Implementation of a Low-Cost SIGINT Platform

Kyle Weetman
Network and Computer Security
SUNY Polytechnic Institute
Utica, New York
weetmak@sunypoly.edu

***Abstract*—This capstone project creates a signal intelligence platform (SIGINT) with a low barrier to entry. The SIGINT platform is implemented using an RTL-SDR software defined radio dongle and free open source software. This platform is demonstrated through collecting automotive remote key fob signals. The key fob signals are analyzed and compared to the make and model of the cars they belong to. It is determined that remote key fob signals can be associated with their corresponding cars based on attributes of the captured signals. The privacy and security implications of this SIGINT platform are discussed.**

I. Introduction

This paper will cover a proposed implementation for collecting and analyzing wireless signals utilizing cheap, readily available equipment and open source software. It will also discuss radio frequency (RF) fingerprinting concepts and its application to automotive key fobs. The motivation for this project was curiosity about the local radio frequency spectrum and the desire for a signal intelligence workflow for personal tactical preparedness.

Signal Intelligence (SIGINT) refers to intercepting and interpreting electronic signals from communications and information systems [1]. SIGINT is a broad term that encapsulates communications intelligence (COMINT), electronic intelligence (ELINT) and foreign instrumentation signals intelligence (FISINT) [1]. COMINT refers to any signal that contains speech or text [1]. FISINT refers to machine to machine communication from systems like missiles, satellites, drones, and remote access and control systems [1]. ELINT refers to all electronic emissions outside of COMINT, intentional transmissions or otherwise [1]. SIGINT is an important part of the intelligence gathering used by military and intelligence agencies [1].

Software defined radio (SDR) is a radio communication system that uses digitized radio signals [2], [3]. This enables a computer to use software to replace the analog components that are used in conventional radio systems [2], [3]. SDR systems can utilize a wide range of software to receive and transmit various different radio protocols [2], [3]. The digital signals produced by SDR can be saved and analyzed using other software [2], [3].

Radio frequency (RF) fingerprinting refers to classifying a radio transmission based on physical attributes of the transmission [4]. Once a signal is fingerprinted and its source is known, similar signals encountered in the wild can be attributed to a similar source.

Modern automotive vehicles use radio frequency transmissions from handheld key fobs to control the door lock and sometimes ignition. Because mass production is commonly utilized in the automotive industry, fingerprinting a few vehicle key fobs signals and correlating them to the make and model of the car enables one to make educated decisions about the identity of key fob transmissions encountered in the future. This paper will discuss SIGINT, SDR, and RF fingerprinting in greater detail followed by a walk-through for implementing a system for collecting and analyzing wireless signals utilizing inexpensive, readily available equipment and open source software. Finally, we will discuss the security implications of this work and conclusions drawn from it.

II. Background

A. Radio

Radio refers to the technology used to communicate using electromagnetic radio waves. A transmitter connected to an antenna radiates electrical energy through the air. Another antenna, connected to a receiver, can absorb electrical energy to receive information. Wireless radio remote controlled devices like drones, garage doors, and car key fobs use this technology to remotely control the device.

B. SIGINT

Every since electronic communications were in use, traffic on telegraph cables have been recorded and analyzed [5]. Signal intelligence (SIGINT) was instrumental in the outcome of the First World War [5]. In fact, the interception of the Zimmerman Telegram by the British was the final event that convinced the United States to join the war [5]. World War II brought new electronic signals outside of communications [5]. Analysis of radar, sonar and munition guidance systems signals provided important insight to the enemy's capabilities [5]. During the Cold War, SIGINT became permanently cemented as a cornerstone in intelligence efforts [5]. Many interception stations were established and land, sea, and air based mobile systems were deployed with the express purpose of collecting SIGINT [5]. SIGINT satellites were launched with the capabilities to capture RF transmissions that emanated from deep in enemy territory [5]. Towards the end of the twentieth century the target of SIGINT operations changed from military targets to terrorist groups [5]. The proliferation of the internet forced the practice of intercepting communications outside of the scope of SIGINT and into the realm of cyber operations [5].

SIGINT has some intrinsic advantages over other fields of intelligence [1]. SIGINT primarily utilizes passive collection and can be done from many miles away, reducing potential political or physical risk [1]. SIGINT is also objective and more reliable than intelligence collected from humans [1]. Sometimes, SIGINT can stand on its own without the need for analysis or correlation with other sources [1]. For example, if a communication transmission is intercepted between two known parties it could be decrypted and taken as genuine intelligence. SIGINT is often the fastest source of intelligence as the NSA can deliver SIGINT to the field in near real-time [1]. SIGINT also produces intelligence reports on the broadest range of subjects compared to other intelligence methods [1]. While people need to sleep, signals can be intercepted 24 hours a day, 365 days a year, regardless of the weather or other environmental conditions [1]. Finally, SIGINT is considered the most cost effective method of intelligence collection [1].

C. Frequency Spectrum Plot

A frequency spectrum plot is a graph where the amplitude is plotted along the y-axis and frequency is plotted along the x-axis.

D. Waterfall Plot

Waterfall plot refers to a graph where the time domain is plotted along the y-axis and frequency is plotted along the x-axis. Amplitude is represented with a color gradient. Warm colors represents a higher amplitude and cool colors represents a lower amplitude.

E. Debian and Ubuntu

Debian is a free and open source Linux based operating system. Ubuntu is based on Debian and is widely used for desktop, server, and internet of things (IoT) devices. Ubuntu Desktop was chosen because of its stability and compatibility with many RF signal applications and the RTL-SDR

F. RTL-SDR

RTL-SDR refers to a inexpensive ~$30 USB device that can be used as a computer based software defined radio [3]. The RTL-SDR dongle can receive frequencies from 500 kHz up to 1.75 GHz [3]. The maximum usable sample rate of the RTL-SDR is 2.56 MS/s [3]. RTL-SDR dongles cannot transmit [3]. The dongle originated from a mass produced TV tuner dongle based on the RTL2832U chipset [3]. It was discovered that raw data received by the dongle could be access directly [3]. This enabled the RLT-SDR to be used as a software defined radio [3]. Because of the wide availability of the dongle, a large community has been built around it [3]. This community has developed many free, open source software applications for use with the dongle [3].

Figure 1— RTL-SDR dongle with antenna

G. SoapySDR

SoapySDR is an open source software library for interfacing with SDR devices [7]. Many open source applications rely on SoapySDR to operate [7]. SoapySDR is vendor neutral, meaning it offers support to a wide range of consumer SDR devices and applications [7]. SoapySDR offers a plugin for use with the RTL-SDR dongle and software that utilizes SoapySDR [7].

H. SDR++

SDR++ is a cross platform and open source SDR program that is compatible was many SDR receivers [8]. SDR++ enables the user to use an SDR to tune to various frequencies and filter the received signals to a desired strength [8]. SDR++ features a spectrum display and waterfall plot of the live tuned signal [8]. This visualizes the radio spectrum for quick identification of signals [8].

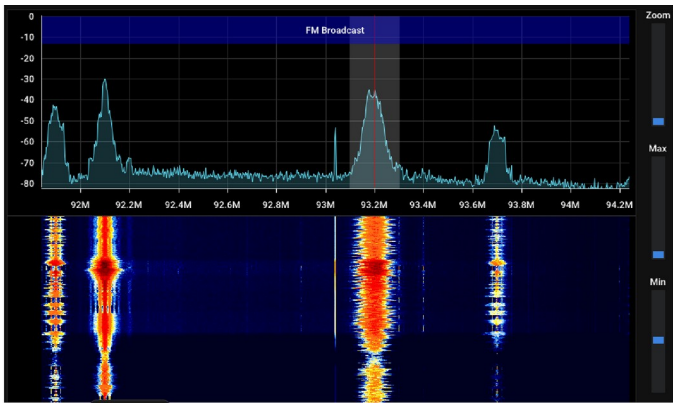

Figure 2— SDR++ spectrum and waterfall plot [8]

I. GNU Radio

GNU Radio is a free and open source software development toolkit that enables signal processing for use with SDR [9]. GNU Radio is widely used in research, industry, academia, government, and hobbyist environments to support both wireless communications research and real-world radio systems [9]. GNU Radio uses signal processing blocks that are linked together to take a raw signal and process it [9]. GNU Radio Companion is a graphical user interface that is used to combine signal processing blocks to create and execute signal processing applications [9]. A group of linked signal processing blocks is referred to as a flowgraph [9]. GNU Radio Companion has many signal processing blocks by default but more functionality can be added by coding new blocks [9]. The GNU Radio community has created many blocks that can be installed to add functionality.

J. FISSURE

FISSURE stands for Frequency Independent SDR-based Signal Understanding and Reverse Engineering and it is still in the early stages of development [10]. Included with FISSURE are many software tools including RF analysis tools, WiFi hacking tools and general radio tools [10]. FISSURE enables a workflow for detecting an RF signal, understanding its characteristics, collection and analysis and transmission all in one software package [10]. The signal conditioner feature can be used to isolate and normalize recorded signals [10]. The feature extractor feature can be used to gather statistical values from a signal recording [10]. A default FISSURE installation included GNU Radio and rtl_433 [10].

K. *rtl_433*

rtl_433 is a generic data receiver that works with RTL-SDR and SoapySDR [11]. Included with rtl_433 are many supported device protocols used to decode their corresponding signals [11]. A few tire pressure monitor (TMPS) protocols are included [11].

L. *Initial Desktop Configuration*

A refurbished Lenovo Thinkpad T470 laptop was purchased for use for this project. This was chosen for its affordable price, appropriate I/O ports and sufficient processing power. The laptop shipped with Windows 10 pro but Ubuntu was immediately installed to replace it. To install Ubuntu the provided installation tutorial was followed [12]. The latest stable Ubuntu ISO image was downloaded from the website and flashed onto a 16GB flash drive using balenaEtcher [12], [13]. The laptop was powered off and the flash drive was inserted. The laptop was booted and during startup, when the Thinkpad logo appeared, the F12 key was pressed to open the boot menu. The flash drive was selected and the installation setup began. During the installation, automated installation was selected and the default app selection was chosen. All previous data on the laptop was erased during the Ubuntu installation. Login credentials were selected and the installation was completed. After the first boot of Ubuntu on the Thinkpad, Ubuntu was updated using the *sudo apt update* and *sudo apt upgrade* commands.

M. *RTL-SDR Dongle Setup*

An RTL-SDR (RTL2832U) dongle was acquired to use as the primary software defined radio for this project. The dongle was purchased in a kit including some antennas and coaxial cables. Ubuntu includes some default RTL-SDR drivers. These drivers are not compatible with the RTL-SDR and need to be removed. The previous drivers were purged using the following commands in the terminal:

```
sudo apt purge ^librtlsdr
sudo rm -rvf /usr/lib/librtlsdr* /usr/include/rtl-sdr* /usr/local/lib/librtlsdr* /usr/
local/include/rtl-sdr* /usr/local/include/rtl_* /usr/local/bin/rtl_*
```

Figure 3— Commands to purge default drivers [14]

The correct RTL-SDR Drivers were then installed:

```
sudo apt-get install libusb-1.0-0-dev git cmake pkg-config
git clone https://github.com/rtlsdrblog/rtl-sdr-blog
cd rtl-sdr-blog
mkdir build
cd build
cmake ../ -DINSTALL_UDEV_RULES=ON
make
sudo make install
sudo cp ../rtl-sdr.rules /etc/udev/rules.d/
sudo ldconfig
```

Figure 4— Commands to install RTD-SDR drivers [8]

The unnecessary, conflicting DVB-T TV drivers were blacklisted:

```
echo 'blacklist dvb_usb_rtl28xxu' | sudo tee --append /etc/modprobe.d/blacklist-
dvb_usb_rtl28xxu.conf
```

Figure 5— Commands to blacklist DVB-TV drivers [8]

The laptop was then rebooted. After the reboot, the *rtl_test* command was run to confirm the dongle was working properly.

```
[redacted]:~$ rtl_test
Found 1 device(s):
  0:  Realtek, RTL2838UHIDIR, SN: 00000001
```

Figure 6— Successful rtl_test command

N. *SDR++ Install*

SDR++ was installed using the manual included on the SDR++ website. The appropriate version of SDR++ was downloaded from the GitHub. The version is determined by the version of Ubuntu installed. The Ubuntu version is checked with the "lsb_release -a" command [8]. Once the correct release version had downloaded, the following command was run:

```
sudo apt install libfftw3-dev libglfw3-dev libglew-dev libvolk2-dev
libsoapysdr-dev libairspyhf-dev libiio-dev libad9361-dev librtaudio-dev
libhackrf-dev zstd
```

Figure 7— Command to install SDR++ dependencies [8]

SDR++ was then installed using this command (The .deb file should be replaced with the file that was downloaded):

```
sudo dpkg -i sdrpp_debian_amd64.deb
```

Figure 8— Command to install SDR++ [8]

Once SDR++ was installed, the application was opened and configuration began. In the source tab, the refresh button was clicked. Afterwards, the RTL-SDR source was selected. Upon initial testing, it appeared only noise was received but after further research, the RTL AGC, Tuner AGC and IQ Correction options were ticked.

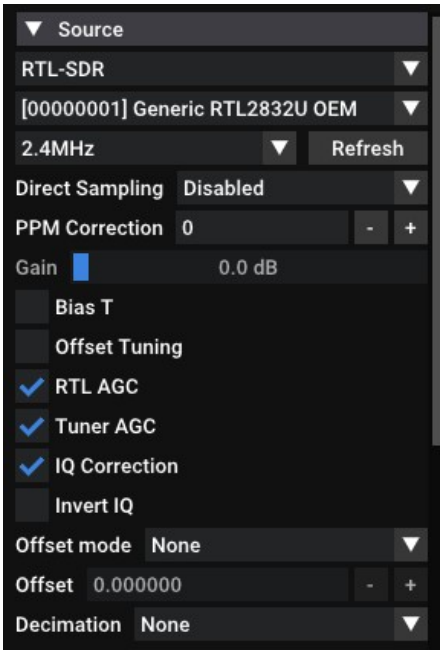

Figure 9— SDR++ source tab

At this stage the RTL-SDR could be used with SDR++ to browse the RF spectrum and discrete signals could be identified. FM radio could also be tuned and listened to.

