Carleton University

COMP 4905 : Honours Project

# Analysis of RF Remote Security Using Software Defined Radio

By: Spencer Whyte

Supervisor: Dr. Michel Barbeau, School of Computer Science

August 19th, 2013

## Abstract

The applications of RF remotes are numerous; they unlock our cars, open our garage doors, and arm our home security systems. These are some very important items; therefore the security on which the systems are built must be solid. This report outlines a few different attacks against these systems and then delves into the details about how a cost effective attack was implemented against a real world system. The hybrid jam and replay attack is a sub $50 attack that was shown to grant unauthorized access to a recent automobile under certain conditions. The key to success was to use a jammer of slightly higher frequency than the authorized remote. Then band-pass filtering was applied to the received signal to eliminate the jamming signal and obtain the signal from the authorized remote. For this reason and many others, bidirectional challenge-response systems are desirable and unidirectional rolling code systems should be avoided.

**Acknowledgments**
Many thanks to Dr. Barbeau for providing a USRP2 and TVRX daughterboard for experimentation.

## Table of Contents

## List of Figures

## List of Tables

# 1. Introduction to Software Defined Radio (SDR)

Typically, when designing a system that involves some sort of radio, the radio hardware will be custom built and designed for one specific purpose. It will operate on very specific frequencies, and implement a protocol that is often embedded right in the hardware. The alternative to this is to use software defined radio hardware which is general-purpose radio hardware that has the ability to operate on a multitude of frequencies and the implementation of protocols can be done in software.

## 1.1 Signal Representation

Signals can be described via what is called sampling, the idea behind sampling is to look at a signal at a particular point in time and to record some characteristics as discrete data. One such characteristic that can be recorded is the amplitude of the signal. The way that a signal is sampled in SDR is I/Q data or in-phase and quadrature data. The I portion of the data is the amplitude of the signal at the time of sampling and the Q portion of the data is the amplitude of the signal 90 degrees out of phase (Barbeau, 2013). An example of one sample of I/Q data can be seen in Figure 1.

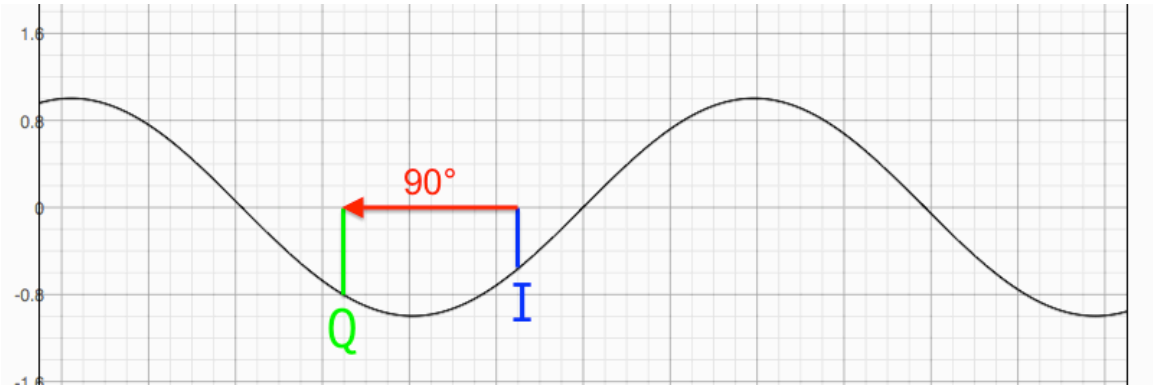


**Figure 1: A visual summary of the I/Q sampling process.**

## 1.2 Amplitude Modulation (AM)

Amplitude modulation is a method of encoding data by varying the amplitude of a carrier wave. Sampling the signal as I/Q data makes the demodulation of an AM signal easy. The amplitude of the signal can be recovered at any point in time by taking the square root of the sum of squares of the I and Q data as per the following equation.

$$A = \sqrt{I^2 + Q^2}$$

This formula can be explained by imagining a circle of radius A, placed on an I/Q axis as shown in Figure 2.

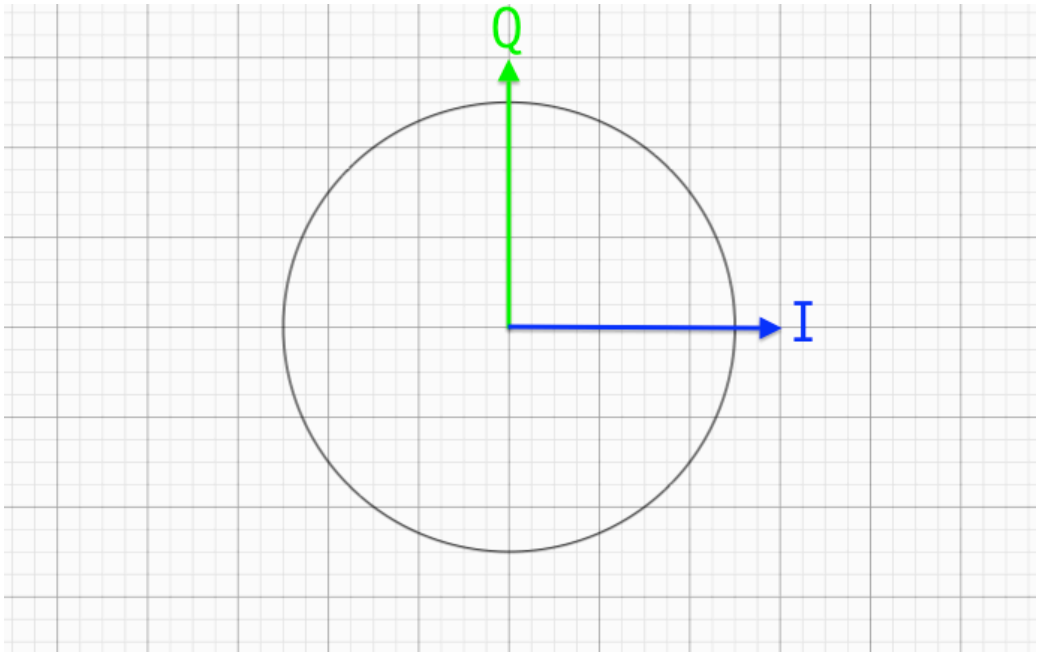


**Figure 2: Visualization of a circle of radius A, placed on an I/Q axis.**

The following equations can therefore be used to describe the points on that circle.

$$I = A \sin \theta$$
$$Q = A \cos \theta$$

We should recognize that these equations could also be used to describe a wave of amplitude A because the cosine function is 90 degrees out of phase with the sin function. If we take any given point on the circle in Figure 2, a right angle triangle is formed between that point, the origin, and the I axis as seen in Figure 3.

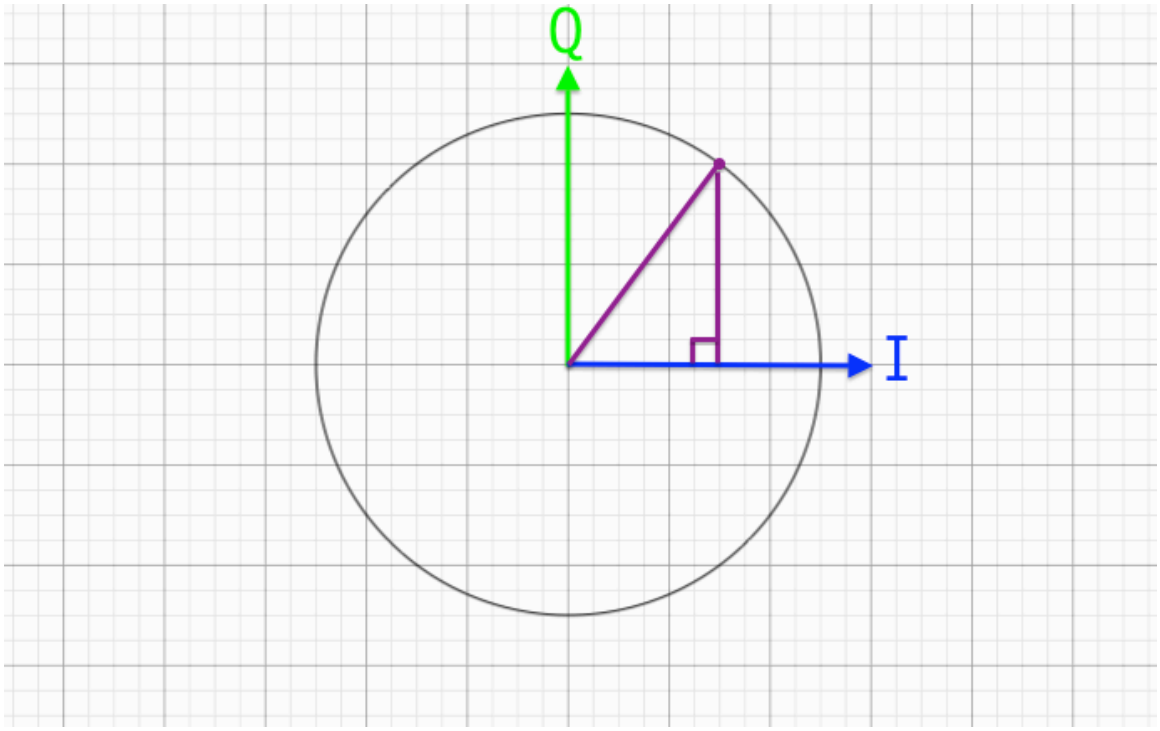


**Figure 3: Visualization of a thought experiment whereby a random point on the circumference of a circle of radius A is chosen and the Pythagorean theorem is applied.**

By the Pythagorean theorem, it must be the case that the following equation holds.

$$A = \sqrt{I^2 + Q^2}$$

### 1.2 Fast Fourier Transform (FFT)

The Fourier transform is a mathematical operation that is able to transform a function from the time domain to the frequency domain. Because computers work with samples rather than functions, a discrete Fourier transform (DFT) can be used to compute the component frequencies that exist within a signal that is being sampled. The fast Fourier transform (FFT) is an algorithm that can compute the DFT very quickly and is considered "the most important numerical algorithm of our lifetime" (Kent, 2002).

### 1.3 Band-Pass Filtering

Filtering is a technique that is used in an attempt to remove some aspect of a signal. Band-pass filtering is used to remove components of a signal above and below a certain band of frequencies. The result is that only the components of the signal corresponding to frequencies of f±α remain. A FFT view of band-pass filtering can be seen in Figure 4.

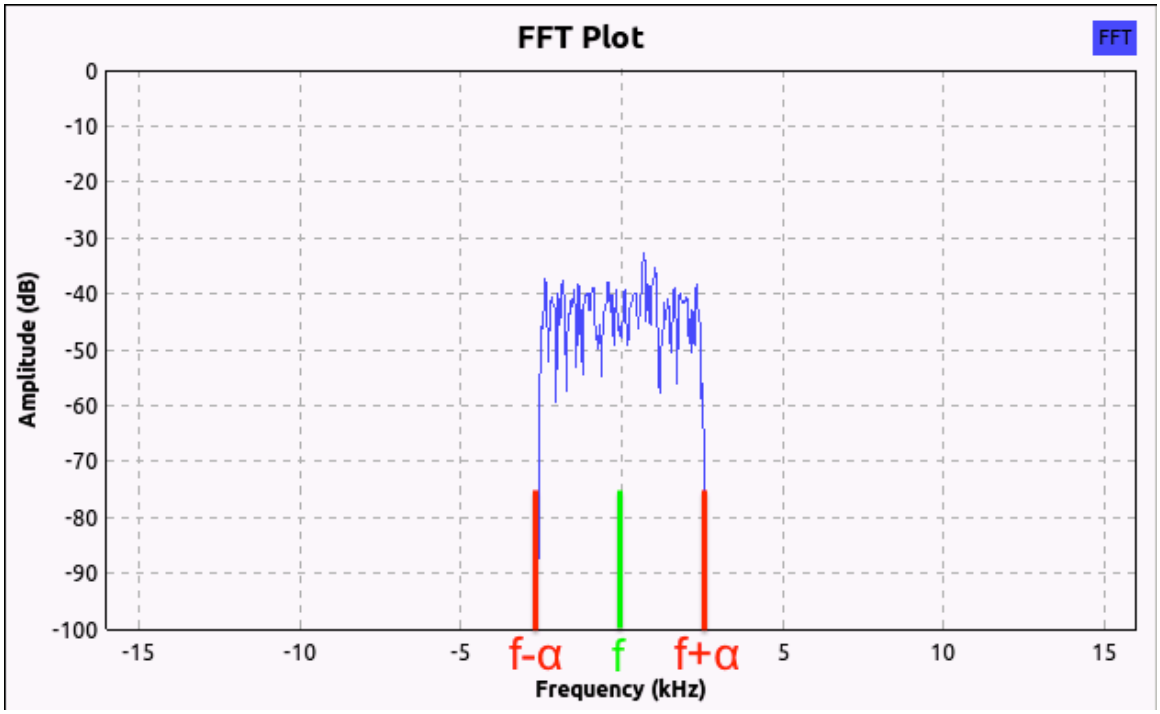


Figure 4: Illustration of band pass filtering using a FFT.

## 1.3 GNU Radio

GNU Radio is an open source software development toolkit that provides the resources to develop software defined radios. There is a multitude of hardware that can be used with GNU Radio, one example of such hardware is the Universal Software Radio Peripheral from Ettus Research LCC. One model is the USRP2 which can be seen in Figure 5.

Figure 5: USRP2 designed and sold by Ettus Research.

This piece of hardware contains two 100 MS/s 14-bit analog to digital converters which allow for sampling of a received signal, and two 400 MS/s 16-bit digital to analog converters which allow for the realization of a signal from digital samples. The hardware also comes with a Xillinx Spartan 3-2000 FPGA which can be used for signal processing on board. The USRP2 also features a gigabit Ethernet port that allows it to communicate with a computer and have the signal processing performed in GNU Radio. The elementary element of a gnu radio application is the block. There

are three main types of blocks, sources, sinks, and signal processing blocks. A source block generates samples to be consumed by signal processing blocks. One example is the file source block and its purpose is to read samples from a file and produce them. A summary of the file source block can be seen in Figure 6.

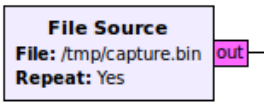


Figure 6: GNU Radio file source block summary.

A sink block consumes samples, one example of this is the file sink, which consumes samples and writes them to a file. A summary of the file sink block can be seen in Figure 7.


Figure 7: GNU Radio file sink block summary.

A signal processing block accepts samples as input, performs some operation on those samples and then produces those samples. An example of a signal processing block is the amplitude demodulation (AM) block. The AM demodulation block accepts samples as input and computes the amplitude demodulated signal as output. A summary of the AM demodulation block can be seen in Figure 8.

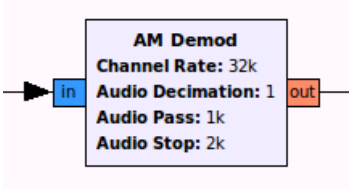


Figure 8: GNU Radio amplitude demodulation block summary.

These blocks are linked together to form a flow graph. The blocks themselves are written in C++ and the flow graphs are written in python. A tool called GNU Radio Companion (GRC) can be used to construct these blocks visually using a drag and drop user interface. An example of a flow graph created in GRC can be seen in Figure 9.

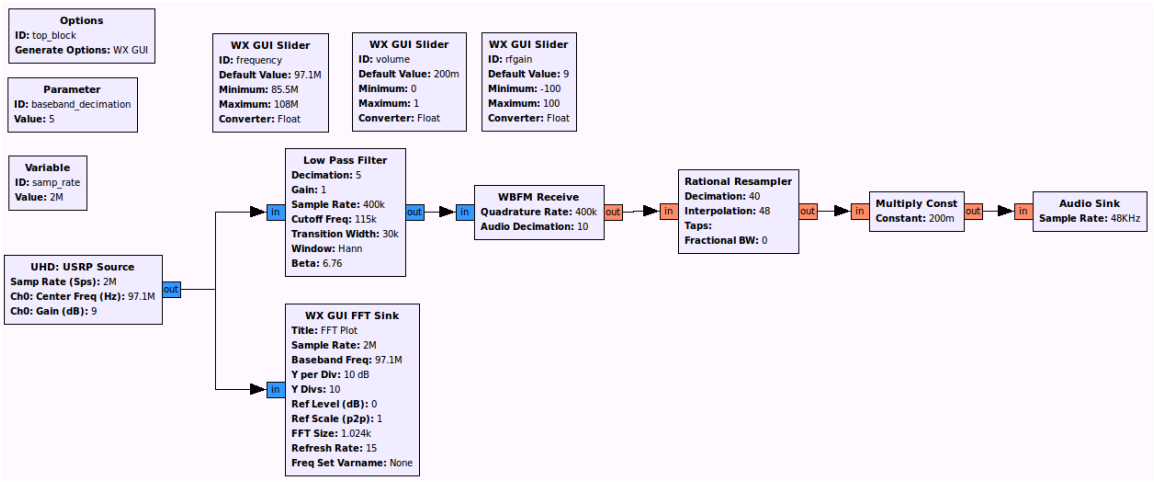


**Figure 9: Sample GRC flow graph that was constructed to listen to FM radio signals.**

# 2. Introduction to RF Remotes

## 2.1 Definition

RF remotes are typically small devices that have a few buttons on them. A press of a button on the authorized remote will invoke a particular operation on the receiver. Due to their small size, the battery must also be small, this gives rise to strict power constraints on the device.

## 2.2 Applications

RF remotes have many different applications; one such example can be seen in automobiles where remotes exist to lock, unlock, and sometimes even start the automobile. Garage door remotes exist so that users can open and close their garage door from inside their car. Home security systems sometimes have remotes that allow the user to arm and disarm their security system at a distance. Each of these remotes is designed to enable protection of things that are worth tens of thousands of dollars. Therefore it is incredibly important that these systems be secure for years to come.

## 2.3 Unidirectional Systems

One type of RF remote system is the unidirectional system. In this system, the remote is strictly a transmitter and the component protecting the resource is strictly a receiver. A system is known as a fixed code system if the transmission produced from a given button press is identical each time. There is another type of system, which is known as a “rolling code”, or sometimes “hopping code” system. In this system, we have that when a button is pressed on the remote, a unique code is transmitted. One key trait of these systems is that an observer cannot predict the next code in the sequence from the previous codes. The transmitter and receiver have previously agreed on what a valid sequence of transmissions is and is kept secret. One problem is that sometimes a remote will experience an accidental button press when out of range of the receiver, causing transmission of code number N in the sequence. When the user is within range of the receiver, they will press the remote button and code number N+1 will be transmitted while the receiver was expecting code number N. The solution to this problem is to maintain a sliding window in which the receiver will accept the next k transmissions as valid to allow for accidental button presses when away from the receiver. An illustration of the rolling code system can be seen in Figure 10.

