

# Breaking a Bluetooth Smart Lock!

We've got a snazzy little smart lock powered by a tiny computer. This smart lock is designed to be opened or closed securely using Bluetooth. However, the development team might have missed some problems during development. Now, it's your job as a quality assurance (QA) engineer to find problems in the Smart Lock's Bluetooth communication software. **Can you imagine how bad it would be for the users if the Smart Lock simply allowed anyone to open a door?** But first, let's dive into the product prototype given to you!

## Testing Target: Wireless Smart Lock System



The smart lock has a brain (ESP32-C6) and some muscles (the servo motor that locks and unlocks the door mechanism). You can talk to the brain using your phone (or Python script) via Bluetooth. The smart lock understands only a few simple commands:

- 1) Authenticate: Tells the lock who you are and enables further commands.
- 2) Open: Opens the lock.
- 3) Close: Closes the lock.

*There might also be some secret, undocumented commands lurking, which some lazy developer forgot to remove....*






## Quickest Start!!

Let's understand how to use the smart lock. As a QA tester, you should initially think like a user!

1. Connect the USB-C from the Smart Lock prototype to the computer. If the thing even works you should see something fancy on the small display screen as shown below:



2. Great, the software team at least made sure the screen works! Now let's try to interact with the smart lock. Wait! We don't even have a mobile app yet? Fine, then let's use some python code!
  - Download the file "**smart-lock-ctf-2025.zip**" from eDimension and extract it.
3. Go to the extracted folder and open a terminal there. **Feel free to use VSCode  here.**
  - For **Windows**: Right-click the extracted folder and choose  Open in Windows Terminal
  - For **MacOS/Linux**: Open the Terminal app  and "cd" to the extracted folder.
4. Let's install and run the software:

- For **Windows**: Run the following commands:


```
.\install.bat  
.\run.bat --gui
```

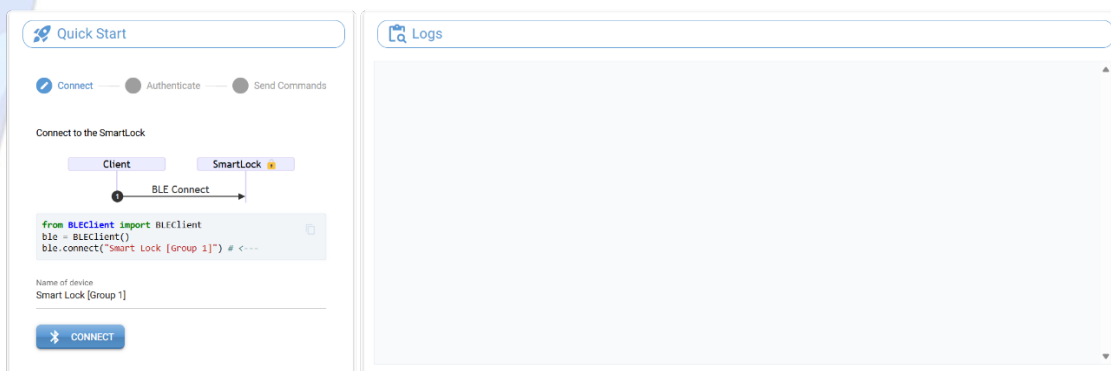
- For **MacOS/Linux**: Run the following commands:

```
bash install.sh  
bash run.sh --gui
```

 **WARNING**  
Do not use folder names with space

5. The browser should open and display a user interface. Follow the instructions shown on the page and keep the smart lock connected to the computer. **On MacOS, you should accept permissions.**

 Wireless Smart Lock



6. Stop the previous command (CTRL+C) and run the benign example (**without --gui**): `.\run.bat`

## 🚀 Your Weapon of Choice: Wireless Fuzzing!!!!!!

Send plain weird stuff when communicating with the smart lock: **Get creative!** Your goal is to discover vulnerabilities by sending different Bluetooth commands, including but not limited to:

- Valid commands: See if sending them in unusual orders breaks things.
- Unknown commands: Try sending bytes the lock isn't expecting.
- Repeated commands: Can you overwhelm the lock with too many requests?

