Keeloq FOB Attack Lab

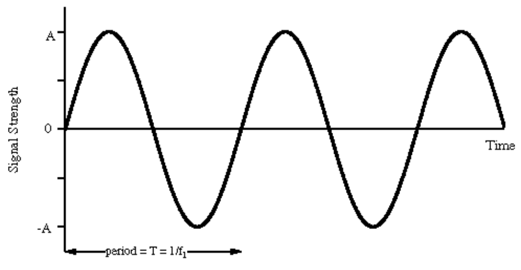
Objectives:

- Introduction to SDR recording
- Basic Signal Characteristics
- Waveform Analysis
- Identification of Rolling Code Signals
- Jamming Signals
- Rolling Code Interception and Replay

Background

Wireless communication is a very complex and deep topic that can encompass many volumes. As a result, this background is going to provide information relevant to the lab so that participants understand what they are investigating and reproducing at a basic level.

With regard to wireless communication, three types of modulation are typically discussed in a basic communications course. In this context, modulation is the process of altering one signal with another to convey information between communicating partners using the air as a transmission medium. The two signals involved are the carrier signal and the information signal. The carrier signal is usually a sinusoidal waveform that operates at a frequency that the sender and receiver must be tuned to in order to exchange information.

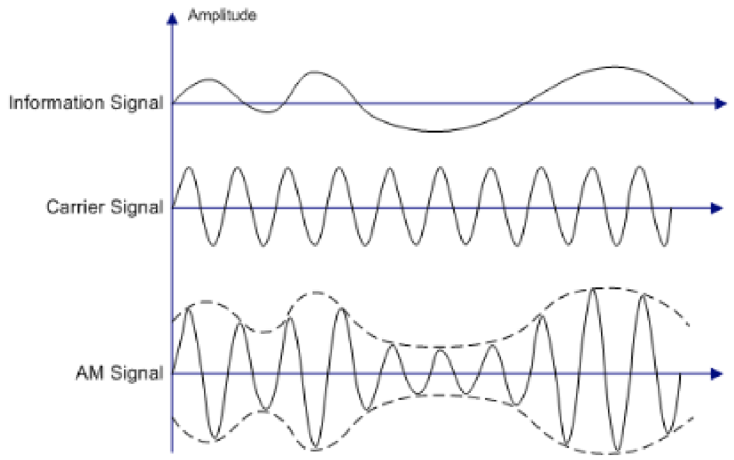


(a) Sine Wave

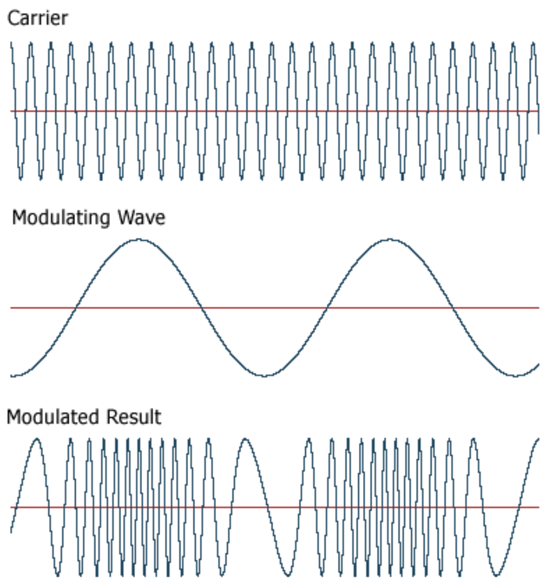
The information signal is used, along with the rules of the modulation technique, to modify the carrier in such a way that the original information signal can be recovered by the receiver of the modulated waveform by demodulating (reversing the modulation process) the received stream. An example information signal could be human speech.

The three basic transmission modulation techniques that are relevant to this lab are amplitude modulation, frequency modulation, and phase modulation. Amplitude and phase modulation should be familiar from the broadcast radio system. AM and FM radio are one application of these modulation techniques.

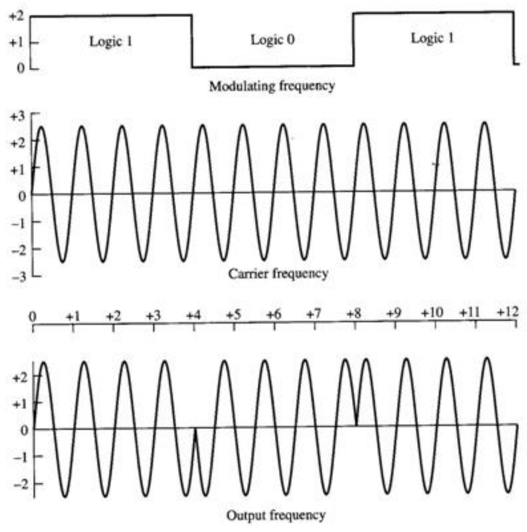
Amplitude modulation uses the information signal to modify the amplitude of the carrier waveform. Graphing the resulting waveform, the height of the peaks mimic the behavior of the input information waveform. A depiction of this behavior can be seen below.



In contrast, frequency modulation uses the information signal to modify the frequency of the carrier waveform. As a result, the modulated waveform appears to compress and decompress based on the input information waveform. This behavior can be seen in the graph below.

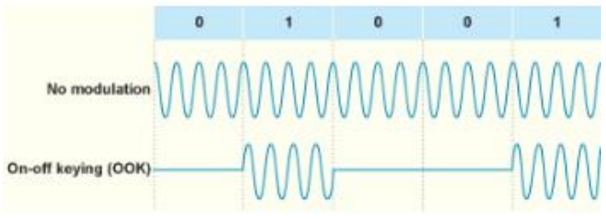


The final transmission modulation technique is phase modulation. Many variants of phase modulation exist. However, they all perform the same operation to transmit digital information. The phase of the carrier wave is modified by the input information signal in order to form symbols that represent the digital input stream. Phase modulation systems with more symbols typically lead to higher throughput and better compression. An example phase modulation technique, Binary Phase Shift Keying (BPSK) can be seen below.

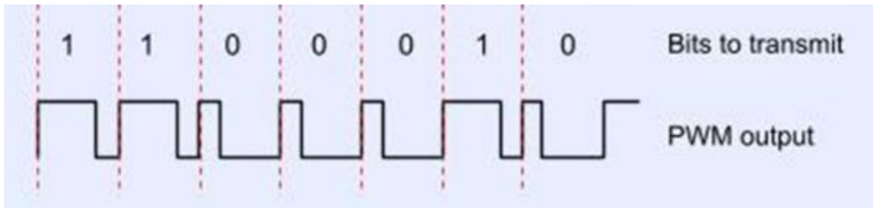


In the diagram above, the phase of the carrier wave is shifted by 180 degrees when a transition between a logic one and a logic zero occur. In BPSK, the carrier wave is inverted at each transition.

Communication systems that have to transmit a small amount of data, like the one we explore in this lab, use On-Off Keying. On-Off keying can be considered an extension of amplitude modulation where the data waveform is digital and the carrier waveform is essentially turned on and off where transitions occur.



The target device in this lab employs AM OOK with Pulse Width Modulation (PWM). Pulse width modulation is a technique to encode the On-Off Keyed signal to form symbols. The width of the pulses and gaps are interpreted by the receiver to recover the transmitted data.



Overview

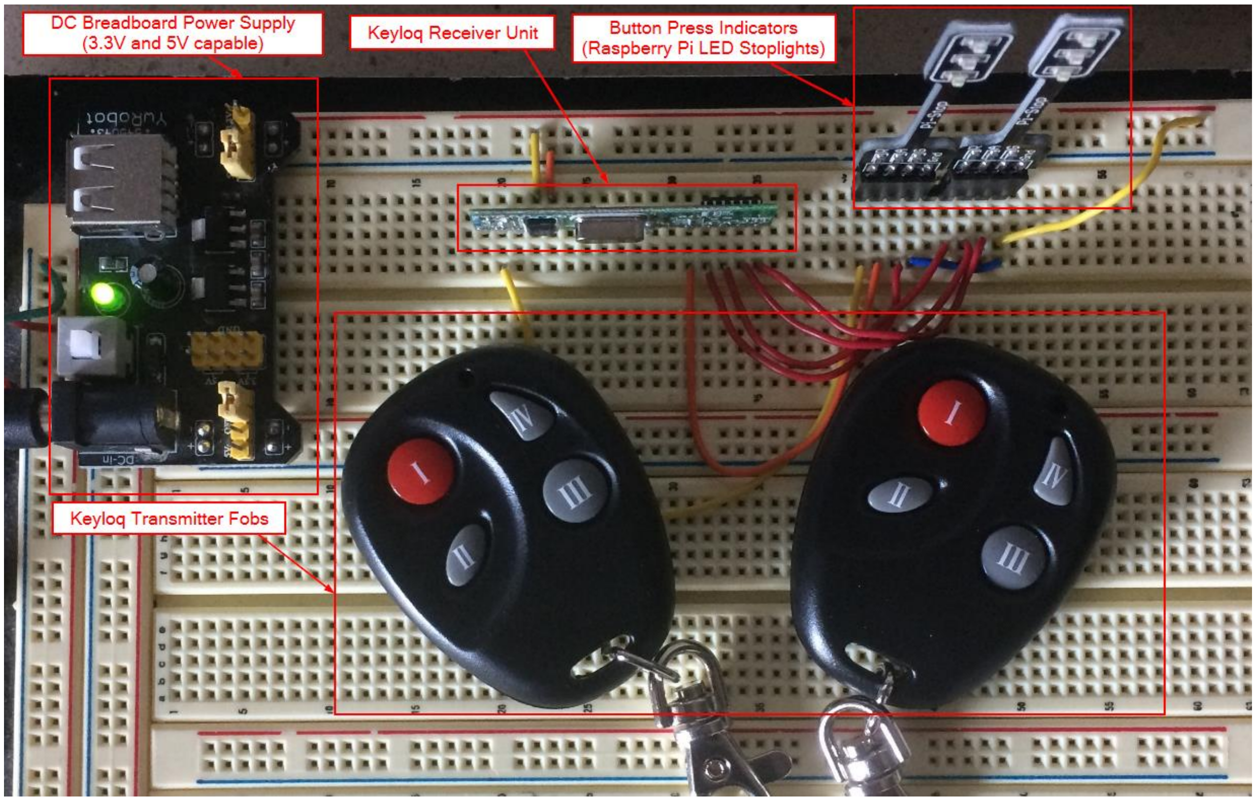
Many simple communication systems that we use for various purposes (doorbells, garage door openers, key fobs, etc) employ Amplitude Modulated On-Off Keying with Pulse Width Modulation for communication. Modern fob transmitters that are used for security purposes (garage door openers and vehicle remote systems) employ a rolling code algorithm to protect the target devices from simple replay or static key discovery attacks.

