Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO DE FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN TECNOLOXÍAS DA INFORMACIÓN

Euro-Inf
Bachelor
awarded by
EQANIE

Security analysis of remote keyless entry systems for vehicles

Estudante: Alessandro Aldrey Urresti

Dirección: Tiago Manuel Fernández Caramés

A Coruña, Septiembre de 2022.

For my loved ones.

Acknowledgements

First of all, I would like to thank my family and my partner, who have always supported me despite adversity and have helped me move forward even if it was complicated and I thought it was not possible to continue.

I would also like to thank the help and dedication of my tutor Tiago, since without him it would not have been possible to carried out this project, from the proposals of different projects to the solution of various doubts.

Abstract

Security in the digital world is a fundamental aspect of our day life but not enough attention is paid to hardware security. Normally, its operation is blindly trusted and it is taken for granted that it is safe, whether it is the key to a garage portal or a key fob of a car. In this project, the level of security of different vehicles as well as a garage will be evaluated and it will be demonstrated how it is possible to open them without cloning the key or physically affecting them. In addition, several countermeasures will be proposed that could stop these attacks and make these environments safer. The remote keyless systems, the amplitude modulation and how it has been possible to open the different vehicles without needing any physical means, such as a remote control, will be studied.

Resumo

La seguridad en el mundo digital es un aspecto fundamental de nuestro día a día, sin embargo, no se presta suficiente atención a la seguridad hardware. Normalmente, se confía ciegamente en su funcionamiento y se da por sentado que es seguro, ya sea la llave de un portal de garaje o la de un coche. En este proyecto, se demostrará el nivel de seguridad de diferentes vehículos así como de un garaje y como es posible abrirlos sin necesidad de clonar la llave o incidiendo físicamente en ellos. Además, se propondrán varias contra medidas que podrían parar esos ataques y hacer estos entornos más seguros. Se estudiarán los sistemas de llave remota, la modulación por amplitud y como ha sido posible abrir los diferentes vehículos sin necesidad de tener ningún medio físico, como un mando a distancia.

Keywords:

- Rolljam attack
- Replay attack
- Brute force
- Car keys
- Keyless entry systems
- Rolling code
- Vehicle security
- RFID

Palabras clave:

- Ataque rolljam
- Ataque replay
- Fuerza bruta
- Llaves de coches
- Llave remota
- Código rotativo
- Seguridad del automóvil
- Identificación por radiofrecuencia

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Currently, technologies of all types such as Artificial Intelligence or data processing are quickly evolving, but not enough attention is being payed to the security factor. Users trust that the products and services they use, comply with its functioning without posing any problem for them. Nevertheless, that is not quite true, as recent scientific papers about security vulnerabilities are not rarely found exposing some vulnerability of some technology which has been widely used and nobody has published something about it until now.

For this reason, it has been implemented a tool capable of reproducing the sent signal, which may be from a garage door or a car remote access key, to open them without the need of stealing them or physically attack the receiver, therefore showing a vulnerability that has been present for years in which a correct and secure solution has not been implemented in many devices.

1.1 Motivation

During my university years I have had to learn about different technologies and in the course of these four years many others have been developed and implemented. Due to its fast and constant evolution it is not possible to bring them to the market with security in mind.

On the other hand, the most commonly used paradigm at the moment, agile methodologies, is not enough conscious of the security factor. The key point is developing a software that works as expected and updating it in the future if necessary. Due to this paradigm, the software ends up in oblivion or being replaced by other projects and leaving it without updating forever. Even worse is the case of hardware devices, which on many occasions cannot be updated unless the device is replaced, which is the case, for example, with car or garage remote access keys.

The radio-frequency field and its study is commonly conceived as an intimidating and very technical one. Nevertheless, this project will serve as a guide and tool to capture and send signals for a specific frequency. In order to carry it out, I will be relying on concepts that I have been taught at university, as well as knowledge from trial and error all along the phases of this project.

To carry out this task and taking time into account, AM radio frequencies are the only subject of study of the project, along with ASK/OOK modulation. These two aspects are the most commonly used in the industry in order to implement a Radio-frequency identification (RFID) lock. The same situation occurs with the remote keyless systems (RKS) used in vehicles. These systems usually operate by sending three different types of codes. In this project, rolling codes will be studied in greater detail and how they can be equally vulnerable through the rolljam attack.

1.1.1 Objectives

The main objective of this dissertation is to analyze the cybersecurity of vehicle opening systems without the necessity of having physical access to its transmitter, nor to its receiver. The implemented tool could be used as a foundation for more sophisticated future work, to simply check the safety of the vehicle towards this type of attacks or, if necessary, to be used in a red teaming exercise.

Moreover, as an additional objective, it will make it possible to raise awareness among the public that shows interest in the area, about the risks that exist in the world of radio-frequency, if it is not implemented correctly or with the necessary mechanisms for safe operation.

1.1.2 Senior thesis structure

The structure of this dissertation is as follows:

1. Introduction: explanation of the motivation and objectives to be met.
2. State of the art:the operation and evolution of the RKS is described, as well as the different attacks that have been implemented throughout its history.
3. Technology foundations: the most common signal modulation and coding systems are explained.
4. Resources: hardware and software resources used for the development of this project are detailed.
5. Planning and cost estimation: the cost of the necessary hardware and human resources for the development of this project is detailed.

6. Development: the methodology used to capture and analyze the signals is described. In addition, it is explained in depth the most relevant parts of the code.

7. Conclusions: it is concluded if the the objectives of the project have been achieved, the knowledge acquired during the development of it and the result on the different technologies studied.

8. Incidents and future improvements: the incidents faced throughout the process are detailed, as well as different improvements for the future are proposed.

9. Additional material: the link to the repository where the source code and evidence of this project are shown.

Chapter 2

State of the art

This chapter details the evolution of keyless entry systems and the techniques used to try to attack them. Due to the presence of these two main topics, it will be divided into their respective sections, focusing on the study of the keys of the vehicles used as well as the most modern systems, making a comparison between the technique used in this senior thesis, the rolljam attack, and the most avant-garde methods for the most modern systems.

2.1 Entry systems

Vehicles in the early 1900s used a mechanical lock, which later would also serve to shut off the engine ignition.

In the years thereafter, in 1980, the remote keyless entry (RKE) system was introduced by the Ford company. This mechanism was based on a keypad on the driver's door that allowed access to the car once a code was entered correctly. Two years later, Renault released the Renault Fuego, which implemented the first RKE using a remote control.

From there on, these systems have evolved, from the most basic RKE systems, through passive entry systems (PES), to finally reach the passive access and secure entry (PASE) system, using different types of coding and modulation, until ending with the implementation of encryption, such as the use of Advanced Encryption Standard (AES).

2.1.1 Remote keyless entry

RKE refers to any electronic device that can be operated through the use of a remote control key, normally having an effect at a distance of less than 50 meters from the receiver. Its first implementation was carried out with the aim of giving greater comfort to the user, not taking security into consideration. The first approach in this area was to implement an anti-theft measure; this was when ignition immobilizers were invented.

Since their appearance, RKEs are required by insurance companies in Europe. These work by approaching the key, which presents a token and allows for starting the engine. In some modern vehicles, the absence of this token or an incorrect sending can activate different security mechanisms such as closing the vehicle or notifying a previously configured security company.

2.1.2 Security of RKE systems

RKE sending systems

There are different types of techniques when it comes to authenticating that the key being used is the correct one for that vehicle [11] [12]. Some of those techniques are the following:

- **Fixed code**. This technique is based on the fact that the remote key has a code embedded within the key chip itself. Once the key is activated, the code is sent and if, the receiving device is the intended one, it will check if the sent code is the same that the receiver also has integrated. If the verification is successful, the system will carry out the operation programmed for that code.

- **Rolling code**. Rolling codes are widely used in remote keyless systems. They consist in sending a different code every time the remote control is activated. In this technique there are different types of variations. The first would be to start from the same seed in both RKS and carry out a certain operation on the value of the counter at that moment, for example by multiplying by a prime number. Subsequently, this value is the one that will be used as the counter value the next time the knob is pressed. The second variation consists in an internal counter that has a sequence of codes: each time the remote access is activated, the counter is incremented and that code is used. The receiver also has a counter programmed equally as the sender. For both cases, the receiver has a range of valid codes to be accepted when received. This is done in case the remote entry key has been pressed without the receiver receiving it and thus to avoid being desynchronized.

  In modern devices, the code is usually previously encrypted before its transmission. When the receiver receives it, it decrypts the received code and compares it with its value. In addition, there are other RKS that once they receive a code, they invalidate all those below that code in the internal counter sequence. This prevents that if a signal has been captured without being heard by the receiver, it can be used later in the future.

- **Challenge–Response Technique**. In this technique the sending device and the receiver have a shared encryption key. When the user approaches the car and tries to open the door, it sends a challenge, a random value. The remote control receives it and encrypts it with the shared key. In turn, the receiver also encrypts the value it has received. Next, the sender sends that encrypted message and the receiver compares it with the one sent: if it is correct the vehicle opens the door, otherwise it will do nothing. These systems are more common in ignition systems.

Attacks against RKE vehicles systems

This section will describe different attacks against the previously mentioned systems [11] [12]:

- **Brute-force attack**. This attack is only viable for the case that the code is fixed and assuming that the length of the code does not have a high value, since testing each combination would take longer than thousands of years. In this project the different keys were analyzed and even without taking into account that they were rolling codes. For example, the later analyzed Volkswagen Passat key car had 80 bits and, although 11 bits were repeated, they are still 69 bits so there would be $2^{69} - 1$ possibilities.

  A variation of this attack would be the dictionary attack, in which an attacker would be capturing signals over a long period of time, either by capturing rolling codes or challenges, trying to open the door. Once a large number of codes has been obtained, the attacker would try to send them to the vehicle: if one of them is repeated, the car will be opened. This is possible due to the impossibility of the infinite generation of codes by the sender or the receiver. However, it is still a number almost as high as in the case of the brute force attack.

- **Replay attack**.The replay attack consists in an attacker getting the recording of a signal emitted by the remote control and repeating it later to be able to open the vehicle, as indicated in Figure 2.1. This case would also serve mainly against fixed codes, since rolling codes and challenge-response schemes were created to avoid them, as the message varies each time. However, it is worth mentioning the case in which the rolling codes do not invalidate the rest of the sequence since one is received correctly: it may happen that it can be recorded as long as it is not received by the vehicle and be able to use it later, even though that chance is low. This last hypothesis is in which the rolljam attack is based on, which is the foundation on this dissertation and which mainly uses this principle, making use of jamming.

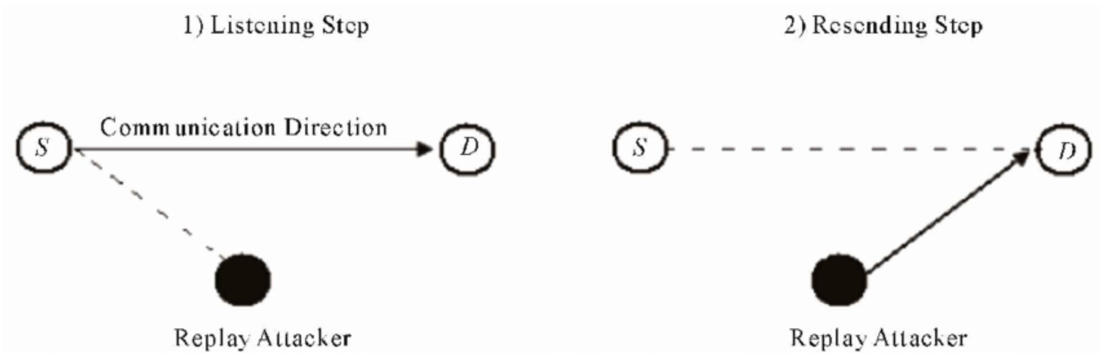


Figure 2.1: Replay attack diagram [1].

- **Two-Thief Attack**. This attack involves the presence of the vehicle, the owner of the key and two attackers who must have a specific and preconfigured equipment according to the characteristics of the vehicle they are going to test. A first attacker should be close to the user with the key, while the second needs to be close to the car, for this attack to work, the user must be enough far away from the car, so that it cannot receive the signals from the remote control.

  The second attacker tries to open the car door, which causes the car to send a challenge. This attacker captures it, amplifies it and sends it to the first attacker.

  The first attacker, once the challenge is received, similarly, forwards it to the user who owns the key and once issued, listens to receive the message sent by the user's car key. Once received, it is sent to the second attacker who replicates the challenge and opens the car, as illustrated in Figure 2.2.

  The mitigation of this attack is based mainly on the distance the user is from the vehicle, being able to be close enough so that the car receives the signals and the command as well, or being at a great distance so that the attackers are not capable of communicating with each other as far as Radio Frequency (RF) is concerned.

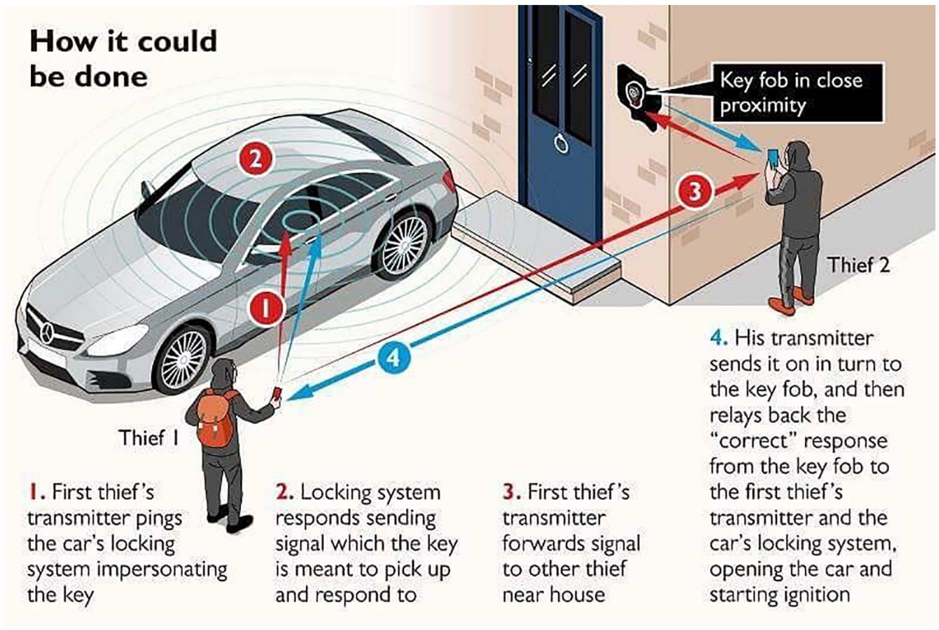

How it could be done
Key fob in close proximity
Thief 2
Thief 1
1. First thief's transmitter pings the car's locking system impersonating the key
2. Locking system responds sending signal which the key is meant to pick up and respond to
3. First thief's transmitter forwards signal to other thief near house
4. His transmitter sends it on in turn to the key fob, and then relays back the "correct" response from the key fob to the first thief's transmitter and the car's locking system, opening the car and starting ignition

Figure 2.2: Two-thief attack diagram [2].

