BlueLock: A Secure Bluetooth Operated Padlock

A Senior Project

presented to

the Faculty of the Computer Engineering Department

California Polytechnic State University, San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science

by

Trever McKee

June 5, 2013

© 2013 Trever McKee
Introduction

In the last decade smartphones have infiltrated every aspect of life, and many everyday tasks have been eliminated by their use including, but not limited to: note taking, surfing the web, and even car alarm systems. Smartphone technology has allowed easy overhauling of countless other technologies as well. One specific item that has not been improved with technology is portable locks. The technology behind traditional locks and padlocks has been around for two centuries, and the earliest known use of the lock dates back to 4000 B.C in Egypt (Rathjen 79). With thousands of years to iterate the design, the mechanics of locks are simple, yet effective. Current portable locks are called pin-tumbler locks because of the mechanics the device uses. Pins are cut in different lengths and once each individual pin is raised to the proper height the lock can open. Unfortunately, the pin tumbler mechanism lends these locks to a very simple, yet proven attack: lock picking. Lock picking allows for a skilled adversary to open locks they should not have access to by exploiting mechanical defects of the lock. These defects allow for a malevolent user to reduce the number of locking mechanisms (or pins) from four, five, or six down to one at a time. Since the user only needs to worry about 1 mechanism at a time, it becomes incredibly easy to defeat the lock.

Almost every smartphone created includes a modern networking standard called Bluetooth. Bluetooth is a WPAN standard used for creating small Wireless Personal Area Networks that allows for communication between devices in relatively close proximity to one another (“Bluetooth Basics”). Bluetooth devices are everywhere and are being used everyday. The most common Bluetooth devices that users interact with are hands-free cellular devices and Bluetooth headsets/speaker systems.

The purpose of this project is to combine the technologies in smartphones and locks to create a secure padlock that is opened by a phone application running on Android OS through the Bluetooth communication standard. This included developing a security protocol to make the device impervious to Bluetooth replay attacks and wireless sniffing. The security protocol was developed and is runnable on both the Arduino Uno (the microcontroller used for prototype development) and the Android phone, the Motorola Droid Razr Maxx. The device is also battery powered and self-sustaining creating a low maintenance locking solution.

Background

Bluetooth Communication

There are many pieces of background information needed to understand the project implementation. The first is the Bluetooth communication specifications. The Bluetooth protocol operates in the license free band from 2.402 GHz to 2.480 GHz. This license free band is divided into 79 different channels of 1MHz, and the device channel changes 1600 times per second (“Bluetooth Basics”). This frequency hopping allows Bluetooth devices to avoid other devices that are within range, and creates a small amount of security for the connection. The band-jumping scheme is determined by a multitude of factors including the device address, timestamps of connection, and a shared key. These values are shared
between the master and slave devices upon the initial connection. The channel-hopping scheme is then calculated for both master and slave and from this point forward all communication happens during channel hopping. Bluetooth communication chips can be purchased at any electronics vendor, and all chips must conform to the Bluetooth standard outlined by the Bluetooth Special Interest Group or SIG.

The Arduino Uno is a development board used in many classes at California Polytechnic State University. The Arduino Uno is based on the ATmega328 chipset and includes access to system interrupts, timers, registers, and other low level interfaces. The Arduino Uno operates at 5V and has a clock rate of 16 Mega Hertz. The board can be ran at other voltages, but these options do not apply to this project. The Arduino Uno also has 32 Kilo Bytes of RAM and has 14 digital I/O pins (“ArduinoBoardUno”). The programs are written in either C or C++, and there is a large, highly supported framework that is used for development on the board.

There are two sleep states on the Arduino Uno; the first is a very low power (in the Pico or $10^{12}$ Watt range) that can only be awakened by an external pin interrupt. Only two digital pins on the Arduino Uno (digital pins 0 and 1) have the capability to be used an external pin interrupt pin change or pin high (“ArduinoBoardUno”). The second sleep cycle consumes more power, but can be controlled using any of the interrupts on the board (timer interrupts, etc.). For this project the low power sleep cycles were used to minimize battery demand, and data transfer from the Bluetooth chip triggers the external interrupt.

Common Attack Vectors for Bluetooth Devices

There are many attack vectors for Bluetooth devices. The first and most common is a replay attack of the data that was sent initially. A malicious user can use listening devices to “sniff” traffic sent across the frequencies that Bluetooth operates on. An attacker then broadcasts the same data over the same frequencies. If the device is not secured then this attack vector is exploited allowing for the device to be compromised. A non-deterministic value in the messages is the best way to prevent replay attacks. Many times this non-deterministic value is a timestamp or some sort of random number.

The second main attack vector is sniffing Bluetooth frequency traffic and decoding the messages that are sent. There is a small amount of inherent security contained within the frequency-hopping scheme of which Bluetooth operation employs, but there are mathematical attacks that can easily determine the hopping scheme. A malicious user could find the addresses of the device then use a mathematical attack to determine the shared key. Knowing the shared key would allow for the user to predict the frequency-hopping scheme and once the hopping scheme is determined it is very easy for the attacker to read the messages that are being sent. Then the attacker can duplicate these messages and exploit the device. Cryptographic systems can lessen the impact of this attack vector.