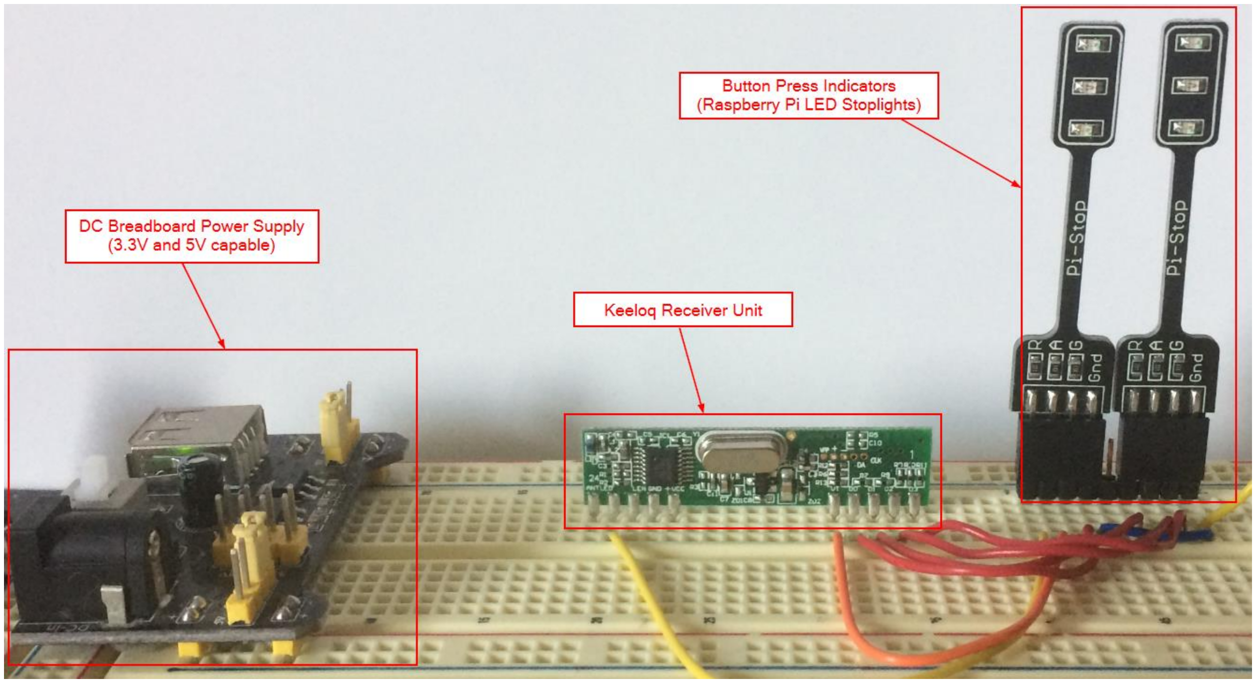
In general, rolling code systems must be synchronized so that the rolling code can be predictably produced on the transmitter side and consistently verified on the receiver side. The synchronization process is similar to setting the seed value on a two-factor authentication token. As long as the seed value is known to the transmitter and receiver, future values can be accurately generated and validated. However, on typical rolling code systems, the current valid key value is used to generate the next value (or set of values). To compensate for transmissions that may be out of range of the receiver (like accidental button pushes) the receiver generates a small list of values that the transmitter may send which should be considered valid. If the transmitter exceeds the number of transformations in the valid list while out of range, the two must be re-synchronized to set a matching seed value.

In this lab, we will explore an implementation of the Keeloq system. Keeloq describes a set of integrated circuits developed by MicroChip Technology Inc. These components are used to implement rolling code transmitter/receiver pairs for various purposes. We will demonstrate attacks against this system using the following hardware and software:

- HackRF One - A relatively inexpensive Software Defined Radio (SDR) capable of half duplex transmit and receive operation within the range of 0-6GHz.
- Rtl-sdr dongle - An inexpensive receive-only software defined radio.
- Yardstick One - A sub-1GHz digital wireless transmitter device.
- GQRX - An open source SDR receiver software.
- Rfcat - An open source SDR transmitter software capable of generating an ASK OOK PWM signal.
- Audacity - An open source audio editing program that can be used to inspect the target waveform.

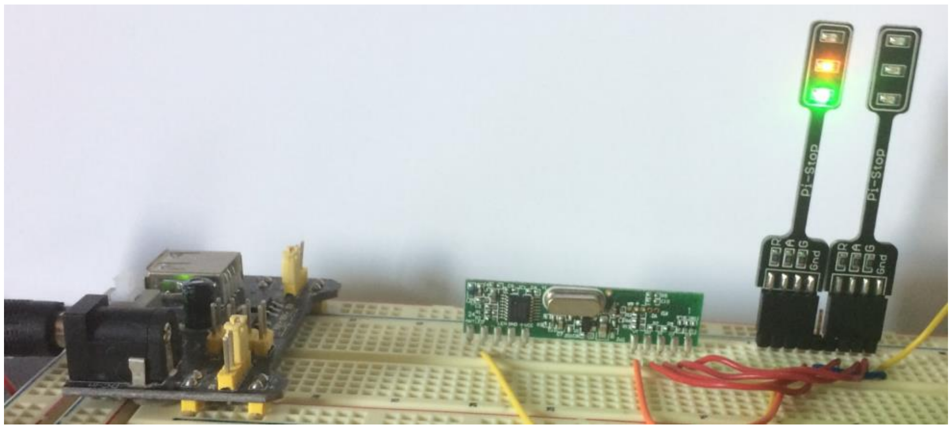
Rather than attacking a remote entry system installed in an actual vehicle, we have purchased an aftermarket kit that uses the same technology. The system is implemented on the breadboard located near the lab workstation. Images describing the setup can be seen below.

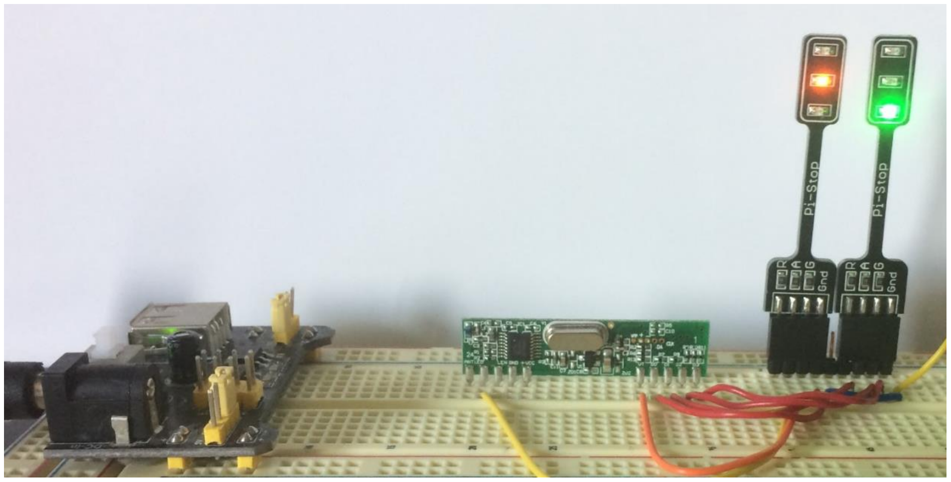

DC Breadboard Power Supply (3.3V and 5V capable)
Keyloq Receiver Unit
Button Press Indicators (Raspberry Pi LED Stoplights)
Keyloq Transmitter Fobs

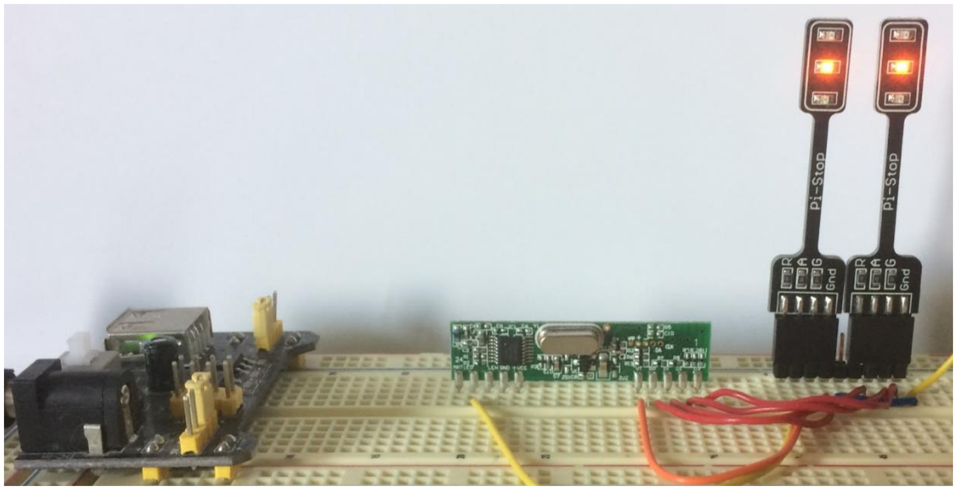

Button Press Indicators (Raspberry Pi LED Stoplights)
DC Breadboard Power Supply (3.3V and 5V capable)
Keeloq Receiver Unit
Pi-Stop
Pi-Stop

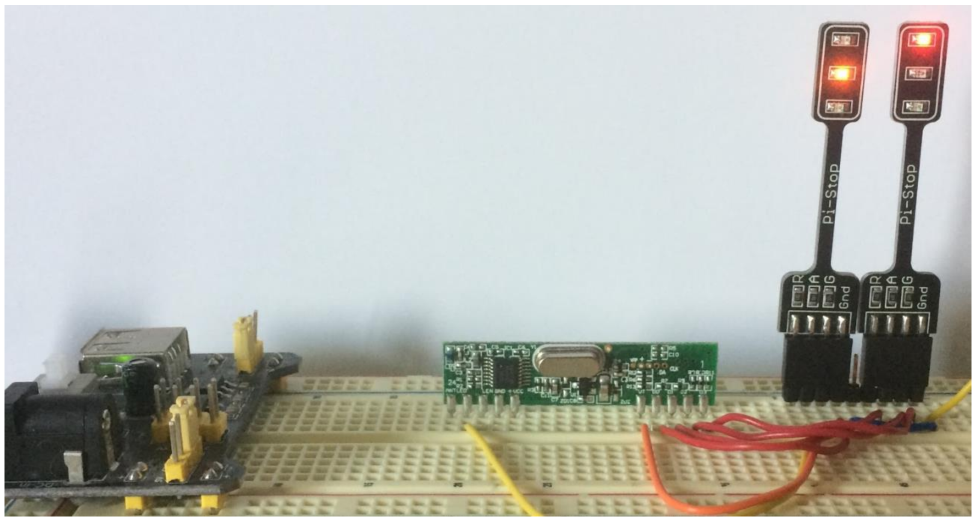
The aftermarket kit is sold as a simple unit that runs off of 5V DC power. In a real vehicle, the pins that power the LEDs would be connected to relays or microcontrollers which control the intended functions of the vehicle. This setup uses six LEDs to identify operation of the device. Four LEDs correspond to the four buttons on the key fob. The remaining two LEDs indicate learning mode operation and RF receive.

The learning mode LED will only illuminate when the learning mode pin is connected to ground. Otherwise, normal operation of the device will cause the corresponding button LED and the RF receive LED to illuminate simultaneously. Pressing each of the four buttons on one of the paired fobs will illuminate the LEDs as seen below.









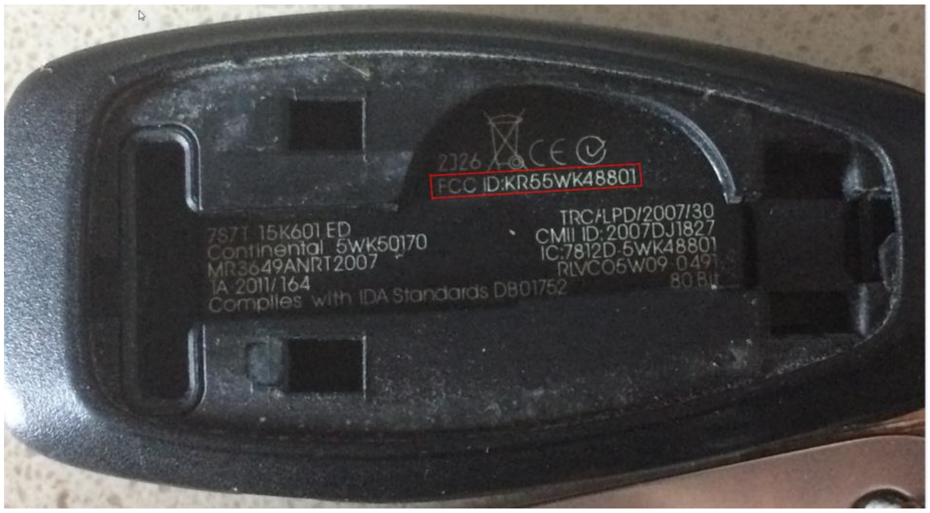
The amber LED on the left stoplight is the RF receive LED. The green LED on the left and all three LEDs on the right represent the four buttons on the key fob.