O. *FISSURE Install*

FISSURE was installed following the instructions on the FISSURE GitHub. It was installed using the following commands in the terminal:

```
git clone https://github.com/ainfosec/FISSURE.git
cd FISSURE
git checkout Python3  # or Python2_maint-3.7
git submodule update --init
./install
```

Figure 10— Commands to install FISSURE [16]

Any requested required dependencies were also installed by selecting "Y" when prompted in the terminal. The fissure installer opened and the correct version of Ubuntu was selected. The recommended software was selected during the installation. Various prompts occurred during installation and the default or recommended options were selected. Once FISSURE was fully installed, it was run using the "fissure" command in the terminal. Some basic exploration within FISSURE was performed and through troubleshooting it was discovered that SoapySDR, which is required for FISSURE to interface with the RTL-SDR, was not detecting the RTL-SDR dongle. Fixing this SoapySDR error would prove useful for use with rtl_433 and GNU Radio in the future. Once this was solved it was determined that the FISSURE's signal detector feature was not the ideal tool for this project.

III. SIGINT Workflow Development

A. *Focusing the Scope*

At this point in the project, it was decided that the scope would be focused from general RF exploration and SIGINT to automotive radio signals with the goal of fingerprinting the corresponding make and model. Tire pressure monitor sensors (TPMS) and remote key fobs were determined to be good candidates for signal analysis. Both TPMS and key fobs typically transmit at 315MHz. A dedicated 315MHz antenna was purchased for use with the RTL-SDR dongle.

The initial signal analysis was attempted using rtl_433. Running rtl_433 tuned to 315MHz using the command "rtl_433 -f 315000000". Upon walking through a parking lot populated with cars, a few TMPS sensors were detected and decoded using the included TMPS protocols in rtl_433. Upon further research into TPMS sensors, the low power and infrequent transmissions when stationary made TPMS sensors less desirable for a lab setup. Conversely, key fobs can be held close to the receiving antenna and can be activated on command. Because of this, remote key fobs were determined to be the focus for the remainder of the project. At this point in the project, GNU radio was determined to be a good option for recording and analysis of the key fobs.

B. *GNU Radio*

GNU Radio and GNU Radio Companion were installed automatically during the FISSURE installation [16]. The beginner tutorials featured on the GNU Radio wiki were completed in order to achieve a basic understanding of how to construct a flowgraph [15]. It quickly became apparent that to utilize GNU Radio to its fullest potential, a large amount of time would need to be spent learning electrical engineering concepts. A simple GNU Radio flow graph was created in order to record key fob signals from the RTL-SDR dongle and save the raw signal data to a file.

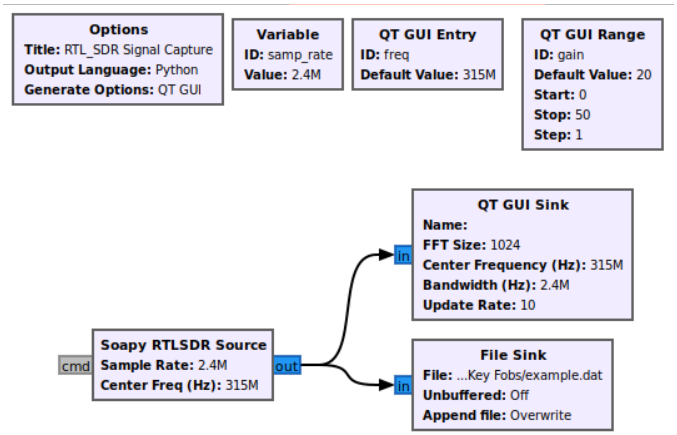

Figure 11— GNU Radio recording flowgraph

Around the same time a GNU Radio module called gr-inspector was discovered. The gr-inspector module describes itself as a signal analysis toolbox for GNU Radio [18]. A simple GNU Radio flowgraph was constructed to replay saved signals. The flowgraph displays recordings and automatically detects signals placing a bounding box around them.

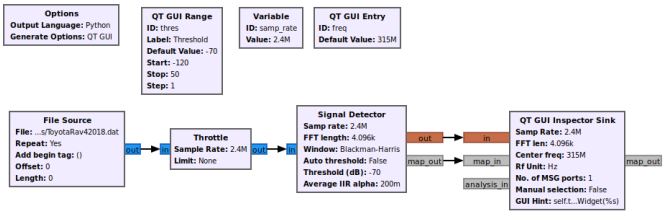

Figure 12— GNU Radio Inspector flowgraph

Another simple GNU Radio flowgraph for replaying the signals was created using the default blocks. This flowgraph enables the recorded signals to be viewed on a waterfall graph.

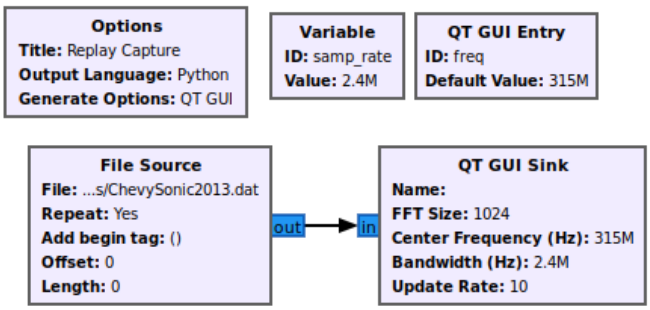

Figure 13— Radio replay flowgraph

At this time during the project, signals had only been recorded from one key fob that belonged to a 2013 Chevrolet Sonic.

C. *Recording Key Fob Signals*

In order to compare key fob signals to each other and eventually fingerprint them to a make and model, many signal recordings were required. Throughout the Spring 2024 semester, individuals were asked to be participants and allow their key fob transmissions to be recorded. This was the done by first confirming the frequency in which the key fob was transmitting at using SDR++ and then using the GNU Radio flow graph to record it. After, the signal was replayed to confirm an acceptable recording. Very quickly it was apparent that some makes and models of cars had key fob transmissions that were similar in appearance and some were very different. A total of 14 key fob signals were recorded from cars of unique makes and model.

D. *Signal Analysis*

Once the key fob signals were collected, the 14 signals were replayed using the GNU Radio flowgraph and visually inspected. In both the frequency spectrum and waterfall plot similarities and differences were noted between the recorded signals. The signals were classified by appearence. The recorded signals were normalized using the FISSURE's signal conditioner feature with the sample rate set to 2.4 MS/s and the frequency set to 315 MHz. Otherwise, the settings were left as default. This created 193 conditioned signals from the 14 original recordings. The conditioned signals were then inputted to FISSURE's feature extractor feature and a spreadsheet of extracted feature values was produced. The values were plotted onto graphs and any correlations were noted.

IV. Results

The proposed hypothesis was that based on aspects of an intercepted automotive key fob signal, the make and model of the corresponding car could be predicted. Visually inspecting frequency spectrum and waterfall plots of the recorded signals yielded obvious notable similarities and differences. An attempt to execute a deep statistical analysis proved inconclusive.

A. *Signal Classifications*

The recorded signals were visually classified into three different classifications: single peak, double peak and multi-peak. Single peak signals were defined as consisting of a single, well defined amplitude peak with no other peaks. For example, the signal recorded from a 2013 Subaru Impreza key fob is classified as single peak signal (see figures A9 and B9). A double peak signal has two well defined peaks. For example, the signal recorded from a 2023 Mazda CX-50 key fob is classified as double peak signal (see figures A8 and B8). A multi-peak signal has more than two well defined peaks. For example, the signal recorded from a 2009 Dodge Durango key fob is classified as multi-peak signal (see figures A5 and B5).

B. *Correlating Signals to Automotive Manufacturer*

While 5 of the 14 signals collected were the sole sample from their respective manufactures, the 9 other signals shared manufacturer with at least one other signal. The signals from the same manufacturer trended towards having similarities when compared to signals from other manufactures

Three Volkswagen key fob signals were collected and analyzed (see figures A12-A14 and B14-B16). These key fobs belonged to a 2013 Volkswagen CC, a 2012 Volkswagen Golf, and a 2022 Volkswagen Jetta respectively. All three of these signals were classified as single peak signals and were centered at ~315 MHz. Very minute differences can be noted from these 3 recorded signals. For example, the 2013 Volkswagen CC key fob transmission appears to have greater spurious emissions than the other Volkswagen key fob signals. Interestingly, the 2012 Volkswagen Golf key fob signal recording includes a significant amount of noise likely emitted from another device nearby at the time of recording.

Four Chevrolet key fob signals were collected and analyzed (see figures A1-A4 and B1-B4). These key fobs belonged to a 2012 Chevy Cruze, a 2018 Chevy Equinox, a 2013 Chevy Sonic, and a 2011 Chevy Suburban. The 2012 Cruze and 2013 Sonic key fob signals were classified as multi-peak signals. The 2011 Suburban key fob signal is very noisy and could not meet the well defined peaks requirement to be classified. However, there are remnants of multiple peaks among the noise. The 2018 Equinox key fob signal was classified as a single peak signal. It is noteworthy that the key fobs signals from the early 2010s all had multiple peaks and the 2018 Equinox key fob signal had a single peak. This is likely due to a change in modulation protocol used in the key fobs between 2013 and 2018. The 2012 Cruze, 2018 Equinox and 2013 Sonic key fob signals were all centered at ~314.8 MHz while the 2011 Suburban key fob signal was centered at ~315.0 Mhz.

Two Toyota key fob signals were collected and analyzed (see figures A10, A11 and B10-B13). These key fobs belonged to a 2023 Toyota Camry and a 2018 Toyota Rav4 respectively. Both of these Toyota key fob signals were classified as double peak signals. Interestingly, both Toyota key fobs transmitted a stronger signal followed by a weaker signal on a different channel. The channels used by the 2023 Camry were centered at ~315.05 MHz for the stronger transmission and at ~314.75 MHz for the weaker. The channels used by the 2018 Rav4

were centered at ~314.35 MHz for the stronger transmission and ~315.44 MHz for the weaker. This dual channel transmission at different power levels appears to be unique to the Toyota key fob signals.

Three of the remaining key fob signals were classified as double peak signals. These double peak signals were from key fobs that belonged to a 2018 Hyundai Tuscon, a 2021 Kia Forte, and a 2023 Mazda CX-50 respectively (see figures A6-A8 and B6-B8). The difference in frequency from the two peaks in the Hyundai and Kia key fob signals appears to be the same while the difference between the peak of the Mazda key fob signal is greater. Both Hyundai and Kia are South Korean companies. It is possible that Hyundai and Kia used the same radio and modulation protocol for their key fobs.

The 2008 Dodge Durango key fob signal, was classified as a multi-peak signal like some of the Chevrolet key fob signals (see figures A1-A5 and B1-B5). However, the difference in frequency between each peak was significantly less in the Dodge key fob signal when compared to the multi-peak Chevrolet key fob signals.

The 2013 Subaru Impreza key fob signal was classified as a single peak signal like the Volkswagen key fob signals (see figures A9, A12-A14 and B9, B14-B16). Overall, the Subaru key fob signal is quite similar to the Volkswagen signals and could prove difficulty to differentiate from one another if encountered in the wild.

C. *Statistical Analysis*

The spreadsheet of signal data produced using FISSURE contained data for 193 conditioned signals. Due to the large volume of data all plotted on one graph and the presence of signals produced by FISSURE that were not representative of the original recordings, no correlations were identified. A single conditioned signal from each source signal was selected that was determined to properly represent the original recorded signal. The data for these 14 selected conditioned signals were entered into a new spreadsheet. All extracted feature data was plotted in graphs and very little corelation was apparent. Two data fields, however, did appear to be slightly correlated to the key fob manufacturer.

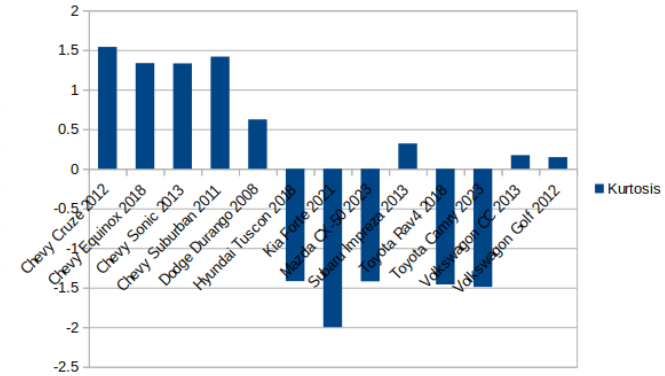

Figure 14— Kurtosis of selected conditioned signals

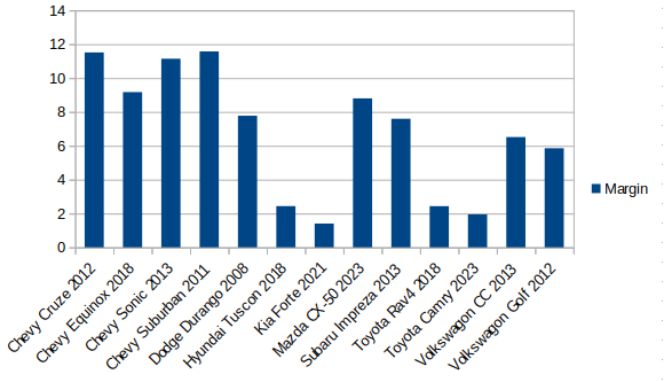

Figure 15— Margin of selected conditioned signals

The values that appear in these graphs trend towards being closer within manufacturer groups but the variance in values and cherry-picking required to create these graphs lead to the conclusion that there is still no statistically significant correlation.

V. Discussion

A. *Experiment Conclusions*

The results presented demonstrated that based on aspects of an intercepted automotive key fob signal, the make and model of the corresponding car can be predicted. The recorded signals that were associated with the same manufacturer shared aspects with each other and were typically significantly different between manufacturer. There were also notable differences within manufacturer groups that could be used to predict with higher precision. Each recorded key fob signal was entirely unique and, as the practice of RF fingerprinting presupposes, each individual key fob could be identified based on their emitted signals.

B. *Wireless Security*

The research performed supports the security concerns associated with radio transmissions and wireless communication. Radio transmissions can be identified and intercepted by anyone within range of the transmission with the appropriate technology and knowledge. Modern technology relies heavily on radio transmissions for wireless network infrastructure and to remote controlled devices. While encryption can be used to mitigate eavesdropping attacks, intelligence can still be gathered from the transmissions without decryption which could be used for future efforts or attacks.

