**BlueLock**

Ethan Trovillion

Ethan Warth

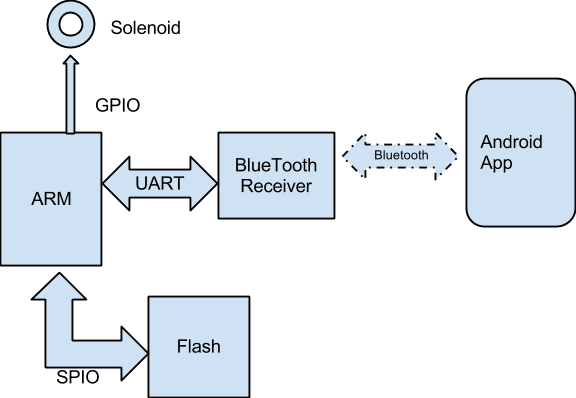
Eric Lachman

Matthew Baker

**Abstract**

We designed a locking mechanism, suitable for a small box, which is controlled via the Bluetooth wireless communication protocol. The design uses an ARM Cortex M0 processor as the microcontroller, which communicates with a Bluetooth transceiver through the UART protocol. A solenoid serves to drive the latch mechanism. For security, we implemented simple password protection. Passwords are hashed and stored persistently using flash. A smartphone application provides the user interface, and serves as the primary controller.

**Overview**



For our ECE 395 project, we set out to create a smartphone controlled lockbox using the Bluetooth protocol as our primary communication medium. The lockbox hardware is primarily composed of a Bluetooth transceiver module, an ARM-based microcontroller, a Flash-based storage unit, and a solenoid to drive a latch mechanism. The microcontroller interfaces with the Bluetooth module using the Universal Asynchronous Receiver/Transmitter (UART) serial protocol, and communicates with the flash unit using the Serial Peripheral Interface (SPI) protocol. The solenoid is driven by the microcontroller’s General Purpose Input/Output (GPIO) signals, and is isolated from the control unit using an H-bridge to prevent the ARM components from being damaged by the solenoid’s high current draw.

Our device’s command protocol is string based; commands are human-readable text, terminated by a carriage-return-line-feed (CRLF)(“\r\n”) sequence, in deference to the serial terminals we used for testing and debugging the system. Our command set is very basic, consisting of “open,” “close,” and “set.” The open and close commands disengage or engage the solenoid, respectively. Set allows the user to specify a password which must then be provided alongside a command, separated by a space (i.e., given password “p@55w0rd”, to open the box one would send “open p@55w0rd\r\n”). In debugging mode, an additional “test” command is available and will reset this password.

To change the password once a password has already been set requires a two line instruction. The first line is set followed by a space and the old password (i.e., “set p@55w0rd\r\n”), and the second is the new password on a line by itself (“n3wP@55w0rd\r\n”). If the second line is empty aside from the CRLF (“\r\n”), the password is unset, and the command parsing behavior reverts to its original state.

**Device Description**

**Microcontroller**

The microcontroller we used in this project is an LPC1114 manufactured by NXP Semiconductors. The microcontroller contains an ARM Cortex M0 processor as well as a number of peripheral devices and interfaces, including UART and SPI. This component is the unifying element for the lockbox hardware. It communicates with the Bluetooth receiver through a UART interface, the Flash chip through an SPI interface, and the solenoid through a generic output signal. It also runs the logic that implements the protocol of the lockbox itself. The code that is loaded onto the chip is written in C, and this C code is broken up into several files in order to help modularize and abstract the code at various levels. The main.c file of the ARM source code contains the code that initializes the processor and implements the lockbox’s communication protocol. The hasher.c/h files encapsulate the SHA code that is used to implement hashing and automate the write through based storage of password hashes onto the Flash chip. The other files each implement some driver, protocol, or function that is used by the ARM; they are described throughout this document under the descriptions of the components that use the interface supplied by each file.