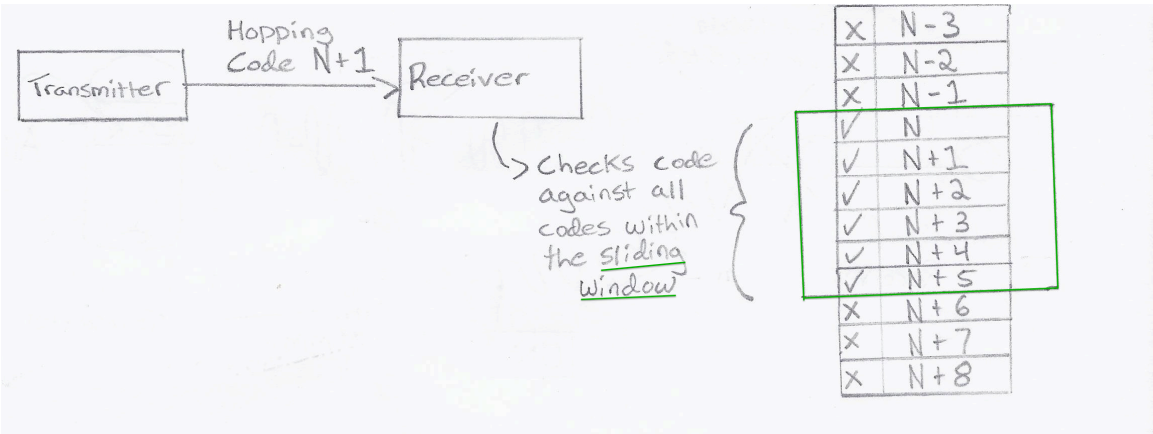


**Figure 10: Summary of a rolling code system that includes a sliding window of six codes.**

## 2.4 KeeLoq

KeeLoq is a block cipher that was purchased by Microchip Inc. for use in its rolling code RF remote systems. Due to the strict power requirements of RF remotes, this cipher was very attractive because it can be implemented in hardware using minimal power. Each remote in a Keyloq system has a unique serial number, and each manufacturer(eg. Toyota, GMC.) has a unique 64 bit manufacturer key. The unique serial number, the manufacturer key, and a seed value are used as input to a key generation algorithm to produce what is called a 64 bit crypt key. A summary of this process is shown in Figure 11.

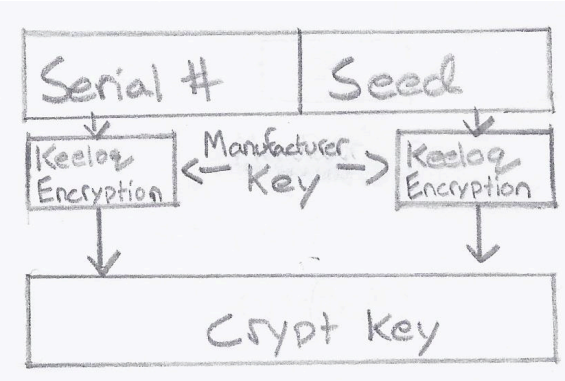


**Figure 11: High-level view of crypt key generation in a KeeLoq system.**

Each remote maintains a synchronization counter, which counts up on each button press. When a button is pressed, the crypt key is used to encrypt the synchronization counter and the desired operation code (ex. 0 for unlock, 1 for lock). This encrypted portion is sent along with the serial number of the remote to the receiver where the encrypted portion is decrypted using the crypt key and the synchronization counter is verified. If the synchronization counter falls within the sliding window, the desired operation is performed. A summary of this process can be seen in Figure 12.

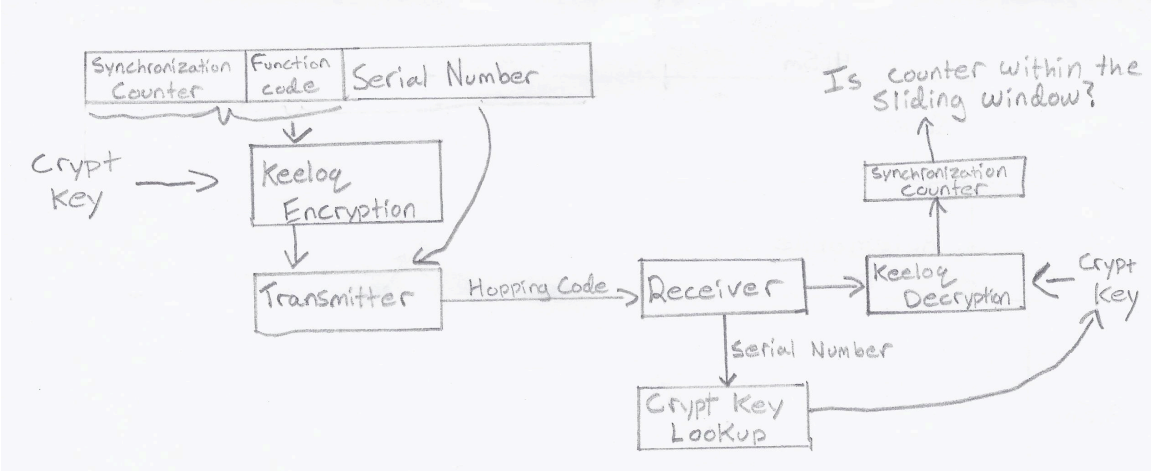


**Figure 12: Summary of a keypress operation in a KeeLoq system.**

## 2.5 Passive Keyless Entry Systems (PKES)

Passive keyless entry systems initiate the unlock process automatically without the need for a button press by the user. The remote and the entry mechanism usually use challenge-response protocol as seen in Figure 13.

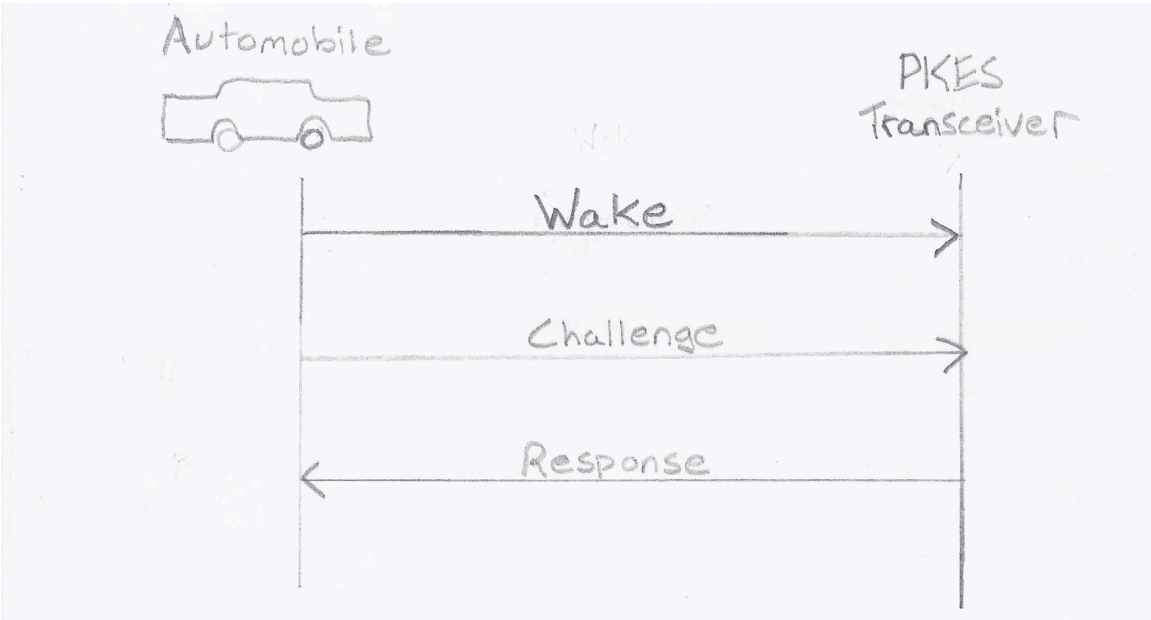


**Figure 13: A typical challenge response protocol used in passive keyless entry systems.**

Once the challenge is sent out by the automobile in this example, it is the job of the user's PKES transceiver to compute the proper response that corresponds to the challenge. If the response is computed correctly, access to the automobile is granted. It has previously been agreed upon secretly how to compute the response; therefore it is impossible for an adversary to compute the correct response from any given challenge.

# 3. Attacks on RF Remotes

## 3.1 Jam

The jam attack can be used to gain access to a multitude of different resources; the following example involving an automobile gives one such example. Alice parks her automobile, and as she is walking away, she presses the lock button on her RF remote. The car does not lock because a malicious attacker Mallory is jamming on the same frequency at which the RF remote operates. If Alice does not notice that her car remains unlocked, Mallory can gain unauthorized access to her vehicle. A summary of the attack can be seen in Figure 14.

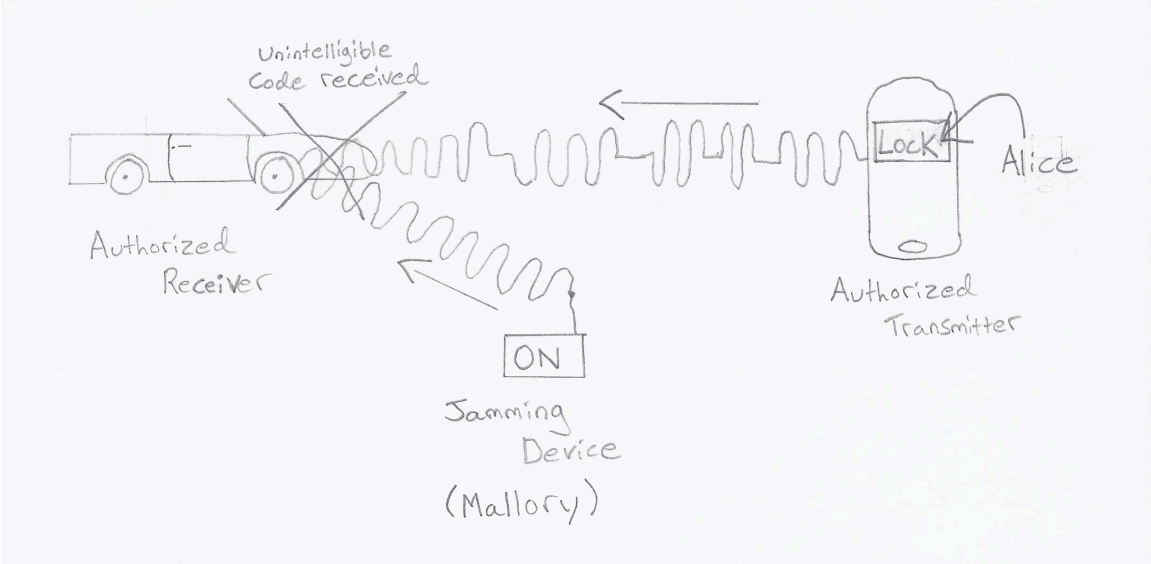


**Figure 14: Summary of the jam attack against an automobile RF remote system.**

The disadvantage of this attack is that it relies on error on the victim's part. The main advantage is that it does not require any prior knowledge of the system being attacked except for the frequency on which it operates. Another advantage of this attack is that it can be implemented in hardware or using a SDR very easily (Xu, 2005) (DeBruhl, 2011).

## 3.2 Replay

The replay attack is typically only useful against unidirectional RF remotes with fixed codes. The idea behind this attack to record a transmission from an authorized remote and then replay it again later to obtain access to the same resources that the authorized transmission allowed the authorized user access to. To give a concrete example, lets say that Alice walks out to her automobile and presses the unlock button on her RF remote, at the same time, Mallory is recording the transmission from the RF remote. The automobile unlocks and Alice obtains what she needed from inside. She then locks it, and leaves the area. Mallory then replays the unlock transmission from the RF remote, unlocking the automobile and giving her unauthorized access. A summary of this process can be seen in Figure 15.

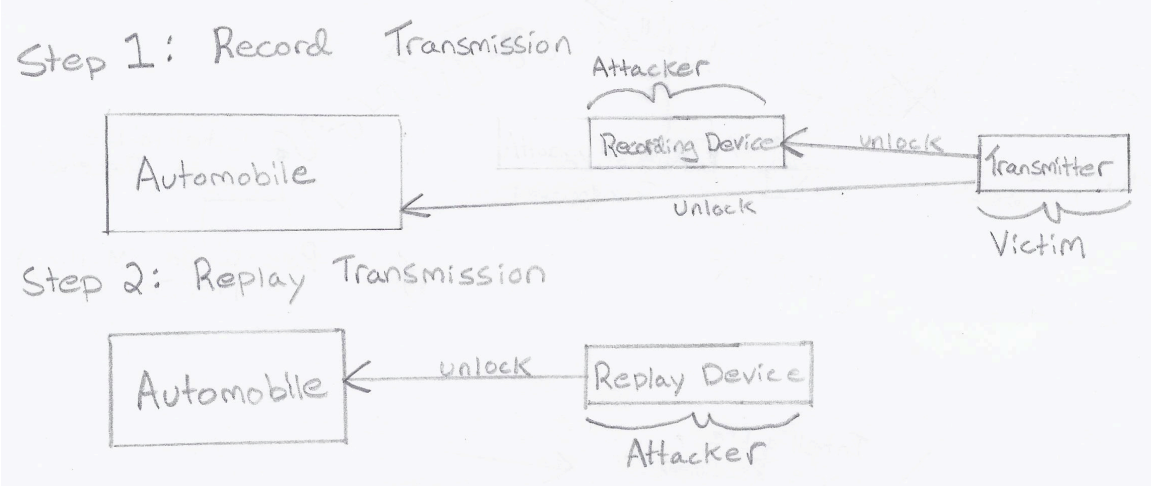


**Figure 15: Summary of the replay attack against a unidirectional fixed code automobile RF remote system.**

The advantage to this attack is that it does not rely on the victim making an error as the jam attack did. One disadvantage is that it requires that the system be using fixed code remotes, for this reason, use of fixed code systems have been discontinued in security sensitive applications for many years. That said, fixed code systems still exist and an attack can be easily carried out against them using both SDR or purpose built replay hardware (Broekhuis, 2011).

### 3.3 Jam and Replay

The jam and replay attack is a process whereby an adversary will jam the frequency on which the RF Remote operates, record the transmission from an authorized transmitter, and then replay the transmission at a later time when it is advantageous for the adversary. To give a concrete example, lets say Alice presses the unlock button on her RF remote to unlock her automobile. Nothing happens because an adversary Mallory is jamming on the same frequency as the RF remote operates. At the same time, Mallory has also recorded the transmission from the authorized remote. Alice manually unlocks her car using the key, retrieves what she needs, and manually locks it using the key. After Alice has left the area, Mallory disables the jammer and replays the authorized transmission; this unlocks the automobile, giving Mallory unauthorized access to the automobile. A summary of the jam and replay attack can be seen in Figure 16.

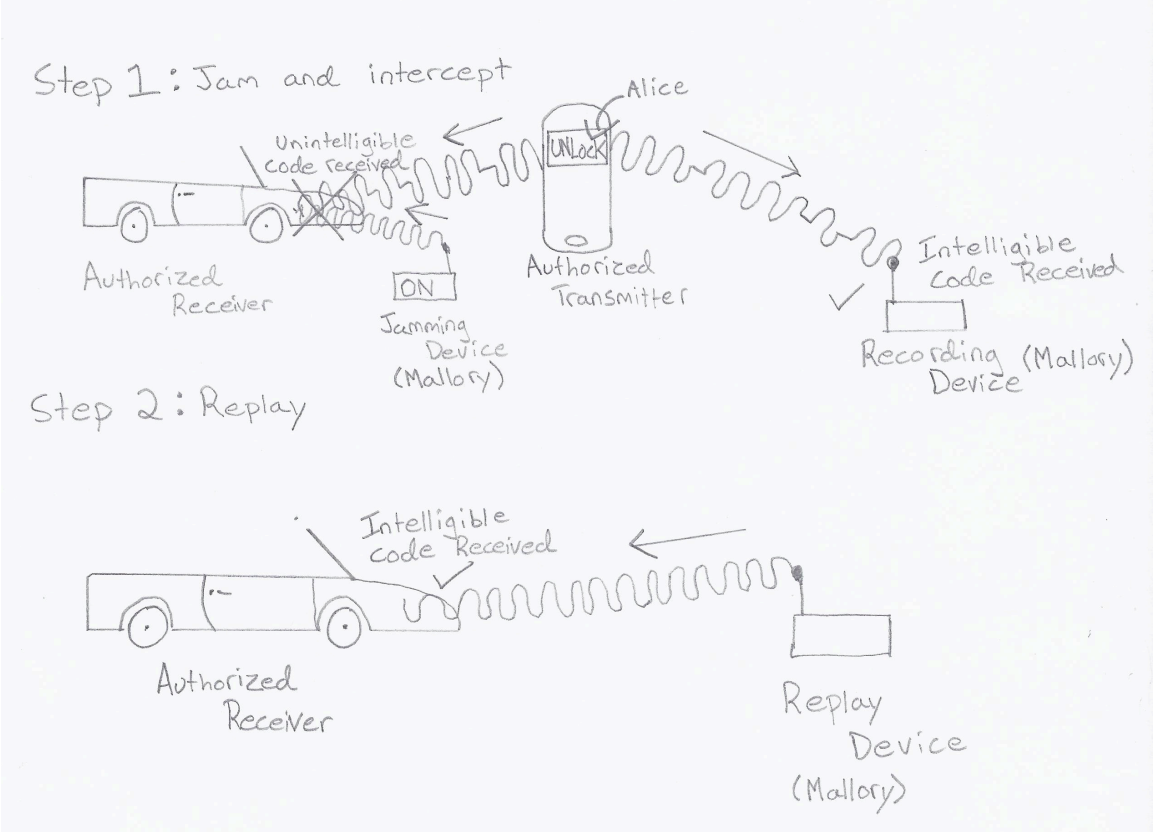


**Figure 16: Summary of the jam and replay attack involving a victim Alice, and an adversary Mallory.**

The jam and replay attack leverages the advantages of both the jam attack and the replay attack. One advantage of the attack is that Alice leaves her automobile feeling safe because she has manually locked the doors, yet Mallory can still obtain access at a later time. Rolling code security was built on the idea that it is computationally infeasible to determine the next code in the sequence from the previous code. This attack bypasses this idea entirely by intercepting the code from the authorized remote and ensuring that the receiver does not receive the transmission through jamming. Therefore this attack also works against unidirectional rolling code systems. Just like the jam attack and replay attack, no prior knowledge about the system is required except for the frequency on which it operates. This attack can be carried out using either custom hardware, or a SDR. Unidirectional rolling code remotes are still in widespread use; therefore this attack is still a threat.

### 3.4 Relay

Modern systems such as passive keyless entry systems can be circumvented via the relay attack. The idea behind passive keyless entry is that a user carries around a small transceiver and when they come within close proximity of the resource they wish to access, an exchange of messages occurs, certifying that an authorized user is within range. The relay attack works relaying messages between the two transceivers over long distances with the goal of obtaining access to a particular resource while the victim is out of range and often out of sight. A summary of this attack can be seen in Figure 17.


**Figure 17: Summary of the relay attack on passive keyless entry systems.**

This attack can be carried out using physical hardware or using two SDRs. The physical hardware attack is very simple and involves just a loop antenna that is long enough to extend from the victim to the desired resource. This gives the transceivers a good medium over which their RF signals can propagate. The end result is that the victim's transceiver is perceived to be much closer than it actually is and the attacker gains unauthorized access to the resource. This attack can also be implemented using SDRs, however current implementations have shown that there are still some latency issues to be overcome (Aurelien, Boris, & Srdjan, 2010). These issues will soon be resolved as SDRs and general purpose computing become faster.

## 3.5 KeeLoq

KeeLoq has been shown to be flawed in many different ways; there is however one flaw that even the latest KeeLoq technology does not protect against. For an adversary to clone an authorized remote, they must obtain a few different pieces of information as shown in Table 1.

**Table 1: Summary of the information required to fully clone an authorized remote under the KeeLoq system.**

| # | Information |
|---|---|
| 1 | Remote Serial Number |
| 2 | Manufacturer Key |
| 3 | Key Generation Seed |
| 4 | Crypt Key |
| 5 | Synchronization Counter Value |

The remote serial number can be obtained from one KeeLoq transmission from an authorized remote because it is sent in the clear and is fixed. The manufacturer key can be obtained via trace power analysis of the Microchip processor used in decrypting a KeeLoq transmission (Kasper, Timo, & Amir, 2009). The seed value used in key generation can be computed from just two sniffed transmissions using a brute force technique on an array of FPGA's (Novotny & Kasper, 2010). The serial number, manufacturer key, and seed can be used to compute the crypt key via the key generation algorithm. The synchronization counter value can be decrypted from just one KeeLoq transmission. A summary of these methods is summarized in Table 2.