- **Rolljam attack**. This attack is what this senior thesis is mainly based on. It mainly consists in a combination of jamming and replay attack. An attacker will need to have a device near the vehicle. Once it detects the preamble of the signal, it begins to jam with enough power so that the receiver does not receive the remote control signal, so the car door will not open. Meanwhile the attacker is listening and will capture the signal to play it back later, as shown in Figure 2.3. A better version of this attack would be when the user reissues a signal: once he has seen that he has not unlocked the car, the attacker captures this new signal and sends the one he had previously captured to unlock the car. In this way, the attacker keeps the last available code. This improvement avoids that if the receiver invalidates the previous codes of the sequence, the attacker is left without any valid code to use.

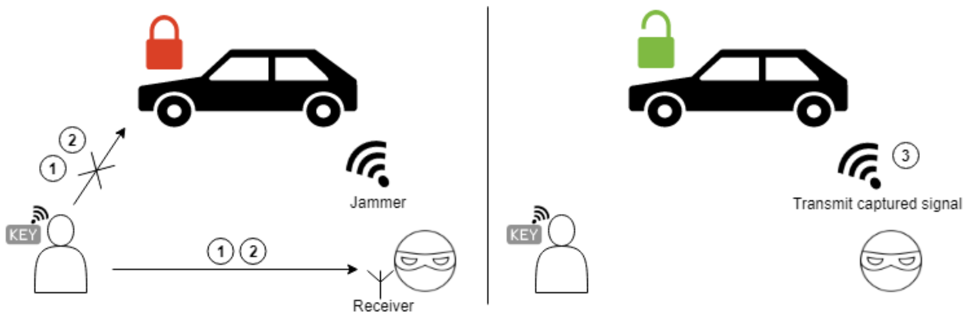


Figure 2.3: Rolljam attack diagram [3].

Chapter 3

Technology foundations

This chapter presents a theoretical basis on the previous concepts that must be known for a complete understanding of the development of the proposed system for the analysis of the security of RKE systems.

3.1 Wireless digital information transmission systems

A modulation is a technique that allows for modifying a high-frequency signal, called a modulator, according to a low-frequency signal called a carrier, with the aim of transmitting data. This is possible because both signals are passed to a modulator device that combines both. In the following subsections some of the more common modulation schemes will be described.

3.1.1 AM modulation

This type of modulation [13] consists in the variation of the amplitude of the transmitted signal, which is composed by the multiplication of two temporal signals. In order to understand this multiplication, it is necessary to know how the Fourier transform works, which performs the translation from the time domain to the frequency domain. A simplification would be that a parameter of a carrier wave changes its value with respect to a variation of the modulating signal, the latter being the message to be transmitted.

On the other hand, this type of modulation allows us to communicate over long distances with cheap technology. However, amplitude modulation is susceptible to atmospheric phenomena or noise in general, so it is not used when a high accuracy is required.

3.1.2 FM modulation

This type of modulation [13] is used when a higher reliability of the information is required than in the case of amplitude modulation. It consists in shifting the frequency of the carrier wave.

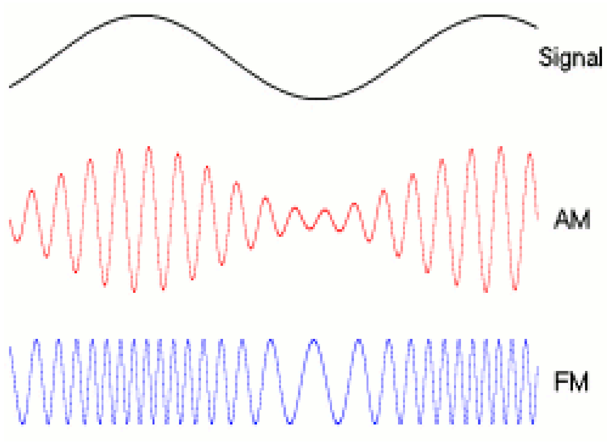


Figure 3.1: Comparison between AM and FM modulations [4].

3.1.3 Amplitude displacement modulation

ASK modulation is mainly used when the signal only has two possible states, on-off. It is capable of representing digital data as amplitude variations of the carrier wave. Depending on the value of the amplitude, the value 0 or 1 is represented, thus functioning as if it were a switch. It can also be called ASK/OOK, Amplitude Shift Keying On-Off Key.

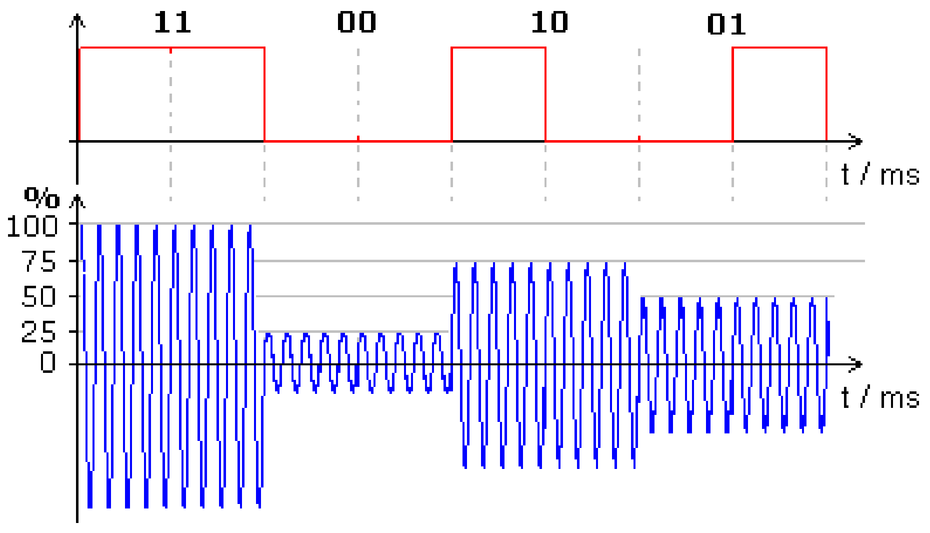


Figure 3.2: ASK modulation [5].

3.2 Transmission coding

In this section only the encoding methods that have been used during the analysis of the vehicles and the garage analyzed during the project are described. All analyzed cars use Manchester coding, while the garage makes use of a coding that varies the length of the high state of the pulse.

3.2.1 Manchester coding

In this encoding system [14] each bit of information is presented in a transition. There are two possible implementations, the first one from E.G. Thomas and the second one corresponds with the IEE 802.3 standard [15]. Where the representation of each one is complementary to the other. In the first the change from low to high is represented with the value 0 and from high to low as 1, while in the second they are the opposite values, 1 and 0 for the respective cases.

In this work it has been assumed that the second implementation was used, despite not being relevant, since as long as the same signal is sent that has been received, the content of the signal effects of applying the rolljam attack is not important.

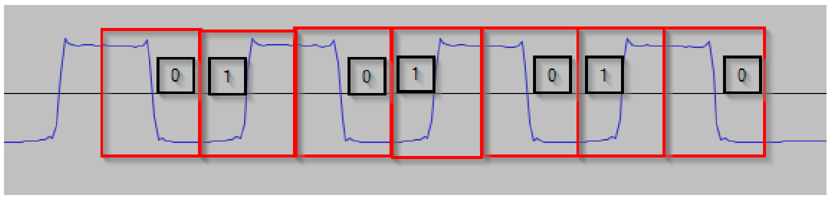


Figure 3.3: Manchester codification.

3.2.2 Return to zero

Return to zero (RTZ) encoding is very simple: each bit of information ends up returning to 0. There are several types of RTZ, among which the inverted and the bipolar stand out. In the first one, if there is a pulse, it will have a binary representation of 0, while if there is no pulse, it will be shown as a 1. In the second, on the contrary, if you want to send a 1, you send a power greater than 0, while if a sender wants to send a 0, a negative value is sent. Empty symbols are used as bit separation or can be used for padding.

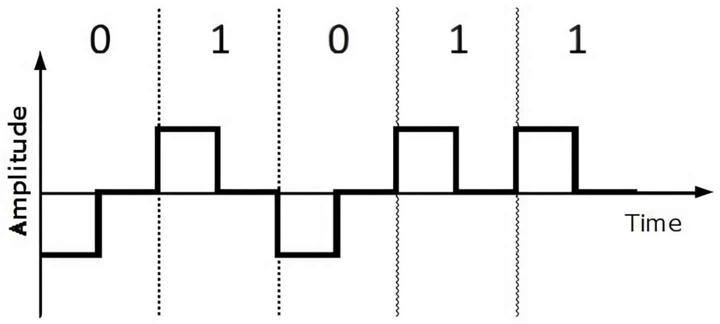


Figure 3.4: Return to zero codification [6].

3.2.3 Length variation of the high pulse state

We could consider this type of encoding as a type of encoding based on return to 0, but it adds a series of peculiarities compared to other types. It can be considered as RTZ since when finished it always returns to that value. The two possible states to differentiate between (let's call them 0 and 1), can be differentiated by how long the signal stays high and how long it stays low. The transmission of each bit has a fixed duration, always sending a low state first and then a high state. For example, if a sender wants to send a "0", it is sent keeping the low signal for a longer time and less high. For this project, the values "0" and "1" have been assigned to the two possible values of the signal message. However, it is not possible to know which value it has at each moment when using this type of encoding. Once again, it is not necessary to know the real value to carry out this work.

As it can be seen in Figure 3.5, for case 1, first it spends more time in a low state and then a short time in a high state, thus representing what has been considered as a 0. For case number 4, it can be observed as it starts with a shorter time in the low state to, subsequently, remain a longer time in the high state, thus representing a 1.

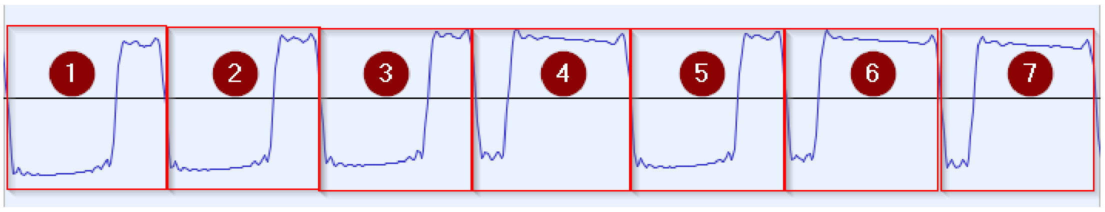


Figure 3.5: Length variation of the high pulse state.

Chapter 4
Resources

In this section it will be generically described the hardware and software used in this project.

4.1 Hardware

To achieve the objectives set for this thesis, it has been necessary to make us of different hardware gadgets, a device to run the developed software, devices capable of sending and receiving signals at different frequencies and modulations, and a Raspberry Pi in order to carry out the rolljam attack.

4.1.1 Development equipment

Firstly, the development of the different scripts has been carried out in my personal computer with the following specifications:

- Model: Lenovo Ideapad 320-15AST
- Processor: AMD A9-9420 Stoney Ridge, 3 Ghz
- RAM memory: 8GB
- Disk: 256GB
- Operating System: Kali GNU/Linux 2022.3

4.1.2 Nooelec NESDR SMArTee v2

Nooelec NESDR [7] is a dongle which can be used as a radio scanner which can receive real time radio signals. It makes use of a RTL2832U chipset as a demodulator and a R820T2 IC tuner. This device is able to capture RF signals among 25MHz-1700MHz, using an ultra-low noise phase Temperature-Compensated Crystal Oscillator with a tolerance of <0.5PPM.

This device has been used during the first phase of the frequency and remote control key signal capture investigation. It allows for filtering the level of noise that is received through the squelch, and record the signal in a file, with a sampling rate of 48000 Hz.

Figure 4.1: Nooelec NESDR SMArTee v2 [7].

4.1.3 YARD Stick One

YARD Stick One [8] is an open-source Half-Duplex Wireless Transceiver that has been created by Mike Ossmann. It has an Subminiature version A (SMA) connector for external antennas that operates at sub-1 GHz and a low-pass filter to eliminate harmonics. This filter has not been necessary for the work developed in this project, since it is for frequencies between 800 and 900 MHz, which are not common in car key fobs.

This dongle comes preconfigured with the RFCat firmware, thanks to which it can be controlled from the computer using a python shell or its corresponding library, rflib.

It allows different frequency ranges:

- 300-348 MHz
- 391-464 MHz
- 782-928 MHz

It supports different types of modulation:

- 2FSK
- GFSK
- ASK/OOK, the only one which was used in the analyzed remote control keys

- MSK

This dongle enables to receive the signal at a specific frequency and transmit the encoded message in bytes. Moreover, its library allows for configuring different parameters, such as bandwidth or emission power which are essential to implement jamming in order to inhibit the vehicle receptor.

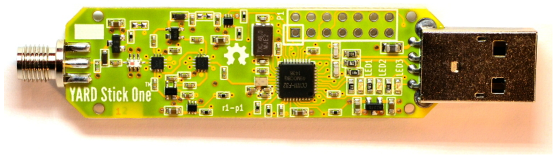


Figure 4.2: YARD Stick One [8].

4.1.4 Raspberry Pi

Raspberry Pi [9] is a single-board computer that has been used as a jamming device thanks to its small size once it detects a remote control key signal. This kind of technology has been chosen as it can be used for this type of attacks in a real scenario due to its reduced dimensions. Moreover, it is not an expensive device, having a price that suits every pocket.

The used model has the following specifications:

- Model: 4 B
- Processor: Arm 8 cores a 1.5 GHz
- RAM memory: 4GB
- Disk: 64GB
- Operating System: Kali GNU/Linux 2022.3-raspberry-pi-armhf

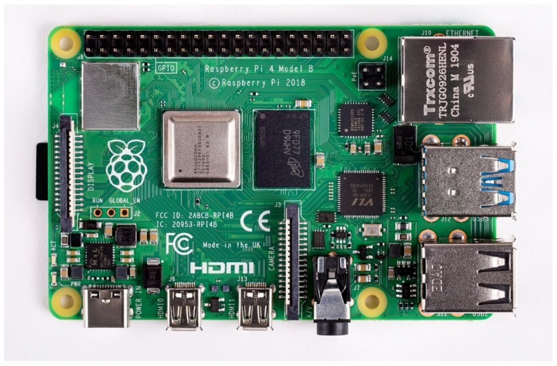


Figure 4.3: Raspberry Pi 4B [9].