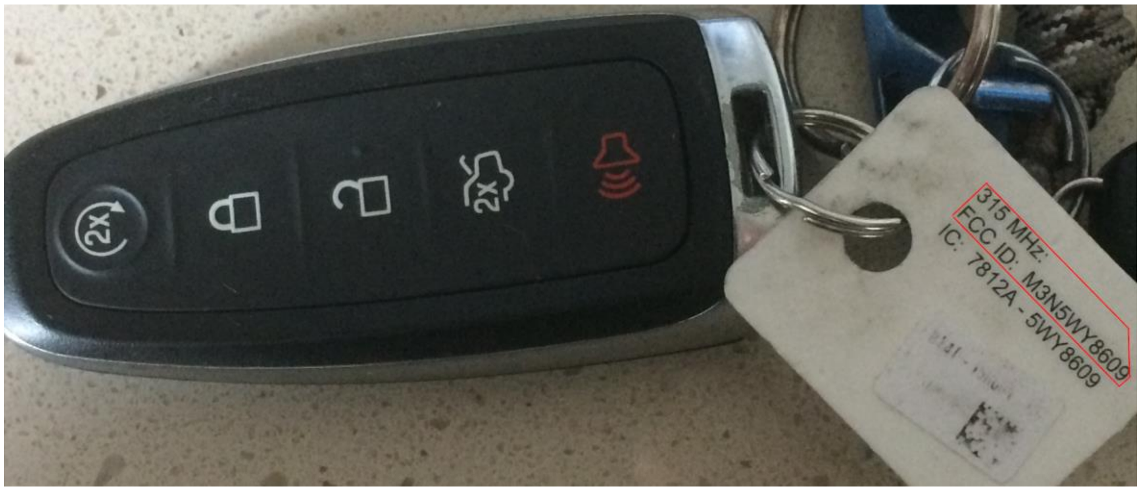
Press all four buttons on the key fob to ensure that the LEDs illuminate in a similar manner.

Reconnaissance

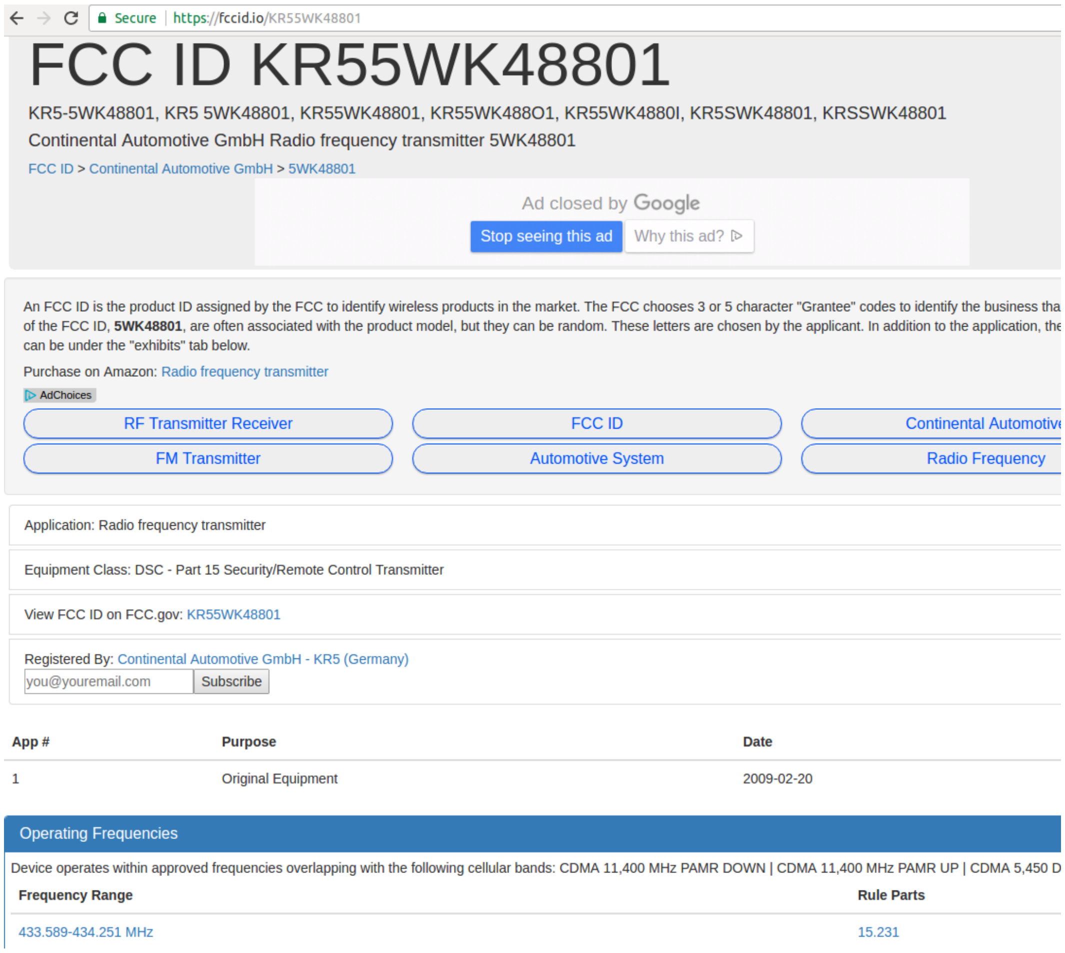
With and understanding of the system under test, we can begin to consider attacking the system. The first step in any well planned attack is reconnaissance. Typically, with an RF generating device such as this, our first step would be to determine the FCC ID for the device and submit that value to http://fccid.io in order to inspect the publicly available enclosures submitted with the device FCC license request.

Typically, this identifier is printed on the transmitter for the system. On other key fobs the FCC ID was found inside the fob or attached to the key ring on a separate tag as seen below.





Submission the FCC ID from one of the example key fobs to http://fccid.io results in details for the fob including the transmission frequency, internal photos, and circuit schematics.



Secure | https://fccid.io/KR55WK48801

FCC ID KR55WK48801

KR5-5WK48801, KR5 5WK48801, KR55WK48801, KR55WK488O1, KR55WK4880I, KR5SWK48801, KRSSWK48801

Continental Automotive GmbH Radio frequency transmitter 5WK48801

FCC ID > Continental Automotive GmbH > 5WK48801

Ad closed by Google

Stop seeing this ad | Why this ad?

An FCC ID is the product ID assigned by the FCC to identify wireless products in the market. The FCC chooses 3 or 5 character "Grantee" codes to identify the business tha
of the FCC ID, **5WK48801**, are often associated with the product model, but they can be random. These letters are chosen by the applicant. In addition to the application, the
can be under the "exhibits" tab below.

Purchase on Amazon: Radio frequency transmitter

AdChoices

RF Transmitter Receiver | FCC ID | Continental Automotive
FM Transmitter | Automotive System | Radio Frequency

Application: Radio frequency transmitter

Equipment Class: DSC - Part 15 Security/Remote Control Transmitter

View FCC ID on FCC.gov: KR55WK48801

Registered By: Continental Automotive GmbH - KR5 (Germany)

you@youremail.com | Subscribe

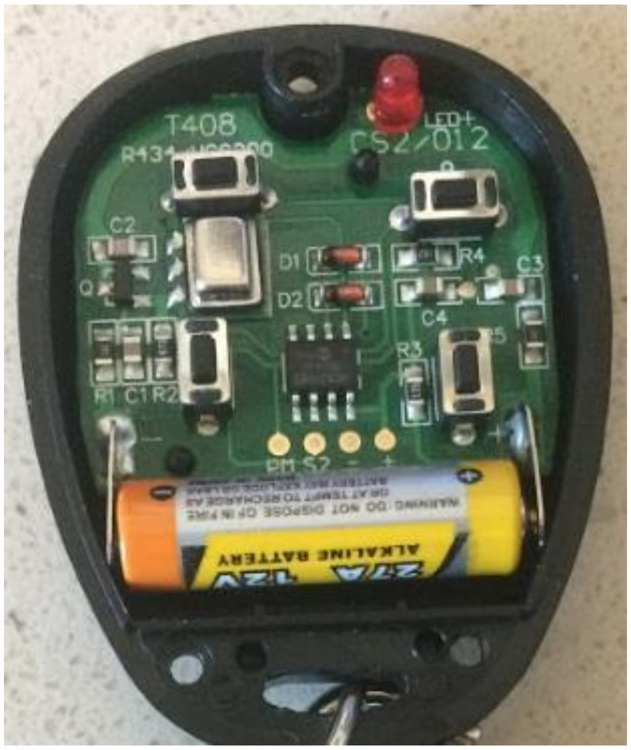
App #	Purpose	Date
1	Original Equipment	2009-02-20

Operating Frequencies

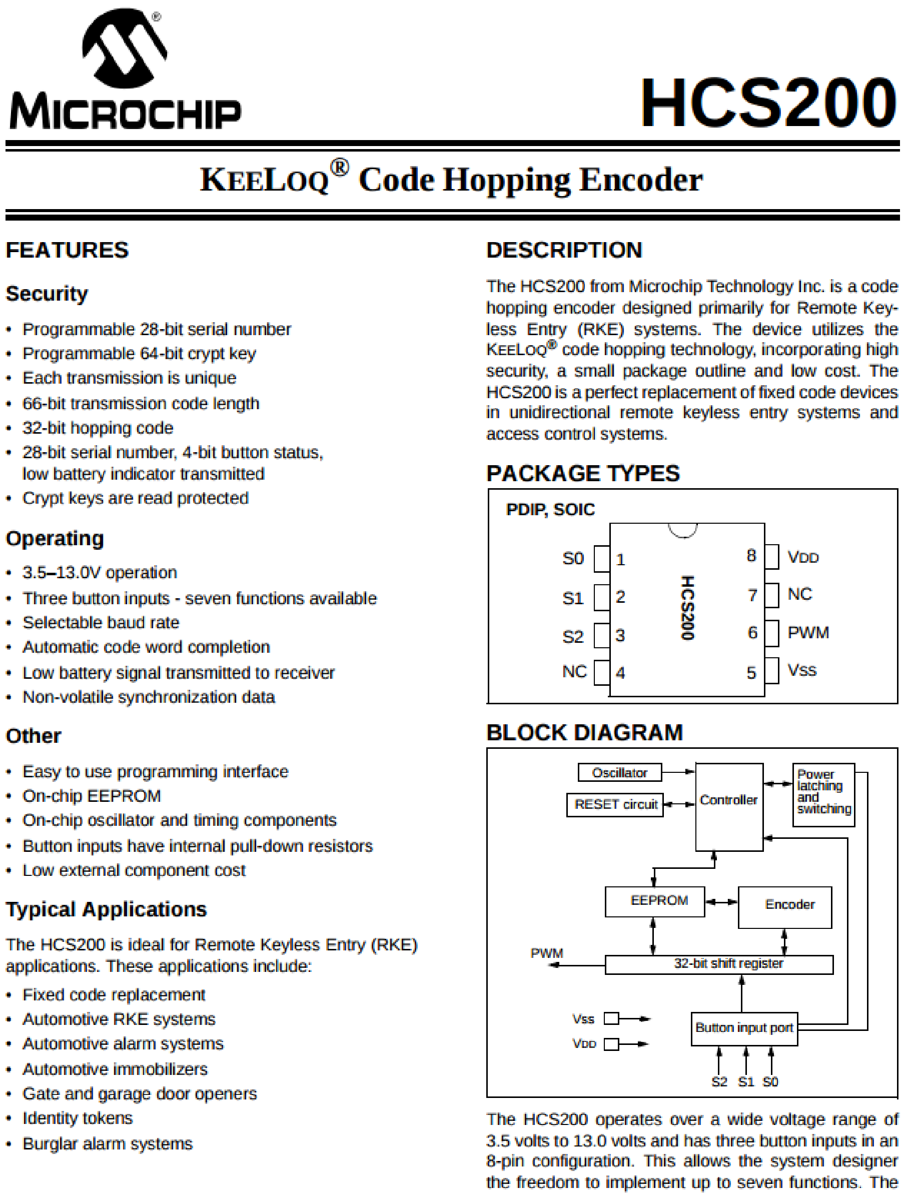
Device operates within approved frequencies overlapping with the following cellular bands: CDMA 11,400 MHz PAMR DOWN | CDMA 11,400 MHz PAMR UP | CDMA 5,450 D

Frequency Range	Rule Parts
433.589-434.251 MHz	15.231

The aftermarket transmitter does not have an FCC ID printed anywhere the device or within the case. As a result, we must resort to other techniques to determine the characteristics of the system. This can include stepping through candidate frequencies while operating the device to determine the transmission frequency. Key fobs typically operate within the Industrial, Scientific and Medical (ISM) frequency bands with a majority operating in the neighborhood of either 315 MHz or 433 MHz. As an alternative, we can also disassemble the device to inspect its internals as seen below.