**UART**

The UART is an asynchronous serial communications protocol. In our project, the ARM processor is connected to a Bluetooth receiver through UART. The uart.c/h C files in the ARM source code implement the software drivers necessary to use the ARM’s UART hardware correctly. These files include an interrupt handler that queues input received from the UART, a blocking read function that returns a buffered line of input, and a write function that sends a string through the UART interface byte by byte.

**SPI**

Serial Peripheral Interface (occasionally referred to as SPIO or SSP) is a synchronous, full-duplex serial protocol designed for bus communications. Devices that use this protocol are designated as masters or slaves; one master can drive multiple slave devices connected to a single bus. There are four primary signals involved in SPI: Master-Out-Slave-In (MOSI) and Master-In-Slave-Out (MISO) are the data transmission lines; CLK is the master-driven clock signal; and SEL designates which slave devices are active on the bus. Communication begins when the master pulls SEL low and begins driving the clock. Each clock period data is moved onto the data lines on the rising or falling edge, and then latched on the complementing edge; which clock edges propagate and capture data is a configurable part of the protocol. When the master has determined the communication is finished, the clock stops and SEL is raised high.

**Bluetooth**

The EGBT-046S Bluetooth Module used in our design contains all of the hardware necessary to produce and interpret valid Bluetooth transmissions. The Bluetooth receiver implements a UART asynchronous serial I/O interface for the sake of allowing other hardware devices to communicate through its Bluetooth antenna. The serial interface abstracts the technical details of the Bluetooth protocol; it is not necessary for a device using the Bluetooth receiver to convert to or from any specific format in order to send or receive data from the device. For that reason, there is no C source that corresponds directly to the Bluetooth receiver; the Bluetooth’s data is simply read as data from a generic UART device.

**Storage**

For persistent storage of password hashes, we turned to a Microchip 25AA020A Serial Electrically Erasable Programmable Read Only Memory (EEPROM) chip. This device communicates using the SPI protocol, which is a synchronous serial connection with a master driven clock signal. Unlike the Bluetooth receiver, the storage chip requires a specific formatting of the bytes sent over the SPI connection in order to be used as a medium to read or write data. The storage.c/h C files in the ARM source code utilize the SPI interface provided by the spio/c.h files to implement the communication protocol that the EEPROM requires and provide a high level interface for reading and writing to the flash to the rest of the code.

**Android Description**

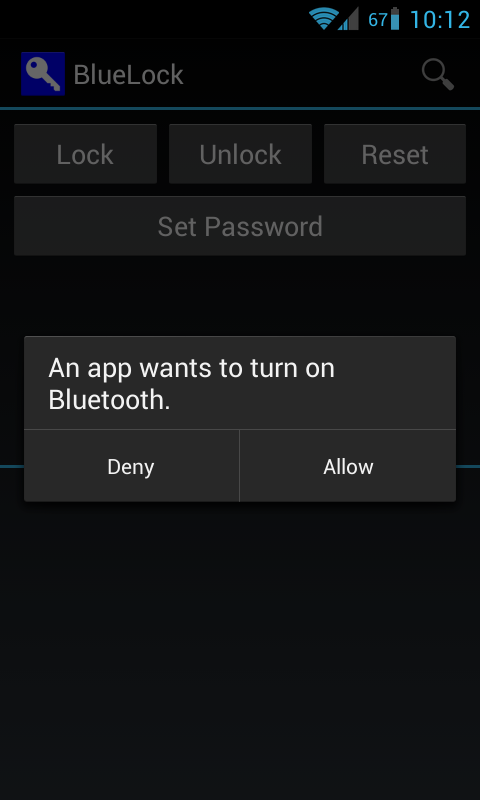
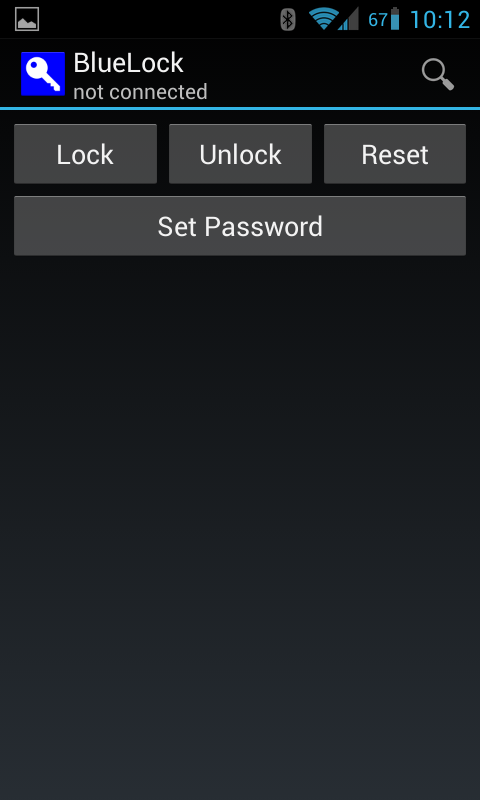
**Application Overview**