AES Encryption

AES stands for Advanced Encryption Standard. AES has been adopted by the United States government for both security and privacy, and is used in many other applications worldwide. According to Schwartz the United States adopted AES encryption in 2001 after previously
using DES. Schwartz continues by saying that “Cryptographers have constructed computer systems that could quickly crack DES messages.” The search for a new encryption algorithm began in January 1997 and in October of 2000 after putting each submission through grueling tests to find the best candidate AES encryption won out (Schwartz). The United States government determined that while all submissions were secure enough for unclassified information, only AES was secure enough for classified information.

AES is based on the Rijndel cipher and includes key rotations, shifts, and many other operations that prevent determining the key from the encrypted message. The starting message is called the plaintext, and the encrypted message is called the ciphertext. There are four operations that occur on the plaintext, and this process is repeated 10 times. The final product of the Rijndel cipher creates a ciphertext that is unique to each key and message combination. There are currently mathematical attacks against AES, but with a proper implementation the mathematical attacks do not make brute force feasible (Ou).

**Public- Key Encryption**

The fundamental idea of public-key encryption is that it is an asymmetric system, which means that two different keys are given to the sender and the receiver. “Anyone can encrypt a message using the public key, but only the owner of the private key will be able to read it” (“SSL/TLS Strong Encryption: An Introduction”). The public key can be given to anyone who will be sending messages to the receiver. The private key and public key are linked mathematically, but the private key can’t be derived from the public key. This creates a secure message-passing scheme so that only the person intended to receive the message is able to decrypt the ciphertext.

These two schemes are both very secure, but during the design of a low-power lock, everything must be taken into account including clock cycles and running time of each solution. The current written libraries for both of these encryption schemes are stable and well maintained. For this project only a single block of message is being sent between the Android phone and the Bluetooth device. The RSA (public/private key encryption) library takes 500 milliseconds for encryption with 20 seconds decryption per block (Sethi). The AES library takes about 60 milliseconds per block for both encryption and decryption. This is a huge time difference and would greatly affect the time to manipulate the lock along with power consumption of the device. These considerations make AES the prime choice for encryption for this project.

Now, it is necessary to address how to make each message different. Without an initialization vector, timestamp, or similar method messages can be replayed to the device and compromise it. The first solution provided was to include a timestamp in the message. The problem is that the Arduino Uno does not support keeping time during sleep cycles, and the time is reset each time the board resets (“ArduinoBoardUno”). There are hardware add-ons that can be used for keeping track of time, but they are expensive and require constant electrical power to run both of these characteristics are undesirable in a portable lock.

The other commonly used system is challenge response authentication. The challenge response authentication verifies identity by requiring a proper response to be issued from a
given challenge. A very common challenge is requiring that the response be the one-complement of a given random number. This requires that the responder can both calculate the response and decrypt the challenge message. Challenge-response authentication also prevents the need for a cryptographic nonce (non-repeating challenge), which is a fairly complicated implementation on a low power development board. Although a cryptographic nonce is not necessary, there still needs to be a source for a random challenge.

**Android Application Development**

Android application development is done in Java using the Android development toolkit provided from Google. There are a few changes from the standard Java 1.6 library, but all changes are small, and mostly necessary for programming on a mobile device. The most important aspect of programming Android devices is to understand the Android Activity lifecycle (Meier). Every different action you take on an Android phone is considered an activity. There are certain times when activities are created, started, paused, and destroyed. In order to make an Android application that functions well it is necessary to understand the conditions that make an activity restart, pause, or destroy and take the proper action in each case.

**Description of Implementation**

**Hardware**

There are six pieces of hardware that were used during the implementation. The servo that was employed for the unlocking motion was a small servo with low power consumption. The operating voltage of the servo is 4.8-6.0 volts, and the servo outputs 1.4kg/cm torque. This torque is enough to complete the mechanical motion to open the lock. The second piece of hardware was the Li-po battery. The battery operates at 3.7V and has a charge capacity of 2000mAH. The battery is connected to the integrated charging/voltage step up circuit. This circuit board takes the 3.7V output from the battery and steps it up to 5V, which is the running voltage for the Arduino Uno and the Bluetooth chip. The charging circuit connected to the device also includes a step up voltage transformer and regulator for the charging input. There are two ways to charge the device; the first is to use a USB cable connected to the charging circuit and the second is the solar panel connected to the lock. The voltage regulator in the charging circuit is required for charging with the solar cell due to voltage and amperage
fluctuations depending on sky conditions. The final piece of hardware is the Bluetooth chip. This chip controls the Bluetooth connection, pairing, data transfer, and sleeping of the Bluetooth device. This is the key hardware component for the device. The Bluetooth chip used was a Bluetooth Mate Silver Pro designed by Sparkfun Industries. The Bluetooth Mate silver is a low power model running at 5 volts with an average power consumption of 25 milliamps.