Inspecting the components on the internal circuit board, the central component that operates the device is a MicroChip HCS200 integrated circuit. A Google search on this device reveals the datasheet that engineers use to integrate the component into their designs.



MICROCHIP **HCS200**

KEELOQ® Code Hopping Encoder

FEATURES

Security

- Programmable 28-bit serial number
- Programmable 64-bit crypt key
- Each transmission is unique
- 66-bit transmission code length
- 32-bit hopping code
- 28-bit serial number, 4-bit button status, low battery indicator transmitted
- Crypt keys are read protected

Operating

- 3.5–13.0V operation
- Three button inputs - seven functions available
- Selectable baud rate
- Automatic code word completion
- Low battery signal transmitted to receiver
- Non-volatile synchronization data

Other

- Easy to use programming interface
- On-chip EEPROM
- On-chip oscillator and timing components
- Button inputs have internal pull-down resistors
- Low external component cost

Typical Applications

The HCS200 is ideal for Remote Keyless Entry (RKE) applications. These applications include:

- Fixed code replacement
- Automotive RKE systems
- Automotive alarm systems
- Automotive immobilizers
- Gate and garage door openers
- Identity tokens
- Burglar alarm systems

DESCRIPTION

The HCS200 from Microchip Technology Inc. is a code hopping encoder designed primarily for Remote Keyless Entry (RKE) systems. The device utilizes the KEELOQ® code hopping technology, incorporating high security, a small package outline and low cost. The HCS200 is a perfect replacement of fixed code devices in unidirectional remote keyless entry systems and access control systems.

PACKAGE TYPES

PDIP, SOIC

BLOCK DIAGRAM

The HCS200 operates over a wide voltage range of 3.5 volts to 13.0 volts and has three button inputs in an 8-pin configuration. This allows the system designer the freedom to implement up to seven functions. The

This document describes the packaging, electrical characteristics, operating tolerances, breaks down the transmitted data into its component parts, describes the code hopping technique, and the employed synchronization technique.

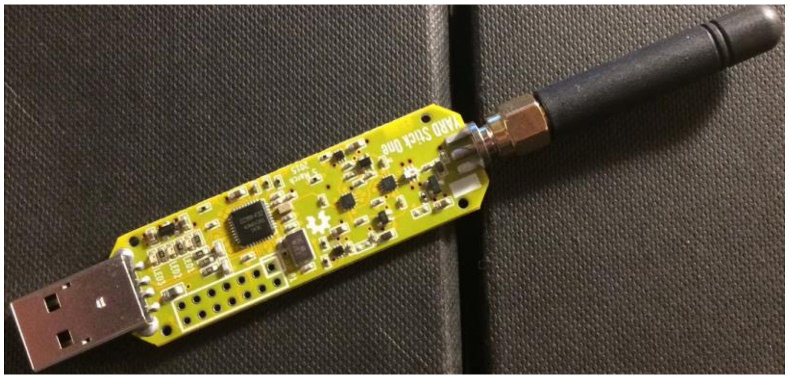
Code Hopping and Replay Prevention Confirmation

This portion of the lab illustrates the code hopping feature through reception and decode of multiple transmitted codes. After confirming that the system does in-fact employ code hopping, we will also confirm that code replay does not result in execution of the targeted feature.

After logging onto the lab workstation using the credentials, open two terminal windows.

- Username:
- Password:

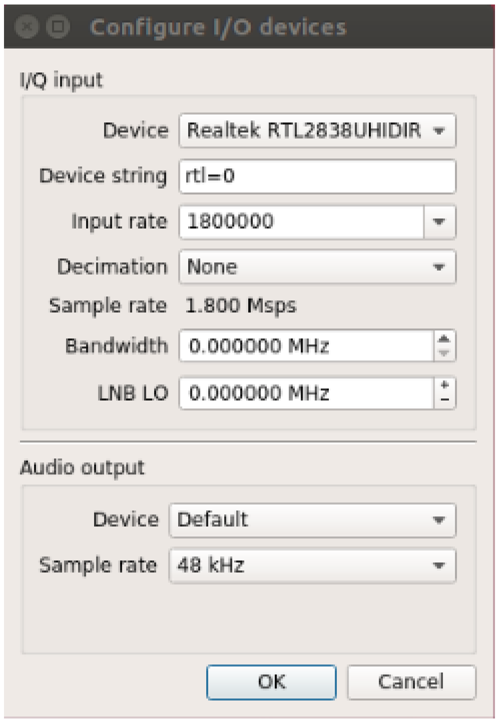
The RTL-SDR will be plugged into one of the computer USB ports while the YardStick One will be plugged into another. These devices can be seen below for reference.



The RTL-SDR will be used to receive our transmitted signal and the YardStick One will be used to replay the signal to confirm that replay attacks are not possible.

In the first terminal window on the lab computer, type “gqrx -r” to launch the GQRX receiver application. Initially, the application will prompt for configuration of the I/O device. This is a consequence of using the “-r” switch and allows the operator to configure the target device before initializing the radio to avoid errors.

Ensure that the “Realtek RTL2838UHIDIR” device is selected and click OK.


Configure I/O devices

I/Q input

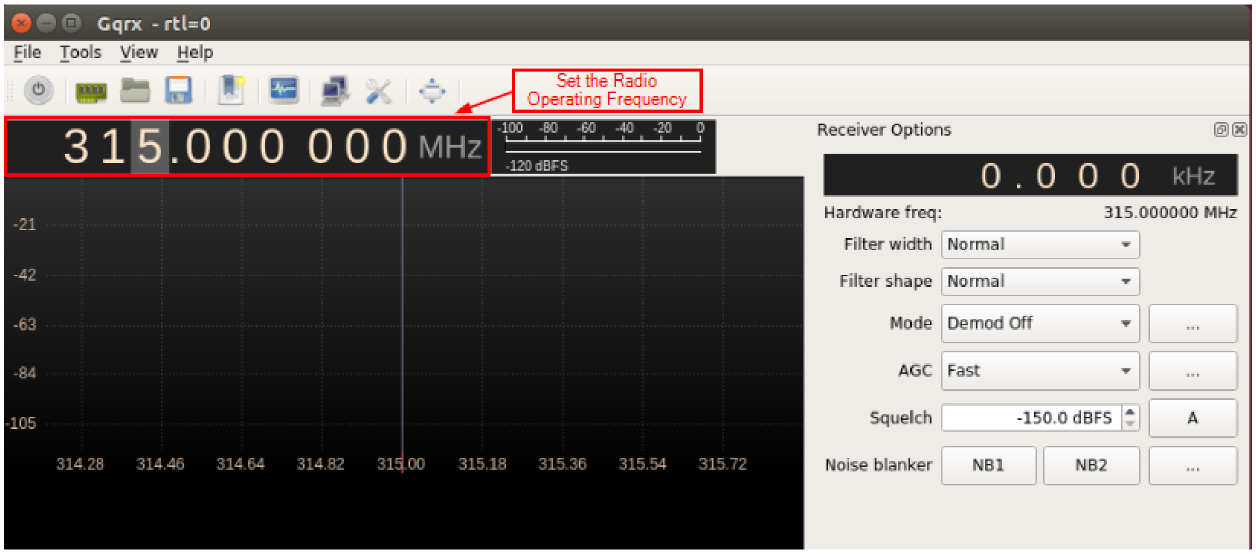
Device: Realtek RTL2838UHIDIR
Device string: rtl=0
Input rate: 1800000
Decimation: None
Sample rate: 1.800 Msps
Bandwidth: 0.000000 MHz
LNB LO: 0.000000 MHz

Audio output

Device: Default
Sample rate: 48 kHz

After clicking OK, the GQRX GUI application appears. Since we were not able to determine the operating frequency of the device from FCC ID reconnaissance or the component data sheet, we must determine the operating frequency through observation.

The most common frequencies for these devices are 315 MHz and 433 MHz. As a result, we will tune our radio to each frequency, click the button and monitor for a transmission. To tune the receiver, click on the numbers in the large frequency display and change the reading to be 315.000 000 MHz as seen below. This is easiest to accomplish by clicking the most significant digit and then using the arrow keys to modify the frequency.


Gqrx - rtl=0

Set the Radio Operating Frequency

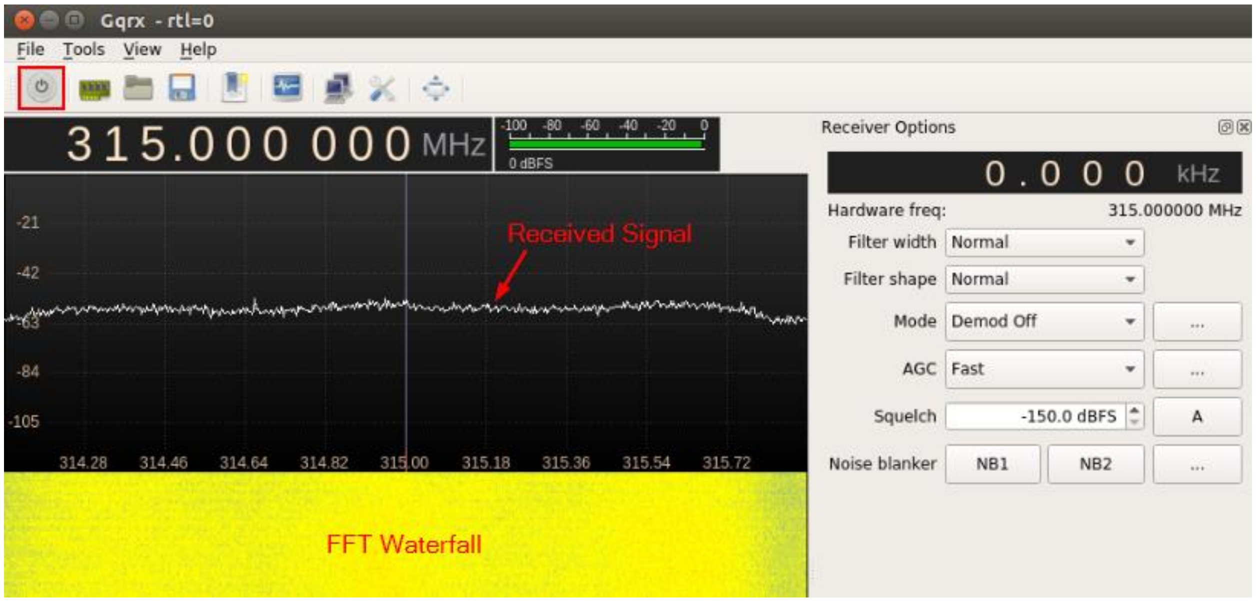
315.000 000 MHz

Receiver Options

0.000 kHz

Hardware freq: 315.000000 MHz
Filter width: Normal
Filter shape: Normal
Mode: Demod Off
AGC: Fast
Squelch: -150.0 dBFS
Noise blanker: NB1 NB2

Next, click the power button on the far left side of the GUI toolbar to enable the radio. You should see the signal appear and the FFT waterfall should turn yellow as seen below.