## 🔍 Getting Feedback: Your Clues!

To help you understand what's going on inside the lock and improve your fuzzing strategy, the smart lock provides feedback through a couple of channels:

- **Serial Port:** Think of this as the lock's internal monologue. It will spit out detailed information about the commands it receives, its internal state, and any errors it encounters. Connect to the serial port of the ESP32 board to see this output. Example serial output:

```
[Bluetooth] Received command: 0x00 (Authenticate)
[Auth] Received authentication data: ...
[Auth] Authentication successful
[State] Device state: Authenticated
[Bluetooth] Received command: 0x01 (Open)
[State] Opening the lock mechanism
[Error] Code: 0xSDNSLK
```

- **Bluetooth Responses:** Spend some time to understand the responses sent by the smart lock. You can use such responses to detect weird behaviour from the software. “Weird” here is any response that deviates from the protocol's technical specifications.  
(The technical manual is attached at the end of this document).
- **Think Like a Hacker (the good kind!):** Your mission is to be a digital detective. By carefully analyzing the feedback from the serial port and Bluetooth responses, you can understand how your fuzzing affects the lock. This will help you refine your attacks and uncover hidden vulnerabilities.

## 🛠 Examples of What You Might Find

- Sending a specific unknown command makes the lock stop working (timeout in connecting).
- Sending commands in a particular order bypasses the authentication.
- Sending too many commands crashes the lock.

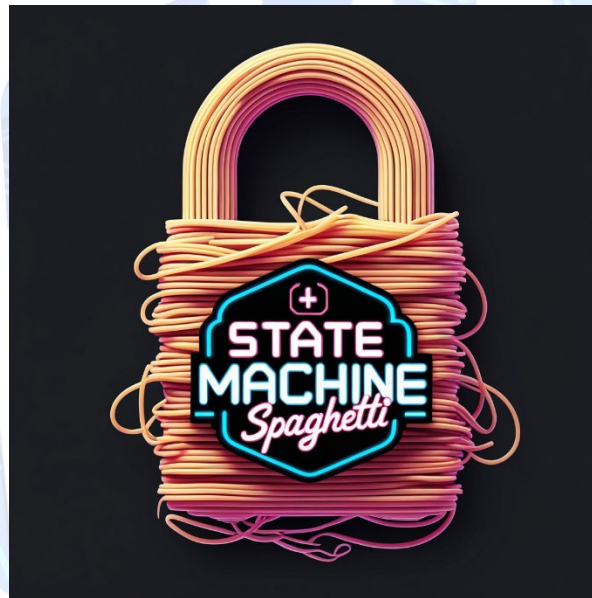
## 📁 Capture-The-Flag (CTF) Cards

As you find vulnerabilities identifiable by **Error codes (flags) via the serial port**, you will unlock illustrative cards that will show lore (more information) about each vulnerability, particularly, how such a mess of vulnerabilities turned out to reach the product's code base. **After getting the error codes, go to <https://asset.sutd.org> and submit the error code with your SUTD email.** A live rank of flags captured by each team will be shown on the website!

**Sometimes the error code is shown as **\*\*\*\*\***. In this case, you need to look at the **display screen** as the smart lock might reveal the error code to you.**

Example of CTF card

### The State Machine Spaghetti



**Development Story:** The developer implemented the BLE command handling using a complex state machine. Due to numerous edge cases and insufficient testing, a specific sequence of valid commands can lead the state machine into an invalid or undefined state. In this state, processing the next incoming command leads to accessing invalid memory or calling an invalid function pointer, causing a crash.

**Attack Vector:** Send the [REDACTED] command, followed by [REDACTED] command, then [REDACTED]

**Impact:** The ESP32 crashes when processing the [REDACTED]

**Side effects:** The device becomes unresponsive after the problematic command sequence.

- The LCD screen might freeze or display an intermediate state.
- The BLE connection might drop.