The Arduino Uno stays in a sleep state that waits for data transfer from the Bluetooth chip to wake; this keeps the power consumption of the development board at minimal levels. The Bluetooth chip is low power and is always on waiting for a connection from an approved device. Once data is transferred from the Bluetooth chip to the Arduino Uno, the Arduino wakes up and reads the message. If the message is the correct message to start an authentication process then the Arduino Uno creates a challenge, which is simply a random number seeded with an analog read of an external unconnected pin. If a hacker had access to the internals of the device this value could be changed to no longer be random, but in the current implementation, access to the device in this way renders all other security useless. An unconnected pin on the Arduino Uno fluctuates quite a bit, and can range from 0 – 4.9 Volts. This voltage is always changing due to outside ambient electromotive forces, outside magnetism, radio communications including Bluetooth, states of connected devices (this includes the solar cell and servo), and current battery charge. The random number (32 bits long) is then appended to the message, and the entire message is encrypted then sent across the Bluetooth connection. The Arduino then awaits the reply message. Once the reply is received it decrypts the message and determines whether the response is correct. If the response is correct the servo is turned on and rotated to open the lock, if the response is incorrect the lock stays closed and the device resumes its sleep cycle. Only users who have access to the correct AES encryption key and device address are allowed to connect preventing denial of service (DOS) for a noisy, or lots of radio signals, area. The channel hopping scheme also prevents denial of service and interference for noisy areas.

Software
The Android application development was designed with user experience in mind. The Android application has two main paths: the storing of addresses and keys, and the opening of the lock. The storing of the addresses and keys is done using a QR barcode scanner. The idea behind this is that a user could buy a lock from their local hardware store and instead of having a combination with their lock there is a QR barcode that can be scanned into the application. This would be a seamless and easy way to contain the necessary information of the AES key and the address of the device (AES key is 16 bytes long, and the Bluetooth address is 32 bytes long). The keys would be encrypted and stored on the provided private database that each Android application has allocated to it. The database for each Android application is private which would prevent other applications from attempting to access the information. The second code path is the opening of the lock.

By operating the interface and selecting any lock from the device list the Android application would begin connecting to the device. Once connection is completed the Android application will then send the necessary message to the hardware to begin the challenge response authentication. The Android application then waits to receive the ciphertext that was encrypted on the hardware. The cipher text is then decoded, the response calculated, and a new message is created which now includes the proper response for authentication. This message is sent to the hardware, and the Arduino app reverts to the home screen waiting for more user input.

**Evaluation**

**Security**

The security design behind this device is similar to many modern security devices currently being used, including devices such as hotel door locks. AES encryption with challenge response authentication prevents replay attacks by changing the challenge during each authentication session. Creating a random number on the hardware for the challenge and requiring that the response be the full 32-bit ones-compliment means that, statistically, for an attacker to guess a response it is required to brute force $2^{32}$ different responses. Each challenge response authentication session has its own challenge that adds another layer of complexity. With encryption built on top of the challenge response authentication, any malicious user attempting to sniff traffic will only have access to the ciphertext (encrypted
message), preventing guessing of the challenge or response. The AES encryption will also prevent derivation of the encryption key due to the non-linearity properties of the Rijndel cipher.

**Power Consumption**

The power consumption of the project was also tested and came in at an acceptable range. During idle states the consumption of the device was, on average, 40 milliamp. A connection and open cycle raised the power consumption to 80 milliamp for about one second. This means that the device could run on the 2000 milliamphours battery for 50 hours, which is over 2 days. The solar cell has an output of 4.94 volts at 94 milliamp that is around .5 watt output. The solar cell could run the device indefinitely with a proper light source, and could reasonably charge the battery during any other condition. There are numerous other ways to lower the power consumption further, but the levels reached were perfectly acceptable for the project specification.

**Conclusion**

The project was successful. The final product is a secure, working, lightweight lock that can be opened from any Android smartphone. The power consumption and recharge rate of the solar panel created a device with an excellent run time, and the application provides a clean user interface. Connecting and unlocking the device is seamless, and takes under a second for the operation to complete. This technology could be marketed in a plethora of ways including adding on to existing systems in houses, padlocks, and even cars. The main point of the system created was a secure way to transfer electronic signals into a form of motion that would unlock a device. This makes the possibilities endless for applications.

There are many directions that this project could take from this point onward. The first, and most important work would be to condense the entire system onto one chip. This would allow for the power consumption to be lowered even more, and perhaps provide seamless integrations with current lock hardware as a drop in replacement module. Another direction is to create web-service that would store the AES keys and addresses of locks that the user is able to open. This would allow for users to share keys with others, and even limit access to certain timeframes or number of open cycles. Another step would be to create the option of password protecting the application so that an unauthorized user that has received access to the phone would not be able to compromise the security of the locking device. The final step would be to explore the possibilities of creating an iPhone application. During this project connection with an iPhone was outside of scope because research found that connecting with an iPhone requires extra hardware to connect via Bluetooth. The process require extra testing to determine whether extra hardware was in fact needed, and whether this would limit the marketability of the product.
Bibliography


