# Smart Lock System

## 1. Introduction

***Project Overview:***

The smart lock system is designed to control door access using a Raspberry Pi 4B as a server and an Android app as the client. Communication between the devices occurs via Bluetooth Low Energy (BLE). The system allows the lock to open based on predefined authentication methods, including automatic pairing, manual password entry, and dynamic password generation via the cloud.

***Purpose of the Document:***

This document outlines the requirements, design, development, testing, and deployment processes for the smart lock system, following the Software Development Life Cycle (SDLC).

## 2. Requirements

### 2.1 Functional Requirements / Configurations

1. The Raspberry Pi (server) and mobile device (client) must pair automatically when within range via BLE.
2. The lock should open automatically when the devices are paired and connected.
3. A hardcoded password must be used to authenticate and open the lock after pairing.
4. The system should allow the password to be retrieved from the cloud, enabling access.
5. The cloud should generate a random password that is valid for a limited time, which can be refreshed periodically.
6. The system should include a visual or auditory indicator (e.g., LED/buzzer) connected to the Raspberry Pi GPIO to show lock status.

### 2.2 Non-Functional Requirements

- *Security:* The system should use secure BLE pairing and password mechanisms.
- *Scalability:* The cloud-based password management should allow multiple users to access the system.
- *Reliability:* The system should have minimal downtime, ensuring consistent communication and lock control.
- *Performance:* BLE communication and lock response time should be under 1 second.
- *Usability:* The mobile app should have an intuitive UI for entering the password and interacting with the lock.

### 2.3 Assumptions and Constraints

- The system assumes stable BLE connectivity within a defined range (e.g., 10 meters).
- The cloud platform used for password management should be reliable and secure.
- The lock system will use basic hardware (LED/buzzer) for testing purposes, but will eventually integrate with a physical lock.

## 3. System Design

### 3.1 High-Level Architecture

- *Client:* Android mobile app, BLE-enabled.
- *Server:* Raspberry Pi 4B, running a BLE service to communicate with the mobile app and control the GPIO pins for lock control.
- *Cloud:* Stores passwords and generates dynamic time-limited passwords.

### 3.2 Components Design

- *Raspberry Pi BLE Server:* Handles pairing, communication, and lock control via GPIO.
- *Mobile App:* Sends pairing requests, user inputs, and interacts with the server via BLE.
- *Cloud Integration:* Manages passwords and synchronizes with the Raspberry Pi when required.

### 3.3 Database Design

- The cloud service may use a database to store and manage dynamic passwords for authorized users, including expiration timestamps.

### 3.4 Interface Design

*Mobile App UI:*

- Pairing screen: Shows BLE pairing status.
- Password entry screen/Tab: Allows the user to enter a password.
- Lock status: Displays whether the lock is open or closed.

## 4. Development

### 4.1 Development Approach

Following an incremental development model. Starting with the basic pairing mechanism, followed by password authentication, and finally cloud integration.

### 4.2 Tools and Technologies

- *Raspberry Pi 4B:* Python for server-side BLE and GPIO control.
- *Android:* Android Studio for mobile app development.
- *Cloud:* ThingSpeak, Firebase, or similar for password storage and generation.

### 4.3 Code Versioning and Management

Use GitHub or GitLab for version control, ensuring all changes are documented and tracked during development.

## 5. Use Cases

### 5.1 Use Case 1: Automatic Pairing and Lock Control

- Actor: User with a mobile device.
- Pre-condition: The mobile device is in range of the Raspberry Pi.
- Main Scenario:
  - The mobile device automatically pairs with the Raspberry Pi.
  - The Raspberry Pi detects the pairing and unlocks the door.
  - The LED/buzzer indicates the lock status (open).
- Post-condition: The door is unlocked.

### 5.2 Use Case 2: Hardcoded Password Authentication

- Actor: User with a mobile device.
- Pre-condition: Devices are paired, but the lock is still closed.
- Main Scenario:
  - The user enters a hardcoded password into the mobile app.
  - The app sends the password to the Raspberry Pi via BLE.
  - If the password matches, the lock opens, and the status indicator (LED/buzzer) activates.
- Post-condition: The door is unlocked after password authentication.

### 5.3 Use Case 3: Cloud Password Authentication (Static)

- Actor: User with a mobile device.
- Pre-condition: Devices are paired.
- Main Scenario:
  - The mobile app retrieves the password from the cloud.
  - The user enters the retrieved password into the app.
  - The Raspberry Pi verifies the password and unlocks the door.
- Post-condition: The door is unlocked after password verification from the cloud.

### 5.4 Use Case 4: Dynamic Cloud Password

- Actor: User with a mobile device.
- Pre-condition: Devices are paired.

- Main Scenario:
  - The mobile app retrieves a dynamic, time-limited password from the cloud.
  - The user enters the password before it expires.
  - The Raspberry Pi verifies the password and unlocks the door.
- Post-condition: The door is unlocked, and the cloud refreshes the password periodically.

# 6. Testing

## 6.1 Test Plan

- *Unit testing* for each function: pairing, password validation, BLE communication.
- *Integration testing:* Mobile app to Raspberry Pi and cloud synchronization.

## 6.2 Test Cases

- *Test Pairing:* Verify that the mobile device pairs with the Raspberry Pi within range.
- *Test Password Authentication:* Verify that the lock opens when the correct password is entered.
- *Test Cloud Password Update:* Verify that passwords stored in the cloud are retrieved correctly.

## 6.3 Bug Tracking and Resolution

- Use a bug tracking tool like Jira or Trello to manage issues found during testing.

# 7. Deployment

## 7.1 Deployment Plan

- Raspberry Pi should be set up with required software (BLE libraries, Python environment).
- Mobile app for ease of access.

## 8. Maintenance

- *Post-Deployment Monitoring:* Ensure the system is operational, monitor BLE connectivity and cloud synchronization.
- *Bug Fixes and Enhancements:* Regular updates to improve security and performance.
- *User Feedback:* Gather feedback from users to refine the app and server functions.

## 9. Conclusion

The smart lock system provides a modern, secure solution for door access control using Raspberry Pi and BLE-enabled mobile devices. Future enhancements could include integration with more advanced hardware locks, multi-user management, and additional security features like biometric authentication.