**Table 2: Summary of the information required to fully clone an authorized remote under the KeeLoq system and methods by which the information can be obtained.**

| # | Information | Method of Determination |
|---|---|---|
| 1 | Remote Serial Number | Sniffed From One Transmission |
| 2 | Manufacturer Key | Trace Power Analysis Of Microchip Processor |
| 3 | Key Generation Seed | Brute Force On FPGA |
| 4 | Crypt Key | Key Generation Algorithm Using 1,2 and 3 |
| 5 | Synchronization Counter Value | Decrypted From One Sniffed Transmission |

Some of these methods, such a trace power analysis, require special expertise and expensive equipment. For this reason, an attack like this is not seen as a great threat.

# 4. Jam and Replay Attack Hybrid

## 4.1 Overview

The basic idea behind the jam and replay attack is that an adversary can jam the frequency on which the authorized remote operates such that the receiver never receives an intelligible transmission. At the same time, the adversary receives the transmission and saves it for a later time. When it is advantageous for the adversary, they replay the transmission, giving them access to whatever resources the original transmission from the authorized remote was intended to. The advantage to this attack is that it is possible to implement an attack using SDR that will work for almost all unidirectional remote systems, regardless of the encoding, modulation and structure of the transmission.

## 4.2 Prevention

The only defense against this type of attack that can be built into unidirectional remote systems is to have the hopping code be a function of time. This way, when the authorized remote sends out a transmission at time $T_1$, that transmission will only be valid for a fixed amount of time, say $T_1 + \varepsilon$. By making $\varepsilon$ sufficiently small, it is possible to ensure that there will not be situations where it would be advantageous for the adversary to replay the transmission at $T_1 + \varepsilon$. This type of implementation is not used in industry because it relies on the timing of the clocks within both the receiver and transmitter to be very accurate. This is often not a problem on the receiving end, but on the transmitting end, there are often power constraints that prevent a reliable clock from being maintained over time. The best solution to this problem would be to not use a unidirectional system at all and to use a bidirectional challenge-response system. This type of system is not susceptible to jam and replay attacks because when each challenge is dispatched, there is a very small window of time during which a response is valid.

## 4.3 Pitfalls

One problem at hand is that when the adversary jams the frequency on which the authorized remote operates, they are also making it such that the signal received by them is unintelligible. There are three ways that an adversary can overcome this problem.

### 4.3.1 Solution: Strategic Positioning

One method would be for the adversaries to position themselves closer to the victims remote than to the jammer and receiver as depicted in Figure 18.

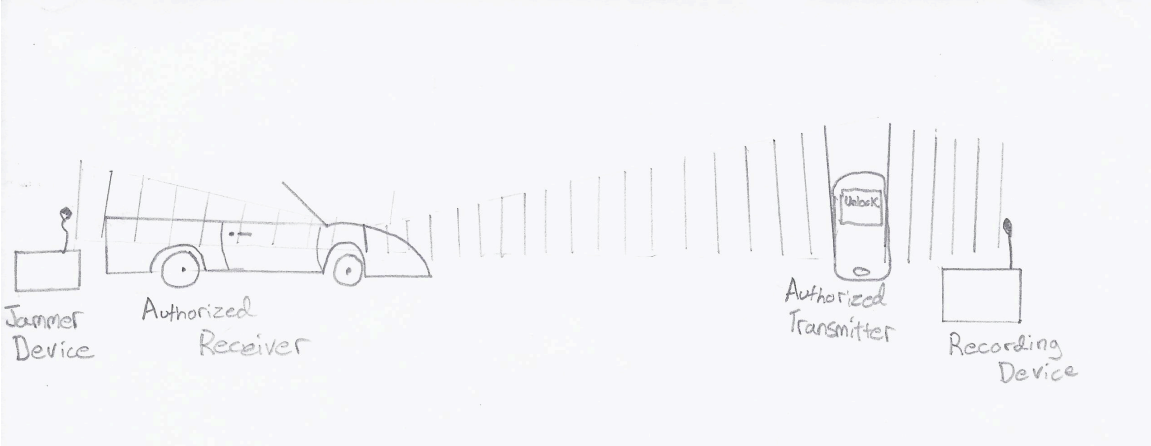


**Figure 18: Strategic positioning of the adversary to avoid jamming signal interference.**

The idea is that the jamming signal strength will be much lower for the adversary, than for the authorized receiver. This way the adversary will be able to capture a valid transmission, but the authorized receiver will not.

### 4.3.2 Solution: Directional Antennae

Another method would be to use a directional antenna such that only the desired signal from the authorized remote will be received. This would involve using a Yagi-Uda antenna on the jammer to direct the jamming signal towards the authorized receiver (Yagi, 1928). An additional Yagi-Uda antenna would positioned orthogonally to the jamming antenna so that the adversary can receive the transmission from the authorized remote as displayed in Figure 19.

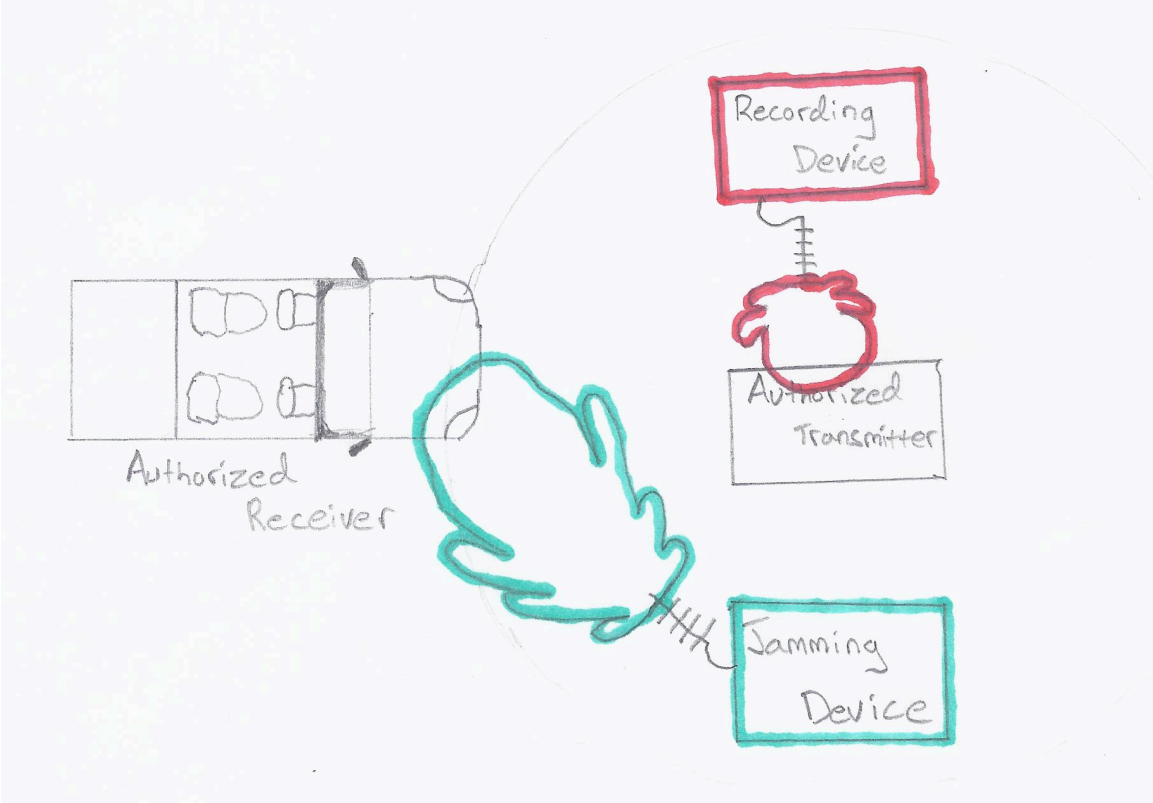


**Figure 19: Configuration of directional antennas to aid the adversary in collecting the authorized transmission. Theoretical radiation patterns have been overlaid to show the advantage provided.**

### 4.3.3 Solution: Band-pass Filtering of Authorized Transmitter Signal

The RF remotes used in these types of systems often employ low quality oscillators in order to keep costs down. This means that any given remote may not operate at its theoretical frequency $f_t$, it is more likely that the remote actually operates at a frequency that is reasonably close to $f_t$ which can be denoted by $f_a = f_t \pm \alpha$. As a result of these variances, the authorized receiver is usually designed to allow any signal that falls within a certain range of possible frequencies $f_t \pm \beta$. This fact can be used to the adversary's advantage by jamming on one frequency within the authorized receivers range, say $f_t + \beta$ and then applying a band-pass filter to only allow reception of frequencies in the range $f_t \pm \alpha$. This way the adversary only obtains the signal from the authorized transmitter and no noise from the jammer. A visualization of this band-pass filtering can be seen in Figure 20.

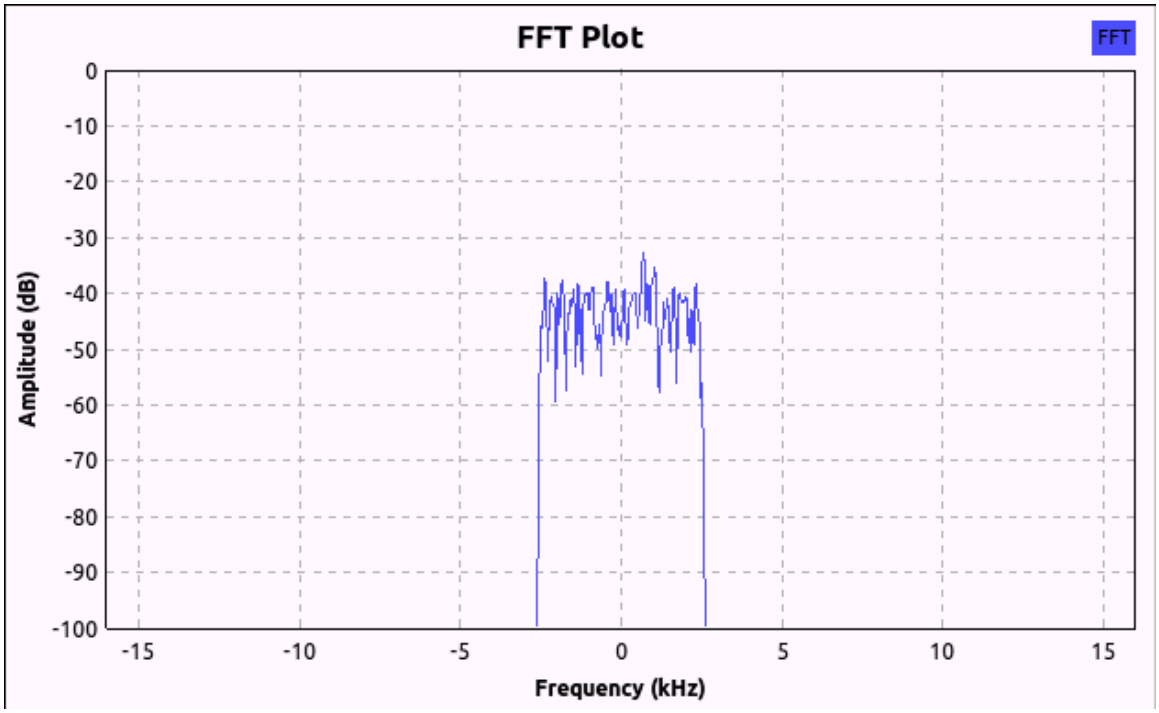


Figure 20: FFT visualization of band-pass filtering to only receive the signal from the authorized remote.

## 4.4 Hybridization

The jam and replay attack can be carried out completely with SDR however the cost of a SDR is very high when compared to the resources that might be protected by a unidirectional remote system. The frequencies used by these remotes are typically either 315MHz or 433MHz. A SDR capable of transmit and receive operations at these frequencies is the USRP2 with a WBX daughterboard. The total cost of this system would be approximately $2045 as shown in Table 3.

Table 3: Approximate costs associated with a pure SDR implementation of the jam and replay attack.

| Item | Cost (CAD funds) |
|---|---|
| USRP E100 SDR | $1495 |
| WBX Daughterboard | $550 |
| **Total** | **$2045** |

Alternatively a hybrid system could be constructed that utilizes an inexpensive SDR for receiving, and additional inexpensive electronic components for jamming and replaying the signal. An upper bound on the cost of such a system is given in Table 4.

Table 4: Approximate upper bound on the costs of carrying out a hybrid jam and replay attack.

| Item | Cost (Canadian Funds) |
|---|---|
| RTL-SDR | <$10 |
| Microcontroller | <$10 |
| 315Mhz Transmitter | <$10 |
| Supporting Components | <$20 |
| Total | <$50 |

By the cost data found in Table 3 and Table 4, the pure SDR implantation would be approximately 40 times more expensive than the hybrid implementation. There are however disadvantages to the hybrid implementation such as the fact that it is specific to the system being attacked.

# 5. Reverse Engineering a RF Remote System

## 5.1 Initial Reconnaissance

The RF system that the attack will be carried out on was initially a black box, the exact details of how the system works had to be reverse engineered. The back of the remote being used in the system can be seen in Figure 21.

Figure 21: Picture of the back of the RF remote for the system being attacked

The FCC ID of the remote can be seen in Figure 21, and then used to query the FCC database for more information about the device (Commission, 2013). The query revealed that the remote operates on 315.0 Mhz. This remote control belongs to an automobile from 2004.

## 5.2 Modulation Identification via GNU Radio

The next step was to construct a GNU Radio flow graph that would allow observation of the activity of the remote at 315MHz. A simple flow graph consisting of a USRP source, a slider to adjust the frequency of reception initially configured to 315MHz, a FFT sink, and an audio file sink was constructed as shown in Figure 22.

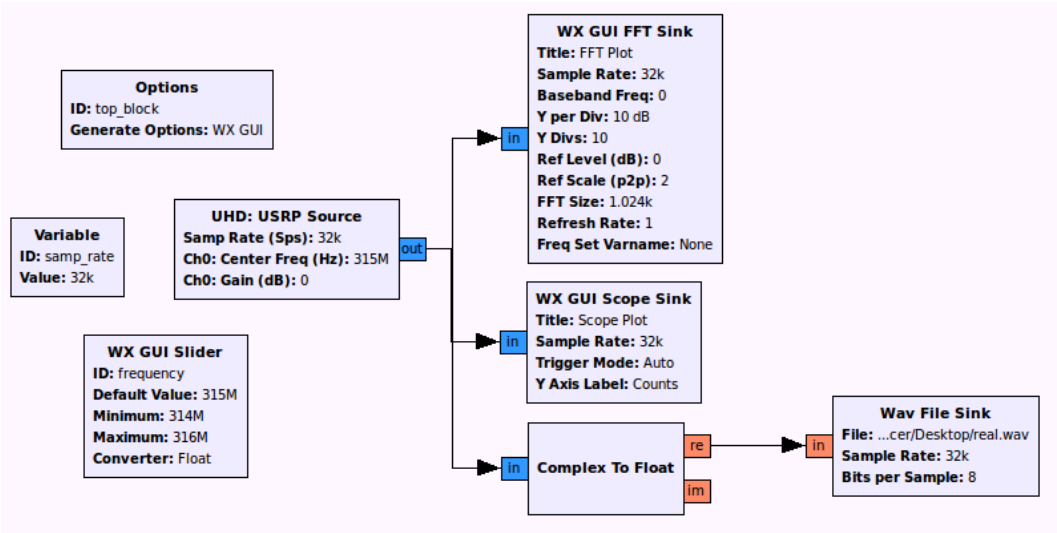


**Figure 22: Flow graph constructed to gather more information on the system being attacked.**

The hardware configuration being used throughout this project is a USRP2 with a TVRX daughterboard installed. The USRP2 communicates with a Macbook Pro which is running Ubuntu 12.10 Linux using an Ethernet connection (Ubuntu, 2013). GNU Radio 3.70 is being used in this case. The hardware configuration can be seen in Figure 23.

**Figure 23: Software Defined Radio hardware configuration.**

The flow graph was executed and a few seconds later, the lock button on the remote was pressed. No large peaks were observed on the FFT. The frequency was adjusted to 314.89 MHz and the lock button was pressed again. This time, a large peak was observed on the FFT indicating activity as seen in Figure 24.

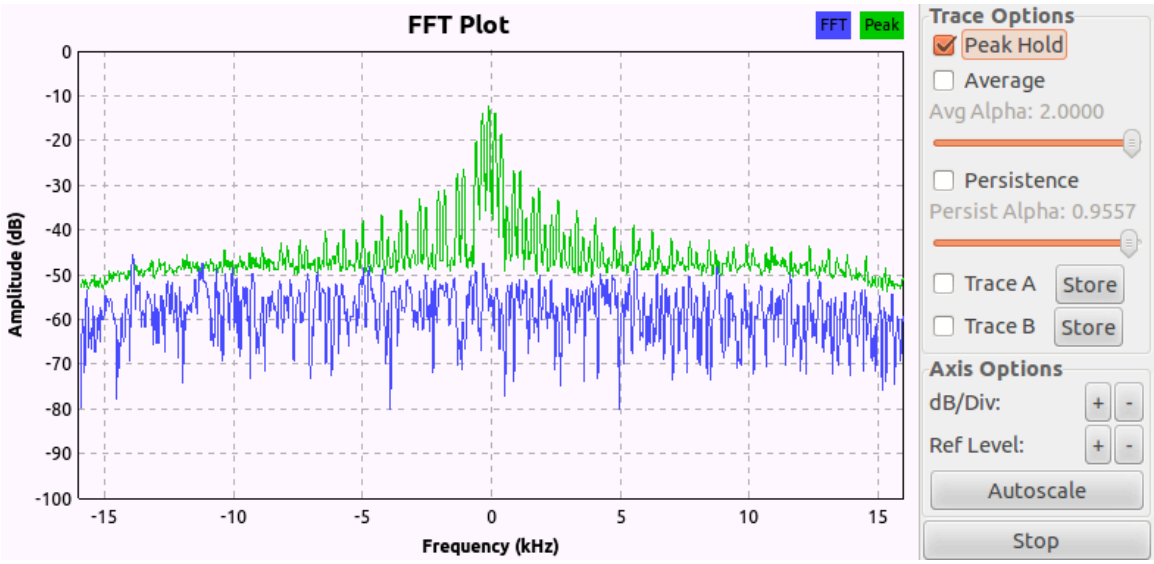


**Figure 24: Peak observed on the FFT during the press of the lock button on the remote.**

This indicated that the frequency at which the remote is actually operating at was 314.89 MHz. At this point in time, it was unknown whether or not the RF system was unidirectional, or bidirectional. The automobile was moved out of range of the remote and the flow graph was executed. After a few seconds, the lock button on the remote was pressed within a 1 m range of the USRP and then the flow graph was terminated. The audio file produced via the audio sink was visualized with audio editing software titled Audacity as shown in Figure 25 (Audacity, 2013).

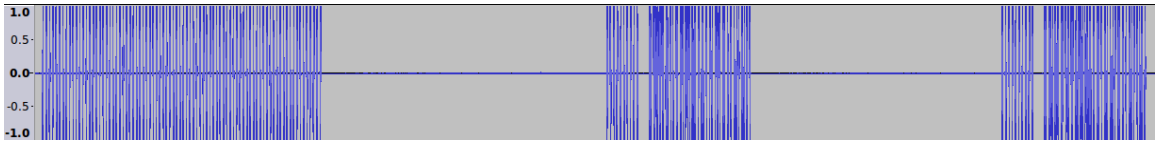


**Figure 25: Audacity visualization of the activity at 315MHz when the lock button was pressed with the automobile out of range.**

The same experiment was repeated again with the automobile within range to obtain the Audacity visualization shown in Figure 26.