C. *Accessible SIGINT Platform*

The basic setup for this project was built using a refurbished laptop that was purchased for under $200, the RTL-SDR dongle with antenna kit that cost ~$40. This enabled the use of free, open source SDR software like SDR++, rtl_433, and FISSURE. Other free, open source software can be to used to gather signal intelligence (SIGINT) like listening to emergency service communications, tracking aircraft, tracking boats, sniffing GSM (2G cellular) signals, decoding

GPS signals, scanning trunked radio, and much more [3]. All of these applications could be done independent of internet functionality and without transmitting signals, only receiving. It is demonstrated that for a very low entry cost, an individual can stealthily gather SIGINT and investigate their local radio frequency transmissions.

D. Privacy and Security Implications

Intelligence can be dangerous when possessed by the wrong parties and lifesaving when possessed by the right people. This SIGINT platform implementation enables anyone with a computer and ~$40 to spend on radio equipment to collect SIGINT. Bad actors can use relevant SIGINT to assist in performing illegal acts or evade security systems. Conversely, actors acting in good faith can use SIGINT to maintain a higher level of situational awareness and potentially detect, predict, and record the actions of bad actors. A low cost SIGINT platform like the one demonstrated here could be distributed across a large area by an organization looking to establish SIGINT listening posts with little infrastructure required. This solution has different advantages and disadvantages when compared to typical, more powerful SIGINT platforms that cost thousands if not millions of dollars.

E. Difficulties Faced

Proper RF signal analysis requires a higher knowledge of electrical engineering and statistical analysis concepts than what was possessed at the time of this project. Much of the features offered by GNU Radio and FISSURE requires knowledge of electrical engineering and signal processing. Because of this, GNU Radio was utilized in a very rudimentary manner and the use of FISSURE did not produce valuable results. Tire pressure monitor sensors (TPMS) initially seemed very useful for the purpose of fingerprinting vehicles due to their unique identifier that is transmitted wirelessly. However, TPMS sensors' low power, infrequent transmissions, and difficulty to isolate made them challenging to use for a lab. This was overcome by focusing on key fobs which a variety could be recorded from willing participants pressing their unlock or lock buttons. FISSURE's signal conditioner feature seemed like a promising ready-made solution to normalize the collected signals, however, this feature did not behave as it was initially assumed. A higher understanding of FISSURE, GNU Radio and signal processing and analysis concepts could have enabled better statistical analysis of the collected signals. Collecting key fob signals required participants that possessed key fobs and were willing to have the signal recorded. Once the initial key fob signals were captured, the analysis portion of the project began. In retrospect, more and greater variety of signals would have enabled a higher level of analysis but by the time this was realized, the population of potential participants were no longer accessible.

F. Future Work

Future work on this topic could add functionality to this proposed SIGINT platform or attempts a similar experiment analyzing car key fob signals. Focusing on a specific protocol and demodulating and decoding the signals is the next logical step for a SIGINT platform like this. This could be achieved using GNU Radio. Gathering more and a greater variety of key fob signals would enable more useful statistical analysis and potentially prove or disprove the conclusions made in this paper. Proper statistical analysis could be performed with more complex signal processing using larger GNU Radio flowgraphs. Deep neural networks trained on waterfall plots of signal transmissions could be deployed to streamline the SIGINT workflow presented in this paper. [19] presents such deep neural network solutions for discovery of Long Range (LoRa) communication signals.

VI. Conclusion

This paper presented a SIGINT platform using readily available equipment and free, open source software. Using a software defined radio (SDR), the ability to capture and analyze wireless signals, specifically automotive key fobs, was demonstrated. The analysis illustrated the capability of this SIGINT platform to attribute captured signals to the device that emitted them. While this platform was effective for visual analysis of signals, proper statistical analysis was not achieved. A SIGINT platform with a low barrier to entry enables anyone to gather potentially sensitive information. This paper expands the understanding of SIGINT capabilities and demonstrates the potential of open source SDR tools to enable individuals to explore the radio frequency spectrum.

References

[1] *M. M. Aid and C. Wiebes, "Introduction on The Importance of Signals Intelligence in the Cold War," Intelligence and National Security, vol. 16, no. 1, pp. 1–26, Mar. 2001, doi: https://doi.org/10.1080/714002838.*

[2] M. Teran, J. Aranda, J. Marin, E. Uchamocha, and G. Corzo-Ussa, "A methodology for signals intelligence using non-conventional techniques and software-defined radio," IEEE Colombian Communications Conference (COLCOM) , vol. 2021, May 2021, doi: https://doi.org/10.1109/colcom52710.2021.9486297.

[3] "About RTL-SDR," rtl-sdr.com, Apr. 11, 2013. https://www.rtl-sdr.com/about-rtl-sdr/

[4] V. Brik, S. Banerjee, M. Gruteser, and S. Oh, "Wireless Device Identification with Radiometric Signatures," 2008. Accessed: Aug. 10, 2024. [Online]. Available: https://www.winlab.rutgers.edu/~gruteser/papers/brik_paradis.pdf

[5] R. Dover, Routledge Companion to Intelligence Studies. Routledge, 2013.

[6] pothosware, "Soapy SDR plugin for RTL-SDR," GitHub, Mar. 15, 2020. https://github.com/pothosware/SoapyRTLSDR/wiki (accessed Aug. 10, 2024).

[7] "SDR++," www.sdrpp.org. https://www.sdrpp.org/

[8] "GNU Radio - The Free & Open Source Radio Ecosystem · GNU Radio," GNU Radio. https://www.gnuradio.org/

[9] C. Poore, "ainfosec/FISSURE," GitHub, May 28, 2024. https://github.com/ainfosec/FISSURE

[10] B. Larsson, "rtl_433," GitHub, May 10, 2022. https://github.com/merbanan/rtl_433

[11] "Install Ubuntu desktop," Ubuntu. https://ubuntu.com/tutorials/install-ubuntu-desktop#1-overview

[12] "balenaEtcher - Flash OS images to SD cards & USB drives," etcher.balena.io. https://etcher.balena.io/

[13] "Quick Start Guide," rtl-sdr.com, Apr. 11, 2013. https://www.rtl-sdr.com/rtl-sdr-quick-start-guide/

[14] "GNU Radio," wiki.gnuradio.org. https://wiki.gnuradio.org/

[15] C. Poore, “FISSURE - The RF Framework — FISSURE documentation,” Readthedocs.io, 2021. https://fissure.readthedocs.io/en/latest/

[16] AlexandreRouma, “Release SDR++ Nightly Build (currently 1.2.0) · AlexandreRouma/SDRPlusPlus,” GitHub, Nov. 22, 2022. https://github.com/AlexandreRouma/SDRPlusPlus/releases/tag/nightly

[17] “GitHub - gnuradio/gr-inspector: Signal Analysis Toolbox for GNU Radio,” GitHub, 2016. https://github.com/gnuradio/gr-inspector

[18] Y. Zou, J. Zhu, X. Wang, and L. Hanzo, “A Survey on Wireless Security: Technical Challenges, Recent Advances, and Future Trends,” Proceedings of the IEEE, vol. 104, no. 9, pp. 1727–1765, Sep. 2016, doi: https://doi.org/10.1109/jproc.2016.2558521.

[19] E. Lo and J. Kohl, “Internet of Things (IoT) Discovery Using Deep Neural Networks,” Mar. 2020, doi: https://doi.org/10.1109/wacv45572.2020.9093371.

APPENDIX A

WATERFALL PLOT IMAGES

The following images are screenshots of GNU Radio Companion running a simple flowgraph that displays the corresponding signals in a waterfall plot. The waterfall plot is centered at 315 MHz and the frequency values on the X axis represent deviation from 315 MHz.