**Recovery:** The device needs to be power-cycled.



# 📄 Technical Protocol Specification: Wireless Smart Lock

**Version:** 1.0  
**Date:** December 6, 2025  
**Target Device:** ESP32-C6 based Smart Lock

## 1. Introduction

This document outlines the communication protocol for interacting with the Bluetooth Smart Lock. The protocol is designed for simplicity and security, utilizing a command-based structure over a Bluetooth Low Energy (BLE) connection. The primary goal is to provide a robust and well-defined interface for controlling the lock's core functions: authentication, opening, and closing.

## 2. Communication Overview

- **Technology:** Bluetooth Low Energy (BLE)
- **Role:** The Smart Lock acts as a BLE Peripheral (Server), and the controlling device (e.g., smartphone, computer) acts as a BLE Central (Client).
- **Data Format:** Little-endian byte order.

## 3. Protocol Commands Structure

The protocol supports three primary commands. Each command sent from the Central to the Peripheral (Smart Lock) follows a specific frame structure:

### 3.1 Authenticate Command

Byte Offset	Description	Data Type	Value/Example
0	Command Code	uint8_t	0x00 (Authenticate)
1-6	Passcode	uint8_t[8]	6-byte passcode (e.g., {0x01, 0x02, ...})

**Total Frame Length:** 7 bytes

### 3.2 Open Command

Byte Offset	Description	Data Type	Value/Example
0	Command Code	uint8_t	0x01 (Open)

**Total Frame Length:** 1 byte

### 3.1.3 Close Command

Byte Offset	Description	Data Type	Value/Example
0	Command Code	uint8_t	0x02 (Close)

**Total Frame Length:** 1 byte

## 3.2. Response Frame Structure

The Peripheral (Smart Lock) will send responses back to the Central. Responses will follow this structure:

Byte Offset	Description	Data Type	Value/Example
0	Response Code	uint8_t	See the Response Codes table below

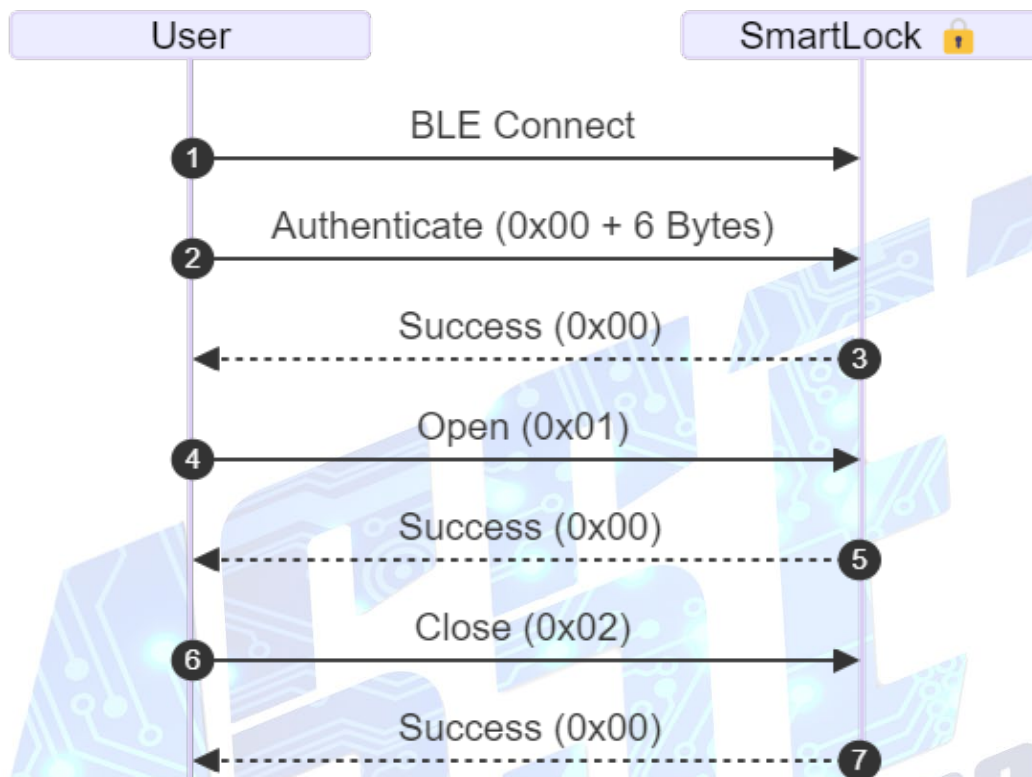
Response Code	Description
0x00	Success
0x01	Authentication Failed
0x02	Invalid Command
0x03	Command Not Allowed (Not Authenticated)
0x04	Lock Error (e.g., mechanical failure)