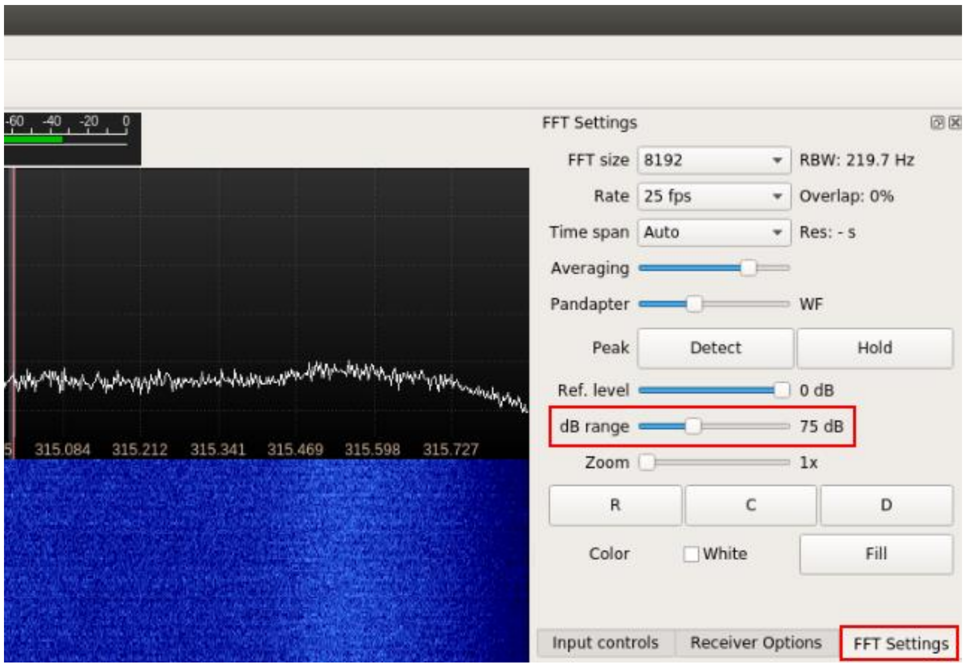


315.000 000 MHz

0 . 0 0 0 kHz

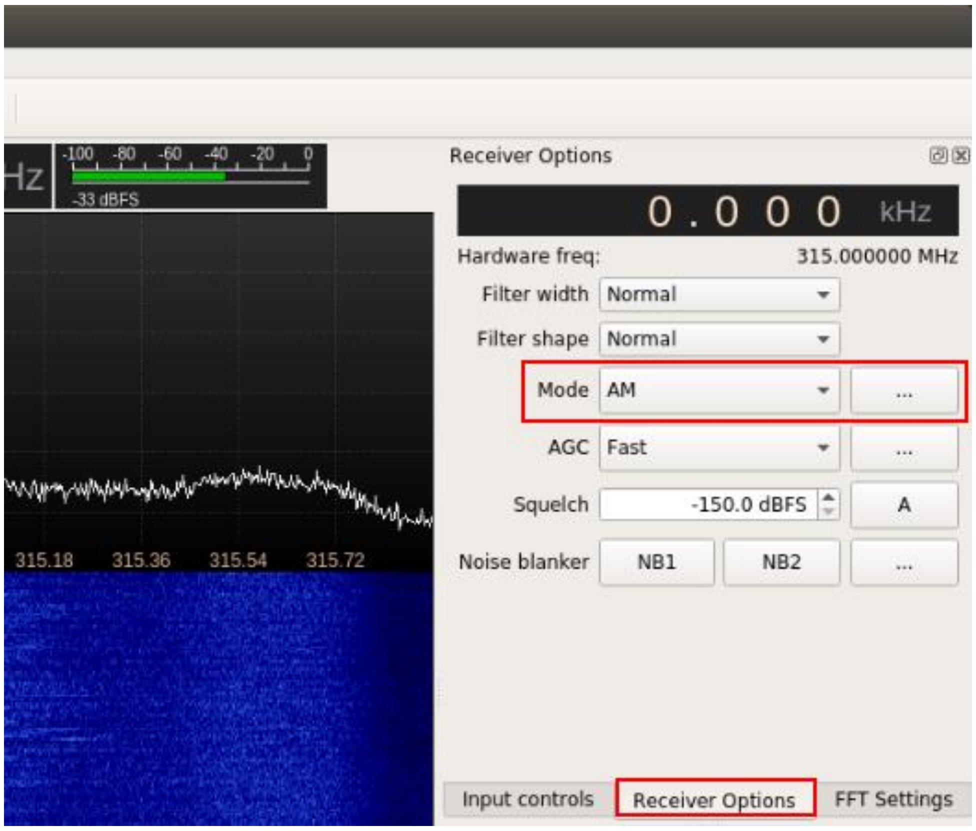
Hardware freq: 315.000000 MHz
Filter width: Normal
Filter shape: Normal
Mode: Demod Off
AGC: Fast
Squelch: -150.0 dBFS
Noise blanker: NB1 NB2

Next, select the FFT settings tab on the right hand side of the display and adjust the dB range parameter to be about 75 dB. The display should turn blue and this will make it easier to distinguish the transmitted signal from noise.



FFT size: 8192 RBW: 219.7 Hz
Rate: 25 fps Overlap: 0%
Time span: Auto Res: - s
Averaging
Pandapter WF
Peak: Detect Hold
Ref. level: 0 dB
dB range: 75 dB
Zoom: 1x
R C D
Color: White Fill

Finally, select the Receiver Options tab and set the mode to AM. This will demodulate the AM transmission so we can observe the OOK PWM signal. As a side effect, we will also hear the pulses generated by the transmitter as an audible signal.



0 . 0 0 0 kHz

Hardware freq: 315.000000 MHz

Filter width: Normal

Filter shape: Normal

Mode: AM

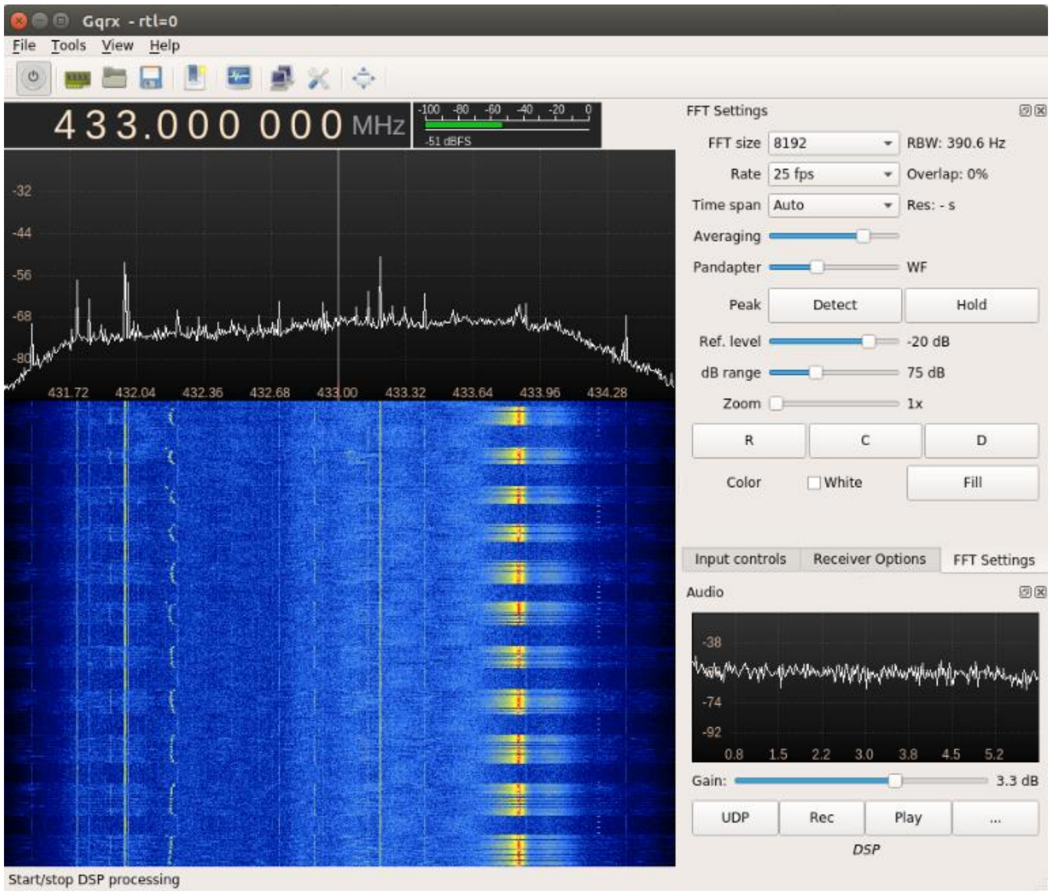
AGC: Fast

Squelch: -150.0 dBFS

Noise blanker: NB1 NB2

With the receiver configured, press the “I” button on the transmitter fob. At 315MHz you should observe that the corresponding LED on the receiver illuminates but there is no spike on the GQRX display.

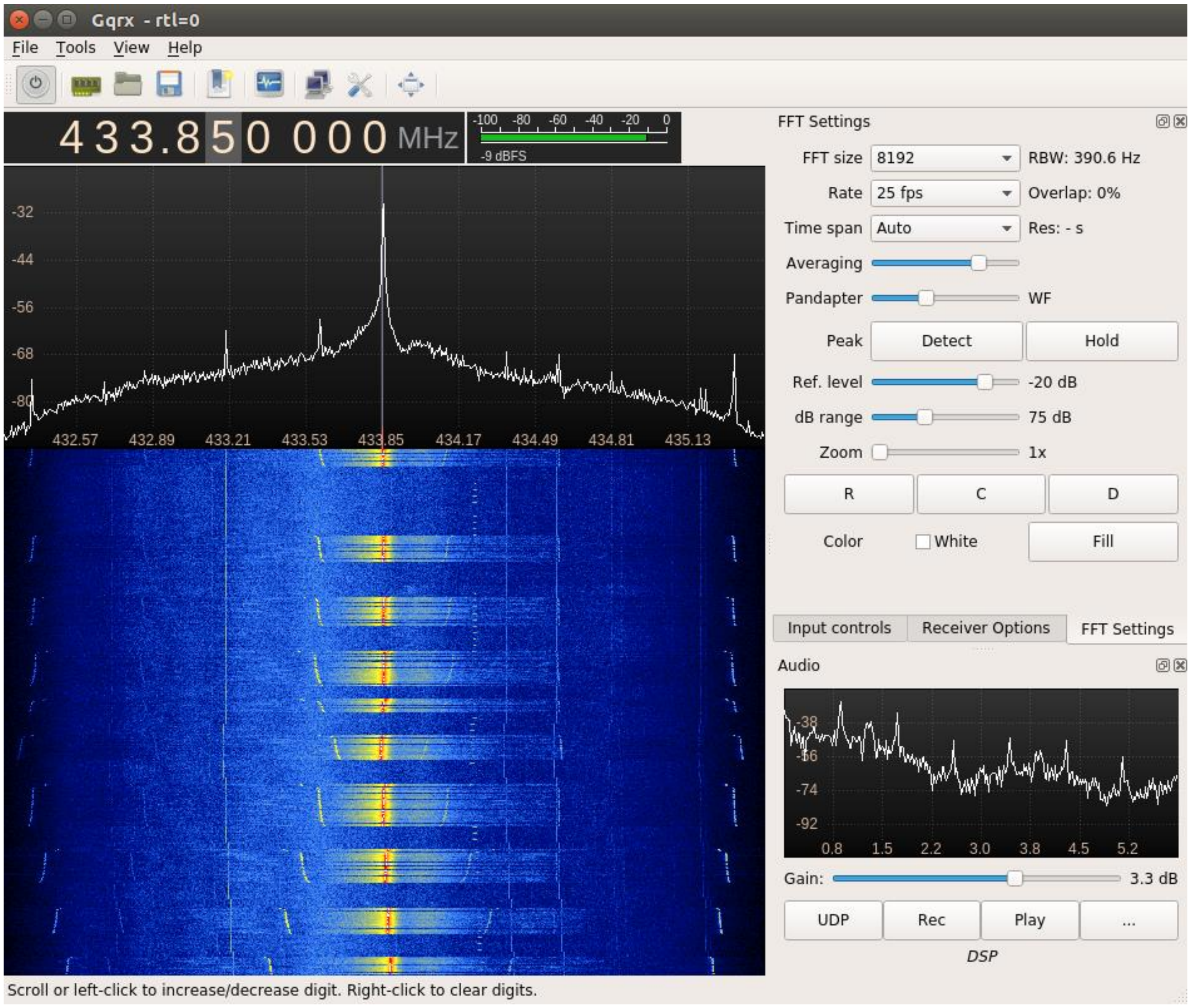
Retune the radio to 433MHz and press the “I” button once again. At this frequency you will see a spike appear on the left side of the frequency plot and a bright yellow and red line appear in the FFT waterfall display. The output below is the result of several successive button pushes so the output can be seen clearly.