4.1.5 Aprimatic key fob

- Receiver type: Garage
- Brand: Aprimatic
- Model: TX2M
- Generation: 2014
- Frequency: 433952000 HZ
- Modulation: High state pulse length variation, Return to zero (RTZ) type

Figure 4.4: Aprimatic TX2M key fob.

4.1.6 Volkswagen key fob

- Receiver type: Car
- Brand: Volkswagen
- Model: Passat
- Generation: 2002
- Frequency: 434412100 Hz
- Modulation: ASK/OOK

Figure 4.5: Volkswagen Passat key fob.

4.1.7 Audi key fob

- Receiver type: Car
- Brand: Audi
- Model: Q2
- Generation: 2021
- Frequency: 434421100 Hz
- Modulation: ASK/OOK

Figure 4.6: Audi Q2 key fob.

4.1.8 Mercedes key fob

- Receiver type: Car
- Brand: Mercedes

- Model: A Class
- Generation: 2006
- Frequency: 433945600
- Modulation: ASK/OOK

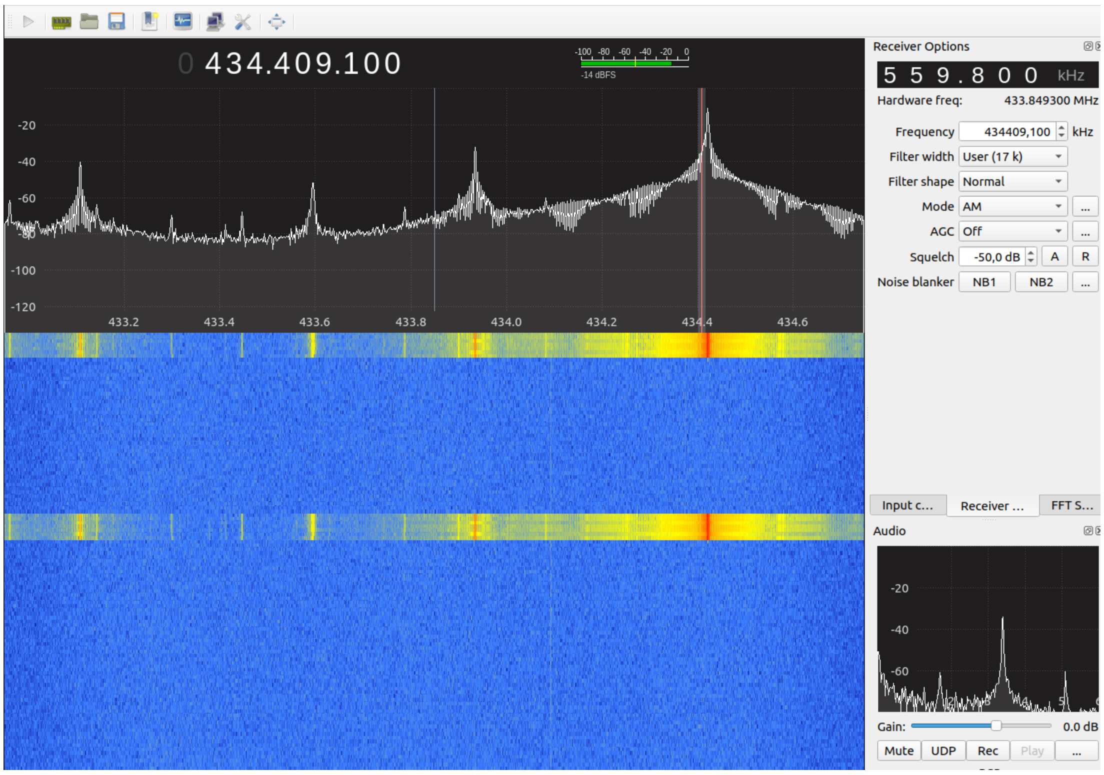
Figure 4.7: Mercedes A Class key fob.

4.2 Software

4.2.1 Gqrx

Gqrx is an open-source Software Defined Radio (SDR) software [16]. The functionalities that have been used during this project were:

- AM demodulation
- Change frequency, gain and apply various corrections
- Automatic Gain Control (AGC), squelch and noise blankers
- FFT plot and waterfall
- Record audio to a WAV file



Receiver Options

5 5 9 . 8 0 0 kHz

Hardware freq: 433.849300 MHz

Frequency 434409,100 kHz

Filter width User (17 k)

Filter shape Normal

Mode AM

AGC Off

Squelch -50,0 dB

Noise blanker NB1 NB2

Gain: 0.0 dB

Figure 4.8: Gqrx capturing signal.

4.2.2 Audacity

Audacity is an open source audio processing tool. Its was mainly used for WAV file visualization, which had been previously recorded with gqrx tool. This software allows for obtaining a very detailed granularity and also allows for measuring, in number of samples, a selected fragment.

On the other hand, it is possible to divide a stereo track into mono channels and compare different previously recorded tracks at the same time, which has been of special relevance since it has allowed us to observe the differences between the original key and the emission of each one of the scripts. This has also been of great help at the moment of sending, in order to correct the code as any differences in terms of the duration of the preamble or the size of the bits could be seen.

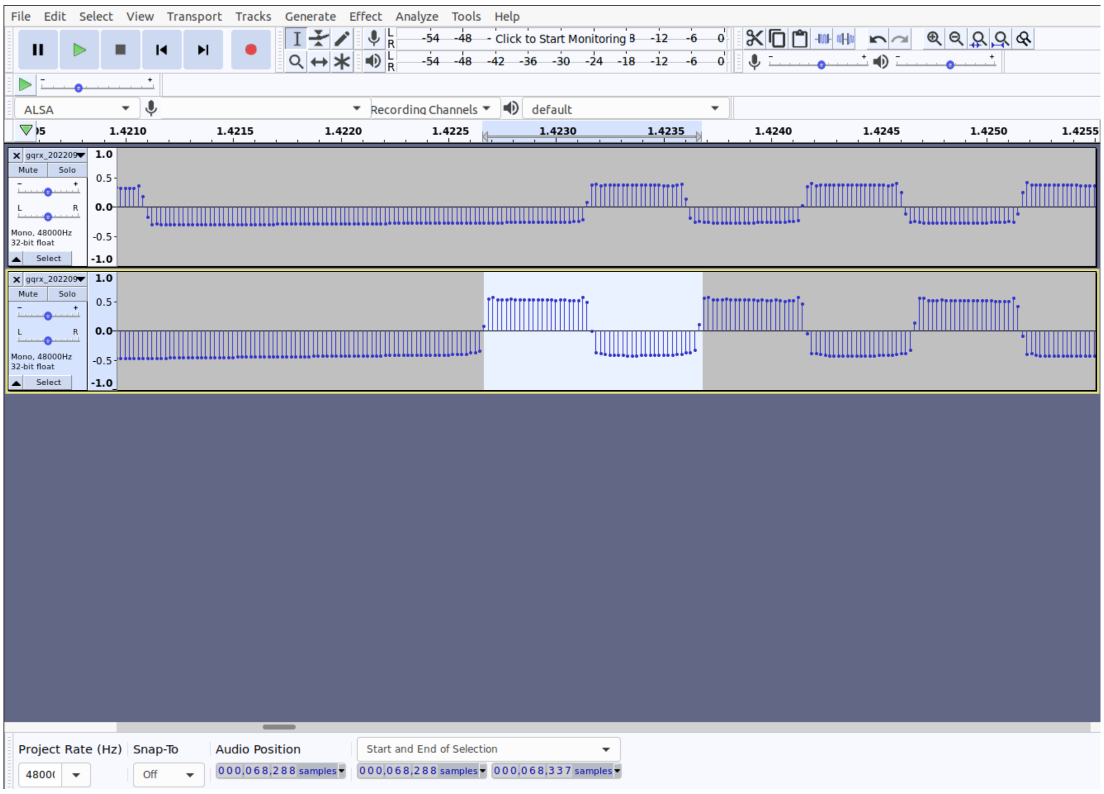


Figure 4.9: Audacity analyzing Passat's signal.

4.2.3 Programming language

4.2.4 Python 3.9.13

The programming language chosen to carry out this project has been Python [17] due to the pre-installed firmware in the YARD Stick One. In addition, this language is easy to learn and is versatile in its implementation. On the other hand, the student had previous experience with it.

4.2.5 Python libraries

JSON

This library allows for writing and reading JSON files. It has been used to store the captured signal once was transformed into a specific format, a format that needs the signal's transmission functions.

Binascii

Binascii converts from binary to ASCII or vice versa, as necessary. In the project, its hexlify() and decode() functions were used to decode the received signal and transform that information into hexadecimal.

Exists

This library determines if a file exists. It has been used to determine the existence of the JSON file before creating a new one in the project.

Bitstring

This library has been used to convert text strings to bytes so that they can be sent.

Time

This library allows for obtaining the current time of the system. It was used when creating the name of the JSON file, as well as to being able to add delays in three functions: jam_with_delay, echo y echo_with_delay.

Rflib

It is the main library of this project. It is in charge the communication with the YARD Stick One. The most relevant functions of this library that have been used are:

- setMdmModulation(), fixed the type of modulation to be used.
- setFreq(), fixes the base frequency.
- setMdmDRate(), fixes the data rate.
- discover(), marks in listening mode the device and allows for seeing the information captured by the dongle in real time.
- RFrecv(), this function is similar to the previous one with the difference that it listens simultaneously.
- RFxmit(), it handles the transmission of the payload.

Chapter 5

Planning and cost estimation

5.1 Project planning

The following figure shows the project planning using a Gantt chart, thus showing the start and end date of each of the phases.

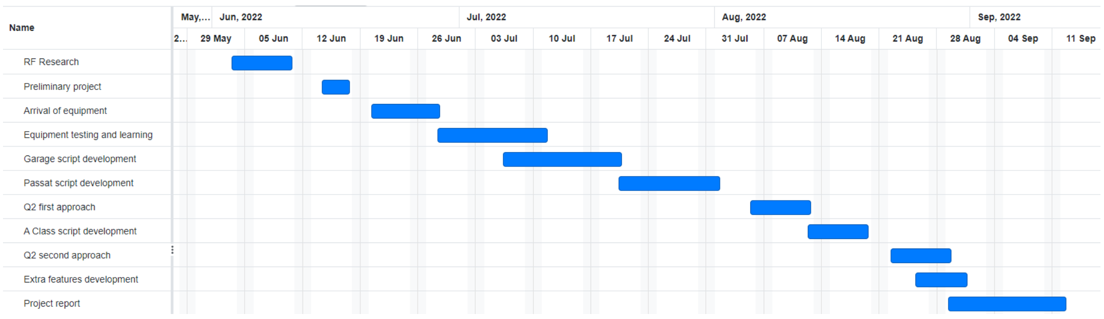


Figure 5.1: Gantt's project diagram.

A brief description of the phases is found below:

- **RF Research**: initial phase of the project, which consists in searching for information on issues related to radio frequency, focusing on the security aspect.
- **Preliminary project**: The viability and approximation of the project were discussed with my tutor. Once the approach was defined, the preliminary project was carried out and delivered.
- **Arrival of equipment**: The necessary equipment, mentioned in the resources section, was purchased and its arrival was awaited.
- **Equipment testing**: different tests were performed and got acquainted with the different devices. Once its functioning was understood, it was configured with the needs of the project.

- **Garage script development**: this phase was the longest of the project. It was decided that the garage would be the first implementation since it exposes the less secure technology and easier operation.
- **Passat script development**: this phase was also especially long since, although the garage code was available, it was not fully compatible due to the type of coding used by the garage. This caused having to completely develop certain part of the code and make it simply adaptable for the rest of the vehicles.
- **Q2 first approach**: a first implementation for the Q2 script was attempted without success in its operation.
- **A Class script development**: the script for the Mercedes was implemented based on the Passat's code.
- **Q2 second approach**: an attempt to implement a useful script for the Q2 car was made without success.
- **Extra features**: different modes were added for each vehicle, in order to have more automated tests.
- **Project report**: the final report of the project was written.

5.2 Cost estimation

5.2.1 Human resources

The main cost of this project has been the remuneration for the hours spent in it, which were approximately 6 hours a day for 3 months. The total number of hours, as they were taken into account during the course of the project, consisted of a total of 540 hours. In addition, we must take into account the hours of the project manager, which we estimated to be about 20 hours. For the cost in human resources, following this guide [18], they would be: 22,92 euros/hour for the student [19] plus 45 euros/hour for the project director. This results in a total cost in terms of human resources of $540 * 22,92 + 20 * 45 = 13175(€)$.

5.2.2 Hardware resources

In this project it was also necessary to use special equipment, which is detailed along with its value in the following table 5.1:

Resource	**Units**	**Cost(€)**	**Total cost(€)**
YARD Stick One	2	149,24	298,48
Nooelec NESDR	1	57	57,00
Raspberry Pi	1	124,87	124,87
		Total	480,36€

Table 5.1: Total cost of hardware resources.

Chapter 6

Methodology and development

6.1 Methodology

An overview of the methodology that has been followed in this senior thesis, can be shown in flow diagram below 6.1 and is thoroughly described in the following sections:

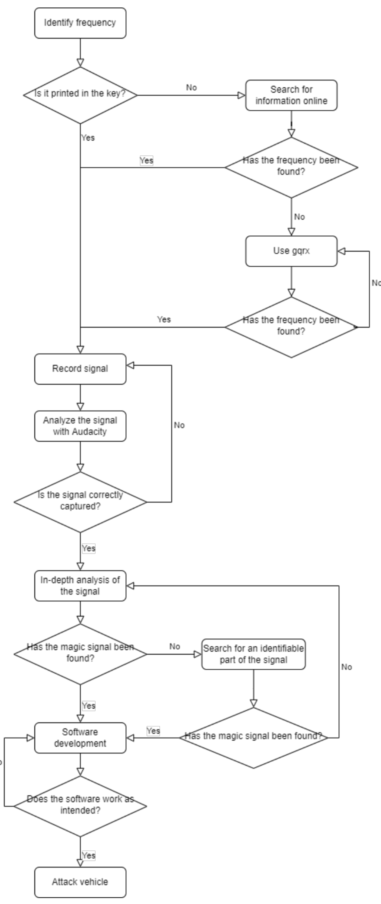

Identify frequency
Is it printed in the key?
No
Search for information online
Yes
Yes
Has the frequency been found?
No
Use gqrx
No
Yes
Has the frequency been found?
Record signal
Analyze the signal with Audacity
No
Is the signal correctly captured?
Yes
In-depth analysis of the signal
Has the magic signal been found?
No
Search for an identifiable part of the signal
No
Yes
Software development
Yes
Has the magic signal been found?
No
Does the software work as intended?
Yes
Attack vehicle

Figure 6.1: Methodology flow diagram.