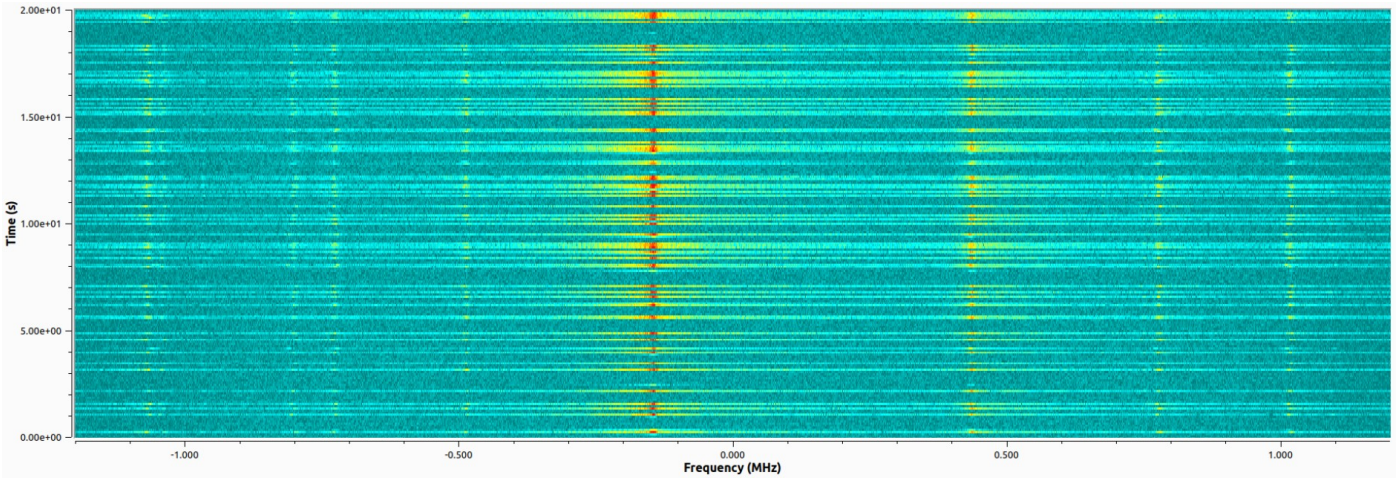

Figure A1— 2012 Chevrolet Cruze key fob signal waterfall plot

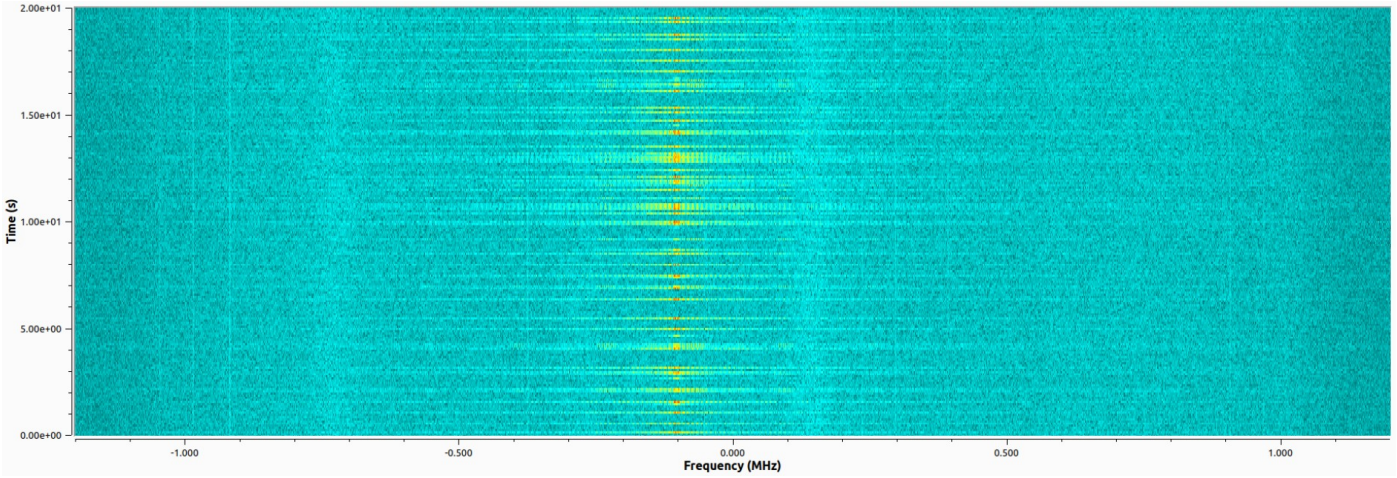

Figure A2— 2018 Chevrolet Equinox key fob signal waterfall plot

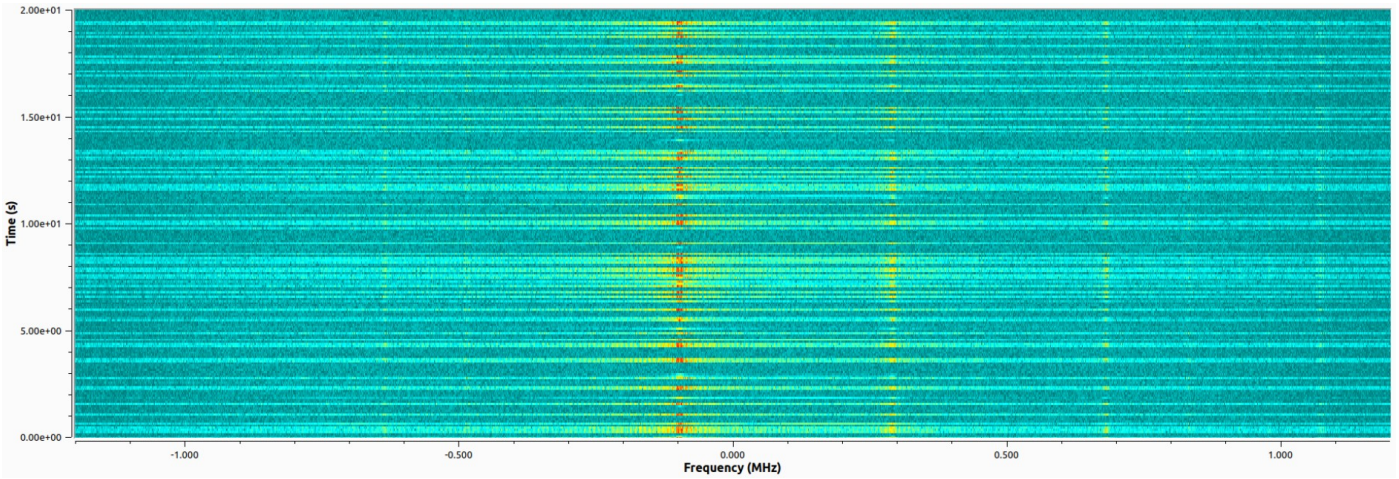

Figure A3— 2013 Chevrolet Sonic key fob signal waterfall plot

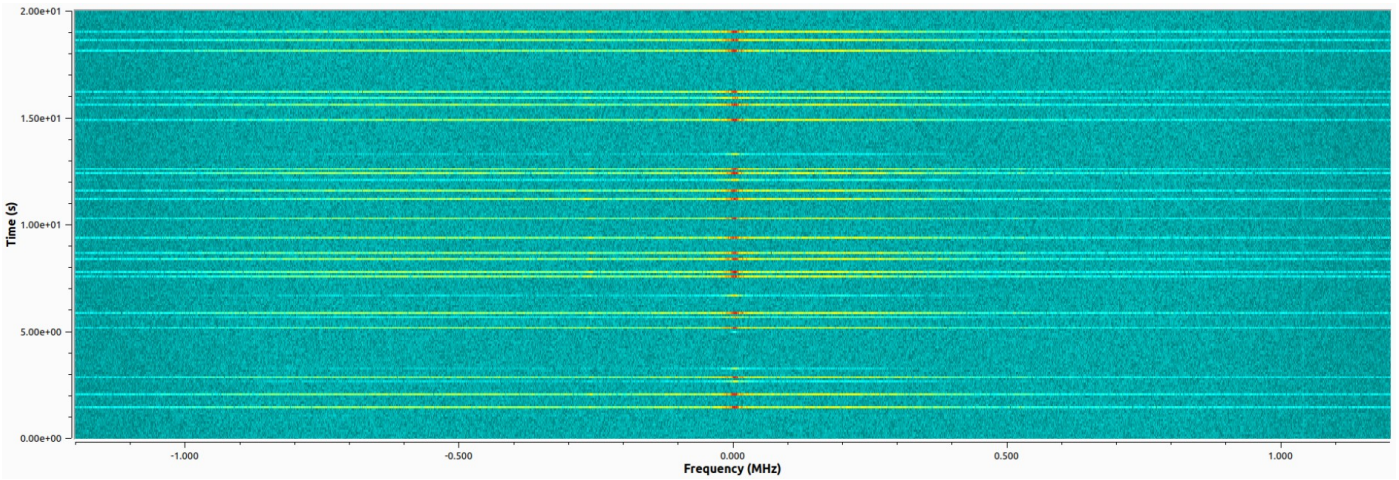

Figure A4— 2011 Chevrolet Suburban key fob signal waterfall plot

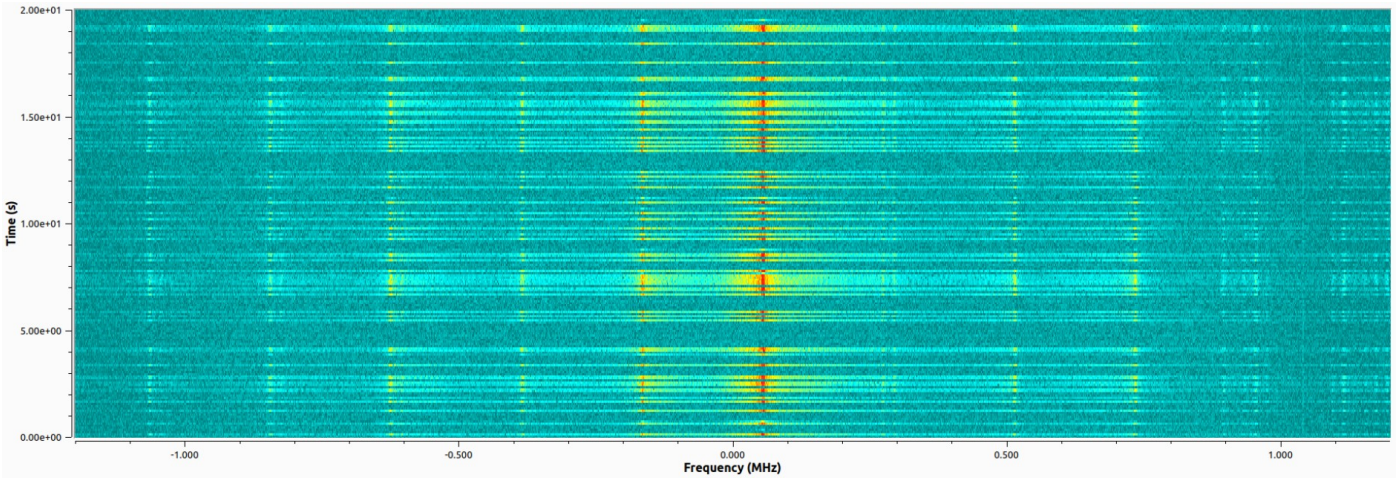

Figure A5— 2008 Dodge Durango key fob signal waterfall plot

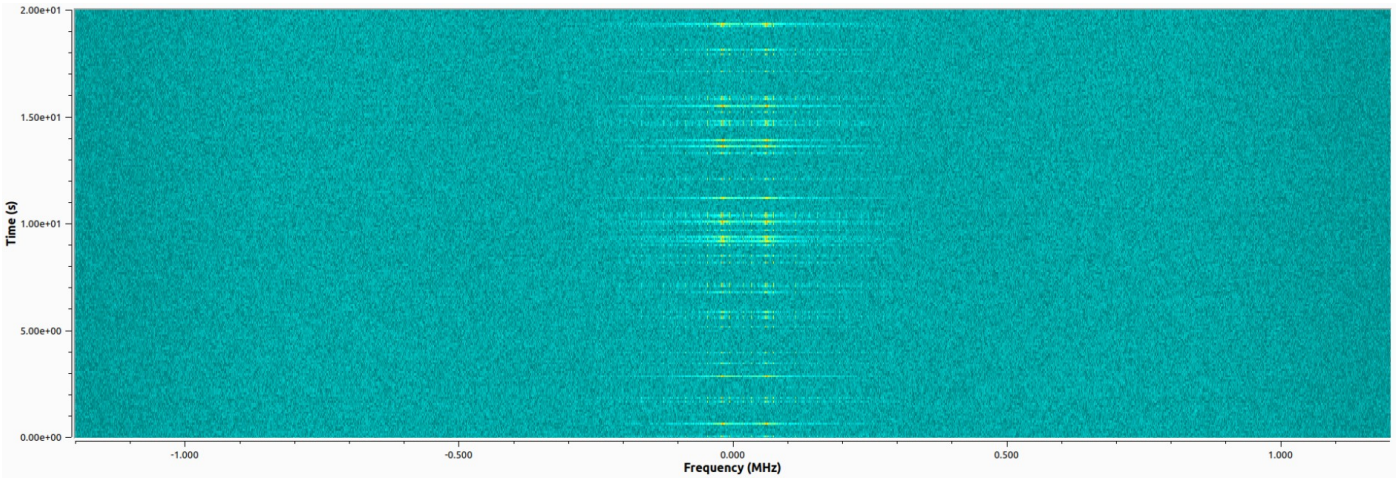

Figure A6— 2018 Hyundai Tuscon key fob signal waterfall plot

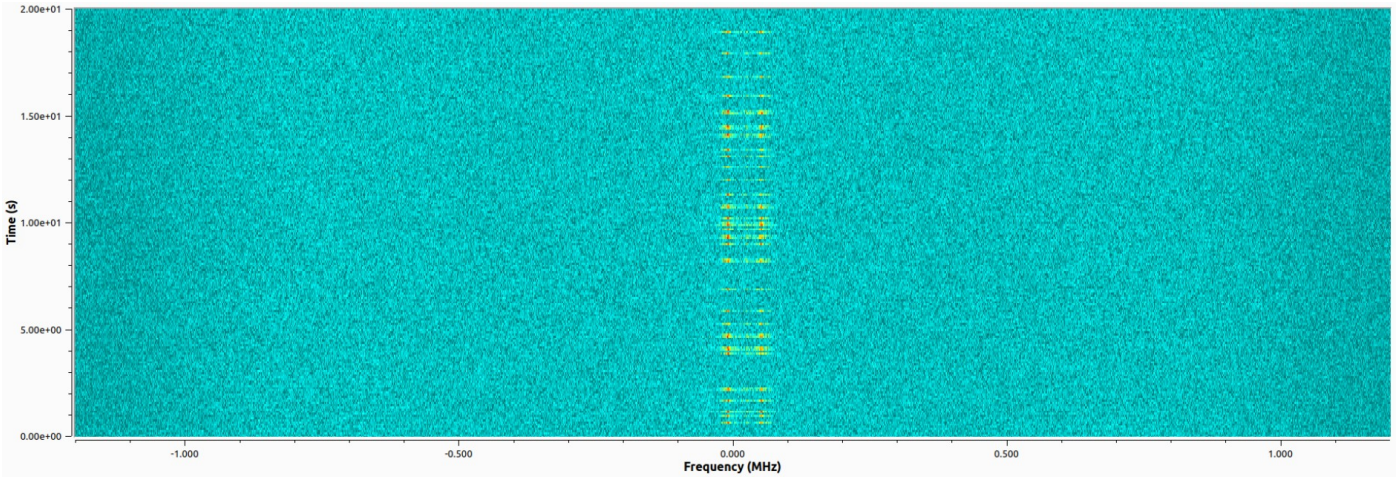

Figure A7— 2021 Kia Forte key fob signal waterfall plot

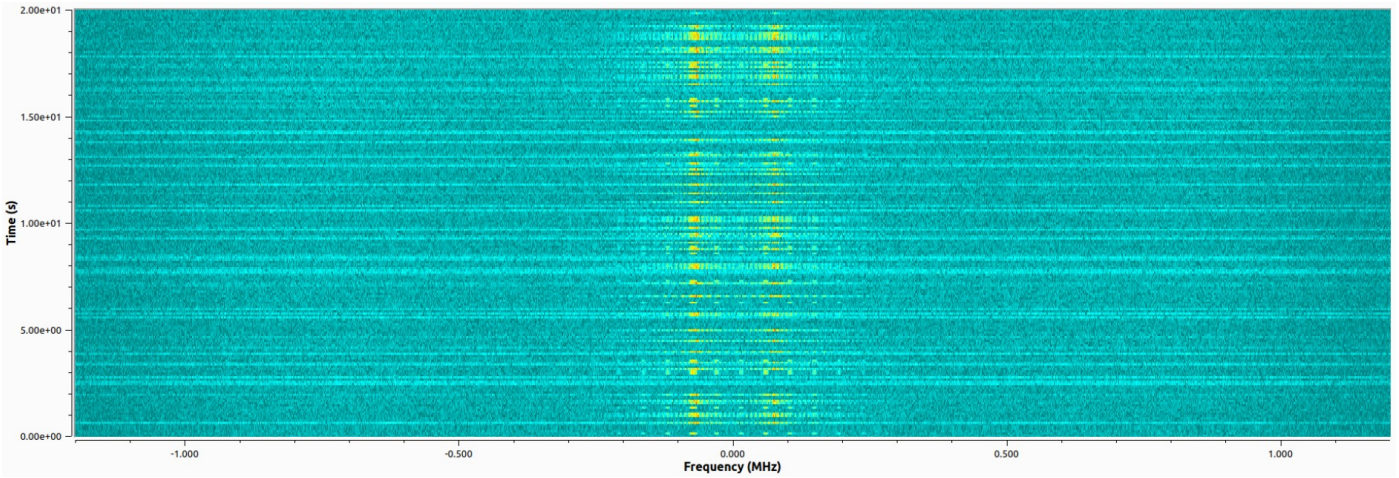

Figure A8— 2023 Mazda CX-50 key fob signal waterfall plot

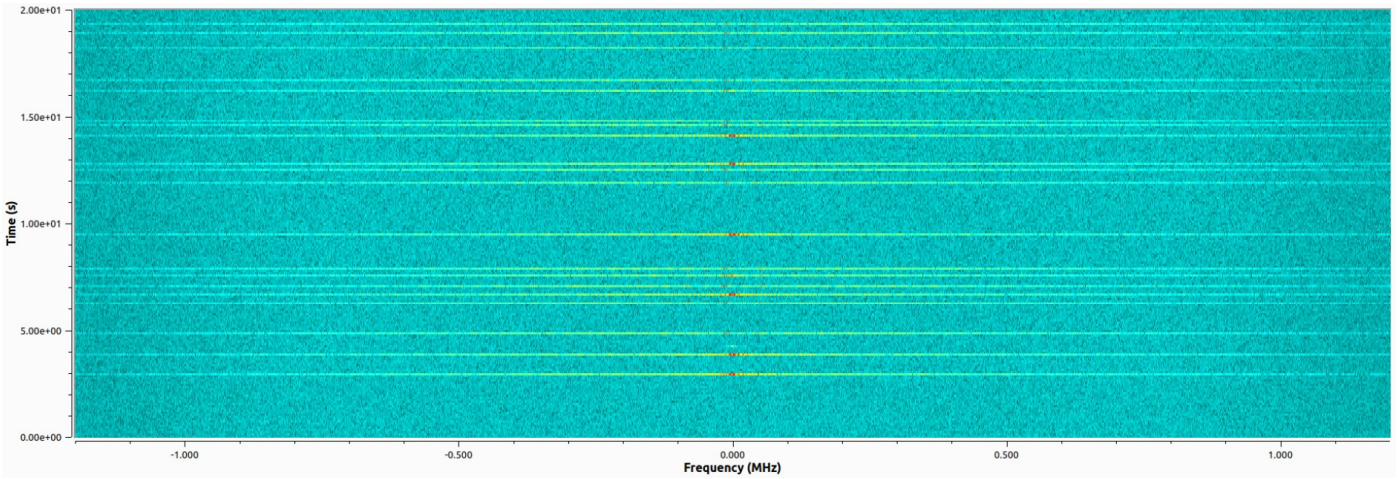