FFT Settings

FFT size 8192 RBW: 390.6 Hz
Rate 25 fps Overlap: 0%
Time span Auto Res: - s
Averaging
Pandapter WF
Ref. level -20 dB
dB range 75 dB
Zoom 1x
Color White Fill
Gain: 3.3 dB

After determining the rough location of the transmission frequency we need to fine tune the radio so that the peak is aligned with the blue center frequency line in the display. The final frequency will be around 433.85 MHz as seen below.



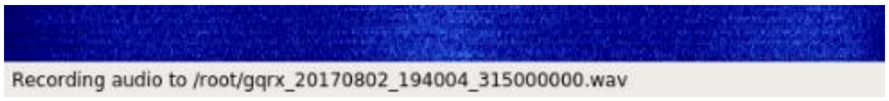
FFT Settings

FFT size 8192 RBW: 390.6 Hz
Rate 25 fps Overlap: 0%
Time span Auto Res: - s
Averaging
Pandapter WF
Peak
Ref. level -20 dB
dB range 75 dB
Zoom 1x
Color White

Gain: 3.3 dB

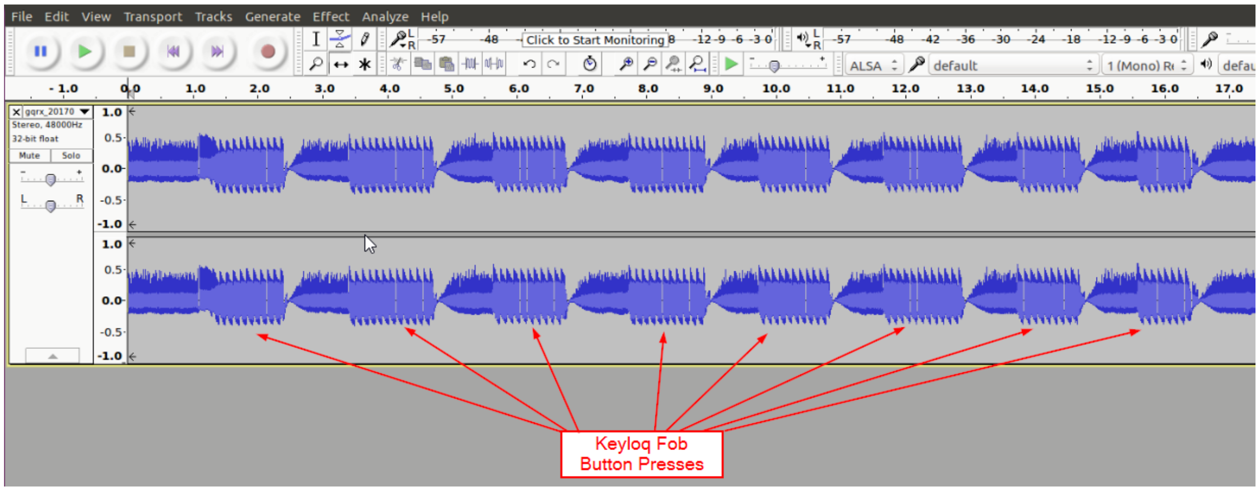
Scroll or left-click to increase/decrease digit. Right-click to clear digits.

After fine tuning the radio, press the button several times in succession. However, this time, click the "Rec" button to record the demodulated waveform to a "WAV" file. The filename and path of the recording are displayed in the status bar of the GUI and in the terminal window you launched GQRX from for reference.



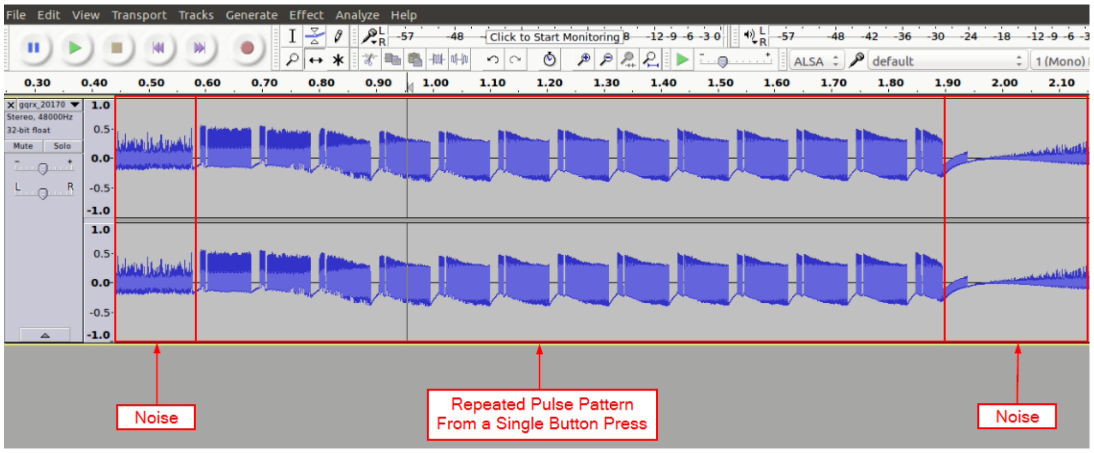
Recording audio to /root/gqrx_20170802_194004_315000000.wav

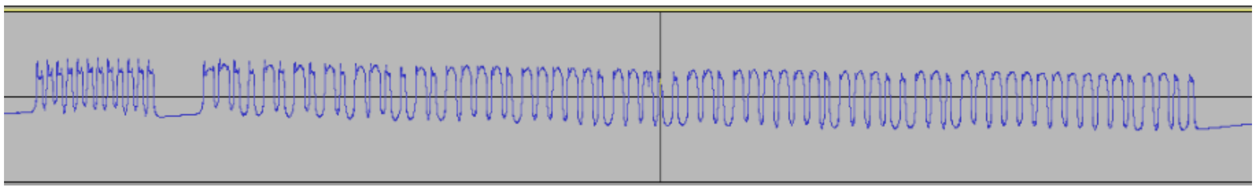
Next, use the launcher to open Audacity on the laptop and open the resulting "WAV" file using the File > Open menu option. Choose the default import method when prompted and click OK. The waveform will open looking similar to the display seen below.



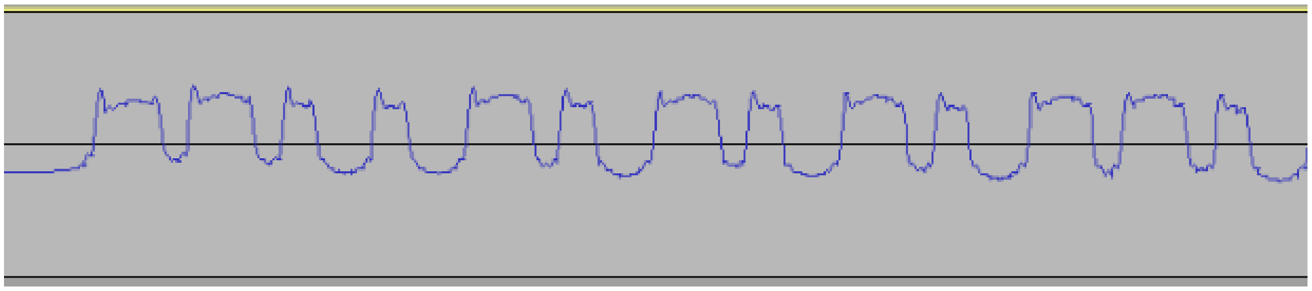
The waveform above shows eight distinct button presses using the “I” button on the key fob. Next, we will extract a single code transmission from each of the button presses present then analyze the result to confirm that the code hopping technique is working.

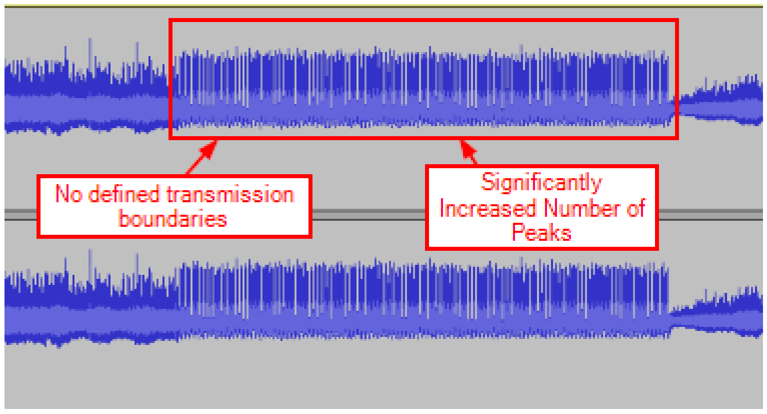
Click in the signal editor area inside one of the pulse regions and press Ctrl + 1 until the signal looks like the one seen below. The first image shows the pulse train zoomed in and the second shows a single iteration of the key being transmitted to the receiver. The transmitter simply repeats the same sequence over and over until the button is released. A second button press should produce a different code.

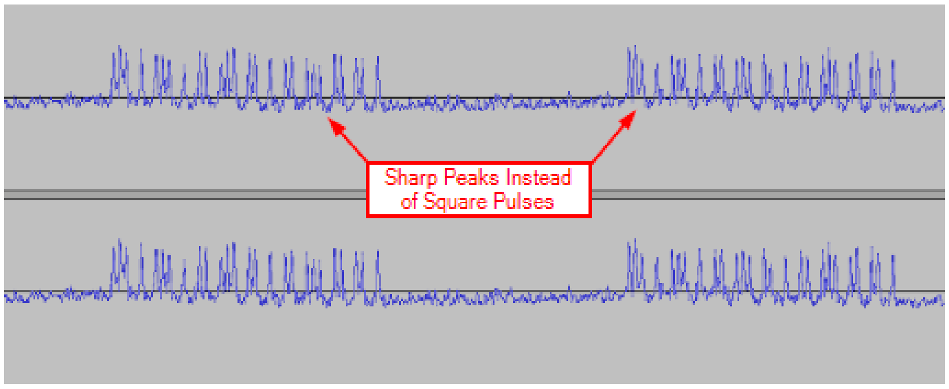




With each individual transmission there will be a sync word followed by the actual data stream that the receiver is supposed to process. Zooming in further on a portion of the large pulse train, you should recognize the Pulse Width Modulated (PWM) digital signal. This signal must be decoded into its binary equivalent so that we can recreate the transmitted signal with the YardStick One. This is most easily accomplished by trying to identify patterns both forward and backward within the signal.

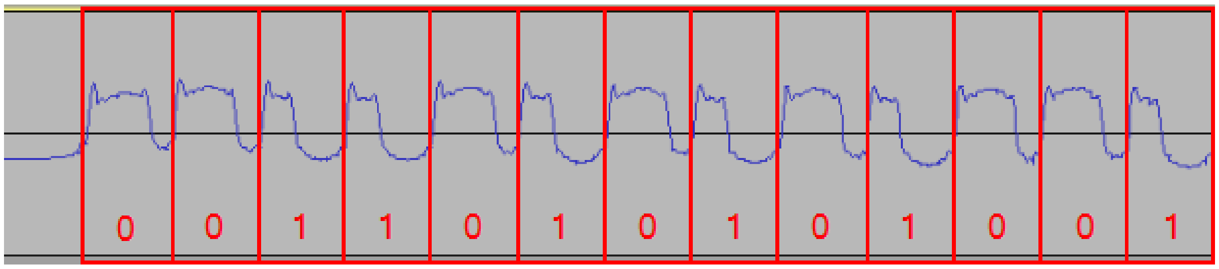

If GQRX needs to be fine-tuned, the output in Audacity will resemble the following graphics.