The main purpose of the Android app was to act as the user interface to operate our lock. Therefore it has to have ability to open and close the lock as well as having all of the necessary functionality to set and use the password. We decided to focus on the core functionality of the app so that the user is presented with a straight forward design.

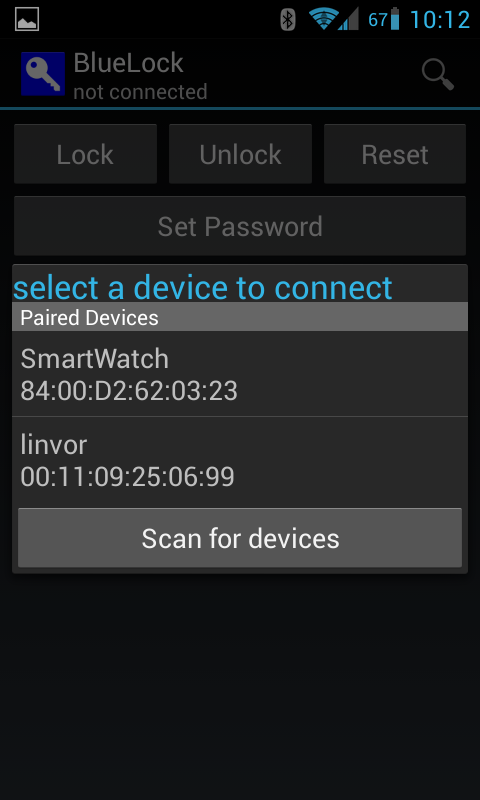
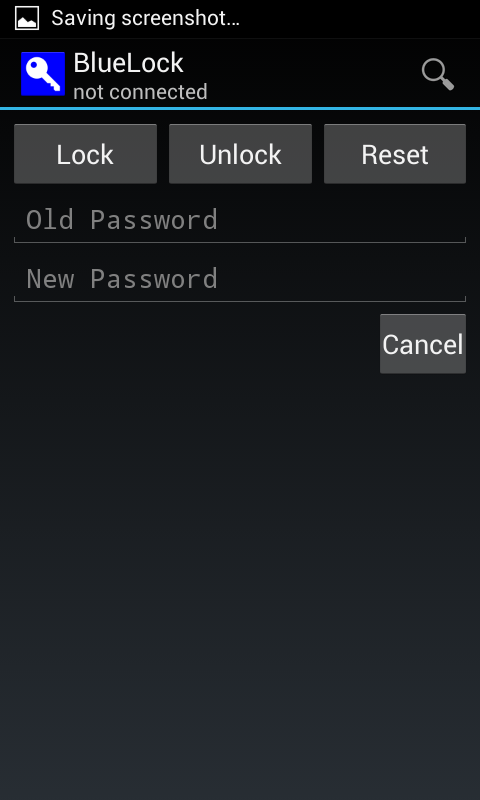
**Use**