Figure A9— 2013 Subaru Impreza key fob signal waterfall plot

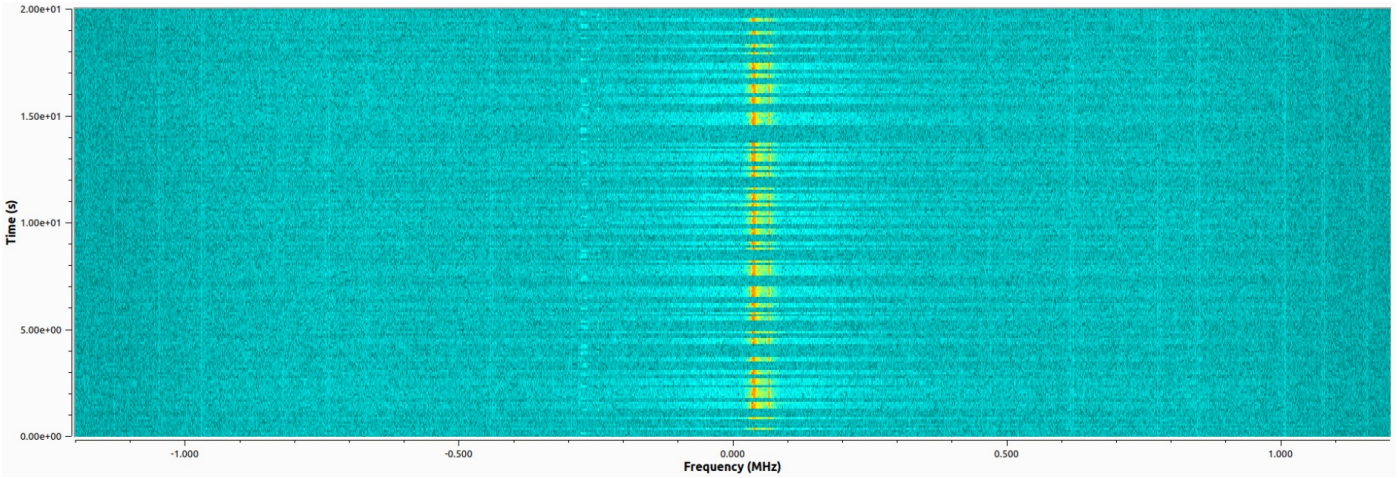

Figure A10— 2023 Toyota Camry key fob signal waterfall plot

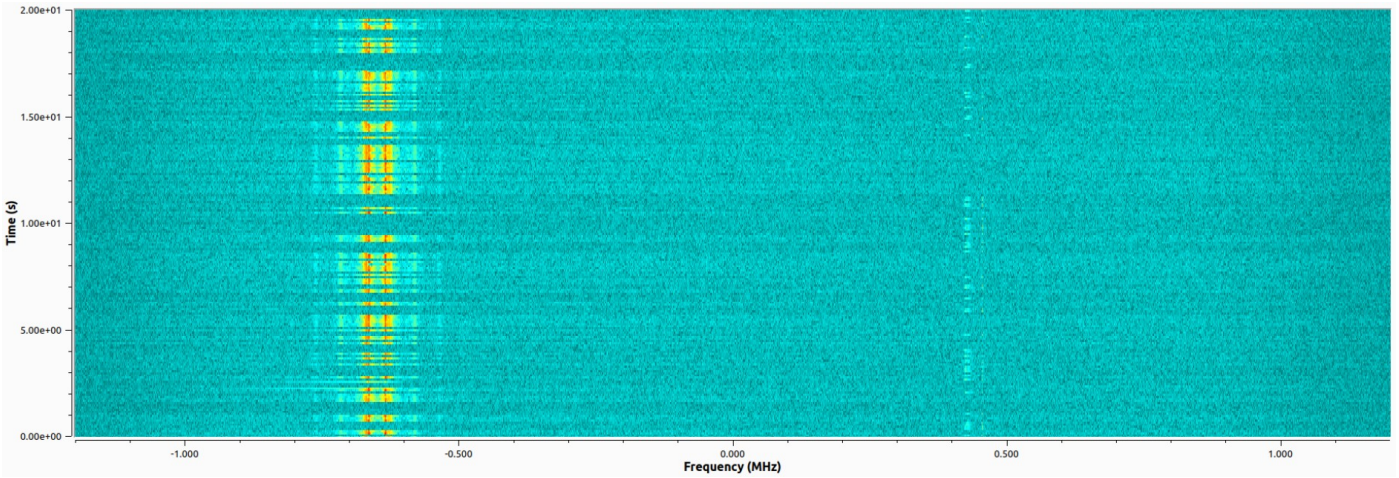

Figure A11— 2018 Toyota Rav4 key fob signal waterfall plot

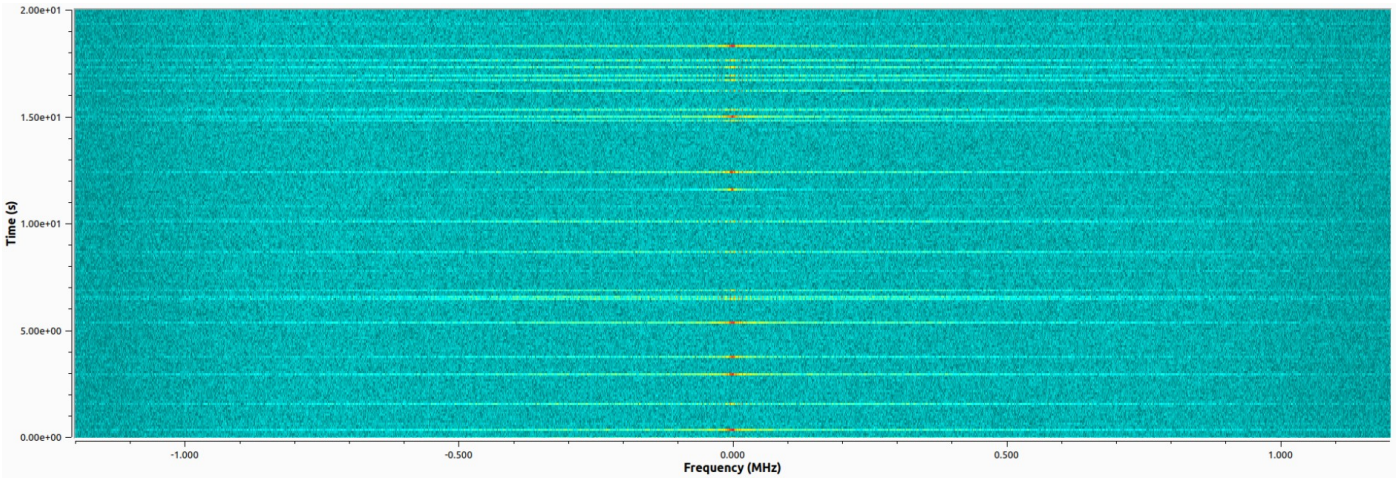

Figure A12— 2013 Volkswagen CC key fob signal waterfall plot

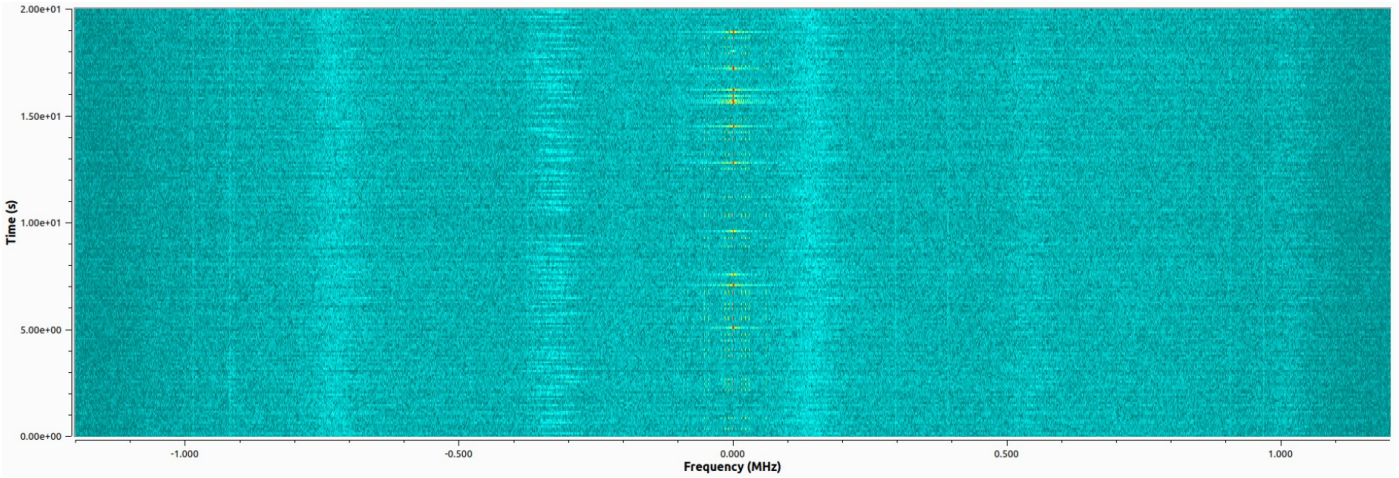

Figure A13— 2012 Volkswagen Golf key fob signal waterfall plot

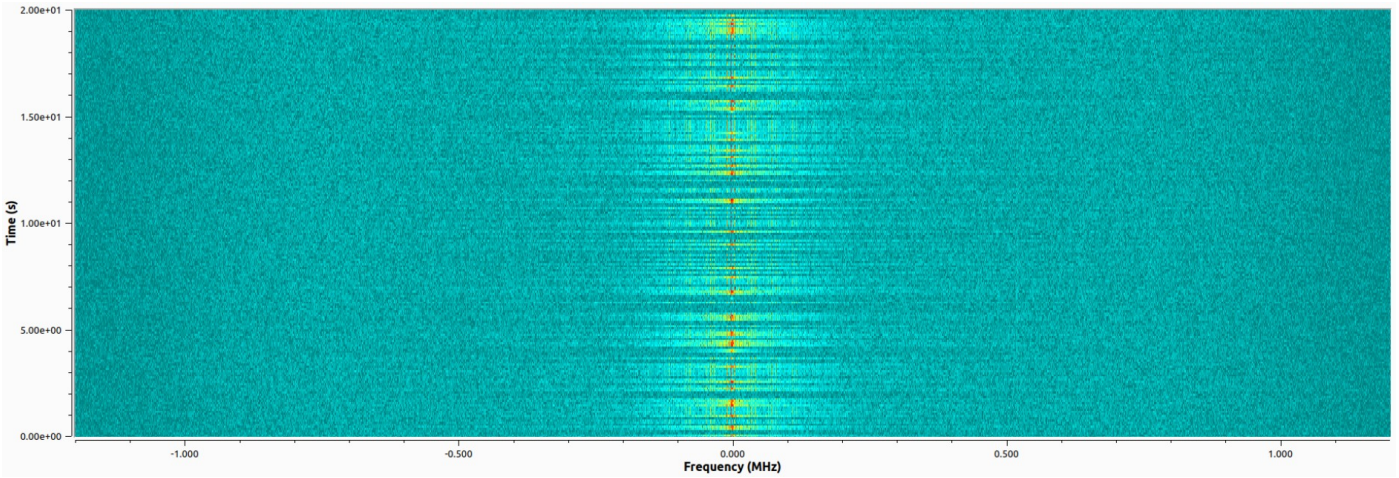

Figure 16— Figure A14— 2022 Volkswagen Jetta key fob signal waterfall plot

VII. Appendix B

Frequency Spectrum Plot Images

The following images are screenshots of GNU Radio Companion running a simple flowgraph using the Inspector GUI that displays the corresponding signals in a frequency spectrum plot. The screenshots capture the signals during a peak amplitude during the recorded transmission.