No defined transmission boundaries
Significantly Increased Number of Peaks



Should your recorded waveform resemble these, or lack defined pulses in the Audacity output, re-tune your GQRX to more closely match the transmitted frequency.

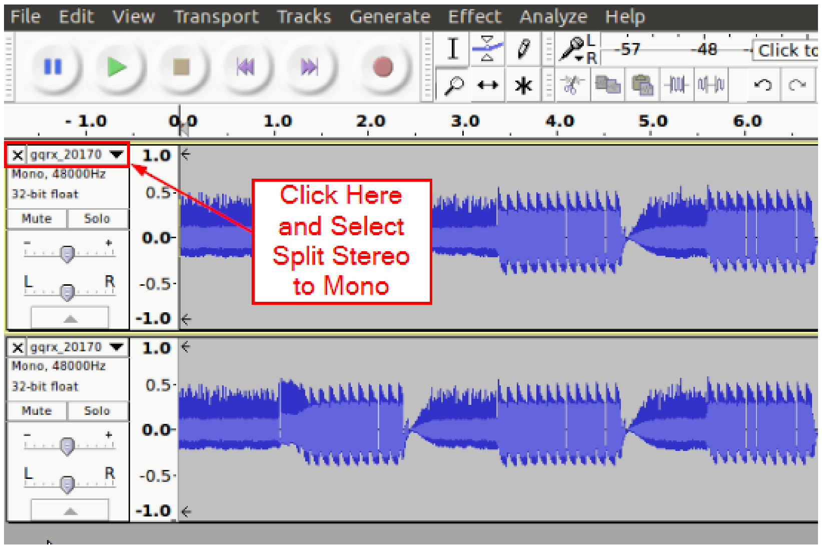
Looking at the waveform in Audacity, you should be able to pick out two distinct symbols. The first is a long pulse followed by a short gap and the second is a short pulse followed by a long gap. After preliminary pattern identification, you should read the signal forward and backward to make sure that are no inconsistencies in transmission (i.e. short pulse short gap or long pulse long gap). A graphical interpretation of a few of the observed bits can be seen below.



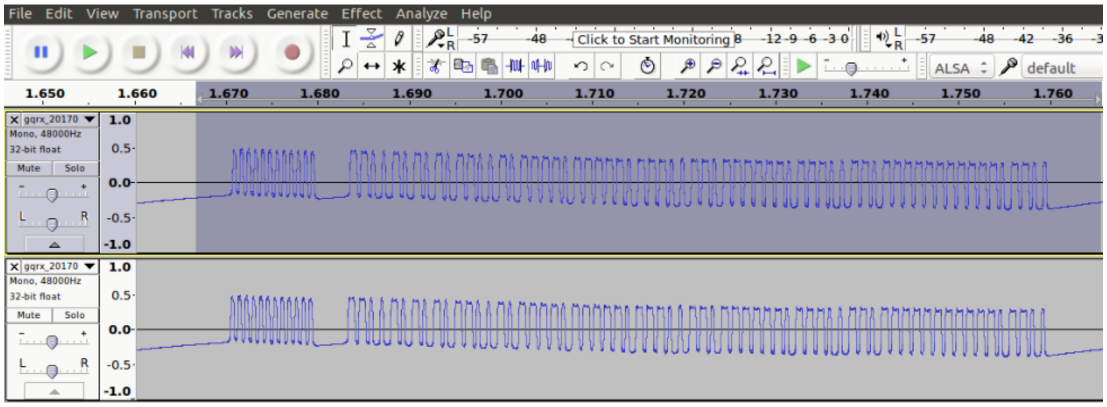
The pulses in this signal appear to be on the order of 25-33% duty cycle for a binary one and 66-75% duty cycle for a binary zero. This means that the signal is being held high for that portion of the repeating period of the waveform. This will become important when we attempt to recreate the waveform.

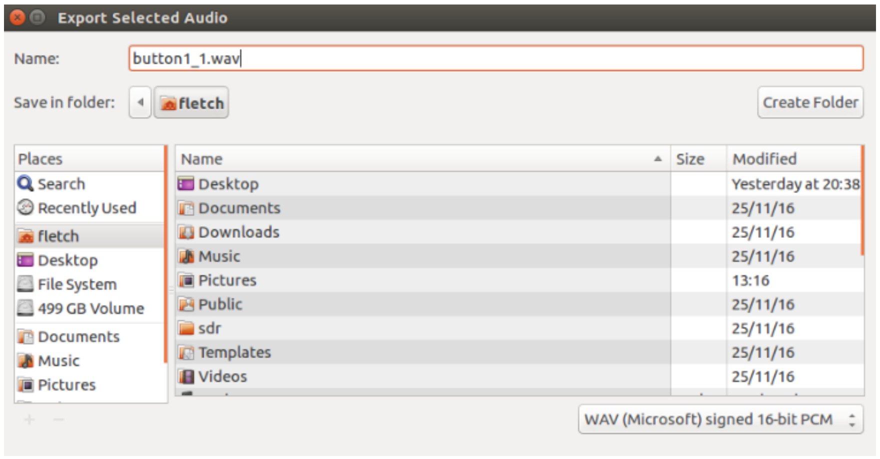
Rather than manually interpreting the full bit pattern and applying the process to multiple key presses to confirm the rolling code pattern is effective, we will use the “ooktools wave binary” command to decode the bit pattern from multiple key presses. Ooktools is a suite of tools for dealing with On-Off Keying bit patterns. The “wave binary” tool will attempt to recover a binary pattern from the PWM signal.

To decode the bit pattern using ooktools, select the track name at the top of the window to the left of the signal display and select the “Split Stereo to Mono” option.


File Edit View Transport Tracks Generate Effect Analyze Help
Click Here and Select Split Stereo to Mono
gqrx_20170
Mono, 48000Hz
32-bit float
Mute
Solo

After splitting the track into mono, highlight a single run of the PWM code as seen below and select File > Export Selected Audio and save the file as “button1_1.wav”. Click OK when prompted for metadata to describe the file.




Export Selected Audio

Name: button1_1.wav

Save in folder: fletch

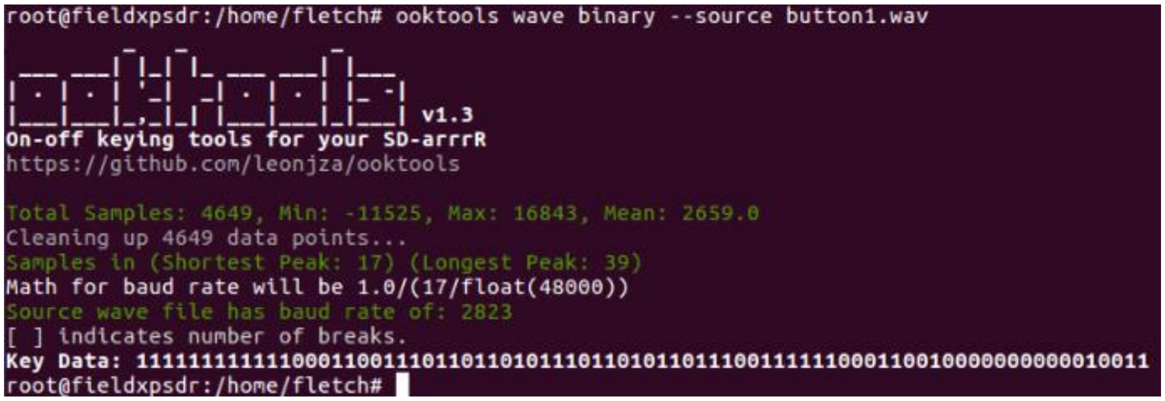
Create Folder

Places
Search
Recently Used
fletch
Desktop
File System
499 GB Volume
Documents
Music
Pictures

Name	Size	Modified
Desktop		Yesterday at 20:38
Documents		25/11/16
Downloads		25/11/16
Music		25/11/16
Pictures		13:16
Public		25/11/16
sdr		25/11/16
Templates		25/11/16
Videos		25/11/16

WAV (Microsoft) signed 16-bit PCM

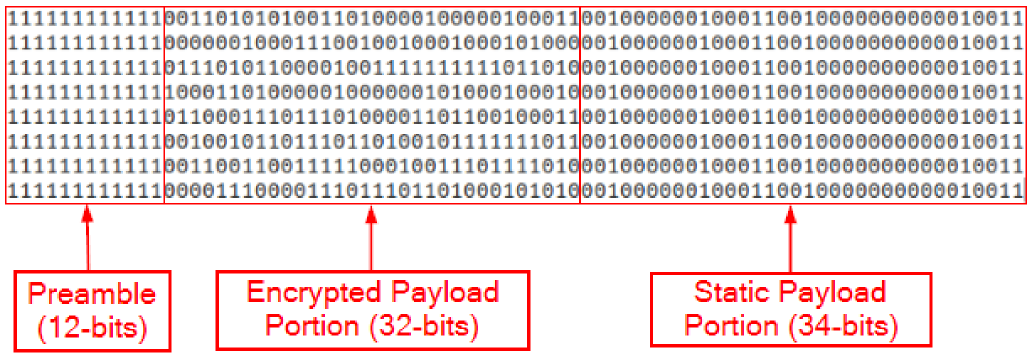
In one of your terminal windows, run the command “ooktools wave binary --source button1_1.wav”. The proper path to the WAV file must be provided in order for this to work. Output from this tool should resemble the graphic below and confirm the manually derived binary code.


```
root@fieldxpsdr:/home/fletch# ooktools wave binary --source button1.wav

___ ___| |_|‾|_ ___ ___| |___
| . | . | '_|  _| . | . | |_ -|
|___|___|_,_|_| |___|___|_|___| v1.3
On-off keying tools for your SD-arrrR
https://github.com/leonjza/ooktools

Total Samples: 4649, Min: -11525, Max: 16843, Mean: 2659.0
Cleaning up 4649 data points...
Samples in (Shortest Peak: 17) (Longest Peak: 39)
Math for baud rate will be 1.0/(17/float(48000))
Source wave file has baud rate of: 2823
[ ] indicates number of breaks.
Key Data: 11111111111100011001110110110101110110101101110011111100011001000000000000010011
root@fieldxpsdr:/home/fletch#
```