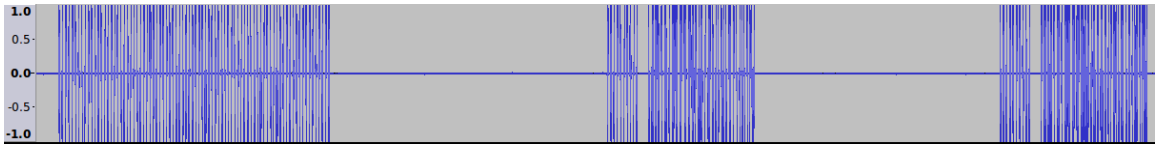


**Figure 26: Audacity visualization of the activity at 315MHz when the lock button was pressed with the automobile within range.**

As can be seen in Figure 25 and Figure 26, the structure of the traffic at 315MHz was similar whether or not the automobile was within range or not. This indicated that the system being analyzed might be unidirectional. The waveform produced by the remote can be analyzed further by zooming in as seen in Figure 27.

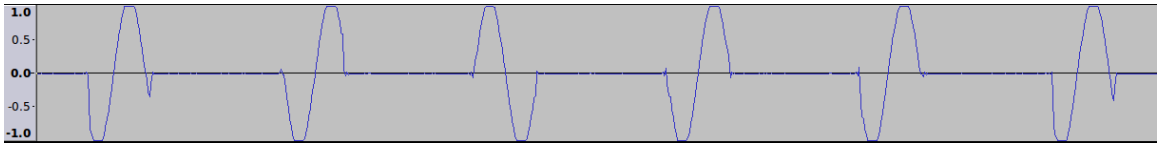


**Figure 27: Magnified portion of the waveform produced by the remote as visualized using audacity.**

From this waveform, it is clear that the frequency is constant, and the amplitude of the wave takes on only two discrete values (high and low for our purposes). This is an indication that some form of amplitude modulation is being used. The flow graph shown in Figure 22 can be modified by the addition of an amplitude demodulation block as shown in Figure 28 to recover the original signal before amplitude modulation.

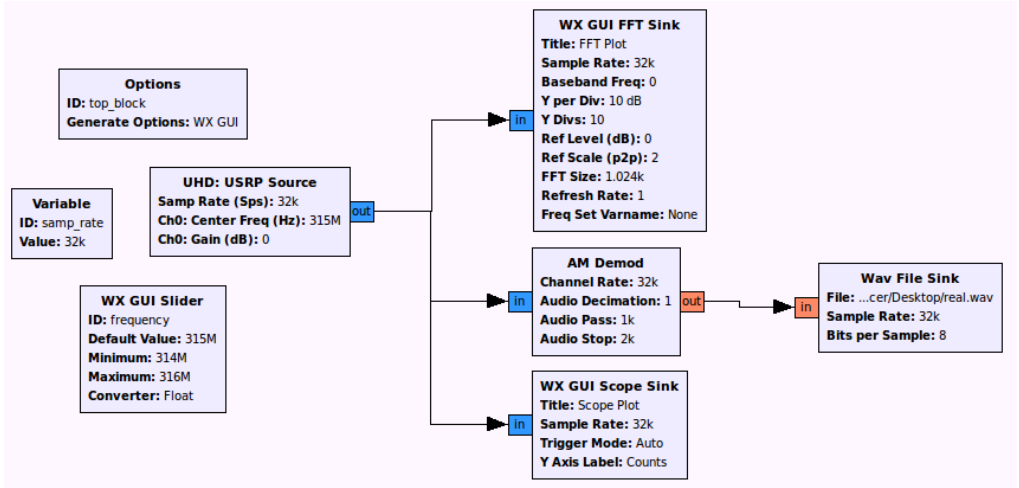


**Figure 28: Modified version of the flow graph depicted in Figure 22 to include an amplitude demodulation block.**

The flow graph depicted in Figure 28 was executed again and the remote button pressed to obtain the original waveform as depicted in Figure 29.

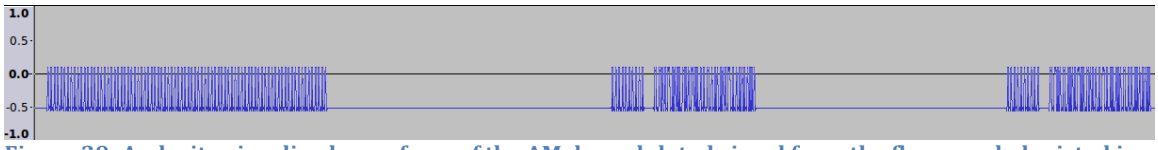


**Figure 29: Audacity visualized waveform of the AM demodulated signal from the flow graph depicted in Figure 28 during a remote button press.**

Zooming in on the waveform shows the possibility of sections of binary encoded data as seen in Figure 30.

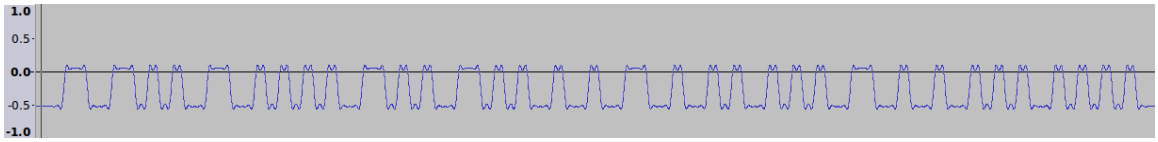

Figure 30: A portion of the amplitude demodulated signal shown in Figure 29.

This portion of the signal displayed in Figure 30 seems to indicate that the data is being encoded by some form of Manchester encoding because there appears to be only four symbols (high, low, high-high, low-low). Some additional information can be derived from the Audacity visualizations if Manchester encoding is assumed and are summarized in Table 5.

Table 5: Summary of the characteristics of the remote observed using the SDR and visualized using Audacity in the order they were observed.

| Observed Pattern | Length |
|---|---|
| **33% Duty cycle** | 250 Time Periods |
| **Silence** | 258 Time Periods |
| **33% Duty Cycle** | 28 Time Periods |
| **Silence** | 10 Time Periods |
| **Variable Data 1** | 32 Manchester Encoded Bits |
| **Fixed Data 1** | 33 Manchester Encoded Bits |
| **Variable Data 2** | 2 Manchester Encoded Bits |
| **Silence** | 258 Time Periods |
| **Variable Data (Matches Variable Data 1)** | 32 Manchester Encoded Bits |
| **Fixed Data (Matches Fixed Data 1)** | 33 Manchester Encoded Bits |
| **Variable Data (Matches Variable Data 2)** | 2 Manchester Encoded Bits |

## 5.3 RF Remote Internals

More information about the remote itself can be obtained by disassembling it and identifying the components used in its construction. The internal design of the remote can be seen in Figure 31.

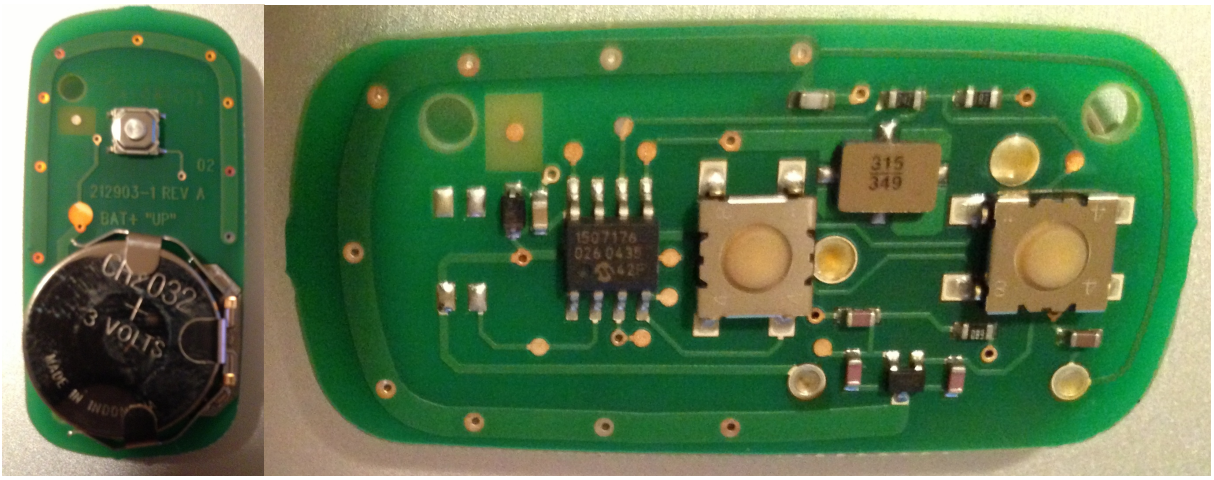


**Figure 31: Internal circuitry (back and front) of the remote used in the system in which the attack will be carried out on.**

There is one main chip on the front of the board labeled 1507176-026 0435 as seen in Figure 31. Unfortunately, a query of the popular datasheet databases for the main chip yielded no results (AllDataSheet, 2013) (Datasheets: Electronic parts info, 2013). This may have been an attempt by the manufacturer to obtain some security through obscurity. A closer look at the main chip on the board reveals the Microchip logo, indicating that Microchip Technology Inc. is the manufacturer of the chip. Microchip owns the rights to the KeeLoq technology and produces a line of unidirectional RF remotes with product codes starting with the letters HCS. A query of the datasheet database for all HCS series chips manufactured by Microchip yields one chip that matches the characteristics observed using the SDR as shown in Table 5 (HCS361 Datasheet, 2002). The HCS 361 from Microchip uses KeeLoq technology with a 32 bit hopping code, 28 bit serial number, a 1 bit low battery warning, and a 2 bit CRC. The 32 bit hopping code varies from transmission to transmission, while the 28 bit serial number remains fixed. Therefore, any transmission can be fully described and reproduced with just 67 bits. According to the HCS 361 datasheet, there are two methods in which data can be encoded, PWM and VPWM. Looking at the definition of VPWM from the datasheet reveals that it is just really Manchester encoding.

# 6. Jam and Replay Hybrid System Design

## 6.1 Jammer

A very simple and cost effective jammer can be designed with only a few components. In this case one will be constructed from a lithium ion battery, a 315Mhz AM Transmitter module, and a switch. The layout of the components can be seen in Figure 32.

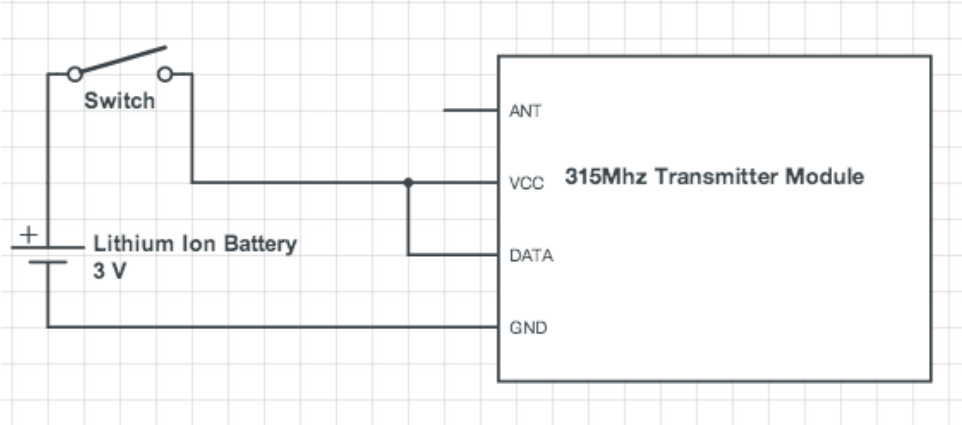


Figure 32: 315MHz jammer schematic based off of a 315Mhz AM Transmitter module.

The actual cost of this design is very small, as can be seen from the bill of materials listed in Table 6.

Table 6: Cost of the components seen in the jammer design of Figure 32.

| Item | Supplier | Cost(Canadian Funds) |
|---|---|---|
| 315 MHz AM Transmitter Module | Canada Robotix | $3.99 |
| SPDT Slide Switch | Canada Robotix | $0.79 |
| Lithium Ion Battery (3.71V - 1600mah) | Ebay | $2.99 |
| **Total** | | $7.77 |

## 6.2 Transmission Interception and Interpretation

The signal from the authorized transmitter can be intercepted with the use of an USRP2 and GNU Radio. The signal will have to be decoded using custom signal processing blocks so that it may be replayed later. The exact details of the transmission protocol are well defined in the HCS 361 datasheet (HCS361 Datasheet, 2002). Each part of the transmission will be handled by parameterized blocks whose descriptions will follow.

### 6.2.1 Duty Cycle Identification block

The 33% duty cycle that begins each transmission from the HCS 361 was originally designed to aid in waking up receiving hardware. It will be used in this context to mark the beginning of the transmission. A GNU Radio block will be developed that is able to identify duty cycles of arbitrary length, percentage and trailing silence.

Before the block sees the duty cycle, it a does not allow samples to pass through. After the duty cycle has been processed by the block, it allows samples to pass through it until a parameterized amount of silence has been seen. This block operates by keeping track of the amount of samples that are above a certain threshold(high), and the number of samples that are below a certain threshold(low). When a parameterized number of peaks have been seen, the block computes the percentage of time the samples were high (the percentage of the duty cycle). If this percentage is within an error value of the parameterized duty cycle percentage, the duty cycle is accepted and samples pass through the block until a parameterized length of silence has been seen. Since the signal being given as input to the block is assumed to already have been AM demodulated, the block accepts floating point samples as input. A summary of the block and its parameters can be seen in Figure 33.

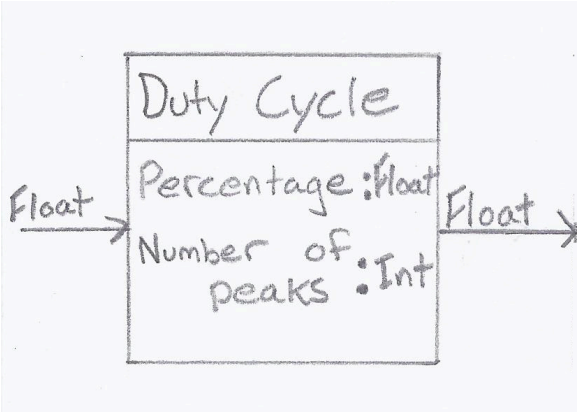


**Figure 33: Summary of the duty cycle identification block and its parameters.**

The block itself is really a finite state machine which is visualized along with it's transitions in Figure 34.

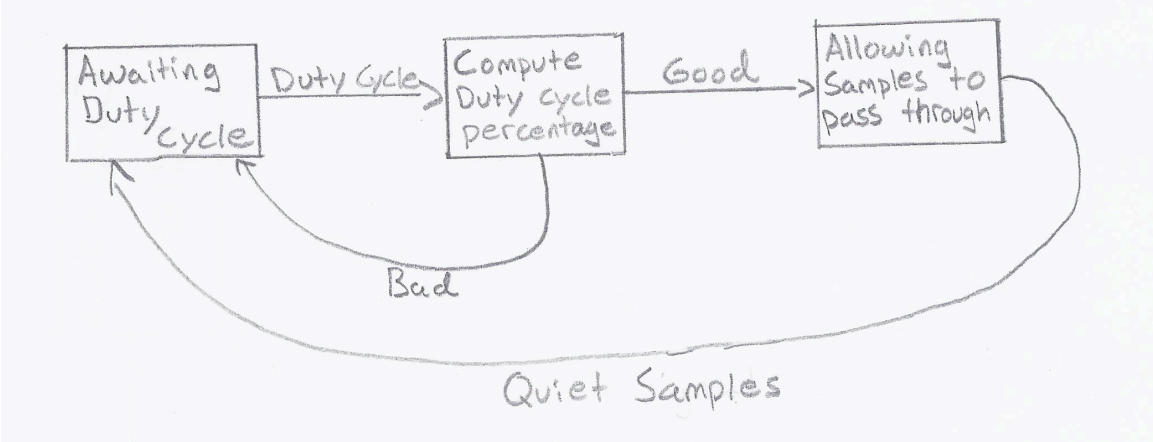


**Figure 34: Finite state machine description of the duty cycle identification block.**

### 6.2.2 Sync Header Identification Block

According to the HCS 361 datasheet, the sync header is an additional 33% duty cycle and period of silence that occurs right before the 67 bits of data is transmitted. In

order for Manchester encoded data to be decoded later on, there must be some definition of how long one time period is. The purpose of the sync header is to provide this definition. The sync header is known to be 28 time periods long by looking at the datasheet. Therefore, if the sync header block computes the length of the header and divides by 28, it will obtain the length of a single time period. A generalized version of the sync header block will be constructed that takes the length of the sync header, silence and duty cycle as a parameter. The sync header block does not actually decode any of the Manchester encoded data itself, the responsibility is left to the next block in the chain. The advantage of this design is that the sync header block can be reused if another remote uses say PWM encoded data instead. Once the elementary time period has been found by the sync header block, how does it notify the block that is actually performing the data decoding? The answer is stream tagging. GNU Radio provides a channel running parallel to the stream of samples that allow blocks to attach metadata to the samples. Therefore, when the sync header identification block determines the elementary time period length, it adds a tag to the stream of samples so that the next block in the chain will receive the tag and be able to decode the data. A summary of the block and it's parameters can be seen in Figure 35.

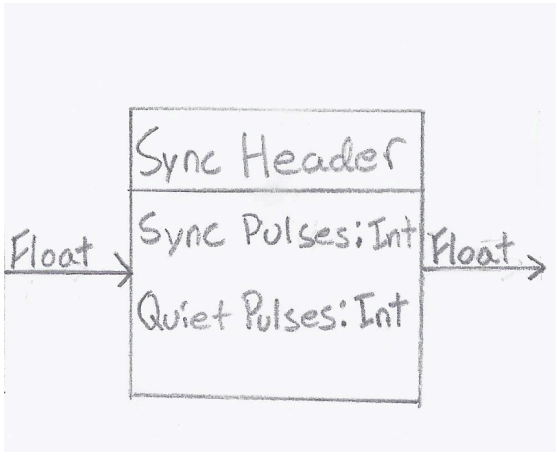


**Figure 35: Summary of the sync header identification block and its parameters.**

This block can also be expressed as a finite state machine as seen in Figure 36.

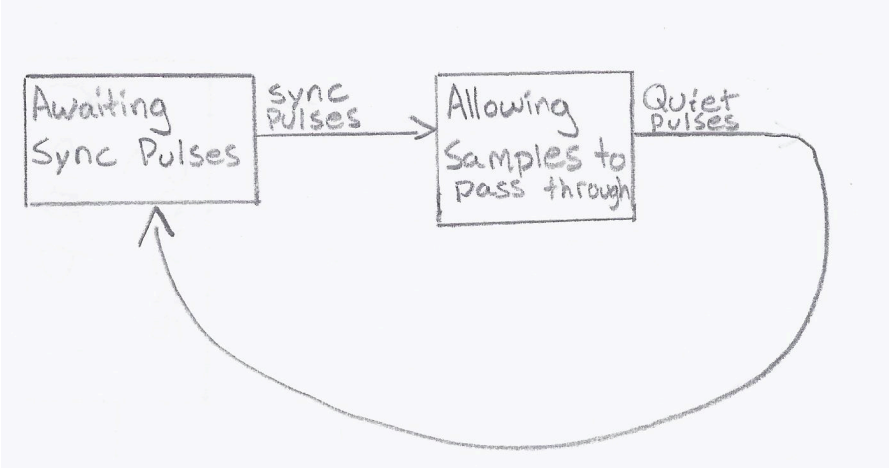


Figure 36: Finite state machine description of the sync header identification block.

### 6.2.3 Manchester Decoding Block

There are 67 bit's of Manchester encoded data as per the HCS 361 datasheet (HCS361 Datasheet, 2002). To decode these bits a signal processing block will be required. The block will take as its input a stream of samples and output binary data. One requirement is that it receives a stream tag notifying it of the length of one time period. Once this has occurred it will begin to output a stream of binary data until the input stream no longer contains Manchester encoded data. If the stream is silent for more than two time periods, this is a good indication that there is no more data because a Manchester signal is quiet for at most two time periods. The block can also be represented as a finite state machine as seen in Figure 37.