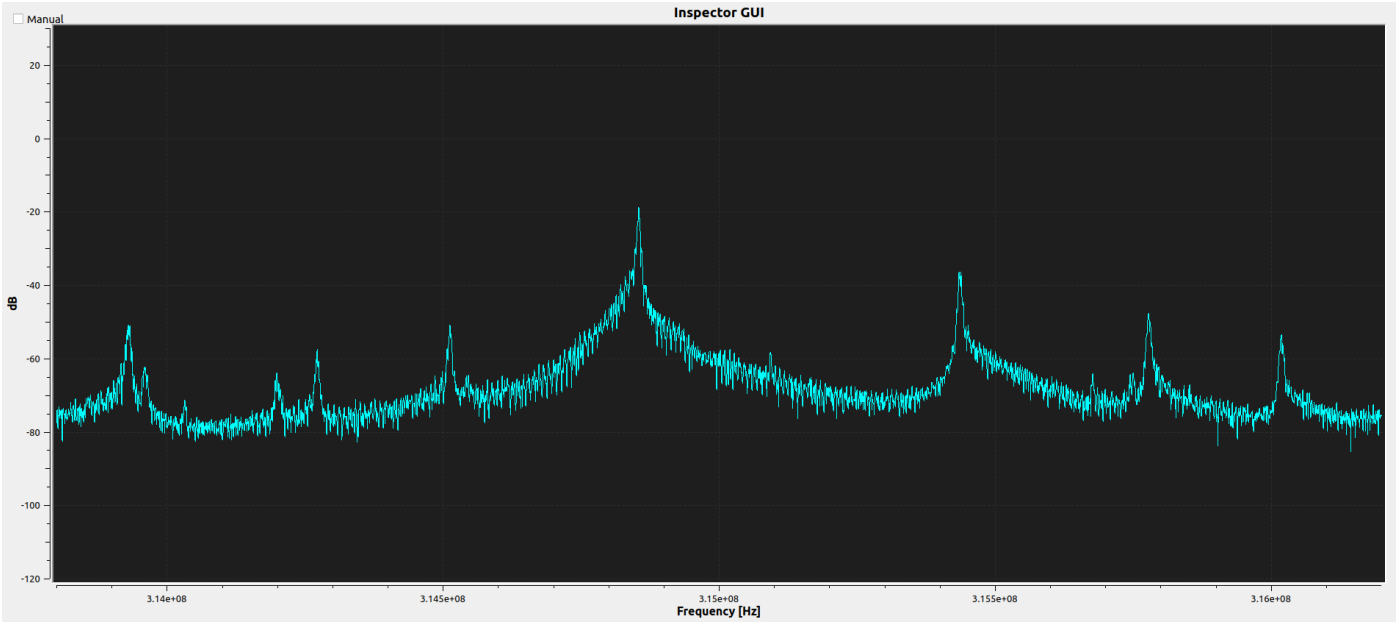

Figure B1— 2012 Chevrolet Cruze key fob signal spectrum plot

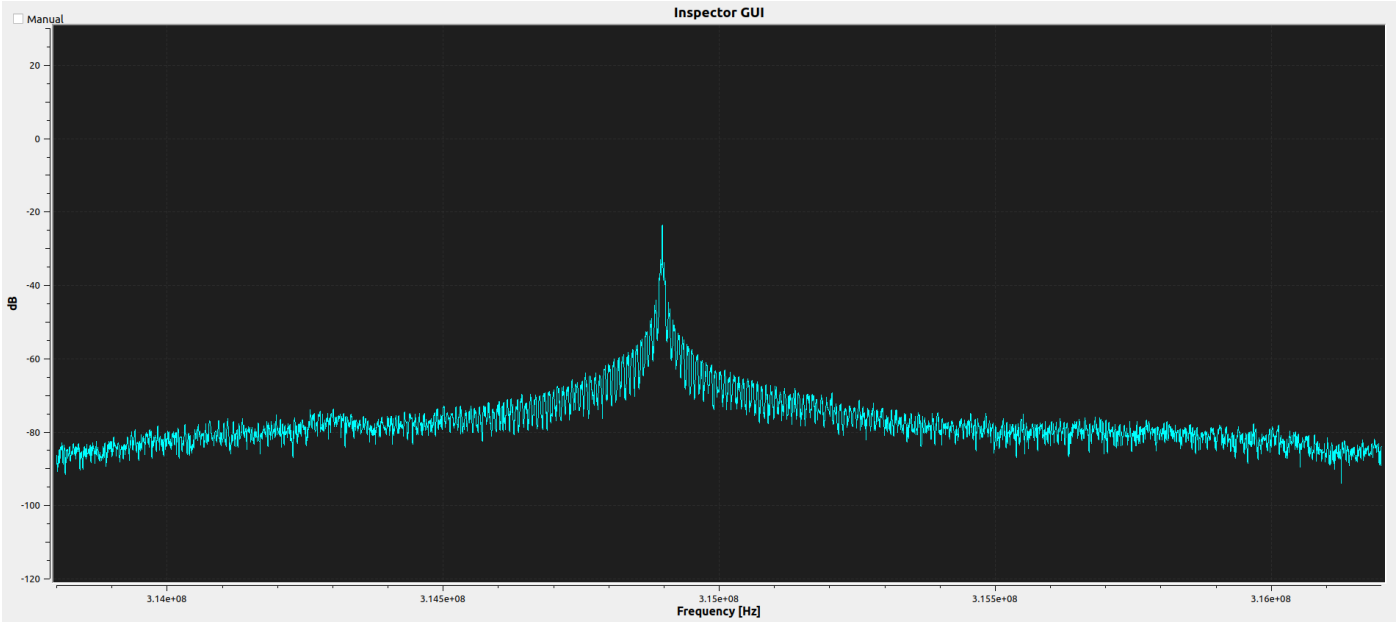

Figure B2— 2018 Chevrolet Equinox key fob signal spectrum plot

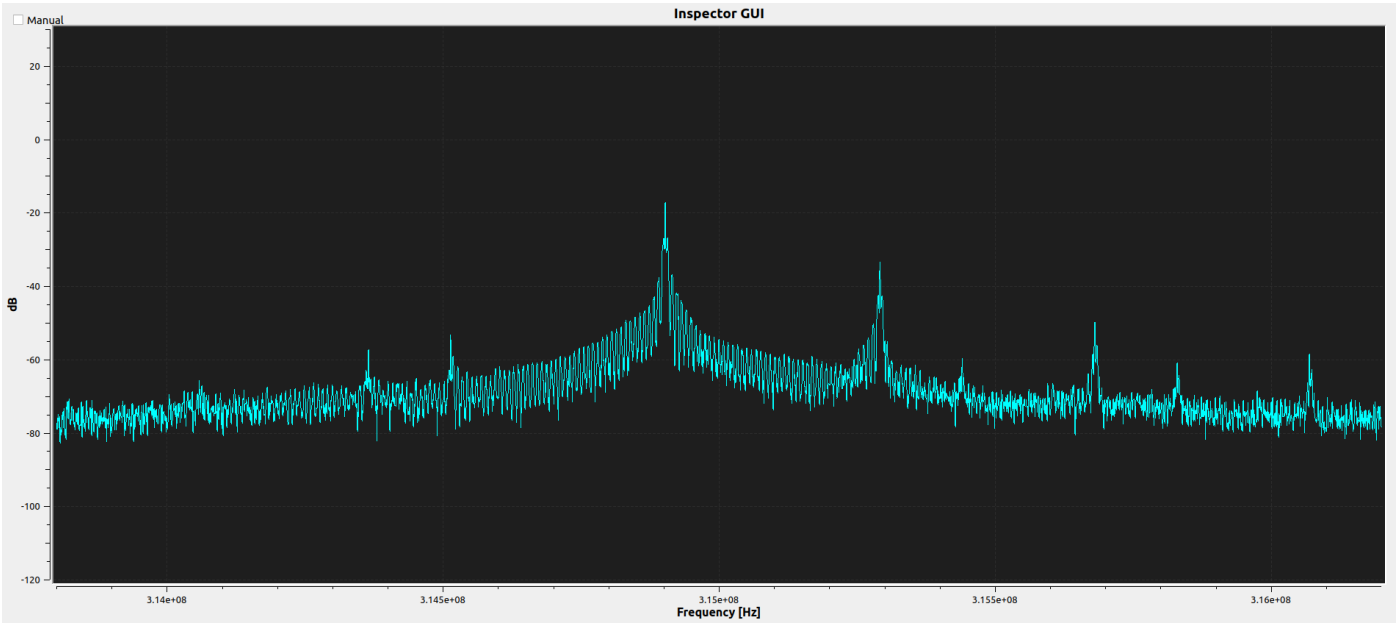

Figure B3— 2013 Chevrolet Sonic key fob signal spectrum plot

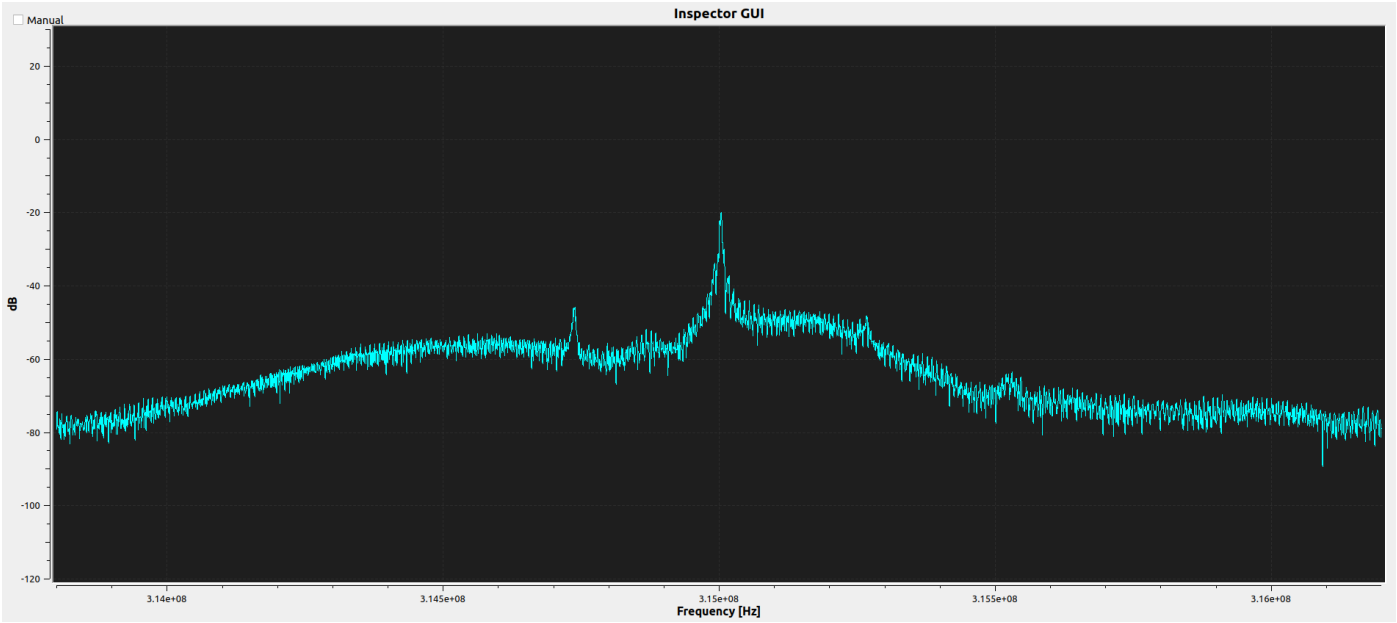

Figure B4— 2011 Chevrolet Suburban key fob signal spectrum plot

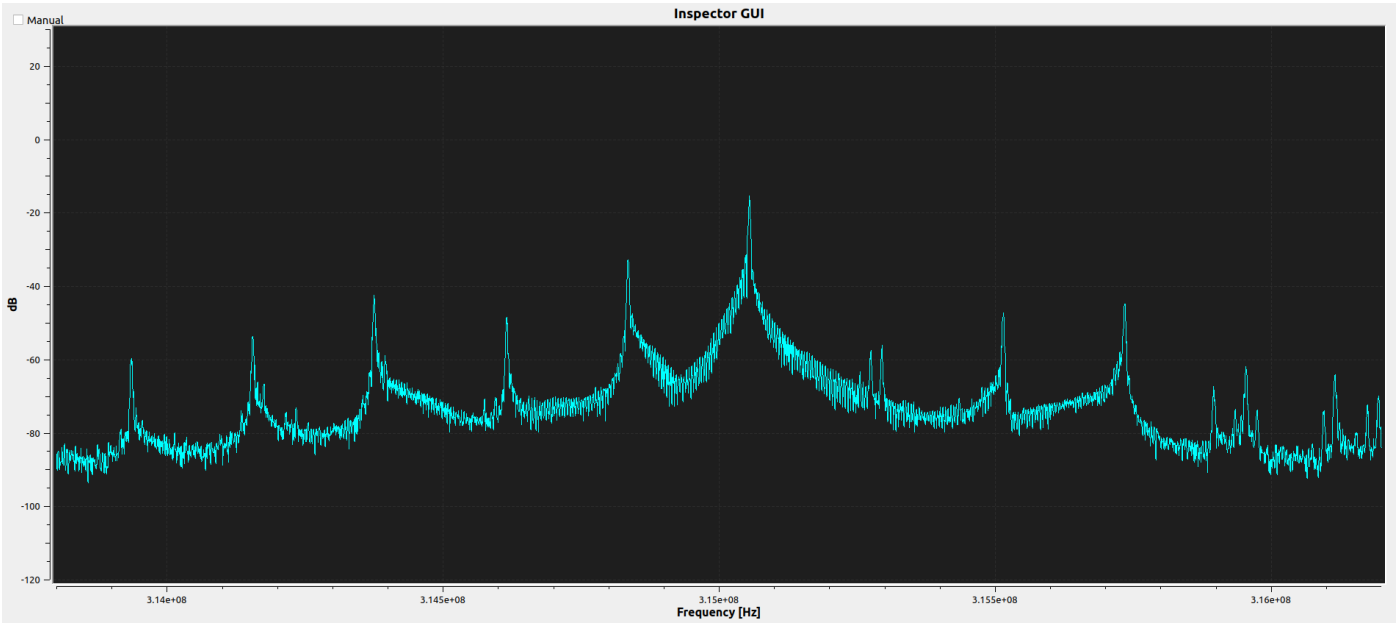

Figure B5— 2008 Dodge Durango key fob signal spectrum plot

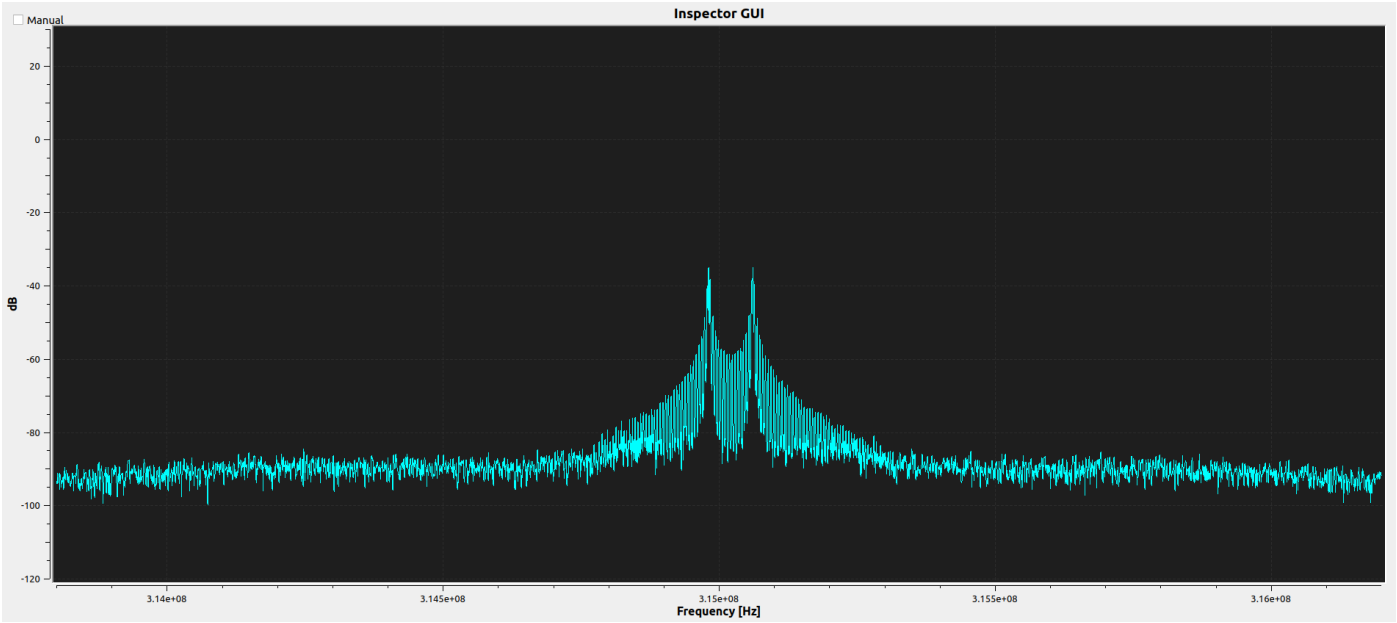