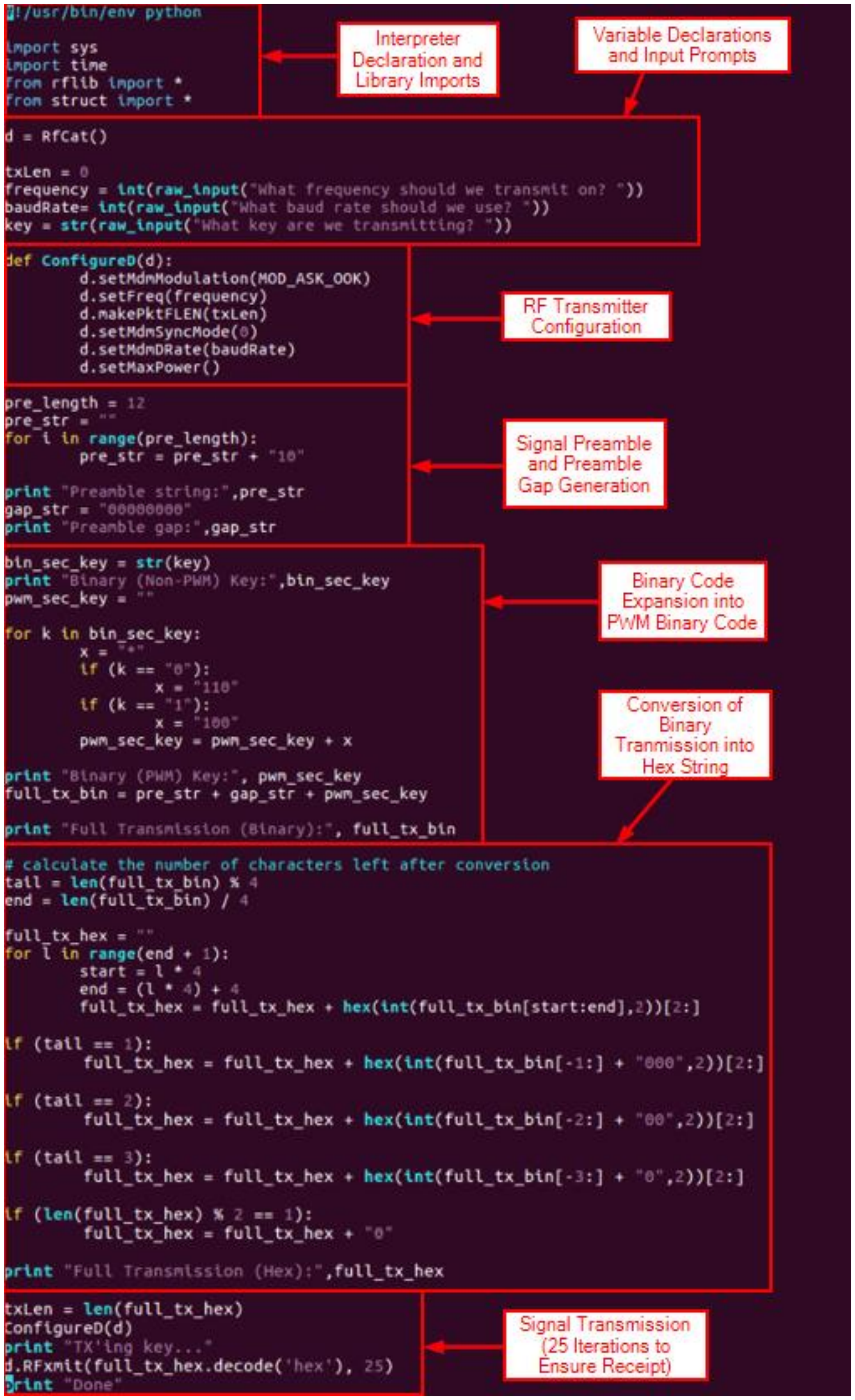
Repeat this process for a single run of the bit pattern in each of the individual button presses. For the eight button presses included in the example signal above, the following code transmissions were observed.



```
1111111111110011010101001101000010000010001100100000010001100100000000000010011
1111111111110000001000111001001000100010100000100000010001100100000000000010011
1111111111110111010110000100111111111101101000100000010001100100000000000010011
1111111111111000110100000100000010100010001000100000010001100100000000000010011
1111111111110110001110111010000110110010001100100000010001100100000000000010011
1111111111110010010110111011010010111111101100100000010001100100000000000010011
1111111111110011001100111110001001110111101000100000010001100100000000000010011
1111111111110000111000011101110110100010101000100000010001100100000000000010011
```

Preamble (12-bits)
Encrypted Payload Portion (32-bits)
Static Payload Portion (34-bits)

As seen above, the encrypted payload portion of the transmission does in-fact change with each successive button press. To confirm that replay of any of the captured codes does not result in execution of the identified button function, we will replay various codes using the script seen below.


Interpreter Declaration and Library Imports
Variable Declarations and Input Prompts
RF Transmitter Configuration
Signal Preamble and Preamble Gap Generation
Binary Code Expansion into PWM Binary Code
Conversion of Binary Tranmission into Hex String
Signal Transmission (25 Iterations to Ensure Receipt)

```
#!/usr/bin/env python

import sys
import time
from rflib import *
from struct import *

d = RfCat()

txLen = 0
frequency = int(raw_input("What frequency should we transmit on? "))
baudRate= int(raw_input("What baud rate should we use? "))
key = str(raw_input("What key are we transmitting? "))

def ConfigureD(d):
        d.setMdmModulation(MOD_ASK_OOK)
        d.setFreq(frequency)
        d.makePktFLEN(txLen)
        d.setMdmSyncMode(0)
        d.setMdmDRate(baudRate)
        d.setMaxPower()

pre_length = 12
pre_str = ""
for i in range(pre_length):
        pre_str = pre_str + "10"

print "Preamble string:",pre_str
gap_str = "00000000"
print "Preamble gap:",gap_str

bin_sec_key = str(key)
print "Binary (Non-PWM) Key:",bin_sec_key
pwm_sec_key = ""

for k in bin_sec_key:
        x = "*"
        if (k == "0"):
                x = "110"
        if (k == "1"):
                x = "100"
        pwm_sec_key = pwm_sec_key + x

print "Binary (PWM) Key:", pwm_sec_key
full_tx_bin = pre_str + gap_str + pwm_sec_key

print "Full Transmission (Binary):", full_tx_bin

# calculate the number of characters left after conversion
tail = len(full_tx_bin) % 4
end = len(full_tx_bin) / 4

full_tx_hex = ""
for l in range(end + 1):
        start = l * 4
        end = (l * 4) + 4
        full_tx_hex = full_tx_hex + hex(int(full_tx_bin[start:end],2))[2:]

if (tail == 1):
        full_tx_hex = full_tx_hex + hex(int(full_tx_bin[-1:] + "000",2))[2:]

if (tail == 2):
        full_tx_hex = full_tx_hex + hex(int(full_tx_bin[-2:] + "00",2))[2:]

if (tail == 3):
        full_tx_hex = full_tx_hex + hex(int(full_tx_bin[-3:] + "0",2))[2:]

if (len(full_tx_hex) % 2 == 1):
        full_tx_hex = full_tx_hex + "0"

print "Full Transmission (Hex):",full_tx_hex

txLen = len(full_tx_hex)
ConfigureD(d)
print "TX'ing key..."
d.RFxmit(full_tx_hex.decode('hex'), 25)
print "Done"
```

This script configures the radio, expands the binary security key to an equivalent pulse width modulated value, converts that value to an equivalent hexadecimal value, and transmits the key on the given frequency using RFCat.

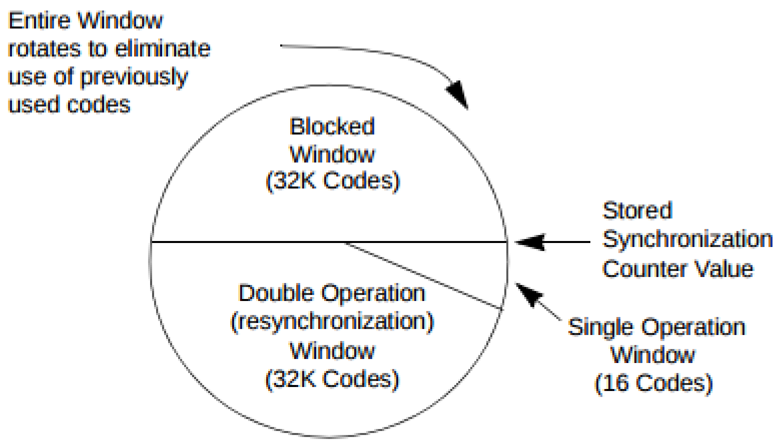
Execute this script by executing “python opensaysme.py”. The script will prompt for the transmission frequency, baud rate, and key discovered while executing the previous portions of the lab.

During the first transmission, you should have GQRX running to ensure that the YardStick One transmits at the appropriate frequency. If not, adjust the transmit frequency according to the observation in GQRX. The transmitted signal should appear at the center frequency where the key FOB signal was recorded.

Use this method to transmit several of the observed keys. Does the corresponding LED illuminate?

Stolen Code Replay

Based on the information found in the data sheet for the HCS200 encoder that this system uses, any of 16 valid codes can be used to activate a given function until a valid code is submitted. Once a valid code is received, a new code window is generated. If the transmitter drifts beyond the 16 code window (typically due to out of range button presses), then the transmitter will still work but requires two sequential correct code transmissions.


Entire Window rotates to eliminate use of previously used codes
Blocked Window (32K Codes)
Stored Synchronization Counter Value
Double Operation (resynchronization) Window (32K Codes)
Single Operation Window (16 Codes)

This means that if a code is captured while the transmitter is out of range, as long as it is replayed before another valid code has been received, then that code will successfully execute the intended function on the target device.

1. Click the “I” button on the transmitter and observe the corresponding LED illuminate on the breadboard to ensure that the transmitter and receiver are synchronized.

2. Push the power button on the DC power supply connected to the breadboard to disable power to the receiver.

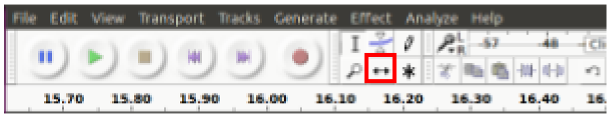
3. Record a single button press using GQRX as previously accomplished.

4. Export a single iteration of the transmitted code using Audacity.

5. Recover the bit pattern of the key using the “ooktools wave binary” command.
6. Push the power button on the DC power supply connected to the breadboard to enable power to the receiver.

7. Replay the stolen code using the Python script and observe the receiver response.

If the code replay is not successful, adjust the transmission frequency and/or baud rate to match the transmitter. It is also useful to display the transmitted waveforms side by side to ensure that the preamble, gap and transmission lengths are roughly equal in terms of time. This can be accomplished by recording the transmissions separately. Open the first using audacity normally, then use the File > Import > Audio menu option to import the second recording. Finally, zoom in on the transmitted waveform using Ctrl + 1. If necessary, the “Time Shift Tool” can be used to align the two waveforms. With the tool selected, you can just click and drag the track right or left. The “Time Shift Tool” menu option can be seen highlighted below.


File Edit View Transport Tracks Generate Effect Analyze Help
15.70 15.80 15.90 16.00 16.10 16.20 16.30 16.40

Transmitter Signal Jamming

Systems like Keeloq are designed for stable operation under extreme conditions. When deployed in a vehicle, the ambient temperature (whether high or low) can affect the operating frequency of the oscillator which provides timing for the microcontroller used to receive and decode transmissions. As a result, the receiver is designed to listen with a wide pass frequency band.

This condition is exploited by devices like the roll jam to prevent the receiver from acknowledging the code emitted by the transmitter. As a result, an attacker can record the code emitted from the transmitter, ignoring the noise signal by filtering the jamming frequency, and store the value for replay as demonstrated in the previous section of the lab.

When two signals are transmitted on or near the same frequency signal loss occurs. This is the equivalent of two people talking at the same time. The receiver of the message does not know how to interpret the data and instead ignores or discards the message. Some communication systems are designed to be jam resistant. However, the Keeloq transmitter and receiver are not in that group.

This portion of the lab will use the RFJammer.py script to demonstrate that the Keeloq transmitter fob can be jammed and that the jamming frequency does not need to be exact.

1. Ensure that the RTL-SDR and YardStick One are plugged into the lab computer.

2. Ensure that the Keeloq receiver is powered on and press the “I” button of the fob to ensure that the transmitter and receiver codes are synchronized.

3. Start GQRX and tune to the frequency of the key fob as demonstrated in other portions of the lab.

4. Open a terminal window and change to the /opt/RFCatHelpers directory where the RFJammer.py script is located.

5. Start the jammer by executing “python RFJammer.py -f [target frequency]”. The signal should appear on the FFT waterfall display.

6. Push the “I” button on the transmitter and observe the response of the receiver.

7. Adjust the transmission frequency incrementally above or below the key fob center frequency.

8. Repeat until the receiver acknowledges the transmitter code. Ensure that between each jamming session, you push the “I” button to keep the transmitter and receiver codes synchronized.

How far away from the key fob transmission frequency could you adjust the jamming signal before the receiver acknowledged the transmitted code?

Cleanup

Please delete your saved WAV file and close all open application and terminal windows.

Thank you!!