6.1.1 Previous investigation

The first step of the methodology was to try to identify all the possible information related to the remote control. The simplest thing is to see if the key itself has the frequency at which it operates in a visually accessible way. For example, if it had its identifier printed on the outside of it, as illustrated in Figure 6.2, sometimes you have to open the remote control. If an identifier is obtained, official pages such as the Federal Communications Commission [10] can be used, as shown in Figures 6.3 and 6.4. However, you could also search online for the manual of the particular key fob or car being examined, otherwise you would have to see if the details of the online stores specify it among its characteristics. If there is no information available on the internet, you can look in the offline manual itself in the remote control section if available.

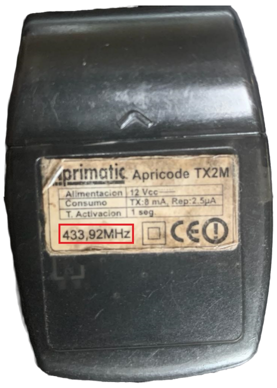


Figure 6.2: Aprimatic frequency printed in the remote control.

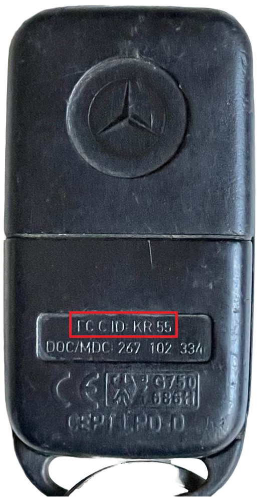
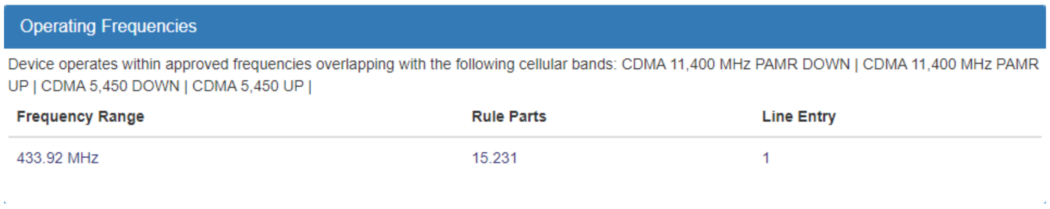


Figure 6.3: FCC Mercedes's remote control ID.


Operating Frequencies

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

Frequency Range	Rule Parts	Line Entry
433.92 MHz	15.231	1

Figure 6.4: FCC Mercedes's search result [10].

On the other hand, if it has not been possible to find any type of identifier or frequency printed on the remote control, the gqrx tool will be used through an rtl-sdr. Firstly, the signal will have to be focused on the usual frequencies, which are usually between 433-434 MHz in Europe, although there may be variations depending on the device to be analyzed. Once the frequency is identified, the signal should be centered where the frequency shows the greatest power in the waterfall panel of the application.

This last step of the methodology was the one used in the case of the Passat, as shown in Figure 6.5, since neither online information was available nor which frequency it operates was printed on the remote control itself.

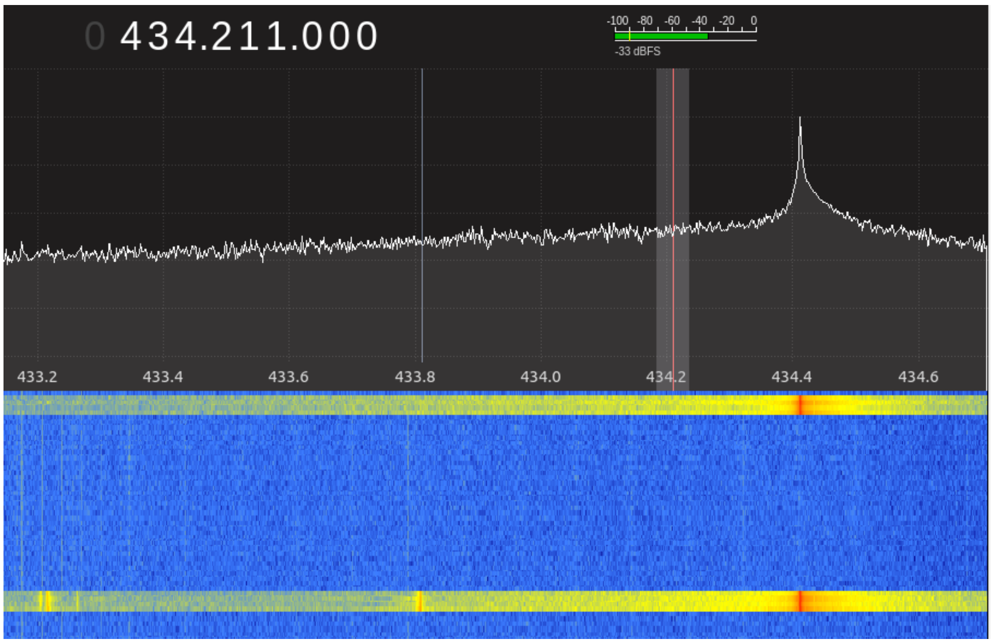
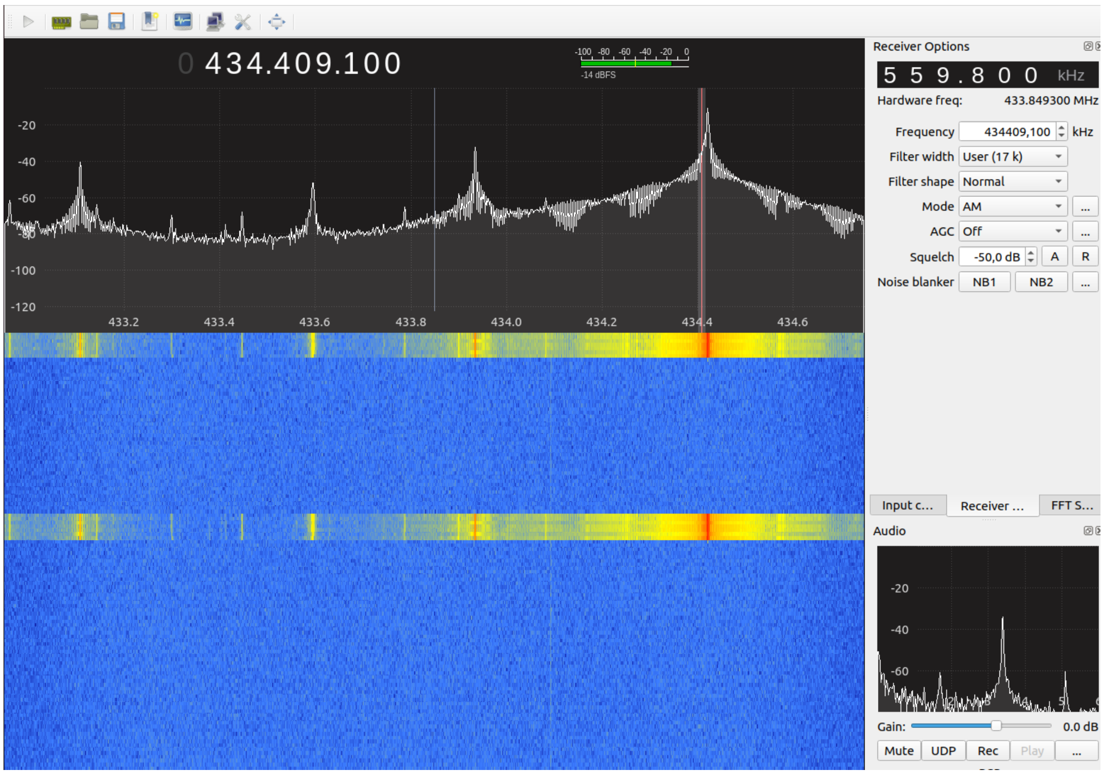


Figure 6.5: Passat obtained frequency through gqrx.

Finally, it would be possible to open the key fob and interact with its modulation chip to see what its behavior is. However, this is risky, since the remote can be easily damaged. This test has not been carried out during the execution of this project.

6.1.2 Signal capture with gqrx

The gqrx capture tool has allowed, among other things, visualizing the signal spectrum, as in its waterfall format. The display is presented on the x-axis as frequency and on the y-axis as time. In the cascading panel, the blue color shows noise while the yellow, orange and red colors represents the signal. The determination of the base frequency can be achieved by setting the center in the last mentioned color. In order to receive the signal correctly, the base frequency, the filter width, the modulation and the squelch had to be configured to eliminate the maximum amount of noise possible. Finally, the hardware autogain system of the device was removed and the Low-noise amplifier (LNA) was manually lowered to a very low value. The latter was done because at the time of the investigation the controls were triggered from a place pretty close to the receiving antenna and therefore, it was not necessary to amplify the signal. It should be noted that the FFT rate has been modified to 60 fps in order to have a better appreciation of the signal details in the evidences and the recording of the signal was made at 48000Hz, as it can be shown in Figure 6.6.


0 434.409.100

Receiver Options

5 5 9 . 8 0 0 kHz

Hardware freq: 433.849300 MHz

Frequency 434409,100 kHz

Filter width User (17 k)

Filter shape Normal

Mode AM

AGC Off

Squelch -50,0 dB

Noise blanker NB1 NB2

Audio

Gain: 0.0 dB

Figure 6.6: Captured Audi's signal using gqrx.

6.1.3 Signal analysis with audacity

As an audio analysis tool, as mentioned above, audacity has been chosen. The first step with it, once the wav file that we want to observe is open, is to see if the signal makes any kind of sense, that is, it can be Amplitude Modulation, Frequency Modulation, Phase Modulation, etc. Otherwise, we have to go to gqrx and adjust to a type that could be the correct one. In this project it was not necessary since all the analyzed devices used AM as their modulation.

The next step would be to observe the behavior of the signal, that is, it will first start with an activation preamble for the receiver. This is used to notify the receiver that a message is going to be transmitted. For these types of remote controls, for the analyzed remote control, the typical message would indicate the opening or closing, either of a garage or a car door. Next, they may have what we could call a "magic" sequence. This has been named in that way because it is the part immediately after the preamble and right before the message to warn that the message is going to be transmitted next. In addition, as will be seen later, this can be used as an anti-copy control measure. After the message, it will be possible to see in a general way if a preamble is sent between each message, in case there is more than one message.

Finally, two aspects of special relevance remain to be observed. The first one would be to see if each part of the message, without considering any of the preambles, use a similar length. It may not be equal due to problems related to the capture tool as explained in the incident section 7.2. The second aspect would be to see if between two preambles are any fixed bits. They can also be variable, assuming only two possible states depending on whether it is open or close. Exceptionally, it was found that there is also a peculiarity in terms of fixed bit frame in the case of Mercedes, as will be explained in its corresponding subsection 6.1.3.

On the other hand, with Audacity it has been possible to observe how a Manchester coding is used for the three cars and a variation in the length of the high state of the pulse for the garage. It must be taken into account that it is unknown in any case which version is being used respect to the Manchester encoding, but it is not relevant when performing a replay or rolljam attack. However, for this project it was decided to opt for the Manchester proposed by the IEEE 802.3 [15]. Using this type of Manchester coding (this would not occur using the differential Manchester), when we see continuous signals of "01" we do not know when it begins or ends, they are indistinguishable. The only way to determine it is when we observe a shift of state between "01" or "10" (when a wider pulse is encountered, oversimplifying the explanation quite a bit). The following images show how Manchester is used for the Volkswagen Passat 6.7 and how it would not fit for the garage remote control 6.8. In addition, it can be seen in the Passat's key fob how fragments 1, 2 and 3 still belong to the preamble. If we only see the first and second fragment, we would not know when it ends, since it is due to fragment 3 and 4, when a change from 1 to 0 that we know it ends.

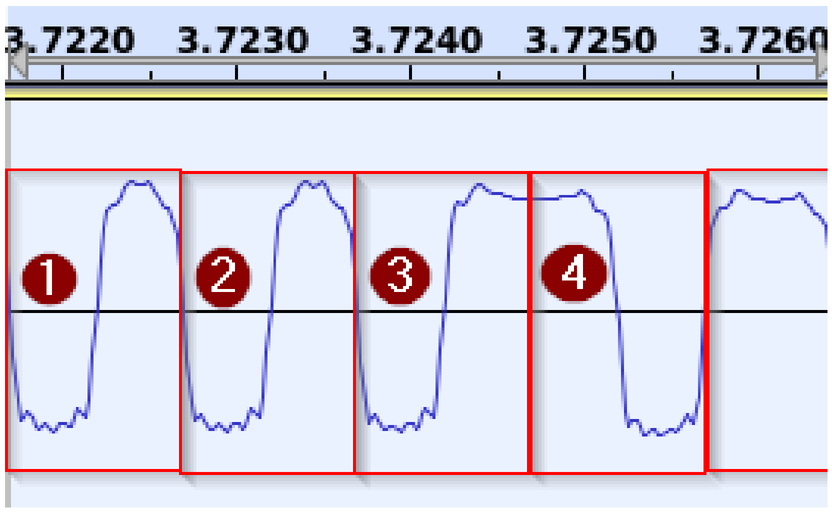


Figure 6.7: Manchester properly fitted in Passsat's signal.

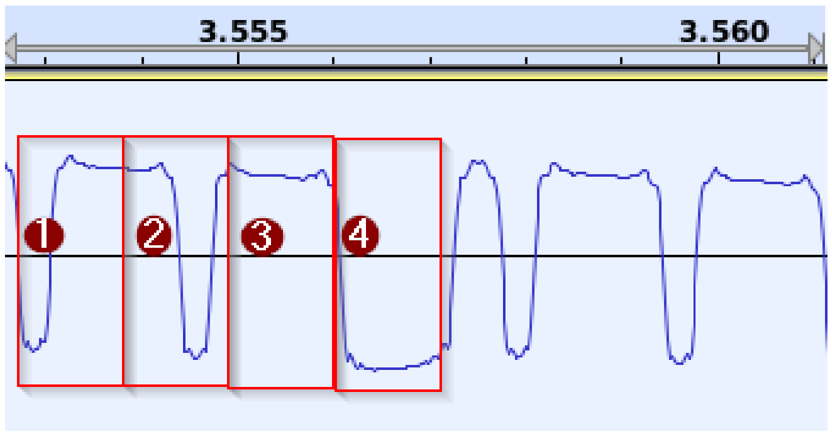


Figure 6.8: Manchester improperly fitted in Garage's signal.