## 4. Error Handling

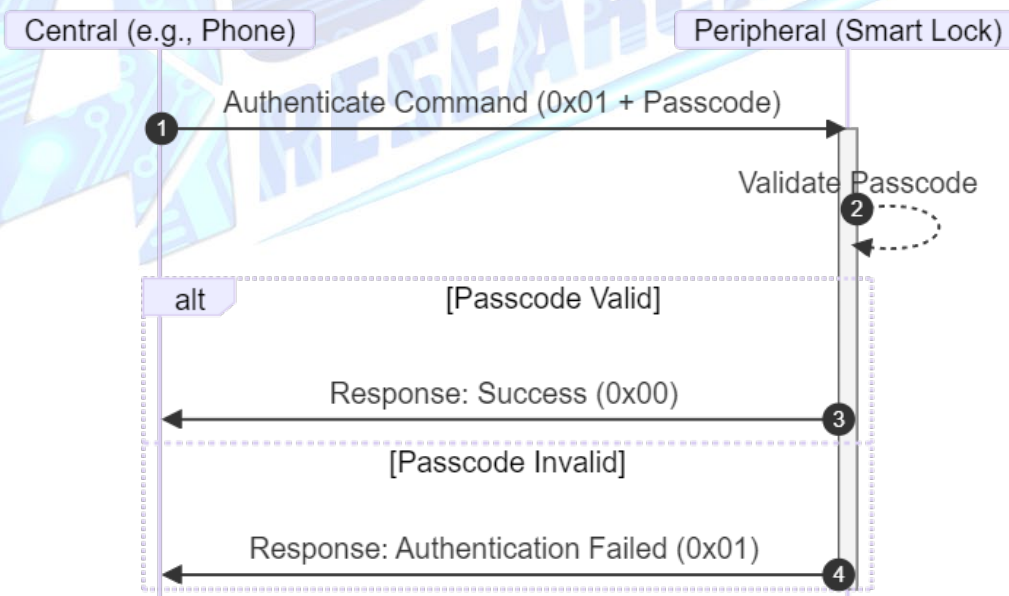
- **Authentication Failure:** If the provided passcode is incorrect, the lock will return an "Authentication Failed" (0x01) response. Multiple failed authentication attempts may trigger a temporary lockout.
- **Invalid Command:** If an unrecognized command code is received, the lock will return an "Invalid Command" (0x02) response.
- **Command Not Allowed:** If an Open or Close command is received before successful authentication, the lock will return a "Command Not Allowed" (0x03) response.
- **Lock Error:** If a mechanical or other internal error occurs during lock operation, the lock will return a "Lock Error" (0x04) response.