Figure B6— 2018 Hyundai Tuscon key fob signal spectrum plot

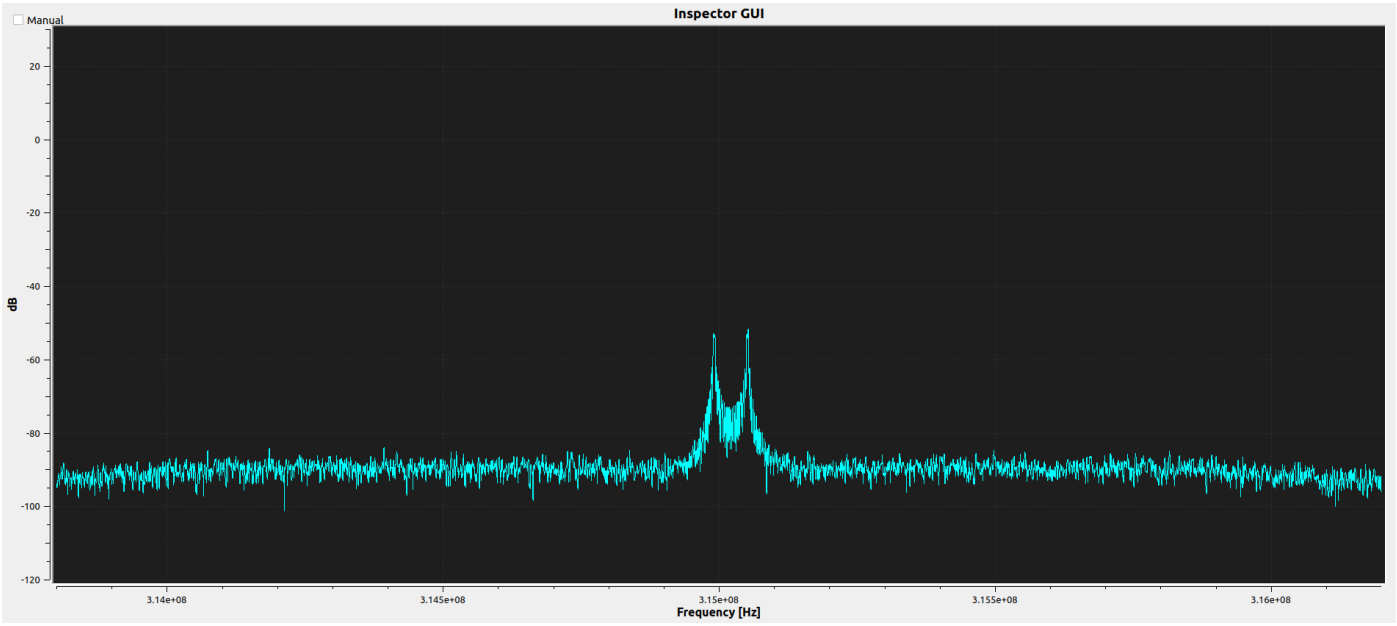

Figure B7— 2021 Kia Forte key fob signal spectrum plot

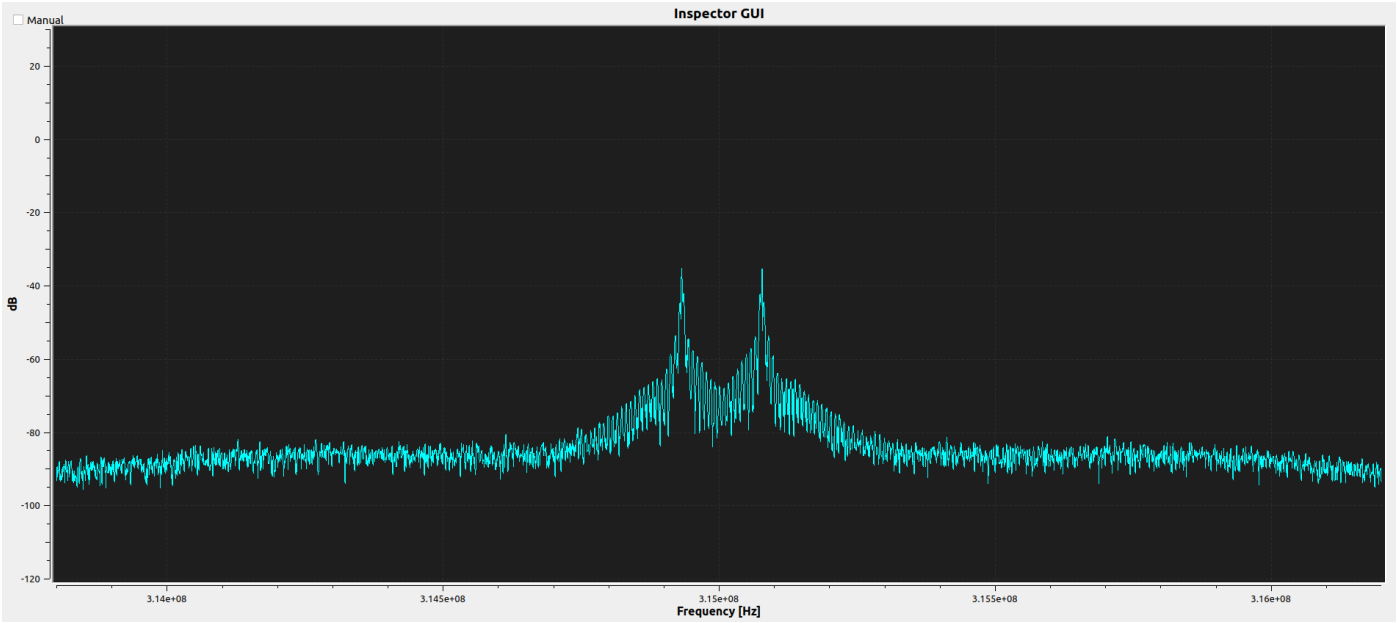

Figure B8— 2023 Mazda CX-50 key fob signal spectrum plot

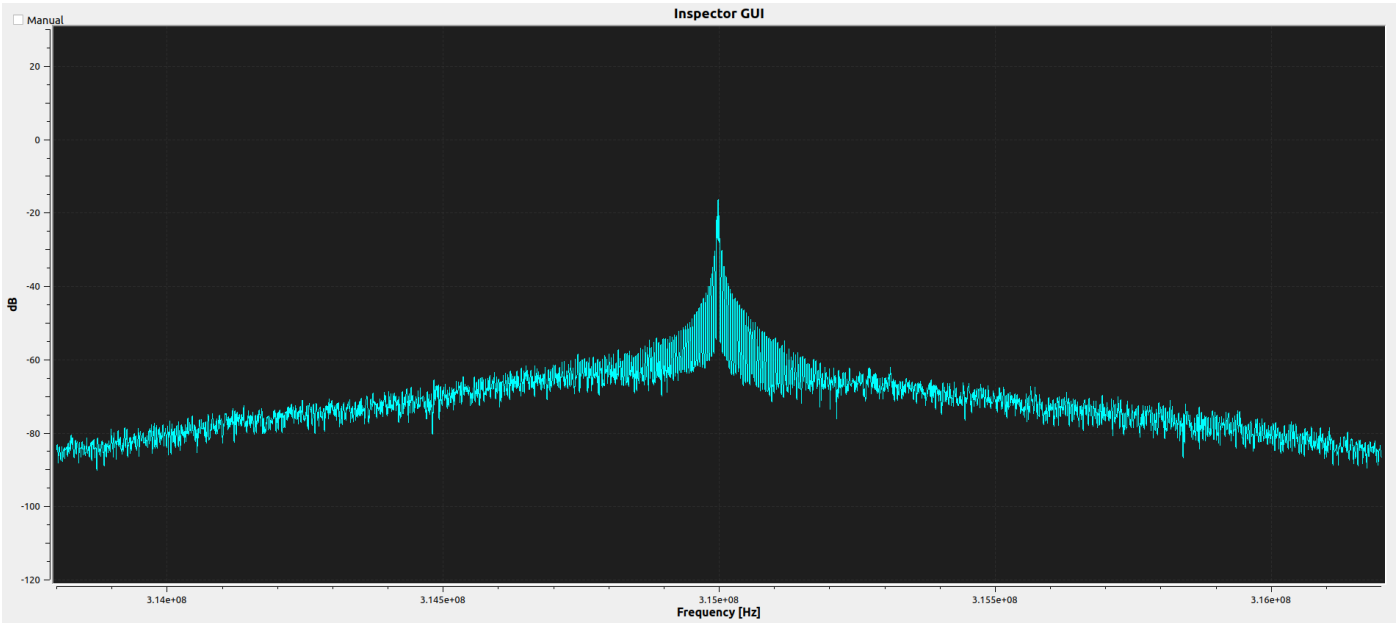

Figure B9— 2013 Subaru Impreza key fob signal spectrum plot

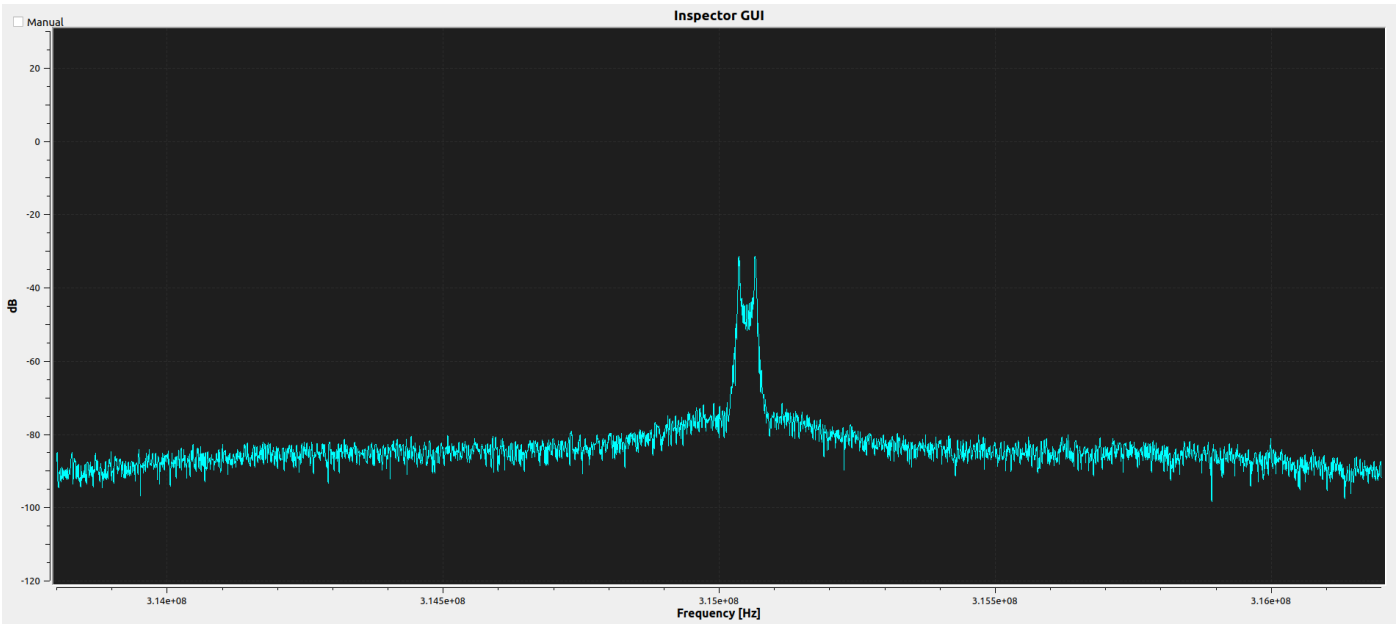

Figure B10— 2023 Toyota Camry key fob strong signal spectrum plot

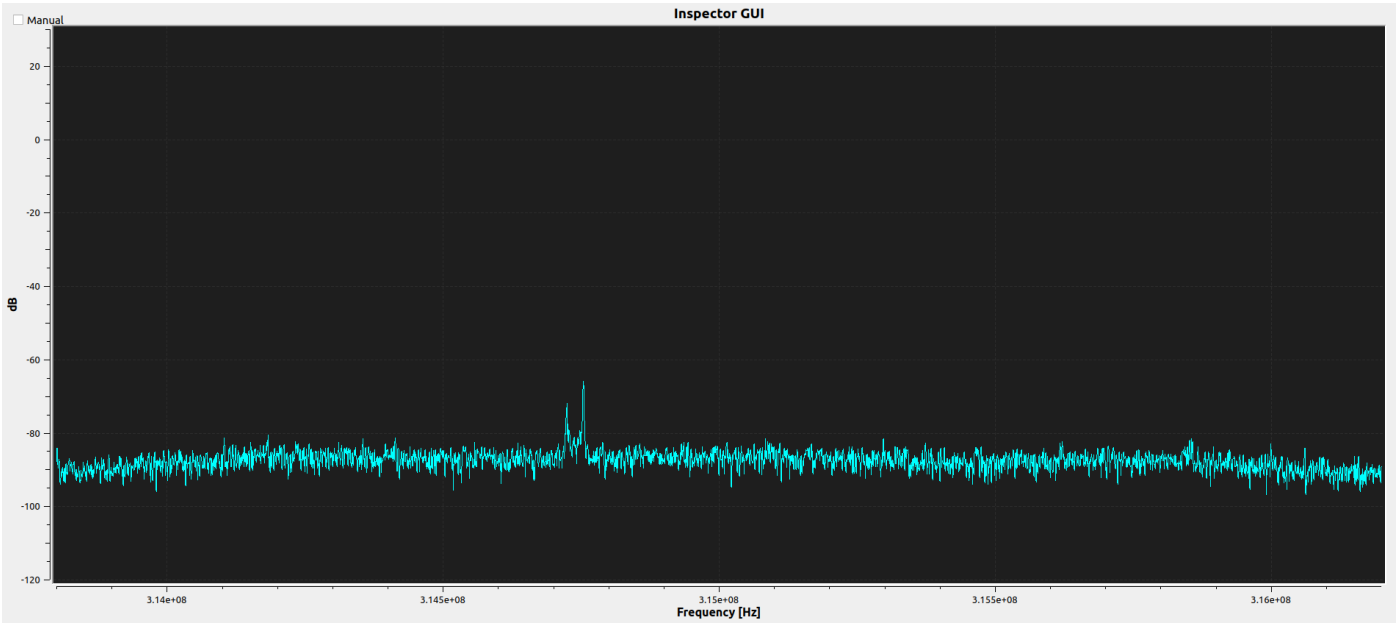

Figure B11— 2023 Toyota Camry key fob weak signal spectrum plot