Finally, in general terms, this tool allowed us to calculate the baud rate. This was done taking into account 10 rising edges (it is equivalent to 10 bits), although they could also be falling. This number of edges was arbitrary but allowed a greater reliability than simply taking a bit as a reference. Once these edges are taken, Audacity allows you to see the number of samples in a selection, so what was done was a conversion factor with the number of samples in 10 bits and the number of bits in 48000 samples (which is the 48000 Hz at which the gqrx records), thus resulting in the desired ratio.

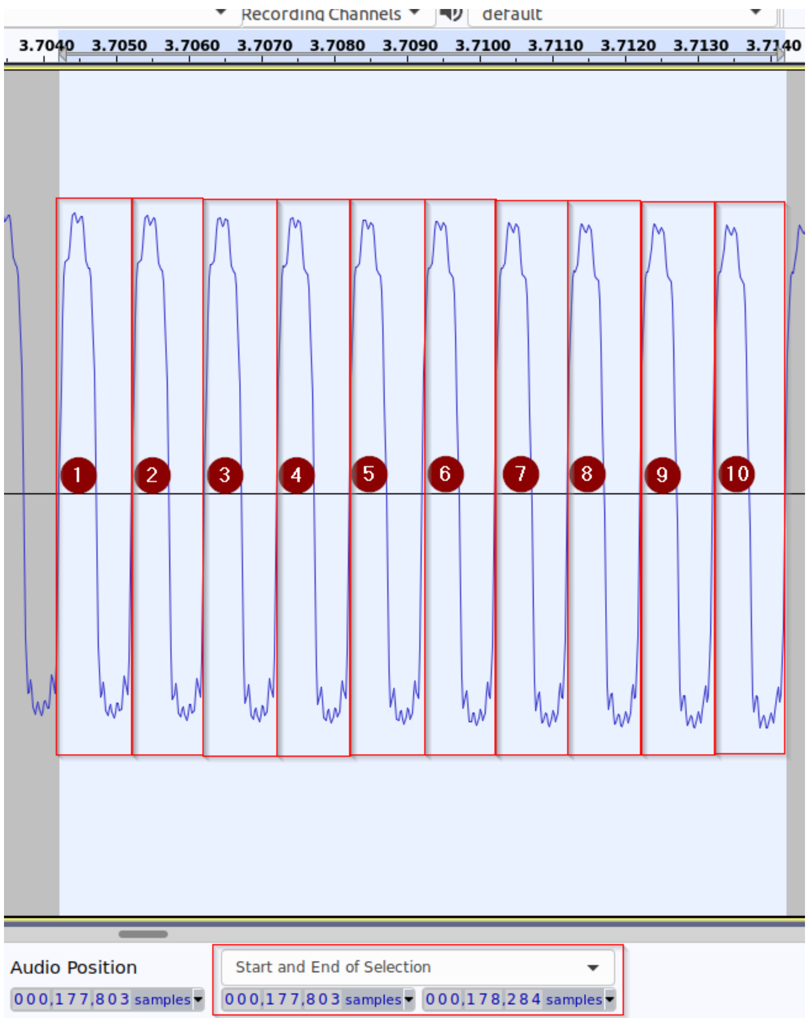


Figure 6.9: Selection of 10 bits to compute baud rate.

Aprimatic TX2M

The analysis of this remote control was one of the most complex, since it was the first one to be analyzed. Therefore, it was not yet known what behavior it was going to have, nor was the audio editing tool, Audacity mastered. For the Aprimatic, it was first observed that the preamble occupied 94 bits and then a 80 bits length message was sent, a very similar length, as explained later, to the Volkswagen Passat 6.1.3 and Mercedes Class A cars 6.1.3. The same message is sent exactly 7 times in total, compose of an initial preamble and then 5 intermediate preambles of shorter duration and a final one of the same duration as the intermediate ones. The initial preamble was composed of 96 times "1" and the rest of the preambles were "0" 26 times. However, no pattern or fixed part was found in 50 analyzed signals 6.9.

On the other hand, for this remote it was discovered that a high state pulse length variation type of encoding was used after multiple tests. A function was made in the script for identifying it, which is described in more detail in the next section. Visually, it can be seen 6.10 how there are two possible states depending on whether you want to transmit a "1" or a "0". One of them first remains in the low state for a longer time and then a short amount of time in the high state, and the other case, which is the other way around. In the Figure 6.10 it can be seen in the first fragment how, what was considered a "1" in this project, is represented. This is due to the fact that it is in a high state for more time than it is in a low state. Also, it can be seen how the second fragment represents the opposite behavior, despite to spend less time in low state compared to the high state of fragment 1, the same occurs with the other state.

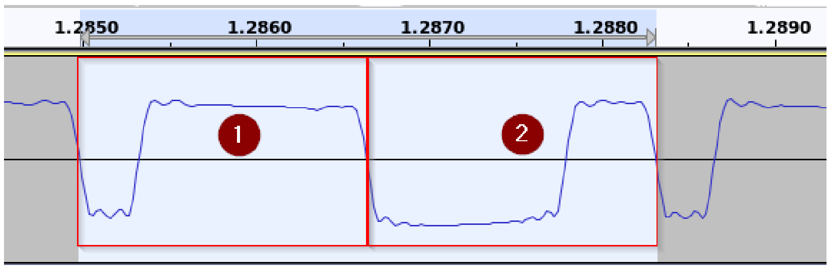


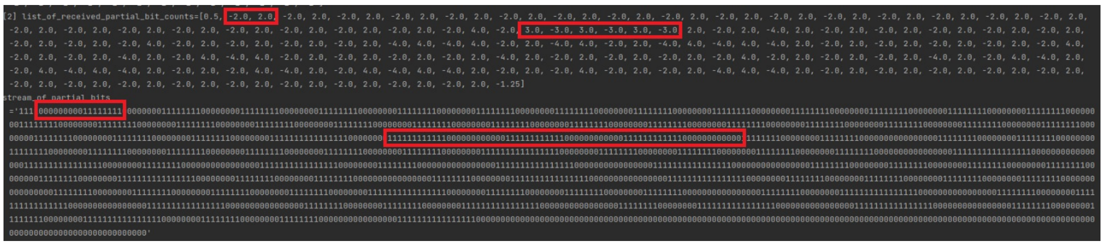
Figure 6.10: Possible states of garage's signal.

To conclude with this remote control, the baud rate was calculated taking into account 5 bits, that is, 10 falling edges 6.9. In this case it turned out to have a value of 600 samples/second. This calculation goes into greater detail in the following key fob 6.1.3 due to the simplicity that its signal provides when calculating it.

Volkswagen Passat

Figure 6.11: Volkswagen Passat car used in the tests.

For the Passat's remote control firstly it was observed that there were 52 initial preamble bits of which 46 were "1100" and the rest were "111100" and then the magic signal, which consisted of 72 "minibits", as illustrated in Figure 6.12 (each bit was sampled 4 times). This magic signal consisted of 3 pulses 50% wider than the normal bits, that is, instead of being 2 high states and 2 low states, they were 3 states each. This, apart from unequivocally identifying the beginning of the message, is used as an anti-copy measure, since if the cloning device works only with a Manchester encoding, at this point it will not be able to continue or it will incorrectly decode the message since the high and low states of the bits would not be synchronized.



```
[2] list_of_received_partial_bit_counts=[0.5, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0,
 -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 4.0, -2.0, 3.0, -3.0, 3.0, -3.0, 3.0, -3.0 2.0, -2.0, 2.0, -4.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0,
 -2.0, 2.0, -2.0, 2.0, -2.0, 4.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -4.0, 4.0, -4.0, 4.0, -2.0, 2.0, -4.0, 4.0, -2.0, 2.0, -4.0, 4.0, -4.0, 4.0, -4.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 4.0,
 -2.0, 2.0, -2.0, 2.0, -4.0, 2.0, -2.0, 4.0, -4.0, 4.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -4.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 4.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -4.0, 2.0, -2.0, 4.0, -4.0, 2.0,
 -2.0, 4.0, -4.0, 4.0, -4.0, 2.0, -2.0, 2.0, -2.0, 4.0, -4.0, 2.0, -2.0, 4.0, -4.0, 4.0, -4.0, 2.0, -2.0, 2.0, -2.0, 4.0, -2.0, 2.0, -2.0, 2.0, -4.0, 4.0, -4.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0,
 -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -1.25]
stream_of_partial_bits
 ='111000000000111111110000000011111111000000000111111110000000001111111100000000011111111000000000111111110000000001111111100000000011111111000000000111111110000000001111111100000000011111111000000000111111110000000001111111100000000011111111000000
 0011111111100000000011111111100000000011111111100000000011111111100000000011111111000000000111111110000000001111111100000000011111111000000000111111110000000001111111100000000011111111000000000111111110000000001111111100000000011111111000
 0000011111111100000000011111111100000000011111111100000000011111111111111111100000000111111111111000000000000011111111111110000000000000111111111111100000000000001111111100000000011111111000000000000000000111111110000000001111111100000000
 1111111100000000011111111000000000111111110000000001111111100000000011111111000000000111111111111111111000000000111111111111111110000000001111111100000000011111111000000000000000001111111111111111100000000000000000
 00011111111111111111100000000011111111000000000000000000011111111111111111100000000111111100000000000000000011111111111111110000000000000000001111111100000000000000000111111110000000001111111100000000011111111000000000111111100
 00000011111111000000000111111111111111110000000001111111100000000011111111000000000000000001111111111111111000000000000000000111111110000000001111111111111111100000000011111111000000000111111110000000001111111100000000011111111000000000
 00000000011111111000000000111111110000000001111111100000000011111111000000000111111111000000000000000111111110000000001111111111111111000000000000000001111111100000000011111111100000000000000000111111110000000001111
 11111111111000000000000000000111111111111111110000000000000000011111111000000000111111111000000000000000001111111111111111100000000000000000011111111000000000111111111111111110000000000000000001111111100000000011111111000000001
 11111110000000001111111111111111000000000111111110000000001111111100000000000000000001111111111111111000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
 00000000000000000000000000000'
```

Figure 6.12: Passat's signal capture.

From this magic sequence, the real message of the key begins to be sent. This message was repeated exactly the same after the intermediary preamble (this is done as a check for errors in the signal). If both messages are not the same, the car will not perform the programmed action which was intended to execute. In total, the length of the signal is 240 bits, which are divided into 80 * 2 message bits since they are repeated twice, plus the three preambles: the initial one, an intermediary one and the final one.

Finally, after analyzing 50 messages of this remote, it has been observed that it has a total of 8 fixed bits, 68 different bits for each message and another 8 variable bits at the end depending on whether it is an open or close signal. An example of each case can be seen in the following table 6.1:

Open/Close	Fixed bits	Intermediate bits	Variable bits
Open	00101111	01001100001110101011110010101000100110110111100111110110100100 11	11010100
Close	00101111	00101010110100111011001110001000011011011000111111110111111101001	11100010

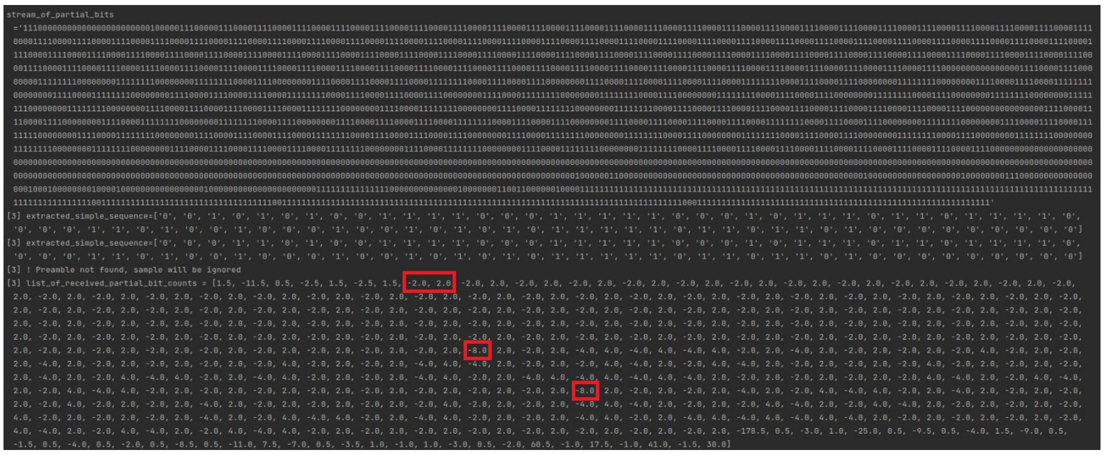
Table 6.1: Comparison between open and close Passat's key fob signals.

For this vehicle, the baud rate was calculated as previously mentioned, taking 10 bits and using 48000 Hz, and it resulted in a total of 480 samples, so the final rate was (48000 * 10 / 480 = 1000 samples/second).

Mercedes A Class

Figure 6.13: Mercedes A Class car used in the tests.