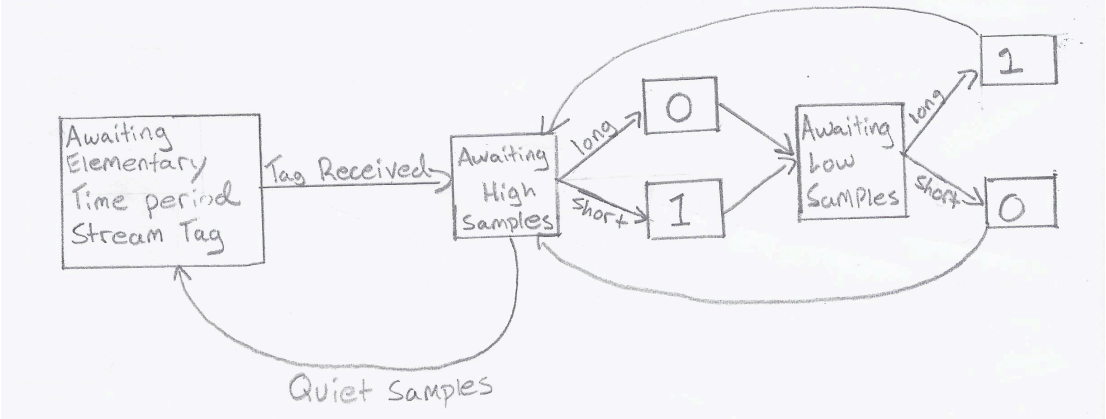


Figure 37: Finite state machine representation of the Manchester decoding block.

### 6.2.4 Interception Flow Graph

The interception process is broken up into two parts. During the first part, the signal will be recorded to a file sink using a high level design as seen in Figure 38.

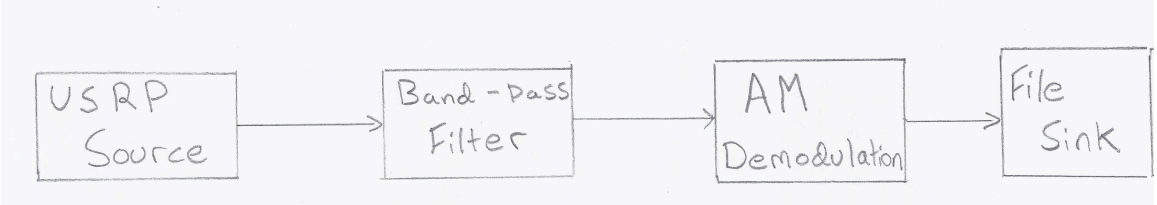


Figure 38: High-level overview of the transmission recording flow-graph.

In the second part, each of the signal processing blocks that are being designed must be linked together to extract the 67-bit sequence that represents the transmission as shown in Figure 39.

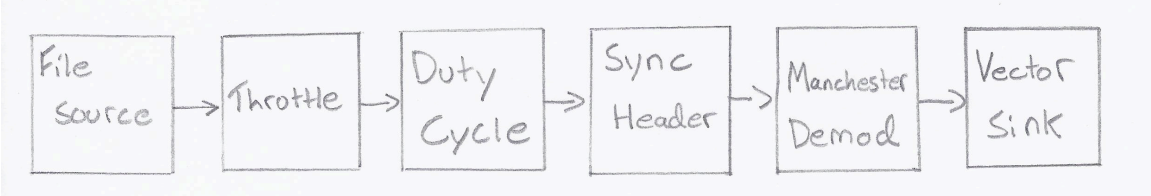


**Figure 39: Flow graph designed to allow decoding of the 67-bit code from the HCS 361.**

The output from this flow graph is the 67-bit code that was transmitted by the authorized transmitter, this will then be transferred to the replay hardware via a serial connection.

## 6.3 Replay Hardware

To do replay of the captured signal without the use of a SDR, a 315MHz AM transmitter will be required, along with a microcontroller to instruct the transmitter on how to reproduce the signal. Each transmission that is decoded by the signal processing blocks of this project produces a 67-bit representation of that transmission. In order to transfer this representation from the PC to the microcontroller, a serial connection will be used. A schematic of the board that will be constructed can be seen in Figure 40.

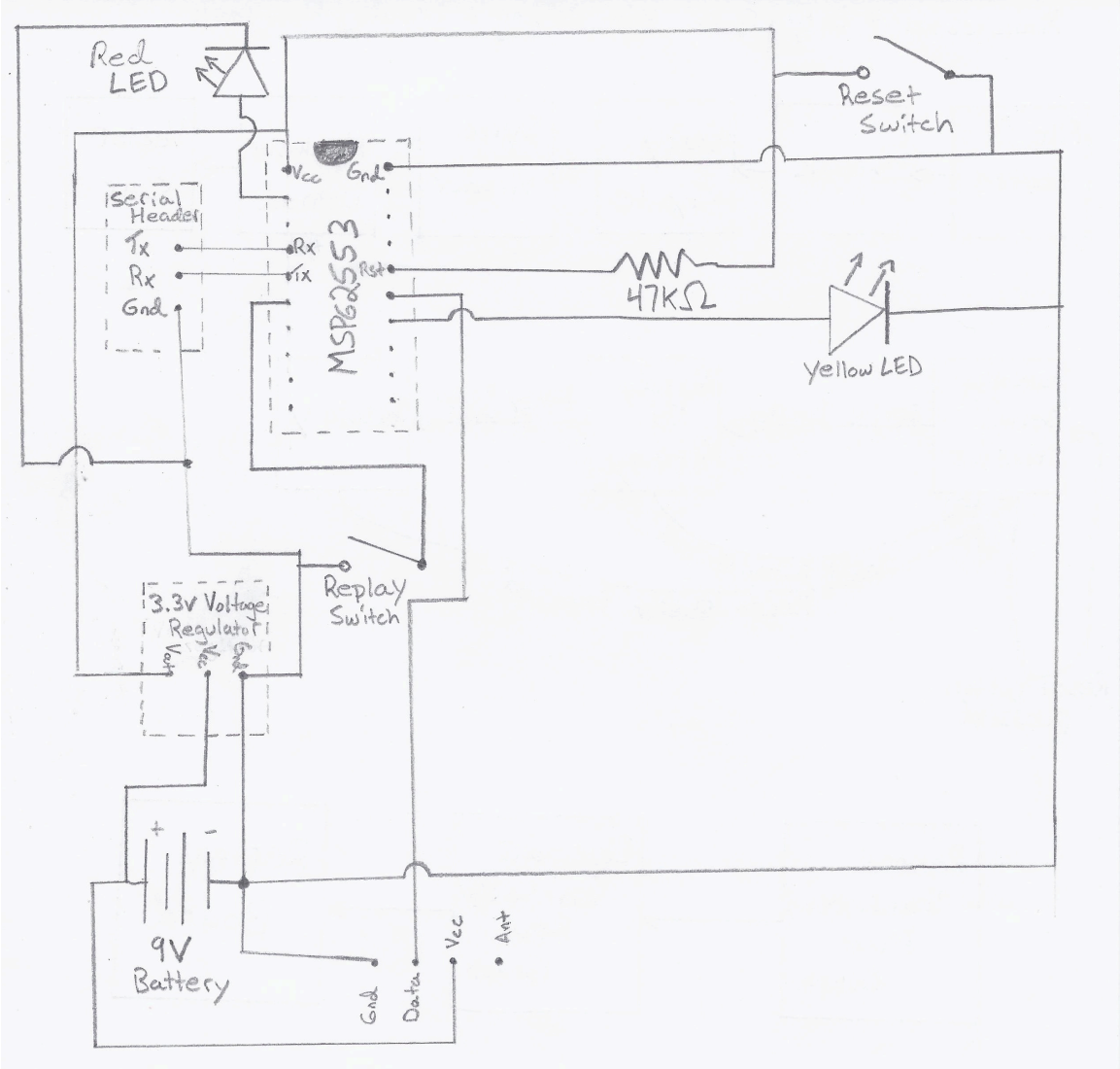


**Figure 40: Schematic of the board that will be used to carry out the replay portion of the attack at 315MHz.**

**The MSP430 from Texas Instruments has been selected as the microcontroller for use in the replay use in the replay attack due to its low cost and low power consumption. This microcontroller is microcontroller is connected to the same 315MHz AM transmission module that was used in the jammer was used in the jammer design. To transfer the 67 bit representation of the transmission to the MSP430, transmission to the MSP430, a USB 2.0 to TLL UART module will be used. A detailed bill of parts for the bill of parts for the replay hardware can be found in**

Table 7.

Table 7: Bill of parts for constructing the replay hardware shown in the schematic of Figure 40.

| Item | Supplier | Cost (Canadian Funds) |
|---|---|---|
| 315 MHz AM Transmitter Module | Canada Robotix | $3.99 |
| LEDx2 | Canada Robotix | $0.60 |
| Momentary Pushbutton x 2 | Canada Robotix | $0.98 |
| Voltage Regulator (3.3V, 0.8A) | Canada Robotix | $0.99 |
| 47kΩ Resistor | Canada Robotix | $0.10 |
| MSP430 Launchpad Value Line Development Kit | Texas Instruments | $4.30 |
| Multi-purpose PC Board | The Source | $4.99 |
| USB 2.0 To TTL UART Module | Ebay | $7.25 |
| 9V Battery | Walmart | $2.99 |
| Plastic Enclosure | Polycase | $3.49 |
| **Total** | | **$29.68** |

There will be a red and yellow LED on the board. In the initial state, the red LED will be lit, indicating that the board is still waiting for a serial transmission. When the 67-bits of transmission information has been received, the light changes its color to yellow, indicating that it is waiting to replay. When the replay pushbutton is pressed, the replay process will begin and the transmission will be made every five seconds indefinitely. By pressing the reset button, the board returns to the initial state of waiting to be programmed.

## 6.4 Replay Software

The replay hardware contains a MSP430 microcontroller which will be responsible for accepting a 67-bit transmission information block serially, as well as replay the transmission corresponding to that information. As previously stated, there are two states in which the board can be. A high level overview of the replay software operation can be seen in Figure 41.

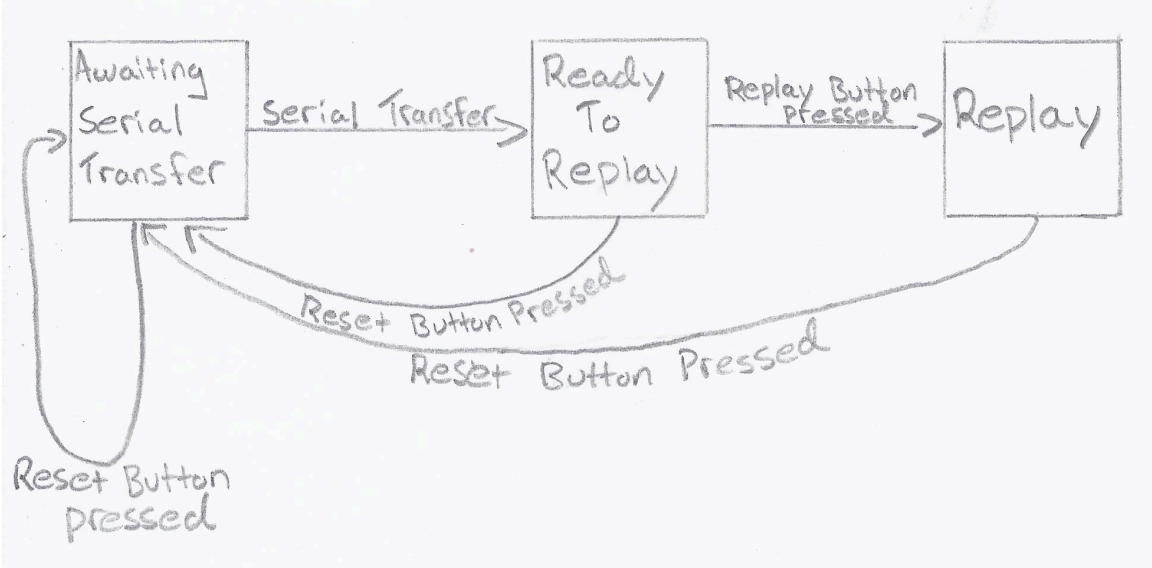


**Figure 41: High-level overview of the operation of the software executing on the replay board.**

## 6.5 System Overview

A high level overview of the components of the system can be seen in Figure 42.

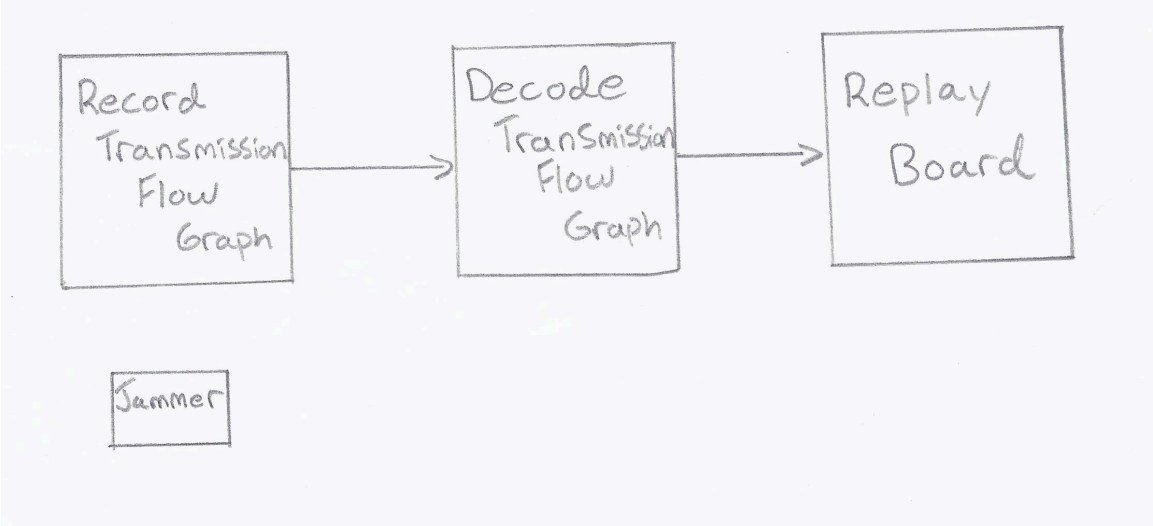


**Figure 42: High-level overview of the components of the hybrid jam, intercept, and replay system**

# 7. Jam and Replay Hybrid System Implementation

## 7.1 Jammer

Before the jammer shown in Figure 32 was finalized, it was prototyped on breadboard using jumper wires as shown in Figure 43. This design included additional components such as an antenna and a 9V battery.

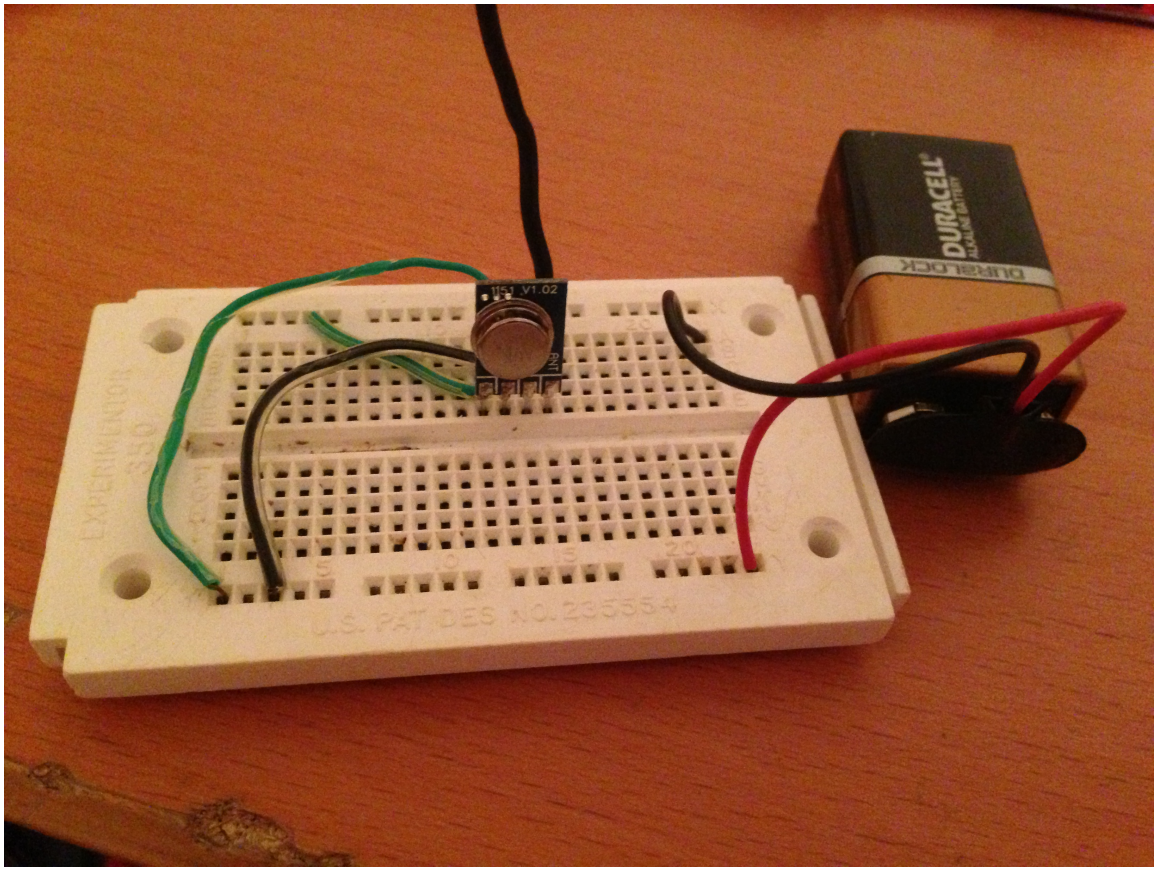


**Figure 43: Prototype jammer laid out on breadboard.**

This design was found to be overkill as it not only stopped entry to the target automobile, but also automobiles within a radius of approximately 10m. For this reason, the antenna was removed from the design and the battery was switched from 9V to 3.71V. The new lithium ion battery is rechargeable via the USB mini B connector. The components of the final device were soldered together with minimal connecting wires to create a low profile product.