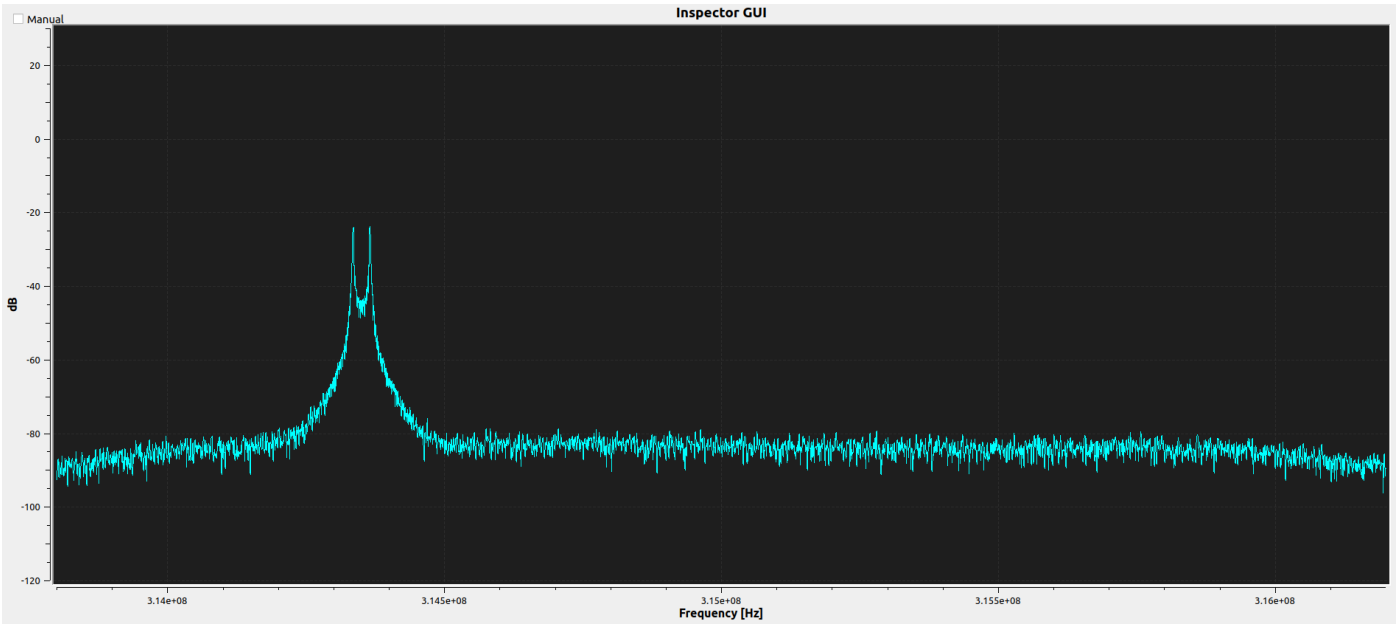

Figure B12— 2018 Toyota Rav4 key fob strong signal spectrum plot

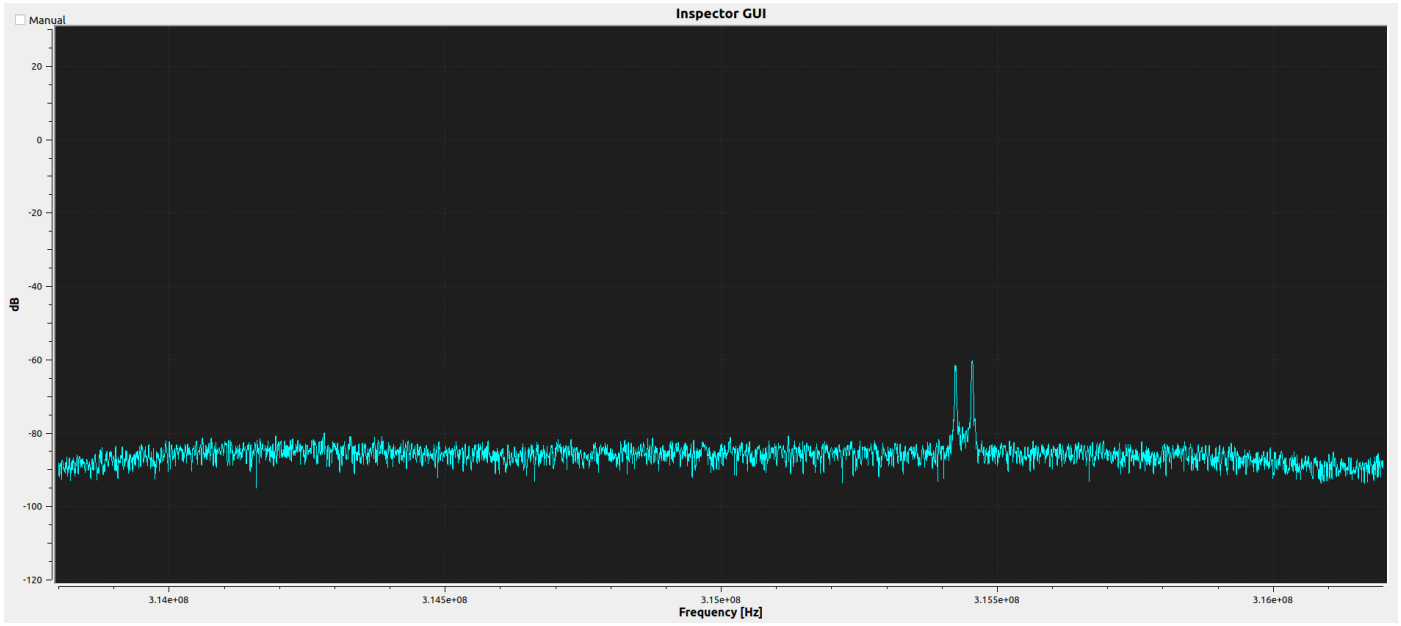

Figure B13— 2018 Toyota Rav4 key fob weak signal spectrum plot

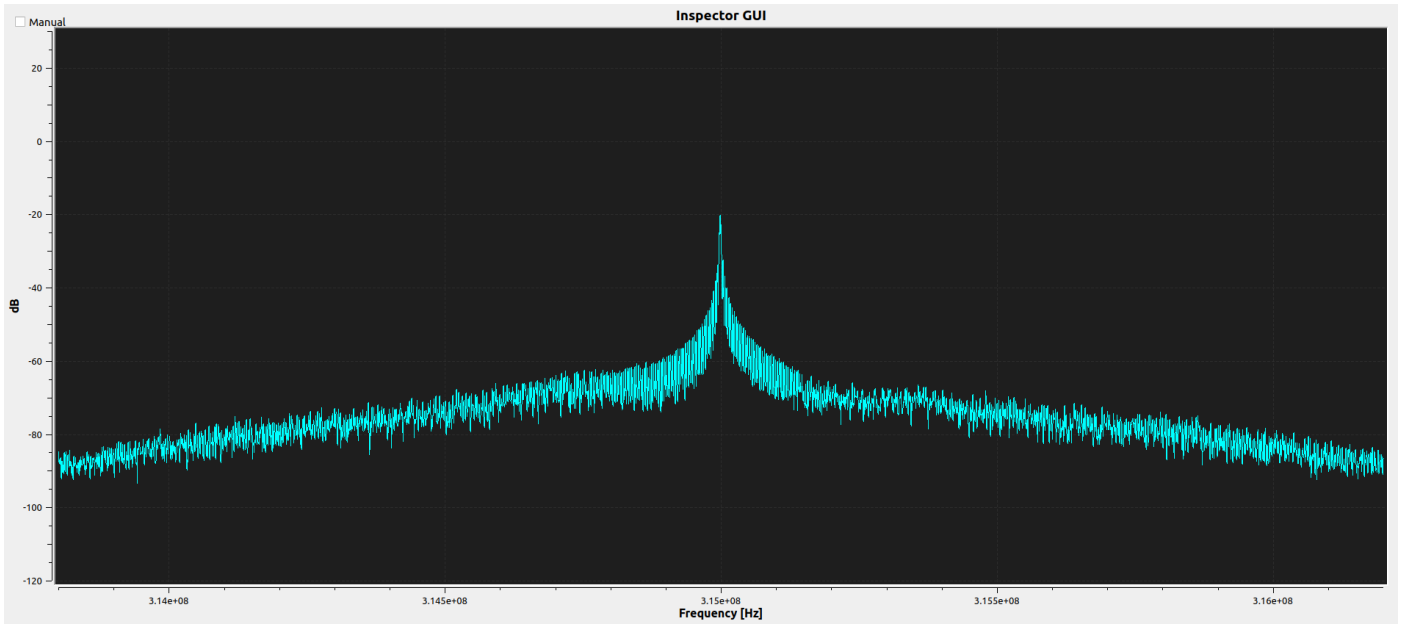

Figure B14— 2013 Volkswagen CC key fob signal spectrum plot

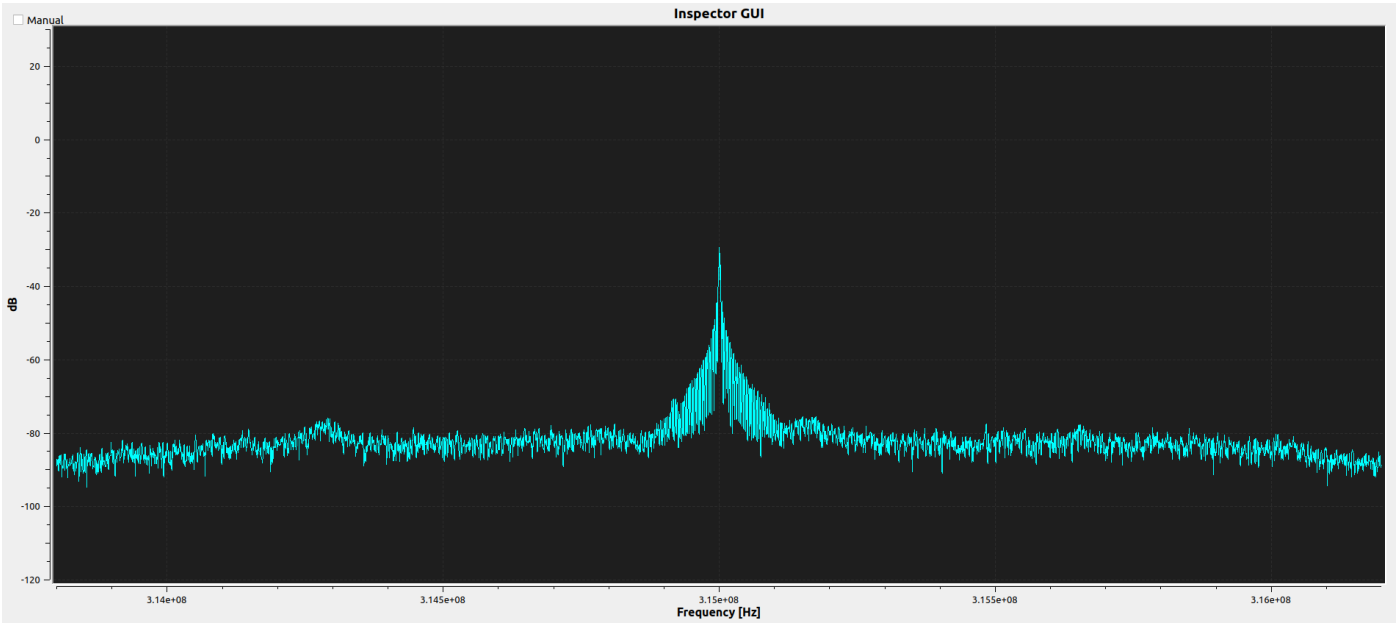

Figure B15— 2012 Volkswagen Golf key fob signal spectrum plot

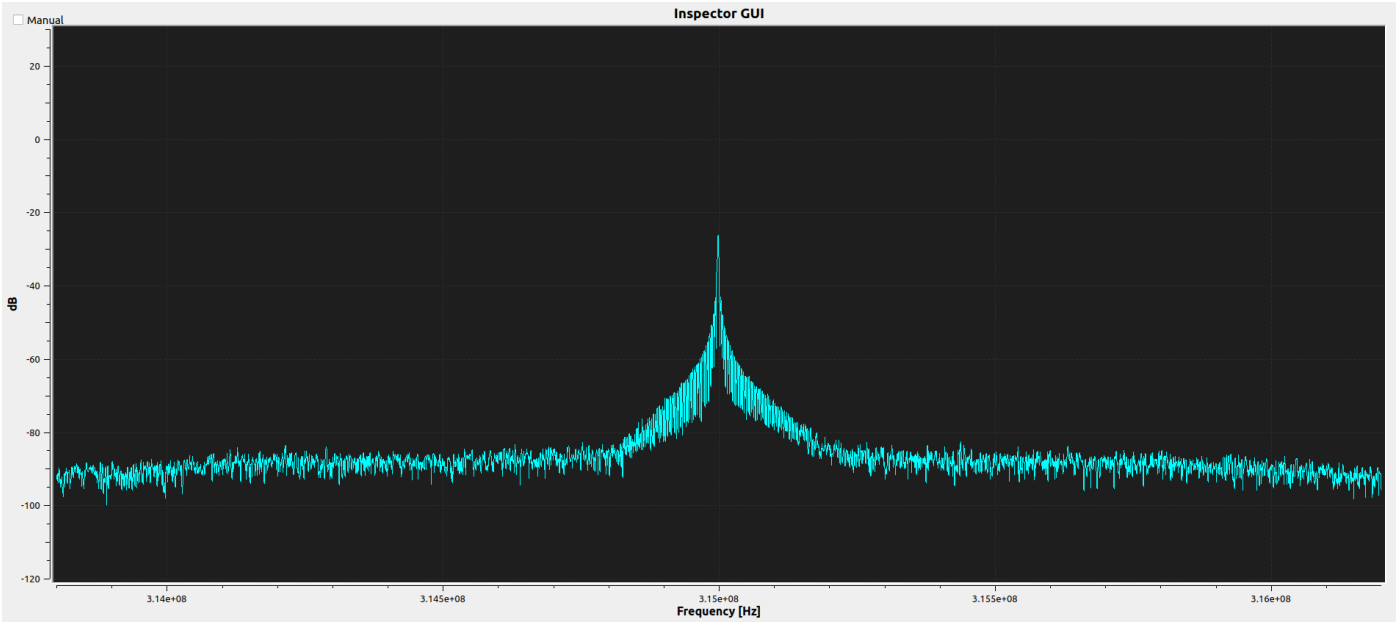

Figure B16— 2021 Volkswagen Jetta key fob signal spectrum plot