In the class A car you could see an initial preamble of 172 bits made up of the values "1100" for each bit and then the magic signal, as shown in Figure 6.14 (each bit was sampled twice). In this case, the magic signal was 8 low states in a row, of which 2 were distributed to complete the previous bit and the other 6 are the magic signal itself, as in the previous case, it serves as an anti-copy measure for the same reasons than before.



```
stream_of_partial_bits
 ='1110000000000000000000000001000001110000011100001111000011110000111100001111000011110000111100001111000011110000111100001111000011110000111100001111000011110000111100001111000011110000111100001111000011110000111100001111
 0000111100001111000011110000111100001111000011110000111100001111000011110000111100001111000011110000111100001111000011110000111000011110000111100001111000011110000111100001111000011110000111100001111000011110000111100001
 1110000111100001111000011110000111100001111000011110000111100001111000011110000111100001111000011110000111100001111000011110001111000011110000111100001111000011110000111100001111000011110000111100001111000011110000111100
 0011110000111100001111000011110000111100001111000011110000111100001111000011110000111100001111000011110000111100001111000011100001111000011110000111100001111000011110000111100001111000011110000000000000000001111000011110000111100
 0000011111111000000000111111110000000001111111100001111000000000111100001111000011110000111111110000111100001111000000000011110000111100001111000011110000111111110000111100001111000000000111111110000000001111000011110000111111
 0000000011110000111111110000000011110000111100001111000011111111000011110000111100001111000000000111100001111111100000000011111111000011110000000001111111100001111000011110000000001111111100001111000000000111111110000000001111
 1110000000011111111000000000111100001111000011110000111100001111111100000000011110000111111110000000001111000011111111100000000011111111000011110000111100001111000011110000111100001111000011110000111100000000000000000011110000
 1100001111000000000111100001111111100000000011111111000011110000000001111000011110000111100001111111100001111000011110000000000111100001111000011110000111100001111111100001111000011110000000001111111100000000011110000111100001111
 1111100000000011110000111111110000000001111000011110000111100001111111100001111000011110000111100000000011110000111111111000000000111111110000111100000000011111111000011110000111100000000011111111000011110000000001111111100000000
 1111111100000000111111110000000011110000111100001111000011110000111111110000000001111000011111111000000000111100001111111111000000000111111110000111100001111000011110000111100001111000011110000111100000000000000000000000
 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000100000000110000000000000000000000000000000000000000010000000000000000000100000000011100000000000000000
 0001000100000000010000100000000000000000001000000000000000000000000011111111111111100000000000000001000000001100110000001000001111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
 11111111111111110011111111111111111111111111111111111111111001111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111110001111111111111111111111111111111111111111111111111111111111111'
[3] extracted_simple_sequence=['0', '0', '1', '0', '1', '0', '1', '0', '0', '1', '1', '1', '1', '0', '0', '0', '1', '1', '0', '1', '1', '1', '1', '1', '0', '0', '0', '1', '0', '1', '1', '1', '0', '1', '1', '0', '1', '1', '1', '1', '0',
 '0', '0', '0', '1', '1', '0', '1', '0', '0', '1', '0', '0', '0', '1', '0', '0', '1', '0', '1', '0', '1', '0', '1', '1', '0', '1', '1', '1', '1', '1', '0', '1', '1', '0', '1', '1', '0', '1', '0', '0', '0', '0', '0', '0', '0', '0', '0']
[3] extracted_simple_sequence=['0', '0', '0', '1', '1', '0', '1', '0', '0', '1', '1', '1', '1', '0', '0', '0', '1', '1', '0', '1', '1', '1', '1', '1', '0', '0', '0', '1', '0', '1', '1', '1', '0', '1', '1', '0', '1', '1', '1', '1', '0',
 '0', '0', '0', '1', '1', '0', '1', '0', '0', '1', '0', '0', '0', '1', '0', '0', '1', '0', '1', '0', '1', '0', '1', '1', '0', '1', '1', '1', '1', '1', '0', '1', '1', '0', '1', '1', '0', '1', '0', '0', '0', '0', '0', '0', '0', '0', '0']
[3] ! Preamble not found, sample will be ignored
[3] list_of_received_partial_bit_counts = [1.5, -11.5, 0.5, -2.5, 1.5, -2.5, 1.5, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0,
 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0,
 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0,
 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0,
 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0,
 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -8.0, 2.0, -2.0, 2.0, -4.0, 4.0, -4.0, 4.0, -4.0, 4.0, -2.0, 2.0, -4.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 4.0, -2.0, 2.0, -2.0,
 2.0, -4.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 4.0, -2.0, 2.0, -2.0, 2.0, -4.0, 4.0, -4.0, 2.0, -2.0, 2.0, -2.0, 4.0, -4.0, 2.0, -2.0, 4.0, -4.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 4.0, -2.0, 2.0, -2.0, 2.0, -2.0,
 2.0, -4.0, 2.0, -2.0, 4.0, -4.0, 4.0, -2.0, 2.0, -4.0, 4.0, -2.0, 2.0, -2.0, 2.0, -4.0, 4.0, -2.0, 2.0, -4.0, 4.0, -4.0, 4.0, -4.0, 4.0, -4.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 4.0, -4.0, 2.0, -2.0, 4.0, -4.0,
 2.0, -2.0, 4.0, -4.0, 4.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -8.0, 2.0, -2.0, 2.0, -2.0, 2.0, -4.0, 2.0, -2.0, 4.0, -4.0, 4.0, -2.0, 2.0, -4.0, 2.0, -2.0, 2.0, -2.0,
 2.0, -2.0, 4.0, -2.0, 2.0, -2.0, 2.0, -4.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 4.0, -2.0, 2.0, -2.0, 2.0, -4.0, 4.0, -4.0, 2.0, -2.0, 2.0, -2.0, 4.0, -4.0, 2.0, -2.0, 4.0, -4.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0,
 4.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -4.0, 2.0, -2.0, 4.0, -4.0, 4.0, -2.0, 2.0, -4.0, 4.0, -2.0, 2.0, -2.0, 2.0, -4.0, 4.0, -2.0, 2.0, -4.0, 4.0, -4.0, 4.0, -4.0, 4.0, -4.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0,
 4.0, -4.0, 2.0, -2.0, 4.0, -4.0, 2.0, -2.0, 4.0, -4.0, 4.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -178.5, 0.5, -3.0, 1.0, -25.0, 0.5, -9.5, 0.5, -4.0, 1.5, -9.0, 0.5,
 -1.5, 0.5, -4.0, 0.5, -2.0, 0.5, -8.5, 0.5, -11.0, 7.5, -7.0, 0.5, -3.5, 1.0, -1.0, 1.0, -3.0, 0.5, -2.0, 60.5, -1.0, 17.5, -1.0, 41.0, -1.5, 30.0]
```

Figure 6.14: Mercedes's signal capture.

However, the structure of the signal is different. In this case the same message is also sent twice, with a length of 82 bits. However, the beginning of each part of the message, the first 4 bits are different. Nonetheless, they always have the same values: for the first message it will be “0010” and “00001” for the second message. It could be said that it is an identifier so that the receiver knows which is the first message and which is the second one. Finally, it is worth highlighting a peculiar behavior of this remote control: the length itself of the intermediate values can vary according to the length of the fixed part. This last fixed part can have a length from 9 to 11 bits, this behavior was studied in 50 signals. An example of what is mentioned is shown in the following table 6.2:

Id bits	**Intermediate bits**	**Fixed bits**
0010	1010011101101011001111101101000111011101110111111010000010011001010	100000000
0001	1010011101101011001111101101000111011101110111111010000010011001010	100000000

Table 6.2: Message of A Class’s key fob.

Finally, it is worth mentioning two really interesting aspects about this key fob. The first would be that the baud rate is exactly the same as for the Passat. In addition, they have a pretty similar length and a similar anti-copy mechanism, so it can be assumed that the keys come from the same provider despite being different car manufacturers. On the other hand, it seems that there is a change in both the frequency (as illustrated in Figure 6.15) and the amplitude (very slight, as shown in Figure 6.16) of the carrier signal in the initial preamble of the A class signal. Although, it has neither been an impediment when performing the rolljam attack nor can it be sure if it is not a fault of the tool that has been used to capture the signal 7.2.

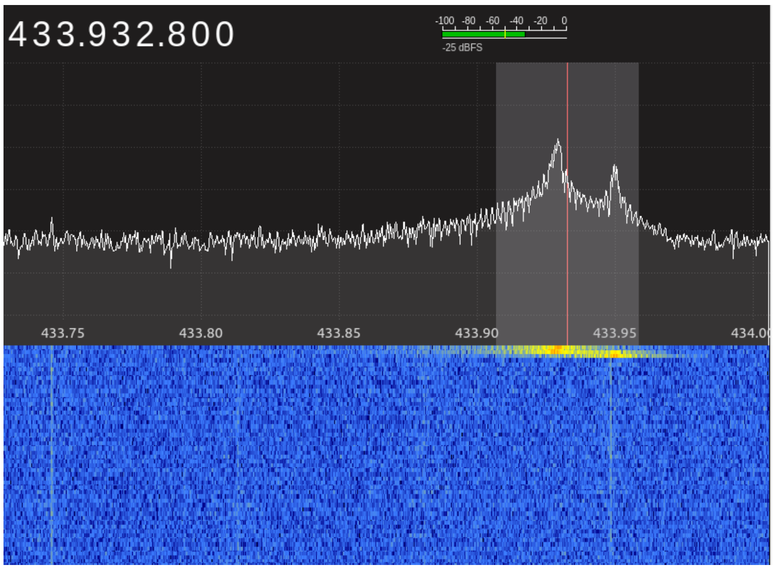


Figure 6.15: Change of frequency in Mercedes’s signal using gqrx.

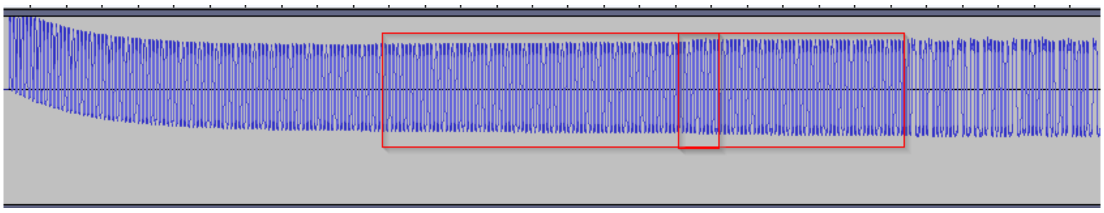


Figure 6.16: Change of amplitude in Mercedes's signal using audacity.

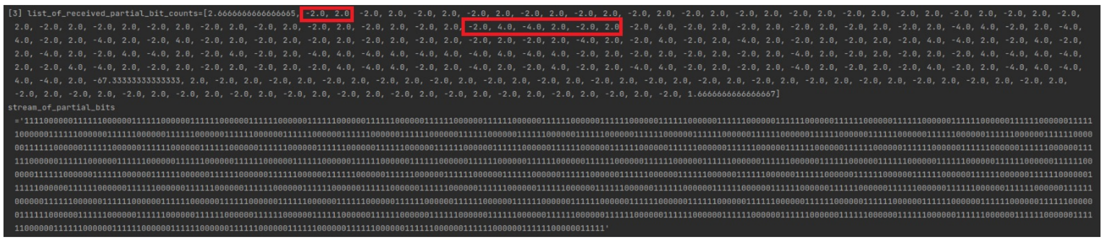
Audi Q2

Figure 6.17: Audi Q2 car used in the tests.

In the remote control of the Audi Q2, a 214-bit preamble consisting of the values "0011" is first sent and no magic signal could be identified, as illustrated in Figure 6.18. Next, the message is sent, which is made up of 96 bits, and finally, before sending the next message, a final preamble is sent. However, in this key fob 3 messages are sent that are independent of each other, but have a similar structure: first 18 bits are the same, then 2 bits that can be "01" or "10" depending on the signal, if it is a closing signal or an open one and the rest are intermediate values of the message. An example of two messages, one is a closing signal and the other one an opening signal, can be seen in the following table 6.3:

Message id	Open/Close	Fixed bits	Open/Close bits	Intermediate values
1	Close	101110000001001001	01	0000101111111001111101001110000000101100010001011000010000100110000101101001
1	Close	101110000001001001	01	0000100000101110001111111000111111000010001110110110011111100000111111111011
1	Close	101110000001001001	01	0000101010011000011000010011100010001010000011000111101101111011111011100000
2	Open	101110000001001001	10	00000001011101111110110010011110001010101000001110010110111111010011101101100
2	Open	101110000001001001	10	0000000101101000010111101101100011000110010101000001010110110001011101010000
2	Open	101110000001001001	10	0000001111000100001111101000001011011001000110100011000000000000110111001111

Table 6.3: First part of messages of Q2's key fob.



```
[3] list_of_received_partial_bit_counts=[2.6666666666666665, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0,
 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 4.0, -4.0, 2.0, -2.0, 2.0, -2.0, 4.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -4.0, 4.0, -2.0, 2.0, -4.0,
 4.0, -2.0, 2.0, -4.0, 2.0, -2.0, 4.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -4.0, 2.0, -2.0, 4.0, -2.0, 2.0, -4.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 4.0, -4.0, 2.0, -2.0, 4.0, -2.0,
 2.0, -4.0, 2.0, -2.0, 4.0, -4.0, 2.0, -2.0, 4.0, -2.0, 2.0, -4.0, 4.0, -4.0, 4.0, -4.0, 4.0, -4.0, 4.0, -4.0, 4.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -4.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 4.0, -4.0, 4.0, -4.0,
 2.0, -2.0, 4.0, -4.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 4.0, -4.0, 4.0, -2.0, 2.0, -4.0, 2.0, -2.0, 4.0, -2.0, 2.0, -4.0, 4.0, -2.0, 2.0, -2.0, 2.0, -4.0, 2.0, -2.0, 2.0, -2.0, 4.0, -2.0, 2.0, -4.0, 4.0, -4.0,
 4.0, -4.0, 2.0, -67.33333333333333, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0,
 -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 2.0, -2.0, 1.6666666666666667]
stream_of_partial_bits
 ='111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111
 1000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000
 0011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111
 1110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100
 0000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001
 1111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111
 0000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000
 0111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111
 11000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111100000001111110000000111111000000011111'
```

Figure 6.18: Audi’s signal capture.

This remote control was the only tested vehicle that has a baud rate of 1700, which is almost the double of what the other two cars send. It is worth mentioning that this car was the only one against which neither the replay attack nor the rolljam attack was effective, but it was possible to subtly jam it and thus achieving a denial of service against it.

6.2 Development

The general structure of the development was as follows:

- Signal capture
  - get_stream_of_partial_bits_from_RF(). This function uses an infinite loop that breaks when it hears what could be a remote control signal. To achieve this, the RFrecv() function is used, which, thanks to the loop, does not work momentarily as it is intended to. Once any signal is received, it is decoded from hexadecimal to binary and it is checked if it can be part of the preamble. If it can be part of the preamble and the jam option is activated, the function will end. Otherwise, depending on the device you want to listen to, a loop will be made with different buffer measurements so that the YARD Stick One is able to catch all the parts of the message in one record or if it has a considerable intermediate preamble, take the two messages in different blocks.

```python
def get_stream_of_partial_bits_from_RF(d: RfCat,
                                       samples_per_bit,
                                       jam: bool):
    print(
        "get_stream_of_partial_bits_from_RF(): Entering RFlisten mode...  \
            packets arriving will be displayed on the screen")
    print("(press Enter to stop)")

    # while not keystop():
    if True:
        list_of_streams_of_partial_bits = []
        while True:
            try:
                y, timestamp = d.RFrecv(
                    blocksize=16 * ACLASS_SAMPLES_PER_PARTIAL_BIT_READ)
                yhex = binascii.hexlify(
                    y).decode()
                stream_of_partial_bits = bin(
                    int(yhex, 16))[2:]

                if could_be_part_of_preamble(
                        stream_of_partial_bits,
                        samples_per_bit):
                    _MY_DEBUG and print(
                        "(%5.3f) received %d bytes: %s | %s" % (
                        timestamp,
                        int(len(yhex) / 2), yhex,
                        stream_of_partial_bits))
                    if jam:
                        print('Preamble detected')
                        return None, timestamp

                    list_of_streams_of_partial_bits.append(
                        stream_of_partial_bits)
                    for blocksize in [256 + 128,
                                      256 + 128,
                                      256 + 128]:
                        y, timestamp = d.RFrecv(
                            blocksize=blocksize)
                        yhex = binascii.hexlify(
                            y).decode()
                        stream_of_partial_bits = bin(
                            int(yhex, 16))[2:]
                        _MY_DEBUG and print(
                            "(%5.3f) received %d bytes: %s | %s" % (
                            timestamp,
                            int(len(yhex) / 2),
                            yhex,
                            stream_of_partial_bits))
                        list_of_streams_of_partial_bits.append(
                            stream_of_partial_bits)
                    print(
                        "get_stream_of_partial_bits_from_RF(): BREAK")
                    break
                else:
                    print('.', end="")
                    list_of_streams_of_partial_bits = [
                        stream_of_partial_bits]
            except ChipconUsbTimeoutException as e:
                print(f'\n! {e=}')
                if len(list_of_streams_of_partial_bits) > 2:
                    break
                else:
                    list_of_streams_of_partial_bits = []
            except BaseException as e2:
                print(f'\n!! {e2=}')
                list_of_streams_of_partial_bits = []

        print(
            "get_stream_of_partial_bits_from_RF(): End")
        print('\n----------------------------')
        return list_of_streams_of_partial_bits, timestamp


```

Listing 6.1: Signal capture function.