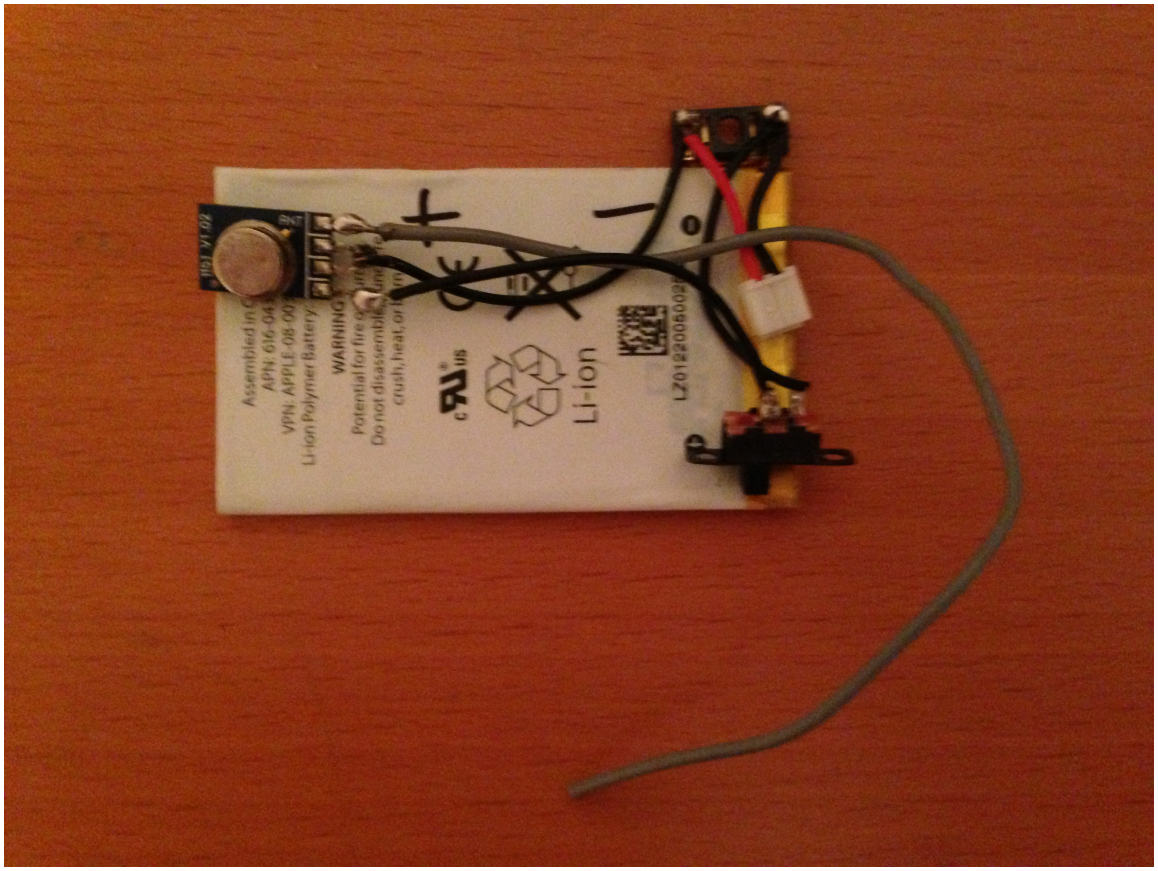


**Figure 44: Final jammer design construction after soldering was completed.**

The battery provides significant rigidity such that a dedicated enclosure is unnecessary. The jammer was finalized by the addition of hot glue and electrical tape. The final jammer can be seen in Figure 45.

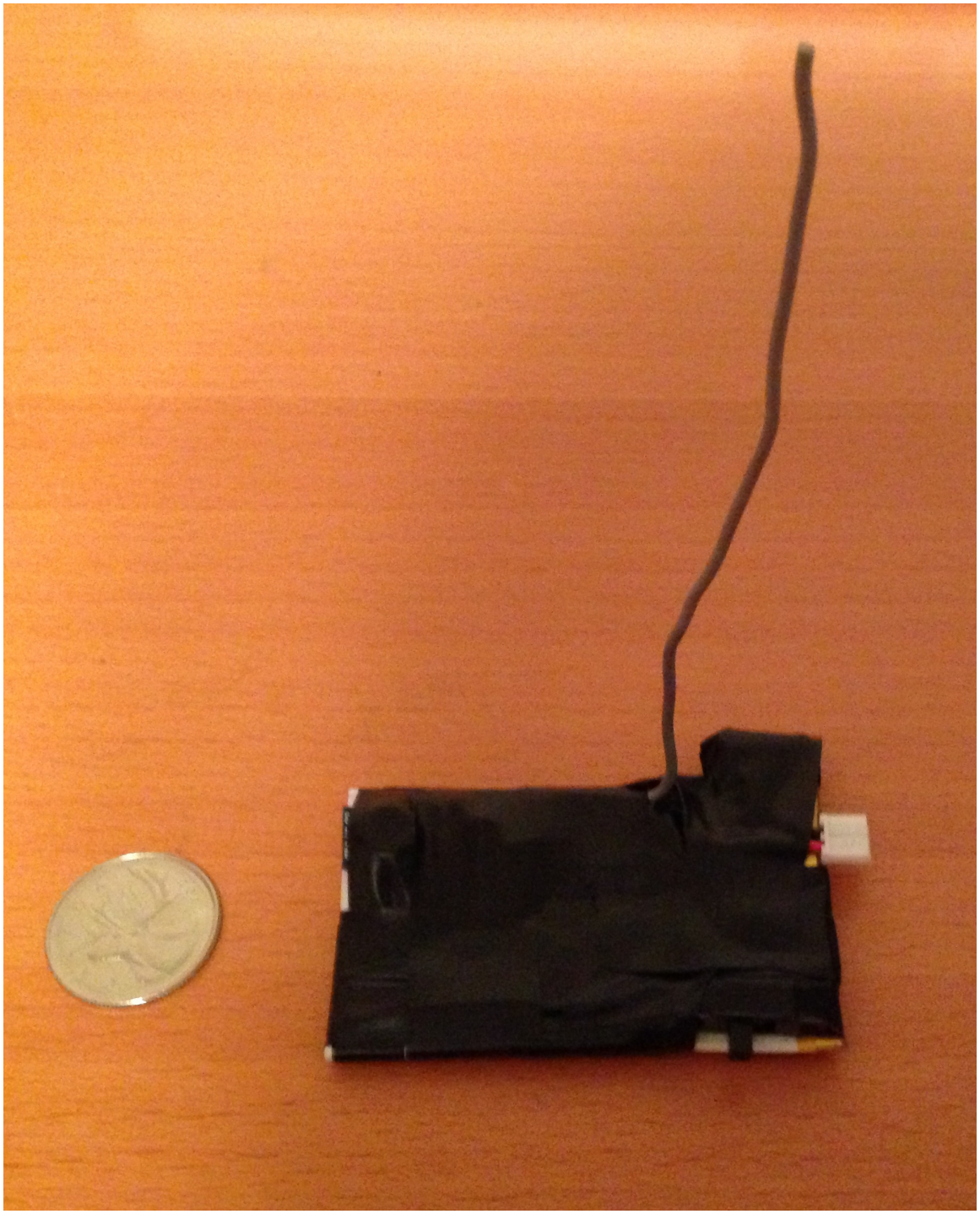

Figure 45: Final jammer construction after encapsulation in hot glue and electrical tape.

The jammer operates by constantly emitting a sinusoidal signal at 315MHz which can be visualized using the GNU Radio scope plot as seen in Figure 46.

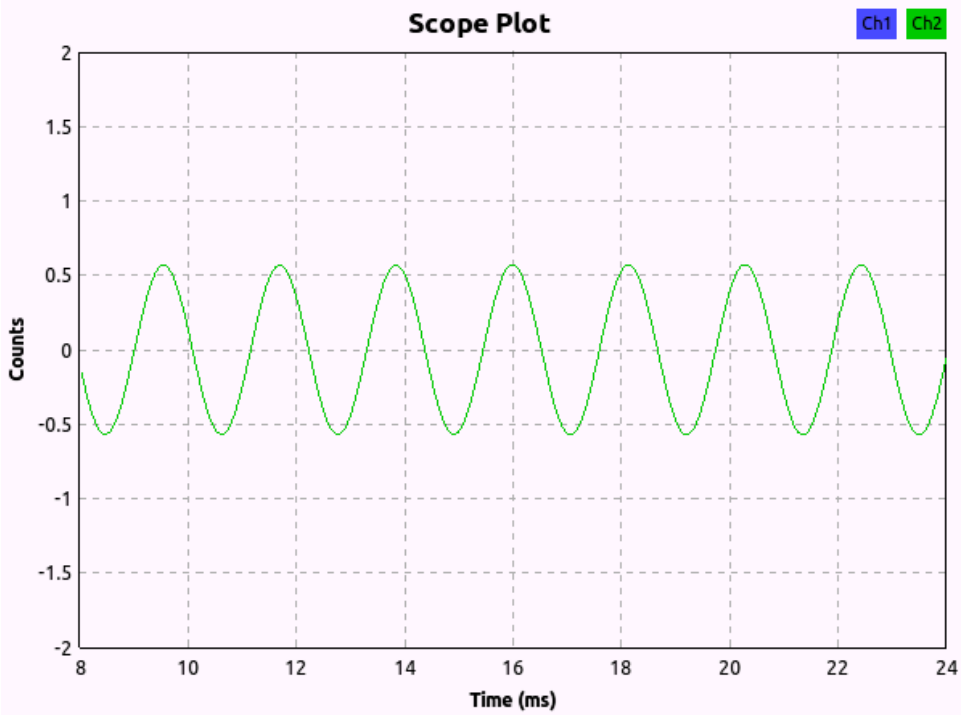


Figure 46: Visualization of the output of the jammer at 315MHz using the GNU Radio scope plot.

## 7.2 Transmission Interception and Interpretation

Signal-processing blocks were implemented in C++ using GNU Radio. Test code for each of the blocks was written in python. There exists a useful tool that automatically sets up the build and test environments for developers of signal processing blocks called 'gr_modtool' (Development Tools, 2013). The blocks themselves need to be created within a module, a module is simply a collection of related signal processing blocks. A new module entitled gr-keyfob was created using gr_modtool via the following command.

```
spencer@spencer-MacBookPro:~/Desktop$ gr_modtool newmod keyfob
Creating out-of-tree module in ./gr-keyfob... Done.
Use 'gr_modtool add' to add a new block to this currently empty module.
```

The tool automatically creates an organized system of folders within your module. Notable folders are the 'lib' folder, which contains all of the C++ code, the 'include' folder, which contains all header files, and the 'python' folder, which contains all of the python code.

### 7.2.1 Duty Cycle Identification block

To create the duty cycle identification block, gr_modtool was invoked again as follows.

```
spencer@spencer-MacBookPro:~/Desktop/gr-keyfob$ gr_modtool add wake_on_duty_cycle
GNU Radio module name identified: keyfob
Enter code type: general
Language: C++
Block/code identifier: wake_on_duty_cycle
Enter valid argument list, including default arguments: float duty, int pulseCount
Add Python QA code? [Y/n] Y
Add C++ QA code? [y/N] N
Adding file 'wake_on_duty_cycle_impl.h'...
Adding file 'wake_on_duty_cycle_impl.cc'...
Adding file 'wake_on_duty_cycle.h'...
Editing swig/keyfob_swig.i...
Adding file 'qa_wake_on_duty_cycle.py'...
Editing python/CMakeLists.txt...
Adding file 'keyfob_wake_on_duty_cycle.xml'...
Editing grc/CMakeLists.txt...
```

Executing this command gives a series of questions about the type of processing block being created, as well as its parameters. After completing this block questionnaire, skeleton code for both the C++ processing block and the python test code was automatically created by the tool. The signal processing block was implemented as a finite state machine as displayed in Figure 34. Implementing the block was done according to the guidelines and instruction given in a tutorial on out-of-tree modules (Martin Braun, 2013). Testing of the block was performed by taking demodulated I/Q data captured from an authorized remote transmission and feeding it through the block as shown in Figure 47.

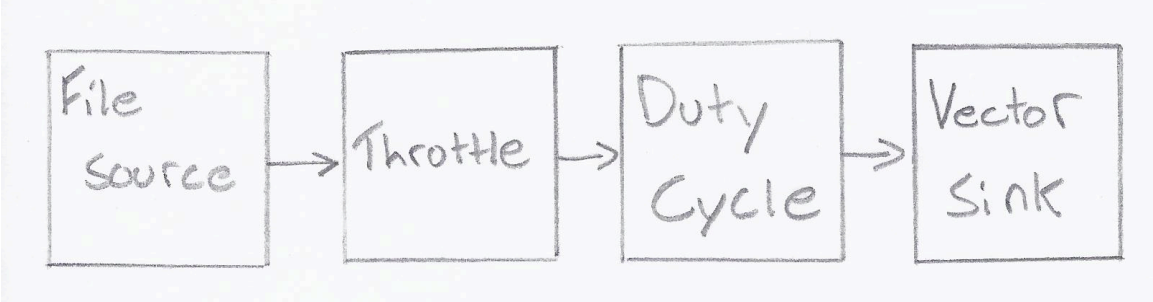


Figure 47: Diagram depicting the signal processing blocks used in testing the duty cycle identification block.

Note the use of the throttle block, this is important because without it, the graph will try and process samples as fast as it possibly can. When the graph is executing without a throttle, it often leads to the system locking up until processing is complete. Initially it was thought that avoiding system lockup was the only reason to include a throttle, therefore it was often left out to speed up the rate at which unit tests could be executed. This was however not the case, when a throttle is omitted, it can lead to samples being dropped, causing the tests themselves to fail. Throttles are not always necessary in flow graphs, for instance, when a USRP source is placed at the head of the flow graph, it generates samples at a particular sample rate. This rate is slow enough that most flow graphs can handle processing the data from it in real time.

### 7.2.2 Sync Header Identification Block

To create the sync header identification block, gr_modtool was invoked again as follows.

```
spencer@spencer-MacBookPro:~/Desktop/gr-keyfob$ gr_modtool add vpwm_sync
GNU Radio module name identified: keyfob
Enter code type: general
Language: C++
Block/code identifier: vpwm_sync
Enter valid argument list, including default arguments: int syncPulsesP, int quietPulsesP
Add Python QA code? [Y/n] Y
Add C++ QA code? [y/N] N
Adding file 'vpwm_sync_impl.h'...
Adding file 'vpwm_sync_impl.cc'...
Adding file 'vpwm_sync.h'...
Editing swig/keyfob_swig.i...
Adding file 'qa_vpwm_sync.py'...
Editing python/CMakeLists.txt...
Adding file 'keyfob_vpwm_sync.xml'...
Editing grc/CMakeLists.txt...
```

The block was implemented as a finite state machine as shown in Figure 36. One of the newer features that was made use of in this block was stream tagging. This allows the header block to take note of the time period length, and to pass it downstream where it can be used at the Manchester decoding block. Information about stream tagging and block coding style was obtained from the blocks coding guide from GNU Radio (Tim Monahan-Mitchell, 2013). Testing of this signal-processing block was performed by passing the block samples from an actual authorized remote whose time period is known. The time period passed downstream via stream tagging was then verified against the known time period. A depiction of the flow graph used for testing this block can be seen in Figure 48.

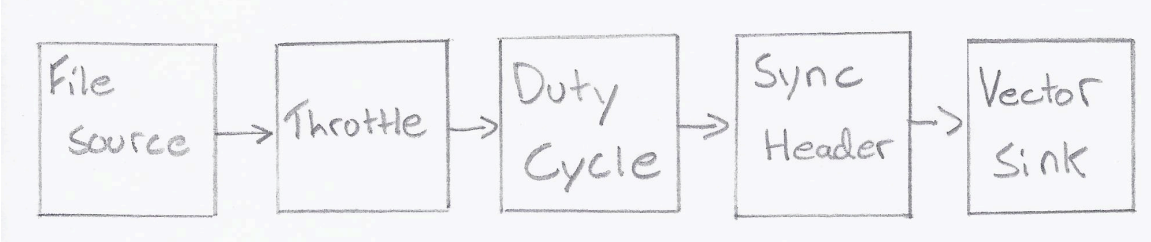


**Figure 48: Visualization of the flow graph used in testing the sync header identification block.**

### 7.2.3 Manchester Decoding Block

To create the Manchester decoding block, gr_modtool was invoked one final time.

```
spencer@spencer-MacBookPro:~/Desktop/gr-keyfob$ gr_modtool add vpwm_demod
GNU Radio module name identified: keyfob
Enter code type: general
Language: C++
Block/code identifier: vpwm_demod
Enter valid argument list, including default arguments: int defaultTimePeriod
Add Python QA code? [Y/n] Y
Add C++ QA code? [y/N] N
Adding file 'vpwm_demod_impl.h'...
Adding file 'vpwm_demod_impl.cc'...
Adding file 'vpwm_demod.h'...
Editing swig/keyfob_swig.i...
Adding file 'qa_vpwm_demod.py'...
Editing python/CMakeLists.txt...
Adding file 'keyfob_vpwm_demod.xml'...
Editing grc/CMakeLists.txt...
```

The block was implemented as a finite state machine as per Figure 37. To construct the unit test for this decoding block, an actual transmission was taken from an authorized remote and fed through the flow graph shown in Figure 49.

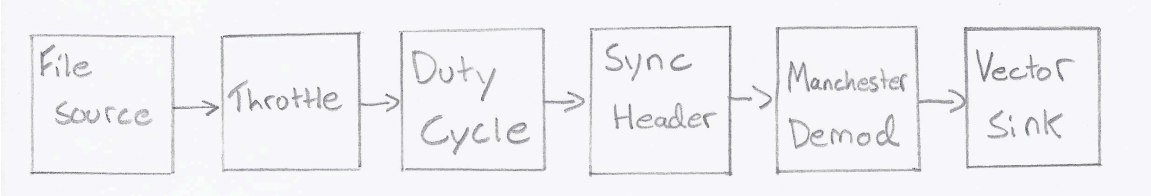


**Figure 49: Visualization of the flow graph used in testing the Manchester decoding block.**

The actual transmission being fed into the flow graph was decoded by hand in Audacity to obtain the 67 bits of data that represents the transmission. The product of the flow graph is then checked against the known 67 bits determined by hand.

### 7.2.4 Interception Flow graph

The interception process was broken up into two steps. In the first step, the transmission from the remote would be isolated from the jamming signal using pass band filtering. AM demodulation was then applied and the final signal was recorded to a file sink. This flow graph was constructed in GRC and can be seen in Figure 50.

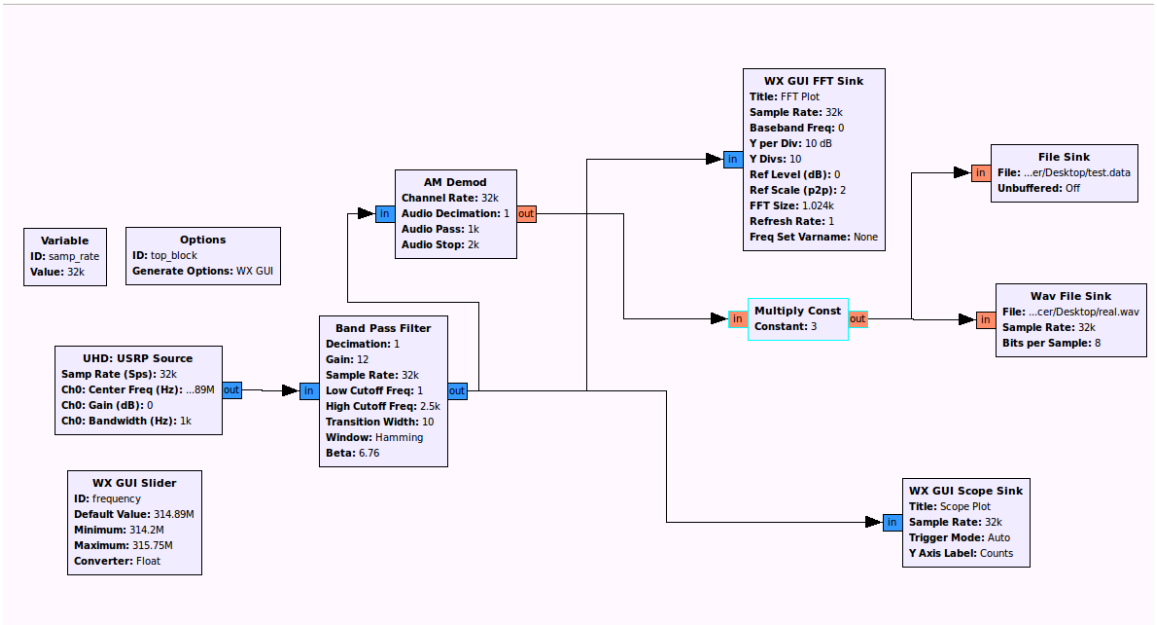


**Figure 50: Flow graph used in part one of the transmission interception process.**

The second half of the interception process was implemented as a flow in python rather than in GRC. When the flow graph completes its execution, the product is the 67-bits representing the received transmission. This transmission information is then transferred serially over to the MSP430 serially using the pySerial library (Liechti, 2010). The serial transfer was executed at 9600 baud with one stop bit.