## 5. State Machine and Sequence Diagrams

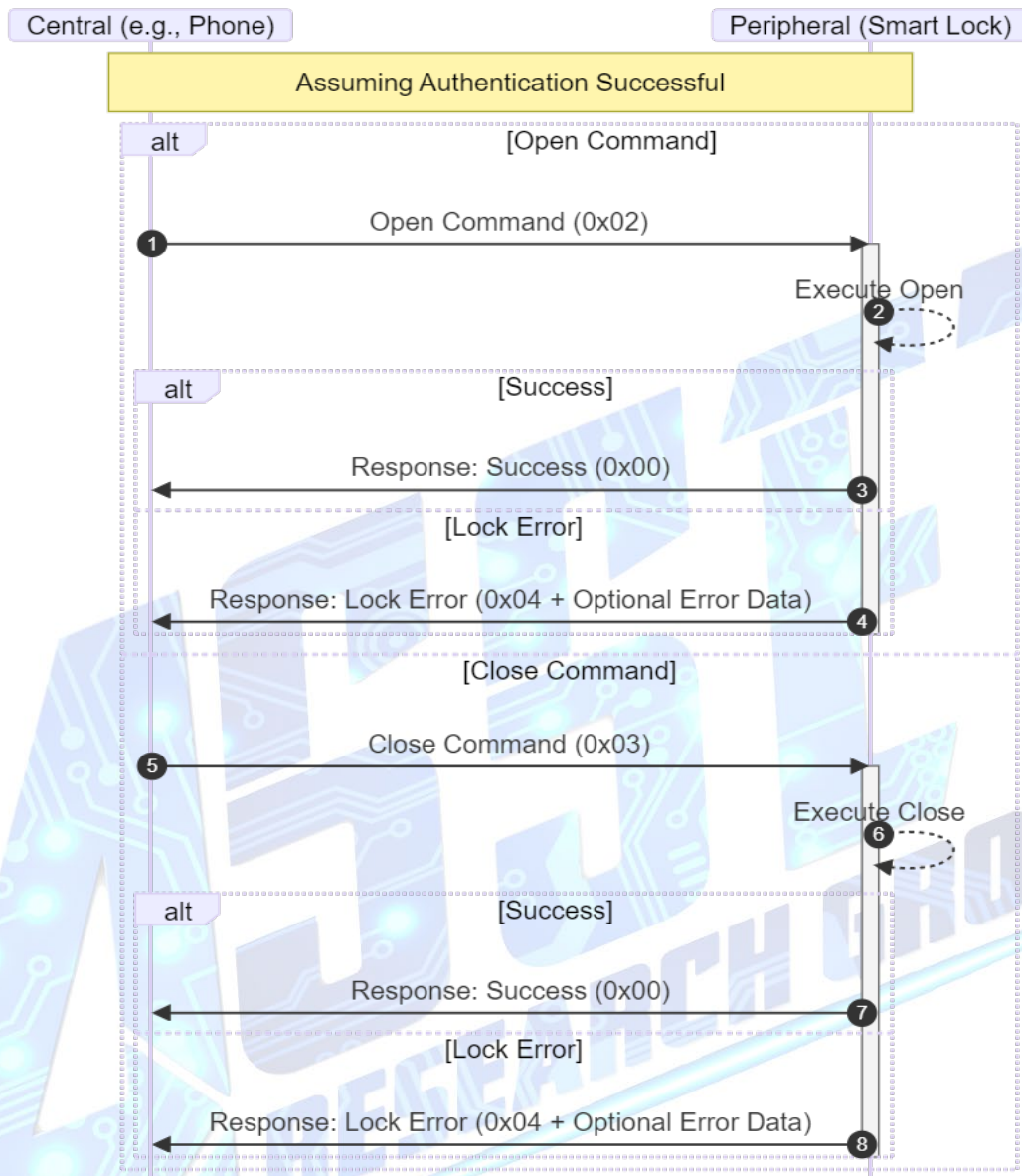
The overall sequence diagram of successful communication with the smart lock is shown below:



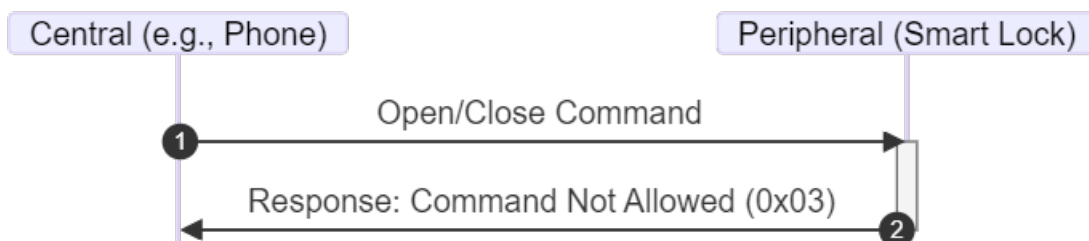
### 5.1. Authentication Sequence Diagram



## 5.2. Open/Close Sequence Diagram (Authenticated State)



## 5.3 Open/Close Sequence Diagram (Not Authenticated State)





## 5.4. Lock State Diagram