– could_be_part_of_preamble(). This function receives the binary data of the signal and counts the number of "1"s and calculates the average with respect to the total length of the data that this function receives. If the number of ones is approximately half of the data block, a function is called which transforms the received data into a list containing the number of continuous "0"s or "1"s in the data. An example would be the following, if that function receives "0011001100110000" it would return [-2, 2, -2, 2, -2, 2, -4] as a result. In an ideal case in which only the preamble was taken and there was no noise, only the values [-2, 2] would be taken. It is worth mentioning that the order has to be in that way, firstly a low state and then a high one.

  However, the most common scenario is to not be able to catch the preamble perfectly, either because it was detected when half of the buffer was already full or because there was noise during the capture, a normal capture example would be [-1.5, 0.5, -3, 2, -2, 2, -2, 2, -2, 2, -2, 1] , the final one also usually appears due to buffer fill issues 7.2.

  Retaking the previous explanation, once it has been detected that, with a certain range, half of the signal values are "1", we proceed to call the previously mentioned function and remove its extreme values on the left and on the right sides. The left value is removed in order to only take into account the main part of the buffer, since it is unlikely to capture the preamble perfectly, and the right value is also removed due to the buffer size, which is usually filled up (it represents the "1" previously mentioned 6.2).

Subsequently, the even positions of that list are taken, which should have a value of "2" and if it turns out to have that value, then it is considered that it is the preamble. It is worth mentioning that for other vehicles it would be necessary to see the sequence of values to be used. However, for the evaluated devices, it was the same sequence for all, except for the garage, which does not use Manchester. It was necessary to count the number of continuous "1"s that were received, since it remains, during its preamble, a considerable amount of time in high state for the garage remote access key.

```
def could_be_part_of_preamble(
        stream_of_partial_bits,
        samples_per_bit):
    count_1s = len(
        [bit for bit in
         stream_of_partial_bits
         if
         bit == "1"])
    fraction_of_ones = count_1s / len(
        stream_of_partial_bits)

    if 0.4 <= fraction_of_ones <= 0.6:
        list_of_received_partial_bit_counts =
            convert_stream_of_partial_bits_to_list_of_partial_bit_counts(
            stream_of_partial_bits,
            samples_per_bit)[-17:-1]
        magic_sum = sum(
            [
                value if pos % 2 == 0 else -value
                for
                pos, value in
                enumerate(
                    list_of_received_partial_bit_counts)])
        magic_fraction = abs(
            magic_sum) / len(
            list_of_received_partial_bit_counts)

        print(
            f'[{round(magic_fraction, 1)}] ',
            end='')
        print(
            f'list_of_received_partial_bit_counts = {[round(value, 1)  \
                for value in list_of_received_partial_bit_counts]}')
        if 1.9 <= magic_fraction <= 2.1:
            return True
        else:
            a = 1
    return False

```

Listing 6.2: Preamble detection function.

– get_list_of_valid_messages(). The objective of this function is to obtain the intermediate bits that make up the variable part of the message, from the binary data received from the get_stream_of_partial_bits_from_RF() function. Firstly, they eliminate the glitches that are in the sequence by calling its corresponding function as detailed below 6.2. Next, the initial position of the message among the received data is searched for. For this propose, the magic signal is searched. If it does not exist, the fixed part of the message is searched, since it is the part immediately before the intermediate bits. If the magic signal has been found, values are taken on-wards from that position and if it is within an approximate range of "-2" or "2", they are added to a list. Subsequently, it is checked that the list has the desired message length for each device and that it does not contain unidentified values, such as "-3" because it would mean that the signal has not been captured correctly. In the case that the message is repeated in the signal, as it may be in the case of the Passat's key fob or a large part of the Mercedes's remote control, it is verified that the first part of the message is equal to the rest of its parts. If they are exactly equal, it is considered valid and it is saved into a json file. In this last step there is a peculiarity in the case of the garage control, as the same message is sent 7 times. It is unlikely that the entire signal will be correctly captured, so what was done was to average all the values of the different messages that have been captured correctly. When the number of correct messages is greater than half of the number of messages composing the signal, corresponding to 4 messages or more.

```
def get_list_of_valid_messages(
        list_of_streams_of_partial_bits,
        samples_per_bit):
    burst_list = []

    for sample_number, stream_of_partial_bits in enumerate(
            list_of_streams_of_partial_bits):
        stream_of_partial_bits = remove_micro_glitches(
```

```python
        stream_of_partial_bits)
    _MY_DEBUG and print(
        f'{stream_of_partial_bits=}')

    list_of_received_partial_bit_counts =
        convert_stream_of_partial_bits_to_list_of_partial_bit_counts(
        stream_of_partial_bits,
        samples_per_bit)

    first_position_to_check = 0
    while True:
        message_start_position = get_next_message_start_position(
            list_of_received_partial_bit_counts,
            first_position_to_check)

        if message_start_position >= 0:
            extracted_simple_sequence = get_simple_sequence(
                list_of_received_partial_bit_counts,
                first_position_to_check=message_start_position)
            _MY_DEBUG and print(
                f'[{sample_number}] {extracted_simple_sequence=}')

            if len(extracted_simple_sequence) < ACLASS_MESSAGE_BITS:
                print(
                    f'Extracted sequence is not long enough,  \
                        {len(extracted_simple_sequence)} <  \
                        {ACLASS_MESSAGE_BITS}, ignoring sequence')
            else:
                is_valid = True
                for symbol in extracted_simple_sequence[
                              :ACLASS_MESSAGE_BITS]:
                    if symbol not in [
                        _ONE,
                        _ZERO]:
                        print(
                            'Error! Extracted sequence containing
unexpected symbols, ignoring')
                        is_valid = False
                        break
                if is_valid:
                    burst_list.append(
                        extracted_simple_sequence[
                        :ACLASS_MESSAGE_BITS])
            first_position_to_check = message_start_position + 10
        else:
            _MY_DEBUG and print(
                f'[{sample_number}] ! Preamble not found, \
                    sample will be ignored')
            break
    print(
        f'[{sample_number}] list_of_received_partial_bit_counts =  \
            {[round(value, 1) for value in  \
            list_of_received_partial_bit_counts]}')
    a = 1
message_matches = {}

for message_on_test_as_list in burst_list:
    message_on_test_as_str = ''.join(
        message_on_test_as_list)
    if message_on_test_as_str in message_matches.keys():
        message_matches[
            message_on_test_as_str] = \
            message_matches[
                message_on_test_as_str] + 1
    else:
        message_matches[
            message_on_test_as_str] = 1

winner_message_count = max(
    message_matches.values())
winner_messages_list = [
    (message, count) for
    message, count in
    message_matches.items()
    if
    count == winner_message_count]

return winner_messages_list


```

Listing 6.3: Get list of valid messages function.

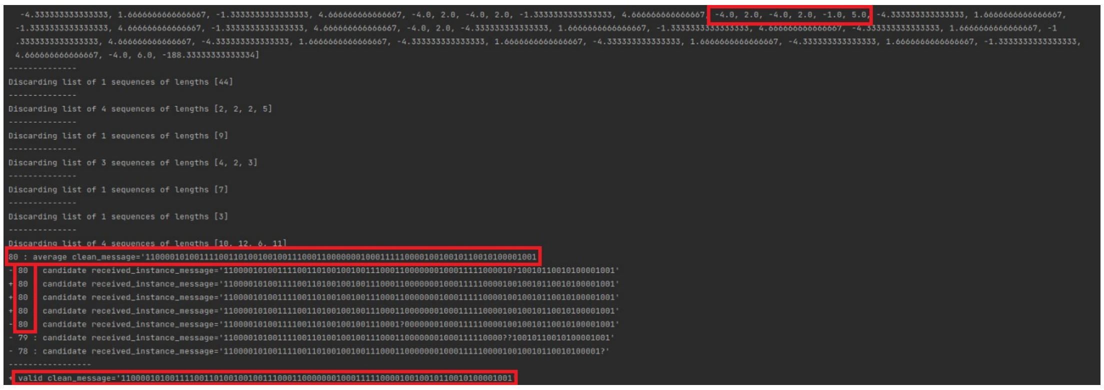
– remove_micro_glitches(). This simple function has the following propose, if a single “0” is found between a segment of “1”s, it is a glitch and its value is changed to “1”. The same occurs the other way around. An example would be the following: assuming that we have the following message fragment from the garage remote “000011 010111 000011”, as we know that the Aprimatic can only be made up of the values “000011” or “011111”, after applying this function the result would be “000011110111000011”.

```
def remove_micro_glitches(
        stream_of_partial_bits):
    stream_of_partial_bits = ''.join(
        [
            stream_of_partial_bits[
                0]] + [
            _ONE if
            stream_of_partial_bits[
                pos - 1] == _ONE and
            stream_of_partial_bits[
                pos + 1] == _ONE else
            stream_of_partial_bits[
                pos] for pos
            in
            range(1,
                  len(stream_of_partial_bits) - 2)] + [
            stream_of_partial_bits[
                -1]])
    return stream_of_partial_bits

```

Listing 6.4: Remove glitches function.

– convert_stream_of_partial_bits_to_sampled_lengths_list(). This function counts the number contiguous "1" or "0" and added to a list. For example, in the case of the garage it allowed to determine what type of encoding was used. This was due to the fact that the signal was observed to be composed of 6 ”minibits”. This selection of 6 was decided after multiple tests and it was found that it was the best fit. For example, if the received message was 00001101111, it would give us the result [(-4,2),(-1,5)], as shown in Figure 6.19. This function, allows for discerning a clear pattern in the message and therefore it could be deduced that it used that particular coding. It may be that with more experience in the field of telecommunications, it would not have taken as many tests or this type of function to see the pattern. However, it was the only valid solution that was found with time and knowledge so far.



```
 -4.333333333333333, 1.6666666666666667, -1.3333333333333333, 4.666666666666667, -4.0, 2.0, -4.0, 2.0, -1.3333333333333333, 4.666666666666667, -4.0, 2.0, -4.0, 2.0, -1.0, 5.0, -4.333333333333333, 1.6666666666666667,
 -1.3333333333333333, 4.666666666666667, -1.3333333333333333, 4.666666666666667, -4.0, 2.0, -4.333333333333333, 1.6666666666666667, -1.3333333333333333, 4.666666666666667, -4.333333333333333, 1.6666666666666667, -1
 .3333333333333333, 4.666666666666667, -4.333333333333333, 1.6666666666666667, -4.333333333333333, 1.6666666666666667, -4.333333333333333, 1.6666666666666667, -4.333333333333333, 1.6666666666666667, -4.333333333333333, 1.6666666666666667, -1.3333333333333333,
 4.666666666666667, -4.0, 6.0, -188.33333333333334]
--------------
Discarding list of 1 sequences of lengths [44]
--------------
Discarding list of 4 sequences of lengths [2, 2, 2, 5]
--------------
Discarding list of 1 sequences of lengths [9]
--------------
Discarding list of 3 sequences of lengths [4, 2, 3]
--------------
Discarding list of 1 sequences of lengths [7]
--------------
Discarding list of 1 sequences of lengths [3]
--------------
Discarding list of 4 sequences of lengths [10, 12, 6, 11]
80 : average clean_message='110000101001111001101001001001110001100000000100011111000010010010110010100001001
- 80   candidate received_instance_message='110000101001111001101001001001110001100000000100011111000010?10010110010100001001'
+ 80   candidate received_instance_message='110000101001111001101001001001110001100000000100011111000010010010110010100001001'
+ 80   candidate received_instance_message='110000101001111001101001001001110001100000000100011111000010010010110010100001001'
+ 80   candidate received_instance_message='110000101001111001101001001001110001100000000100011111000010010010110010100001001'
- 80   candidate received_instance_message='1100001010011110011010010010011100011?0000000100011111000010010010110010100001001'
- 79 : candidate received_instance_message='1100001010011110011010010010011100011000000001000111110000??10010110010100001001'
- 78 : candidate received_instance_message='1100001010011110011010010010011100011000000001000111110000100100101100101000010?'
----------------
+ valid clean_message='110000101001111001101001001001110001100000000100011111000010010010110010100001001
```

Figure 6.19: Garage's signal capture.

```python
def convert_stream_of_partial_bits_to_sampled_lengths_list(
        stream_of_partial_bits):
    sampled_lengths = []
    current_value = \
        stream_of_partial_bits[0]
    current_length = 0
    for bit_value in stream_of_partial_bits:
        if bit_value == current_value:
            current_length += 1
        else:
            sampled_lengths.append(
                current_length if current_value == (_ONE else -
                    current_length))
            current_value = bit_value
            current_length = 1
    # last sample
    sampled_lengths.append(
        current_length if current_value == _ONE else -current_length)
    return sampled_lengths

```

Listing 6.5: Transform stream of bits to sampled lengths of bits function.

– get_next_message_start_position(). This function allows for identifying the magic signal, if it exists (for the Passat and the A Class cars). A characteristic part of the signal is identified for the Q2 and the garage. For example, in the case of the Passat's remote control, the following sequence was searched for: "-3, 3, -3, 3, -3, 3".

```python
def get_next_message_start_position(
        list_of_received_partial_bit_counts,
        first_position_to_check):
    for pos in range(
            first_position_to_check,
            len(list_of_received_partial_bit_counts) - 6):
        if 1.5 <= \
                list_of_received_partial_bit_counts[
                    pos] <= 3:
            if -3 <= \
                    list_of_received_partial_bit_counts[
                        pos + 1] <= -1.5:
                if 1.5 <= \
                        list_of_received_partial_bit_counts[
                            pos + 2] <= 3:
                    if -9 <= \
                            list_of_received_partial_bit_counts[
                                pos + 3] <= -7:
                        if 1.5 <= \
                                list_of_received_partial_bit_counts[
                                    pos + 4] <= 3:
                            if -3 <= \
                                    list_of_received_partial_bit_counts[
                                        pos + 5] <= -1.5:
                                return pos + 4
    return -1

```

Listing 6.6: Search of magic signal function.