## 7.3 Replay Hardware

The MSP430 describes a family of microcontrollers, many of which are capable of the task at hand. The MSP430G2452 is one member of this family and was chosen for its high availability and low cost. The features of the MSP430G2452 are outlined in Table 8.

**Table 8: Summary of the main features of the MSP430G2452.**

| Feature | Specification |
|---|---|
| **CPU Architecture** | 16 bit RISC |
| **CPU Speed** | 16 MHz |
| **RAM** | 256 B |
| **Flash Memory** | 8 KB |
| **GPIO Pins** | 16 |
| **Supply Voltage** | 1.8-3.6V |
| **Price** | ~$1 |

Initially the replay hardware was prototyped on breadboard without the use of a serial connection as seen in Figure 51.

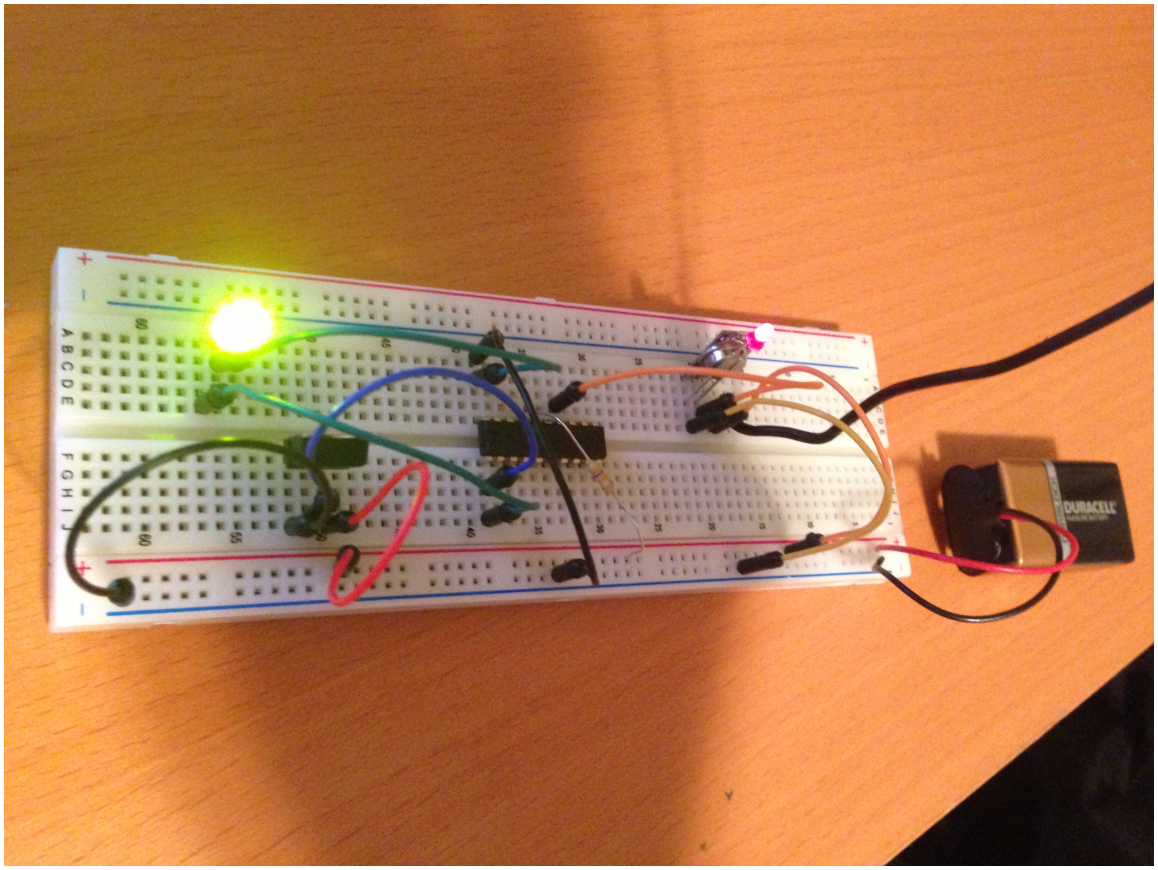


**Figure 51: Initial prototype of the replay hardware used in the hybrid jam and replay attack.**

The green LED used in this prototype was placed to indicate that power was being delivered to the microcontroller, and the red LED is used to indicate that the replay of a transmission was occurring. The 67 bits transmission information was manually input into the source code of the software running on this prototype for testing purposes.  The second prototype was also developed on breadboard and included the USB to TTL serial connection from a PC as shown in Figure 52.

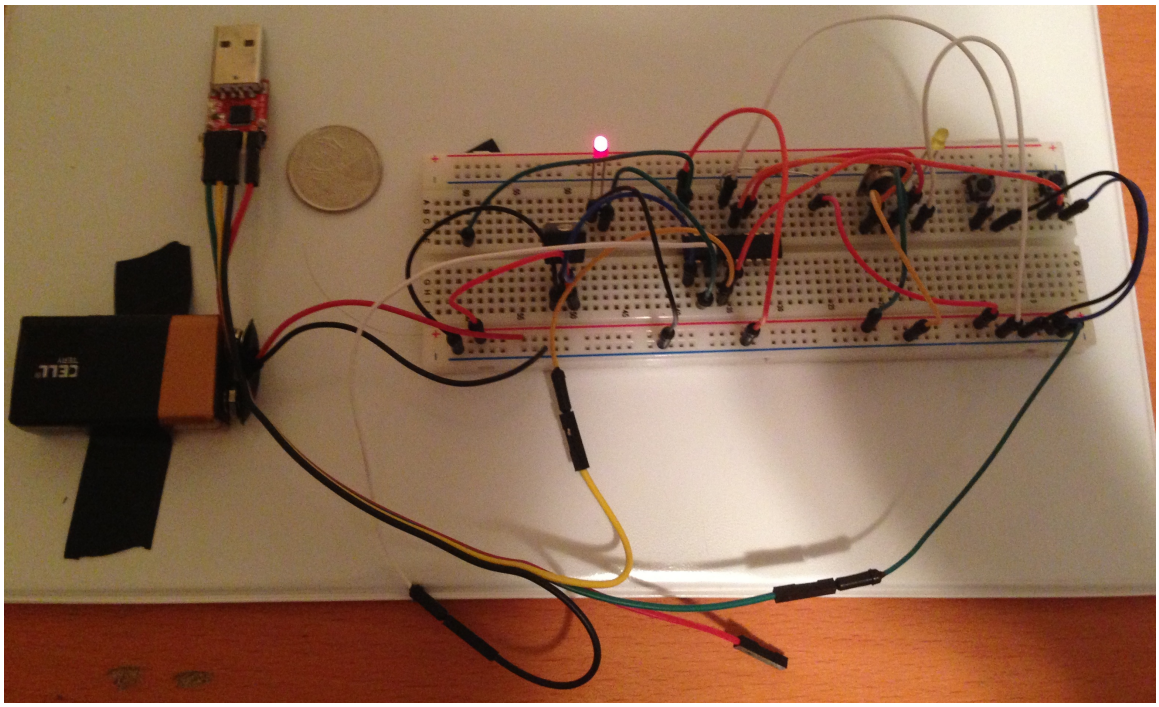

**Figure 52: Second prototype of the replay hardware used in the hybrid jam and replay attack.**

The second prototype was successful the hardware was finalized by placing the components on a PCB and soldering them in place as shown in Figure 53 and Figure 54.

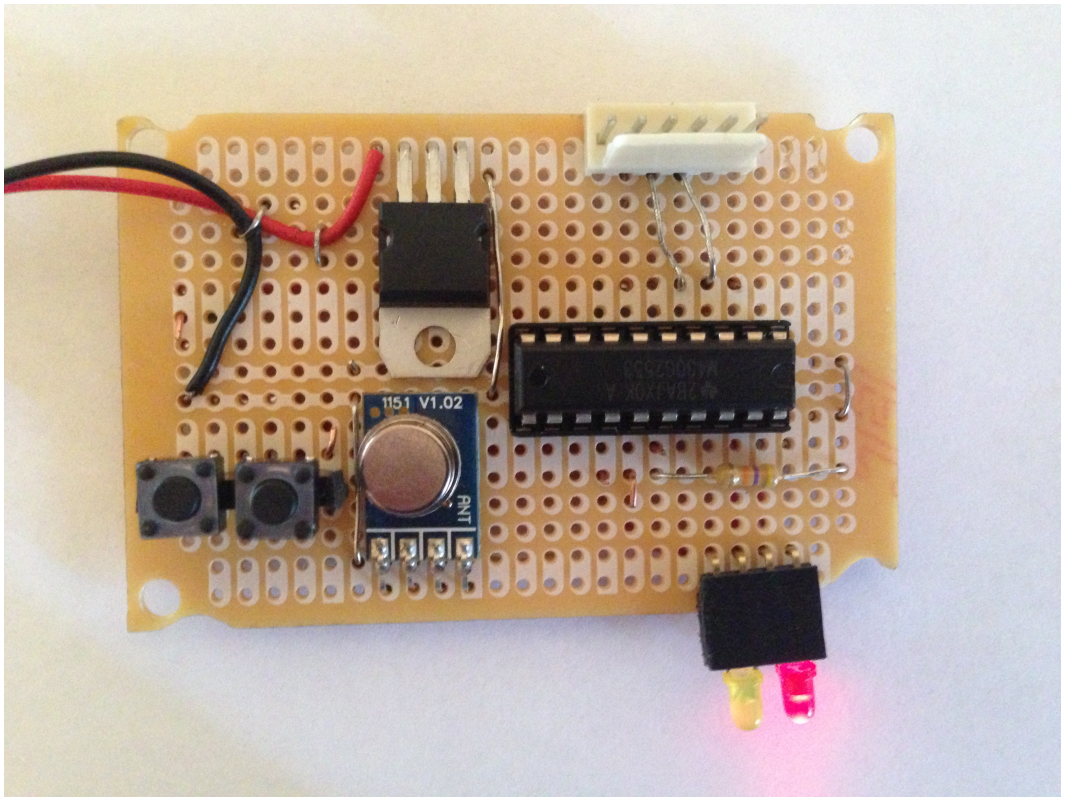

Figure 53: Top side of the final board with all components connected.

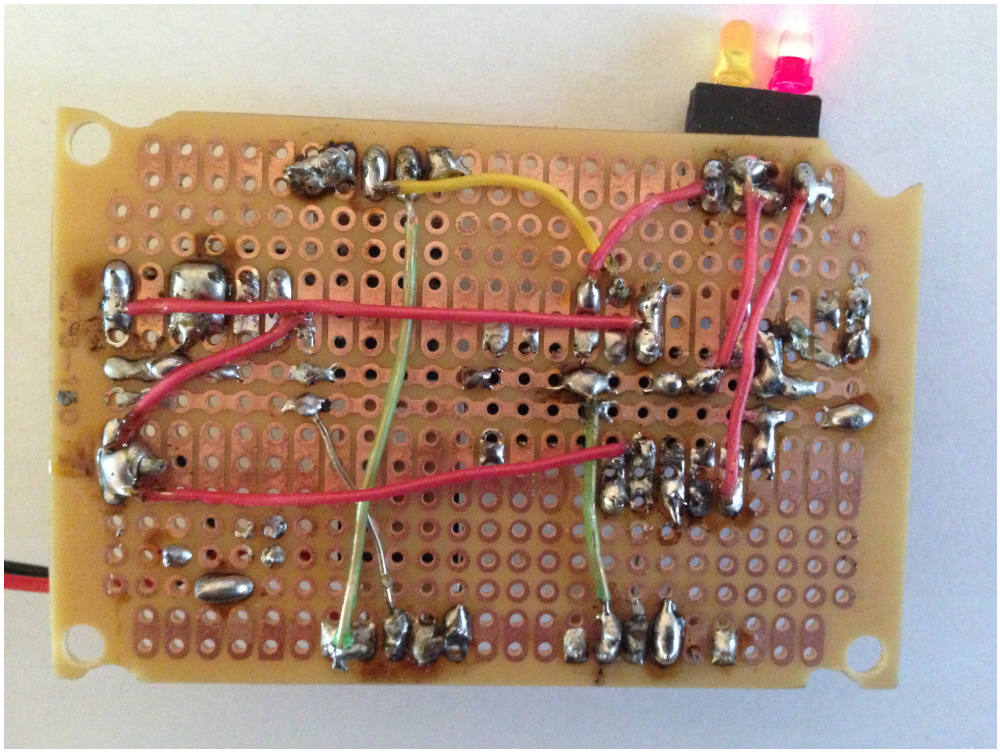
Figure 54: Underside of the final board with all components connected.

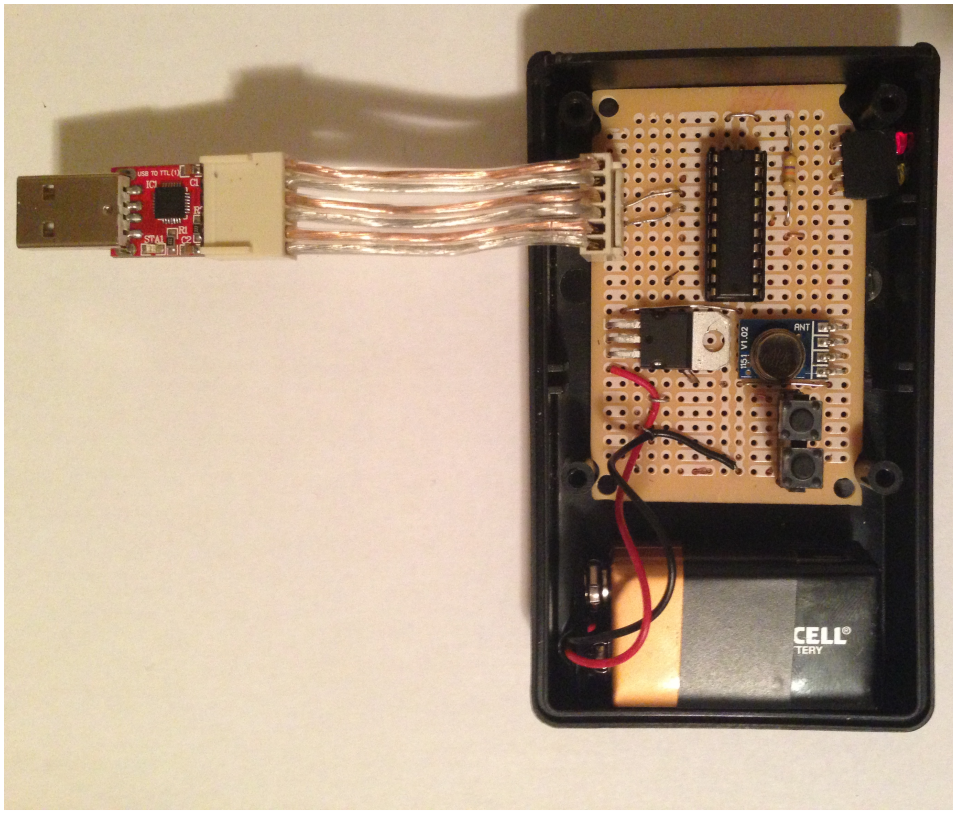



Figure 55: Installation of the hardware into the enclosure.

## 7.4 Replay Software

The MSP430 replay software was developed in C as per to the finite state machine displayed in Figure 41. The software was compiled under Ubuntu Linux using a port of the GCC compiler for the MSP430 (GCC toolchain for MSP430, 2013). The software was then transferred to the MSP430 via the Texas Instruments Launchpad as seen in Figure 56 in conjunction with the open source mspdebug (GCC toolchain for MSP430, 2013).

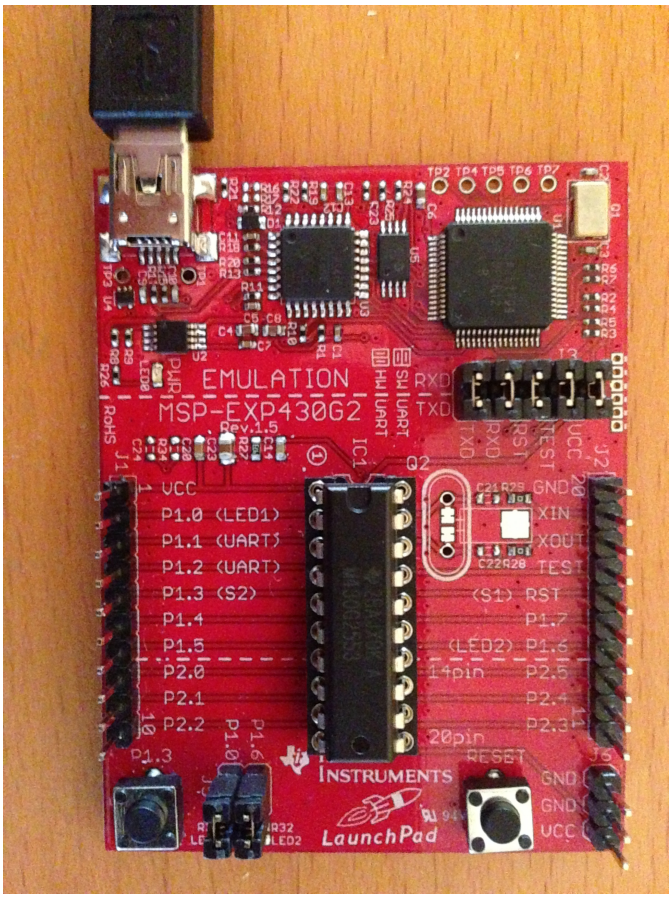


Figure 56: Texas Instruments Launchpad.

Because the microprocessor only operates at 16MHz, CPU cycles must be used sparingly if the desired signal is to be perfectly replicated. Originally, the software was written using the watchdog timer peripheral, when the timer fired, an ISR (Interrupt Service Routine) was called. The idea was to have the ISR called once at the beginning of every elementary time period and then at that time, send the appropriate data to the transmitter module. Unfortunately the clock source for this timer was not accurate enough for the task and was heavily influenced by the value of the data being sent to the transmitter module. The result was a transmission that was very similar to that of the authorized remote, but not close enough to convince the authorized receiver. The solution to this problem was to run the finite state machine from the main application starting point instead of from the ISR. Timing was provided by sleeping for a preconfigured number of clock cycles after each run

through of the finite state machine. This produced a transmission that was almost identical to that of the authorized remote however the software was to slow, leading to a replay that occupied too much time. A comparison of the duty cycles of the authorized transmission versus the replay transmission can be seen in Figure 57.

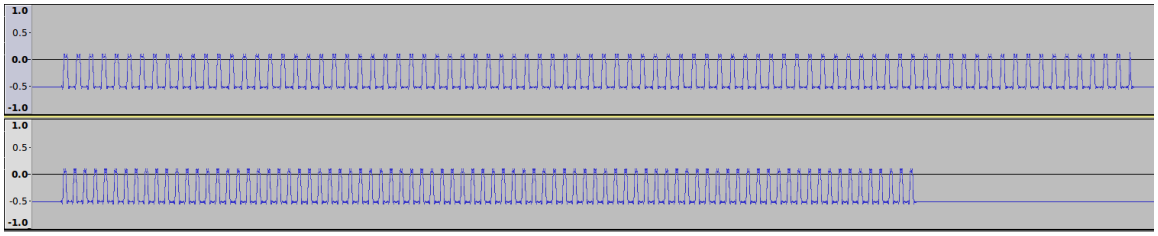


**Figure 57: Duty cycle of the authorized transmission versus the duty cycle of the replayed transmission.**

Many things were optimized in an attempt to speed up its execution, however the main optimization had to do with how inefficiently the modulus (%) operator was being implemented. The removal of the modulus operator lead to a speed increase that allowed the prototype to replay the transmission almost exactly as seen in Figure 58.