When starting the app, it requests that user enables bluetooth if bluetooth is disabled. Once bluetooth is enabled the user must select the lock from the list of available devices, by clicking search button in top right corner. If the lock has never been connected before the user has to input pin 1234 to pair with lock. On any attempt to lock or unlock the device will provide feedback whether the password or accepted or not. If the password was rejected the app will inform the user and clear the internally cached password, requiring the user to enter different password. Once a correct password is entered the app will successfully operate the lock. In order to change passwords the user must click the set password button and enter in the correct old password, if there is one, and the desired new one. If the old password was correct the app cache the password allowing the user to operate the lock.

Enabling Bluetooth Main screen (not connected)

Connecting to device (linvor) Changing the password

**BluetoothActivity.java**

BluetoothActivity is main code for the application. It contains Android life cycle events, user interface controls, and interacts with the service. On launch this is called to populate the interface, load the saved password, and wait for interaction. Once the user initiates the connection to the lock the BluetoothActivity initializes the BluetoothService. On either leaving or closing the app the password is saved to SharedPreferences cache. Also the activity is responsible for handling all interaction with the buttons and views. The activity contains a handler which reacts to user interactions. These changes are converted into string or state changes which are forwarded to the service for handling. Similarly, responses from the service, such as received messages, are passed to the activity to interact with the user.

**BluetoothService.java**

BluetoothService contains the code responsible for handling the bluetooth communication. When called from BluetoothActivity the service launches the necessary threads to connect and communicate to the lock through bluetooth. It uses the standard Android synchronized and secure protocol for the bluetooth connection. Once connected the service provides the functions to read data coming in from the lock and to send write data from the activity to the lock. The service is also responsible for managing connection, meaning that failed or lost connections are properly handled.

**Going Forward**

A central concern for anyone who continues with this project is power. While we did not get past using the on-bench power supply, we discussed using batteries, USB, or some combination thereof to power a completed lockbox. It’s worth mentioning that if a battery only design is chosen, extra precautions must be taken to avoid running out of battery power (a locked Bluetooth lockbox that ran out of power would be highly inconvenient, to say the least). Such precautions might include optimizing components for low power consumption and/or charge measurement mechanisms combined with low power notifications through the Bluetooth interface.

Another improvement that could be made in the future is more robust error handling. Our design currently only checks for error conditions that are critical to the device’s functionality or caused issues when debugging. Adding fault and interrupt handlers to keep the ARM and its peripherals out of illegal states would be a great improvement.

Although we created a circuit diagram and board layout through the Eagle software, the project file may require some additional work before it is ready to be used to print a circuit. There was no component listed for our solenoid (or for any solenoids, for that matter), and it wasn’t clear how such a component would be integrated into a circuit; for that reason, the solenoid of our project is diagrammed as an inductor instead. Because the H bridge component we used was one that is custom made for use by UIUC’s ECE department (it is most frequently used for ECE 110), we had to create our own component in Eagle to model it. In addition, we made our own Eagle component to model the Bluetooth receiver since we didn’t have an easily accessible library that included such a component. The size of these components was acquired through measurement, but they may nevertheless be off slightly. Last but not least, because none of us have had any experience using Eagle before, it is possible that some parts of the Eagle project may be incorrectly configured or otherwise misused.

In addition to the improvements that can be made on the lock side of the project, there are many improvements that can be made with the phone application. Currently we have a fully functional Android app, but there are other features that can be added. Some features we wanted to add include password sharing / guest access, additional security methods (facial recognition, patterns, etc), handling multiple locks, and expanded functionality. Either direct, short-range communication or a web service would be required to implement the password sharing. Additionally we can port the app to other mobile OS’s to reach a wider user base. There are always ways to improve upon our project but in the end it is important to make sure that the app meets its primary purpose of being a secure user interface to the lock.

A copy of our source code is stored on GitHub at <https://github.com/miralexan/ECE395SP13BlueLocker>. Interested parties may download our files from there to continue work on this project.