– write_to_file(). The objective of this function is simply to store the message of the signal and other characteristics of it in a json file. If the file does not exist, it creates it with the name of the device type (garage, Passat, A class or Q2) followed by the date when it was taken and if it exists it simply adds it to the end of file.

```
def write_to_file(
        list_of_streams,
        samples_per_bit,
        timestamp,
        type, state,
        number_of_reads):
    output_path = "/home/alessandro/PycharmProjects/TFG/Samples/JSON/"
    file_name = f'{output_path}/ACLASS.{time.strftime("%Y-%m-%d")}.json'

    message_info = {
        # stream, garage, timestamp, samples_per_bit, state in JSON
        "list_of_streams": list_of_streams,
        "samples_per_bit": samples_per_bit,
        "timestamp": timestamp,
        "type": type,
        "state": state,
        "number_of_reads": number_of_reads
    }

    if exists(file_name):
        with open(
                file_name,
                "r") as log_file:
            messages_info_list = json.load(
                log_file)
    else:
        messages_info_list = []

    messages_info_list.append(
        message_info)

    with open(file_name,
              "w") as log_file:
        json.dump(
            messages_info_list,
            log_file,
            indent=4)

    return None

```

Listing 6.7: Store signal information function.

- Signal transmission
  - Configuration of the YARD Stick One. In general terms the sending function is not as complex as the receiving one. In this first part, some aspects are configured, such as the transmission data rate, which may not be the same as the reception rate, and the maximum power (a relevant aspect when jamming, because if it is set to a very high value, it is possible that also cannot be received correctly by the attacker).
  - convert_message_to_partial_bit_string_to_send(). This function is responsible for transforming the bits of the message to be sent to a ”minibit” format that is part of the format desired by the RFxmit() function, which is responsible for sending the replica of the signal.
  - add_x(). It is the last relevant function and is responsible for converting the replica

of the signal in hexadecimal and adding "\x" in front of each pair of values. It should be noted that the used Integrated development environment (IDE) converts certain characters from bytes to ASCII due to visualizing purposes.

– Transformation example for sending the signal, using the Passat's remote control: preamble + message:

```
11001100110011001100110011001100110011001100110011
00110011001100110011001100110011001100110011001100
11001100110011001100110011001100110011001100110011
00110011001100110011001100110011001111001110001110
00111000 + 1100110000110011001100110011001111000011
00111100110011000011110000110011001111000011110011
00001100111100110000111100001111001100001111001100
00111100110000111100001100111100001111001100110011
00001100110011001111000011110000110011001100111100
11000011001100111100110011000011001100110011001100
110011001111001100110000111100 = \xcc\xcc\xcc\xcc\xcc
\xcc\xcc\xcc\xcc\xcc\xcc\xcc\xcc\xcc\xcc\xcc\xcc\xcc
\xcc\xcc\xcc\xcc\xcc\xf3\x8e8\xcc333\xc3<\xcc<3<<\xc3<
\xc3\xc3\xcc<\xc3\xcc<3\xc3\xcc\xcc33\xc3\xc33<\xc33
\xcc\xc3333<\xcc<
```

– Note that jamming only occurs when a preamble is detected, it is executed for a time similar to that of the original key fob and at a power high enough to inhibit the signal so that the receiver does not validate it as correct but the attacker is able to capture it.

6.3 Countermeasures

This section presents a series of countermeasures against successful implemented attacks:

- Correct implementation of the rolling codes to avoid the replay attack. Therefore, once a message has been sent, even if the next message is different, the receiver should invalidate all the previous messages in the counter so it cannot be reused.

- A possible countermeasure for jamming attacks would be a considerable change of frequency during the sending of the signal. For example, if the remote access key sends two identical messages, the first one could be sent on one frequency and the second message on a different one. According to this, if the jamming device works from the moment it detects the signal's preamble, the receiver would correctly detect the first message. Meanwhile, the second message would not be affected by the jamming and therefore, the vehicle will operate as intended.

- Implementation of a remote key whose emitted signal has a temporary validity. That is, the receiver accepts a different range of valid keys from time to time, incrementing the counter once that time has elapsed.
- A possible solution against the rolljam attack would be to simply unlock the car manually with the physical key itself, if this method is available.
- One possibility to prevent the signal from being captured correctly would be that the remote uses neither a magic signal nor any identifiable part of the signal. In this way, each signal is completely different and the attacker would not be able to correctly capture the signal to carry out the attack.
- The use of a mobile application, by means of Internet, through which the car can be opened or closed, implementing the pertinent security mechanisms, such as the use of encryption.

Chapter 7

Conclusions

To conclude this project, it will be detailed in the following sections if the objectives of the project have been met, incidents that have been faced during the evolution of the development and future improvements to the current implementation for the tool.

7.1 Objectives achievement

The project has been successfully completed in terms of meeting objectives, the following have been carried out:

- **Hardware assembly**. It has been possible to assemble the necessary infrastructure to carry out the various attacks, from a jamming device such as the Raspberry Pi to the device for listening to the signal of the different controls, which has been the personal laptop.
- **Identification and analysis of the systems used in different vehicles**. Through the gqrx and Audacity tools it was possible to study the behavior of the signals, seeing in detail their type of encoding, message structure and baud rate used. Differentiating between two main groups, the garage control and the vehicle control, each group used a different encoding. However, it was concluded that the base frequencies were relatively similar, around 433-434 MHz and they all use the same type of modulation, the amplitude modulation.
- **Tool for implementing the different attacks**. During the project, a different script was developed for each device, which is capable of performing different functions: capturing signals, transmitting, echoing the capture signal, jamming and saving the signal messages in a json file.
- **Evaluation of the security of different vehicles**. It has been concluded that the RKS of the garage is the most insecure since once a signal has been captured, despite

using rolling codes, it could continue to be used indefinitely. However, the Passat's mechanism had a correct implementation of this type of codes and once a signal has been used, it could no longer be reused. Therefore, for this car it was able to perform the replay attack and the rolljam attack. On the other hand, the Mercedes's remote control, despite having a similar structure, could only be opened using the rolljam attack technique. It has been believed that it is because the Mercedes's remote must have a second chip, having two batteries as explained in the incident section 7.2, which keeps a very small time frame in which the transmitted signals are accepted. Finally, in the case of the Audi, it was only possible to jam against it, none of the proposed techniques worked. It was not possible to conclude a probable cause about why the attack failed, a more in-depth study of the signal would have to be carried out, spending more time analyzing it. However, it may have happened due to some type of variation in the intermediate preambles because they did not always represent the exact same length. Although, it may be a failure of the capture tool or a change in the amplitude of the signal in certain parts, again it is not certain if it was the same reason.

To conclude, new knowledge and experience have been acquired in the area of telecommunications as well as in remote keyless entry security, from simpler ones to more modern systems.

7.2 Issues faced during the project

- Lack of online information about the Aprimatic TX2M. There was not information on the Internet about this controller, so finding out what type of encoding it used was more complex. In addition, the most common encodings did not match the received signal, so until the function for counting the occurrences of continuous numbers was implemented, the encoding was not determined.
- Problems in capturing the signal from the Mercedes control, which during its analysis ran out of battery and that is when it was discovered that it used two batteries. This finding allowed us to deduce, not confirm, that it could be using two different chips, one to keep track of time and another one for the normal operation of the controller itself, and that is why it is believed that the replay attack did not work.
- Limitation in the size of the YARD Stick One buffer. It did not have a large buffer, different possible sizes had to be calculated to be able to capture the entire message in one shot so that there were no errors.

- The sudden appearance of noise when capturing the signal, probably due to weather conditions since there were adverse conditions on those days.
- Problem about the key size of the remote controls. The length of the messages sent by all the commands analyzed was too long to carry out a brute force attack, even eliminating the fixed or variable bits depending on whether they open or close.

7.3 Future improvements

- Automation of the sending process. Currently, to perform the replay attack you have to manually take the value of the JSON file and enter it in the send function or use the function of echoing with delay.
- Unify the different scripts into a single one in which you will have to pass to the main function a configuration file depending on the device to be tested.
- Automatic analysis of stored messages, to be able to analyze which bits of the message are fixed, which are variables or intermediate.
- Analysis and development of scripts for other attacks and vehicles, for example the two-thief one against a BMW X1.
- Perform tests with other signal analysis tools such as rtl_443 [20] or other devices such as HackRF One [21].

Appendices

Appendix A

Additional material

A.1 Download tool repository

The repository can be cloned from the following link: https://github.com/AlessandroAldrey/TFG. The content of it is structured as follows:

- **Python folder**. It contains the scripts for each analyzed device.
- **Samples/JSON folder**. This folder contains the captured signal messages in JSON format for each remote access key.
- **Videos folder**. It contains different demonstrations for the different attacks.
- **WAV_Files folder**. Audio recordings of each analyzed device are located in this folder.

List of Acronyms

AGC Automatic Gain Control. 18

Automatic Gain Control Is a volume-operated gain-adjusting device which is used to prevent overmodulation. 18

baud rate Number of symbols transmitted per second, where symbol stands for a detectable state of the transmission medium. 30

IDE Integrated development environment. 48

Integrated development environment Integrated development environment, enables programmers to consolidate the different aspects of writing a computer program. 48

LNA Low-noise amplifier. 28

Low-noise amplifier Low-noise amplifier is an electronic amplifier that is used to amplify signals of very low strength. 28

minibits Transformation of regular bits which are ’0’ or ’1’ to its real representation according to the rflib format Where usually they are shown as ’1100’ or ’0011’. This allows for a more reliable representation of the information when it is processed. 33

Radio Frequency Frequency or band of frequencies located in the range of 104 to 1012 Hz, suitable for use in telecommunications. 7

RFID Radio-frequency identification. 2

script Source code written in any interpreted language. 13

SMA Subminiature version A. 14

Software Defined Radio Software Defined Radio is a radio communication system where components like mixers or filters are implemented by means of software. 18

squelch Circuit function that acts to suppress the audio output of a receiver in the absence of a strong input signal. 14

Subminiature version A Subminiature version A is a coaxial RF connector. 14

Temperature-Compensated Crystal Oscillator A temperature-compensated crystal oscillator is used whenever particularly high stability within a variant temperature environment is required, i.e. when the frequency deviation of the oscillator must be minimal over its entire operating temperature range. 14

WAV file It is a kind of file used to store videos or audios. 18

Glossary

AGC Automatic Gain Control. 18

Automatic Gain Control Is a volume-operated gain-adjusting device which is used to prevent overmodulation. 18

baud rate Number of symbols transmitted per second, where symbol stands for a detectable state of the transmission medium. 30

IDE Integrated development environment. 48

Integrated development environment Integrated development environment, enables programmers to consolidate the different aspects of writing a computer program. 48

LNA Low-noise amplifier. 28

Low-noise amplifier Low-noise amplifier is an electronic amplifier that is used to amplify signals of very low strength. 28

minibits Transformation of regular bits which are '0' or '1' to its real representation according to the rflib format Where usually they are shown as '1100' or '0011'. This allows for a more reliable representation of the information when it is processed. 33

Radio Frequency Frequency or band of frequencies located in the range of 104 to 1012 Hz, suitable for use in telecommunications. 7

RFID Radio-frequency identification. 2

script Source code written in any interpreted language. 13

SMA Subminiature version A. 14

Software Defined Radio Software Defined Radio is a radio communication system where components like mixers or filters are implemented by means of software. 18

squelch Circuit function that acts to suppress the audio output of a receiver in the absence of a strong input signal. 14

Subminiature version A Subminiature version A is a coaxial RF connector. 14

Temperature-Compensated Crystal Oscillator A temperature-compensated crystal oscillator is used whenever particularly high stability within a variant temperature environment is required, i.e. when the frequency deviation of the oscillator must be minimal over its entire operating temperature range. 14

WAV file It is a kind of file used to store videos or audios. 18

Bibliography

[1] "Replay attack." [Online]. Available: https://www.researchgate.net/figure/Typical-scenario-of-replay-attack_fig2_276489970

[2] "Two-thief attack." [Online]. Available: https://carkeysignalblocker.co.uk/what-are-relay-attack/

[3] "Rolljam attack diagram." [Online]. Available: https://github.com/jordib123/replay-jamming-attack

[4] "Comparison between am and fm modulations." [Online]. Available: https://en.wikipedia.org/wiki/Frequency_modulation#/media/File:Amfm3-en-de.gif

[5] "Ask/ook modulation." [Online]. Available: https://www.tutorialspoint.com/digital_communication/digital_communication_amplitude_shift_keying.htm

[6] "Return-to-zero codification." [Online]. Available: http://rahmatmuammar.blogspot.com/2016/01/polar-unipolar-dan-bipolar-encoding.html

[7] "Nooelec nesdr." [Online]. Available: https://www.nooelec.com/store/nesdr-smartee.html

[8] "Yard stick one." [Online]. Available: https://greatscottgadgets.com/yardstickone/

[9] "Raspberry pi 4b." [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

[10] "Federal communications commission." [Online]. Available: https://fccid.io/

[11] H.-l. LIU, J.-s. MA, S.-y. ZHU, Z.-j. LU, and Z.-l. LIU, "Practical contactless attacks on hitag2-based immobilizer and rke systems," *DEStech Transactions on Computer Science and Engineering*, 08 2018.

[12] J. Wetzels, “Broken keys to the kingdom, security and privacy aspects of rfid-based car keys.”

[13] W. Silver, *Communications receivers DSP, software radios and design*, 3rd ed. The McGraw-Hill Companies, 2004.

[14] A. S. Tanenbaum, *COMPUTER NETWORKS*, 5th ed. Prentice Hall, 1944.

[15] “Institute of electrical and electronics engineers.” [Online]. Available: https://standards.ieee.org/

[16] “Gqrx.” [Online]. Available: https://gqrx.dk/

[17] “Python 3.9.13.” [Online]. Available: https://www.python.org/

[18] “Salaries of jobs related to it in galicia.” [Online]. Available: https://es.scribd.com/document/288511179/Guia-Salarial-Sector-TI-Galicia-2015-2016

[19] “Cybsersecurity analist.” [Online]. Available: https://www.glassdoor.com/Job/spain-ingeniero-de-ciberseguridad-jobs-SRCH_IL.0,5_IN219_KO6,33.htm

[20] “Generic data receiver rtl433.” [Online]. Available: https://github.com/merbanan/rtl_433

[21] “Hackrf one.” [Online]. Available: https://greatscottgadgets.com/hackrf/one/