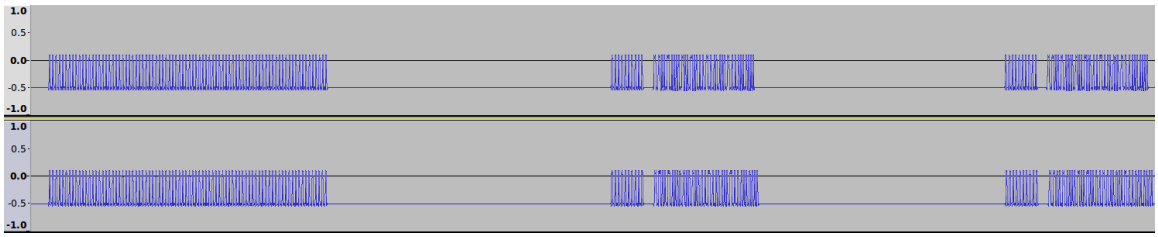


**Figure 58: Original transmission from authorized remote vs. Replay board transmission.**

A closer look at a portion of the authorized transmission versus the replayed transmission can be seen in Figure 59.

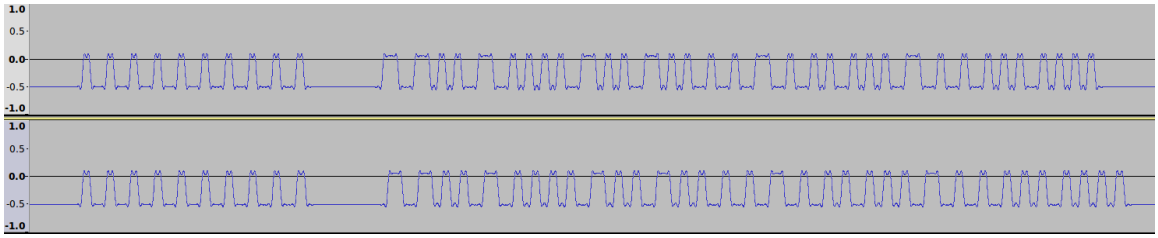


**Figure 59: Authorized transmission versus transmission replayed with the custom replay hardware.**

# 8. Results and Testing

## 8.1 Jammer

The jammer was tested by activating it and placing it in different concealed locations on the automobile. It was determined that the antenna on the roof of the automobile used for FM radio reception was also being used to pick up transmissions from the remote. The jammer was successful, no matter how close the remote was brought to the antenna on the roof, as long as the jammer was anywhere within a 3m radius of the automobile.

## 8.2 Close Range Attacker

One attack scenario that was tested involved the attacker being within a reasonably close distance to the automobile being attacked, the first one can be seen in Figure 60.

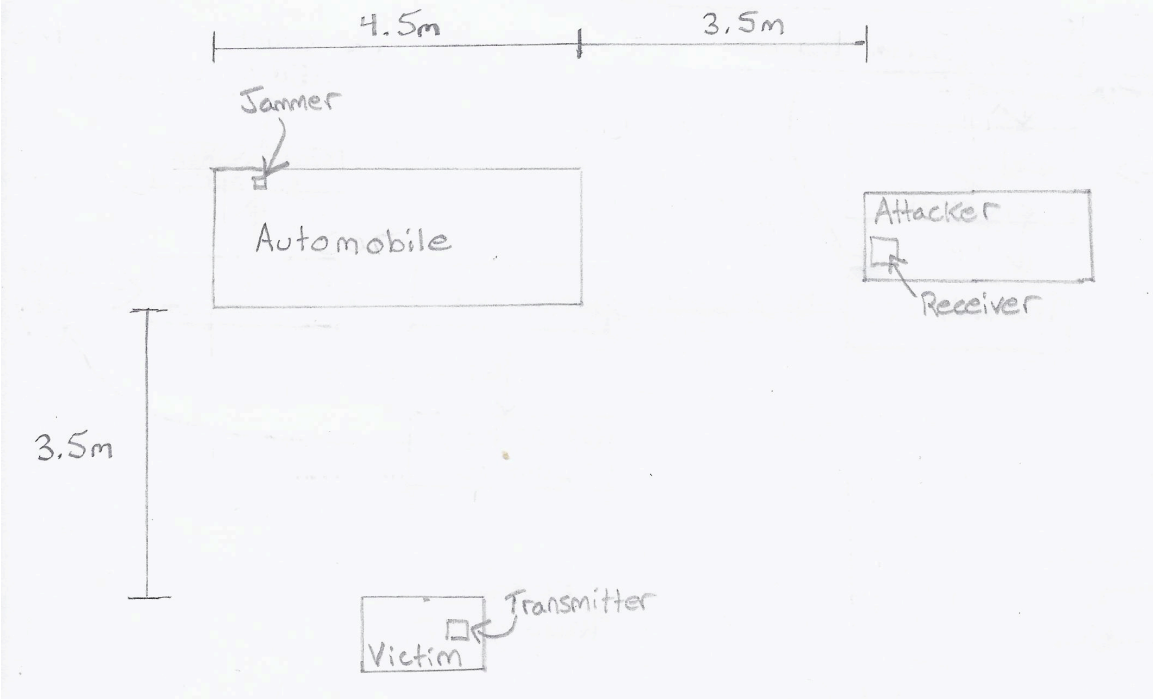


**Figure 60: First attack scenario used for testing the system.**

Because the jammer is small, it was adhered underneath a panel on the trunk lid where it was out of sight and activated. The software defined radio hardware was placed 3.5m from the front bumper of the automobile. The simulated victim approached the car from the side and when they became within range of the automobile, the recording flow graph was executed. When the simulated victim reached the 3.5m mark, they were instructed to press the unlock button on the remote. The automobile did not unlock due to the fact that the jammer was activated. The button press could be observed on the FFT and the flow graph was terminated soon after. The simulated victim could still use their physical key to unlock the car and retrieve what they came for. The simulated victim leaves their automobile thinking that their remote battery is dead and does not fear that their

automobile security has been compromised. The output of the recording flow graph was then used as input to the transmission decoding software, the software successfully decoded the 67-bits of information and transferred it to the replay hardware via a serial connection.

```
spencer@spencer-MacBookPro:~/Desktop/Dropbox/FINAL$ sudo ./test.sh
Decoding Hopping Code From ./test.data
Sending data to replay hardware over /dev/ttyUSB0
Baud Rate: 9600
Code Sent: 1001100010000110111110111011111111011110111010110111111010111010 1001
```

The replay hardware was disconnected from the computer and the jammer was deactivated. The replay button was pressed and within a couple of seconds, the automobile unlocked, giving unauthorized access to it. This process was repeated five times with success each time. The key settings that were used in the recording flow graph for this scenario can be seen in Table 9.

**Table 9: Settings used to successfully record transmissions using the recording flow graph in the short range attacker scenario.**

| Property | Value |
|---|---|
| Frequency | 314.89 MHz |
| Band-pass Filter Gain | 12 dB |
| Multiply Const | 2 |

## 8.3 Long Range Attacker

The distance of the software defined radio from the automobile was increased from 3.5m to approximately 9m as shown in Figure 1.

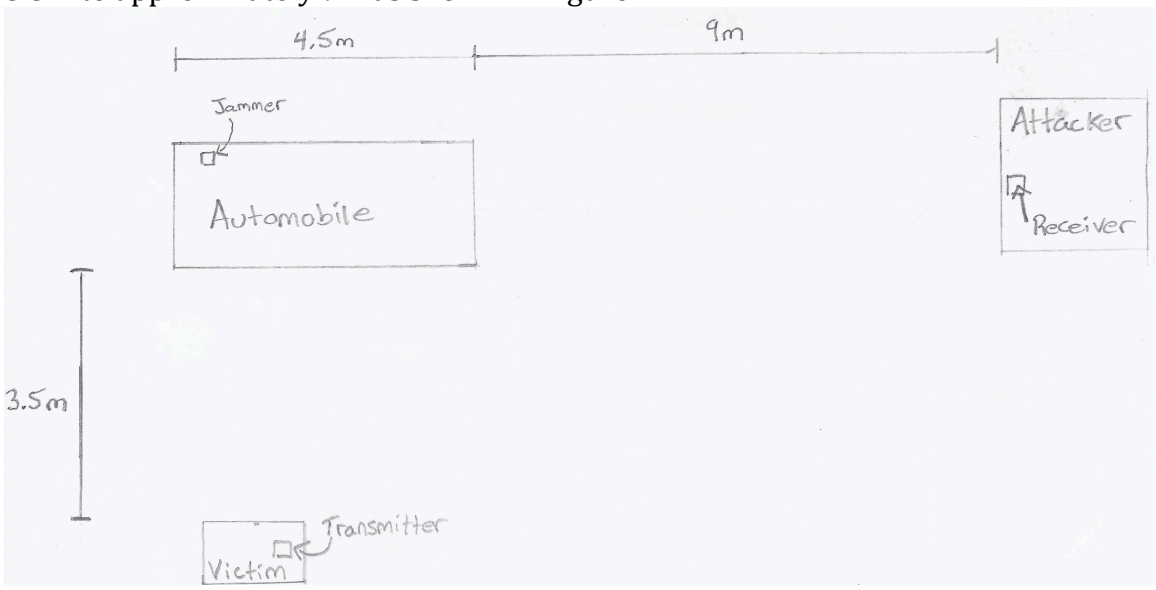


**Figure 61: Second attack scenario used to test the system.**

The testing was conducted again as described above with different flow graph settings as outlined in Figure 10.

Table 10: Settings used to successfully record transmission using the recording flow graph in the scenario described in section 8.3.

| Property | Value |
|---|---|
| Frequency | 314.89 MHz |
| Band-pass Filter Gain | 12 dB |
| Multiply Const | 3 |

Notice that in the longer-range attack, a larger value was required for the multiply const field. This is because the strength of the signal gets weaker as the victim moves further away.

## 8.4 Directional Antenna Failure

Originally the plan for implementing this attack was to use directional antenna for jamming and intercepting the signal as shown in Figure 19. A three director Yagi antenna was constructed for this purpose according to the specifications provided in a document put out by the National Bureau of Standards on Yagi antenna design (Viezbicke, 1976). The Yagi was designed specifically for 315MHz operation and the attempt can be seen in Figure 62.

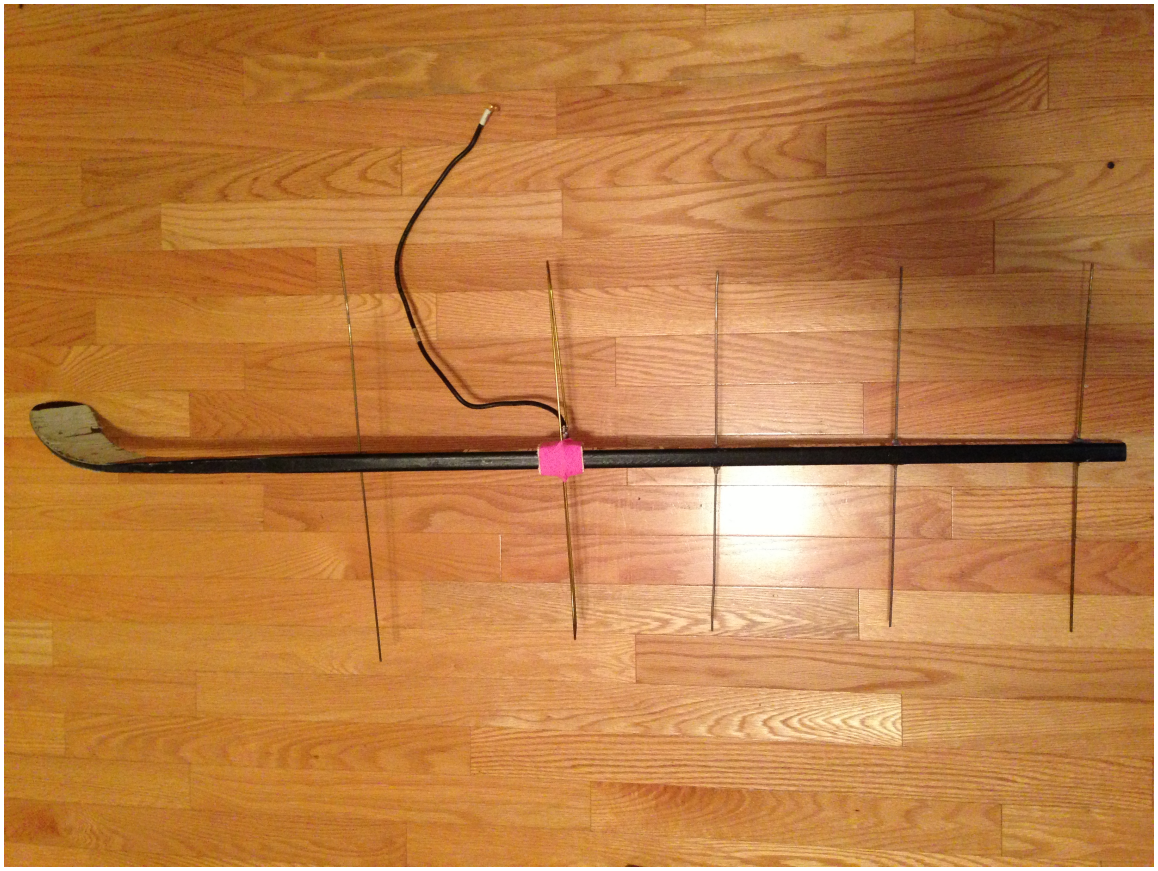

Figure 62: Failed attempt to build an effective Yagi at 315 MHz.

A senior hockey stick was used as the boom, each element was constructed of coat hanger metal, and a right angle SMA coax cable was soldered on to provide a connection from the driven element to the USRP. The driven element and the reflector were capable of moving along the boom to allow for adjustment at the time of testing. When testing the Yagi, the antenna was found to be marginally different from the isotropic antenna. The Yagi did not provide a way to focus on the transmitter's signal and ignore the jammer's signal, it failed horribly at this. This failure was a huge step back for the project however it ended up being the best thing to happen to this project because it led to the use of pass-band filtering which is performed in software and requires no additional hardware.

# 9. Improvements

## 9.1 Improvements to the Hybrid Jam and Replay Attack

### 9.1.1 Band-reject Filtering

The project currently uses pass-band filtering to ensure that the transmission from the authorized remote is picked up and not the jamming signal. The problem with this is that the attacker must first determine the frequency at which the victim's remote operates, this can be done easily enough but is truly unnecessary. The attacker creates the jamming hardware and knows the exact frequency at which it operates, therefore it would be much more appropriate to use band-reject filtering to block out just the jamming signal.

### 9.1.2 Adding Blocks to GRC

The three blocks developed for use in this project can only be used by writing python flow graphs, it would make the blocks more easily reusable and easier to experiment with if they were moved into GRC. Doing this is as simple as creating a few XML files containing metadata that tells GRC about the blocks.

### 9.1.3 Floating Point Bits

Currently the Manchester decoding block produces the 67 bits as floating point values, either 0.0f or 1.0f. This is both inefficient and confusing to other developers. Ideally this block would either produce integers or it could be made into a sink block and the bit stream could be sent to a message queue. Currently, when the decoding flow graph executes, it must iterate over all samples and terminate before the bits can be sent serially to the replay hardware. If message queues were used, the 67 bits of transmission information could be sent to the replay hardware as soon as they are decoded.

### 9.1.4 External Oscillators

The MSP430 is running code that wastes CPU cycles as a timing mechanism. This is very power inefficient because the MSP430 could be put in a low power state and told to return after a certain amount of time instead. The problem with this route was that the accuracy of the clocks used on board the MSP430 was determined to be insufficient. The solution to this problem would be to attach an external 16MHz oscillator between the XIN and XOUT pins which ere designed for exactly this purpose.

## 9.2 Improvements to RF Remote security

Due to the attacks presented in this report, it is recommended that unidirectional remotes be discontinued from use in security sensitive areas. Most PKES systems are still vulnerable to the latest replay attacks. A better solution would be to keep the challenge response protocol and keep the button presses that are required in PKES systems.

## 10. Conclusion

It was shown that implementing a hybrid jam and replay attack can be done for less than 50$. Filtering was determined to be the most effective method for extracting the transmitted signal from the authorized remote when a jammer is in operation. Implementing this hybrid attack in a budget friendly manor was a great learning experience, however it is only a matter of time before software defined radio hardware that can transmit at 315MHz becomes available. When this does happen, it will allow ill willed people to carry out a purely SDR based jam and replay attack without the need for much technical skills at all.

## Bibliography


AllDataSheet. (2013). *AllDataSheet.com - Datasheet search site for electronic components and semiconductors*. Retrieved June 27, 2013, from AllDataSheet: http://www.alldatasheet.com

Audacity. (2013). *Audacity: Free Audio Editor and Recorder*. Retrieved June 26, 2013, from SourceForge.

Aurelien, F., Boris, D., & Srdjan, C. (2010). Relay Attacks on Passive Keyless Entry and Start Systems in Modern Cars.

Barbeau, M. (2013). *Software Readio for Experimenters with GNU Radio, Octave and Python.* Ottawa, Ontario, Canada.

Broekhuis, D. (2011). Feasibility Study of Eavesdropping Using GNU Radio. *15th Twente Student Conference on IT.* Enschede.

Commission, F. C. (2013). *OET - FCC ID Search*. Retrieved May 15, 2013, from Federal Communications Commission: http://transition.fcc.gov/oet/ea/fccid/

*Datasheets: Electronic parts info*. (2013). Retrieved June 27, 2013, from Datasheets: http://www.datasheets.com

DeBruhl, B. a. (2011). Digital Filter Design for Jamming Mitigation in 802.15.4 Communication. *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on Computer Communications and Networks* (pp. 1-6). Maui: Institute of Electrical and Electronics Engineers ( IEEE ).

*Development Tools*. (2013). Retrieved July 6, 2013, from The Comprehensive GNU Radio Archive Network: https://www.cgran.org/wiki/devtools

*GCC toolchain for MSP430*. (2013). Retrieved July 12, 2013, from SourceForge: http://sourceforge.net/projects/mspgcc/

*HCS361 Datasheet*. (2002). Retrieved June 27, 2013, from Microchip Inc.

Kasper, M., Timo, M., & Amir, P. (2009). Breaking KeeLoq in a Flash: On Extracting Keys at Lightning Speed. *Proceedings of the 2nd International Conference on Cryptology in Africa: Progress in Cryptology* (pp. 403-420). Berlin: Springer-Verlag.

Kent, R. D. (2002). Acoustic Analysis of Speech. *American Scientist* , 250-255.

Liechti, C. (2010). *Welcome to pySerials Documentation*. Retrieved August 18, 2013, from SourceForge: http://pyserial.sourceforge.net

Martin Braun, G. W. (2013). *Out of Tree Modules*. Retrieved July 8, 2013, from GNU Radio: http://gnuradio.org/redmine/projects/gnuradio/wiki/OutOfTreeModules

Novotny, M., & Kasper, T. (2010). Crypanalysis of KeeLoq with COPACOBANA.

Tim Monahan-Mitchell, J. B. (2013). *Blocks Coding Guide*. Retrieved July 8, 2013, from GNU Radio: http://gnuradio.org/redmine/projects/gnuradio/wiki/BlocksCodingGuide

Ubuntu. (2013, May 29). *The world's most popular free OS*. Retrieved 2013, from Ubuntu: http://www.ubuntu.com

Viezbicke, P. P. (1976). *Yagi Antenna Design.* Boulder, Colorado, United States.

Xu, W. a. (2005). The feasibility of launching and detecting jamming attacks in wireless networks. *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing* (pp. 46-57). Urbana-Champaign: ACM.

Yagi, H. (1928). Beam Transmission Of Ultra Short Waves. *Proceedings of the Institute of Radio Engineers , 16* (6), 715-740.