and calculated the average time each operation took, as well as the average of the worst 95% and 99% results.

Looking at the results in table 6, it is possible to conclude that the time performance of the communication between

[15]https://www.tek.com/en/products/keithley/digital-multimeter/2001-series

Table 6. Results of time performance of the communication. Units in milliseconds.

RTT Tests	Average	Worst 95%	Worst 99%
Single command (BLE)	77.67	111.62	117.05
RUD operation (BLE)	911.01	1024.15	1067.60
Single command (remote)	335.91	1493.34	2425.30
RUD operation (remote)	1418.94	2586.04	3379.10

the Mobile App and the Smart Lock is good and acceptable for the use case.

6 System Demonstration

For a final evaluation and validation of the system, it is important to show the system working as a whole, to check if the different components interact as intended. For that reason, we did a video demonstration of the system working in a situation near the real world, and we can verify if everything works as intended, and in a user-friendly way.

Demo Video: https://drive.google.com/file/d/1DsX27aJCN70dJIFdr-0tSk9-tjE1fzTj/view

Although this video shows multiple of the system features, it doesn't show all the developed features, for that reason we included a link to a folder with multiple small video demos of each feature, filmed during the development process of the system.

Video Repository: https://drive.google.com/drive/folders/1dkLh4pij-du4HPkIwg8WJwYqWif1iakr

7 Conclusions

In this project, we proposed the design and implementation of a proof of concept for a SDL system that could be controlled by a smartphone. We focused on the development of a secure, reliable and efficient system with low power consumption, and we set ourselves to solve some of the problems of the already existing smart lock systems.

At the end of this project, we successfully created a proof of concept for a SDL system with all the necessary firmware and software for a real-world working system but using an RGB LED to emulate the state of the door, without integrating the system into a real door or deadlock. The developed system could be classified with a level 4 on the Technology Readiness Level (TRL) scale which corresponds to a technology validated in lab, and it would be easily upgraded to a level 5, a technology validated in relevant environment.

With this thesis, we proved the possibility to create a SDL system with a rich feature set, using small and low-power embedded systems and low-power communication technologies as BLE.

8 Future Work

There are multiple possible paths of future work to follow from this project. The most obvious one is to continue the current development of this system, integrating the system with a mechanism to electronically control a door, further develop the Mobile client App, or try to streamline the process to deploy the system and install a smart lock.

Another valid approach for future work would be to experiment with other embedded systems and communication technologies. We used the ESP32-S2/S3 and BLE for this project, however, there are multiple other wireless technologies to experiment with, like Z-Wave, Zigbee, NFC, and multiple others. It would be interesting to test other combinations of embedded systems and communication technologies and evaluate if would be possible to create a more efficient system or a better user experience.

References

[1] Espressif. 2022. *ESP32-S2 Technical Reference Manual.* https://www.espressif.com/sites/default/files/documentation/esp32-s2_technical_reference_manual_en.pdf

[2] Google. 2021. Bluetooth Low Energy - Android Developers. https://developer.android.com/guide/topics/connectivity/bluetooth/ble-overview Accessed 17-Aug-2022.

[3] Norlezah Hashim, N.F.A.M. Azmi, Fakrulradzi Idris, and Norain Rahim. 2016. Smartphone activated door lock using WiFi. *ARPN Journal of Engineering and Applied Sciences* 11 (3 2016), 3309–3312. http://www.arpnjournals.org/jeas/research_papers/rp_2016/jeas_0316_3803.pdf

[4] Philip S. Lee. 1991. Smart card access control system. https://patents.google.com/patent/US5204663A

[5] Ricky Martin. 1977. Electronic combination door lock with dead bolt sensing means. https://patents.google.com/patent/US4148092A

[6] Karthik Patil, Niteen Vittalkar, Pavan Hiremath, and Manoj Murthy. 2020. Smart Door Locking System using IoT. *International Journal of Engineering and Technology* 7 (5 2020), 2395–0056. https://www.researchgate.net/publication/341508373

[7] Rozeha A. Rashid, Nur Hija Mahalin, Mohd Adib Sarijari, and Ahmad Aizuddin Abdul Aziz. 2008. Security system using biometric technology: Design and implementation of voice recognition system (VRS). *Proceedings of the International Conference on Computer and Communication Engineering 2008, ICCCE08: Global Links for Human Development* (2008), 898–902. https://doi.org/10.1109/ICCCE.2008.4580735

[8] Mrutyunjaya Sahani, Chiranjiv Nanda, Abhijeet Kumar Sahu, and Biswajeet Pattnaik. 2015. Web-based online embedded door access control and home security system based on face recognition. *IEEE International Conference on Circuit, Power and Computing Technologies, ICCPCT 2015* (7 2015). https://doi.org/10.1109/ICCPCT.2015.7159473

[9] John C. Schmitt and Dale R. Setlak. 1997. Access control system including fingerprint sensor enrollment and associated methods. https://patents.google.com/patent/US5903225A

[10] Shenzhen Jindouyun Electronic Technology Co. 2018. *Ultra Low Power Bluetooth 5.0 BLE Module - User Manual of JDY-23 Slave Bluetooth Module.* https://fcc.report/FCC-ID/2AXM8-JDY-23/4936741.pdf

[11] Jacopo Tosi, Fabrizio Taffoni, Marco Santacatterina, Roberto Sannino, and Domenico Formica. 2017. Performance Evaluation of Bluetooth Low Energy: A Systematic Review. *Sensors* 17 (12 2017), 2898. https://doi.org/10.3390/s17122898