www.uhasselt.be
Universiteit Hasselt
Campus Hasselt:
Martelarenlaan 42 | 3500 Hasselt
Campus Diepenbeek:
Agoralaan Gebouw D | 3590 Diepenbeek

Faculteit Wetenschappen

School voor Informatietechnologie

master in de informatica

Masterthesis

Security analysis and exploitations of keyless entry systems in cars

Jordi Breuls

Scriptie ingediend tot het behalen van de graad van master in de informatica

PROMOTOR :
Prof. dr. Peter QUAX

2018
2019

Masterthesis

Security analysis and exploitations of keyless entry systems in cars

Jordi Breuls
Scriptie ingediend tot het behalen van de graad van master in de informatica

PROMOTOR :
Prof. dr. Peter QUAX

Acknowledgements

First and foremost I would like to thank my promoter Prof. Dr. Peter Quax for providing me with the opportunity to work on this interesting subject for the past year. Furthermore, I was also given a lot of freedom in choosing the direction of this thesis and my research. Second, I would like to thank my mentor Pieter Robyns for proofreading this thesis, providing constructive feedback and useful answers on a lot of questions asked throughout the year. Like Prof. Dr. Peter Quax, Pieter gave me a lot of freedom in choosing the direction of my research. Last, I would like to thank my parents, family and Mariano Di Martino for allowing me to perform tests on their cars equipped with remote and passive keyless entry.

As this thesis marks the end of 5 years education at the University of Hasselt, I would like to thank all teachers, supervisors and staff that provided me with this rich education about computer science and many other fields. Last, I would also like to thank my student colleagues and friends for helping each other and making the past 5 years a fun experience during and after courses.

Contents

Chapter 1

Introduction

1.1 Motivation

Although the total amount of car thefts and car burglaries are decreasing over the last decade, the way of breaking into cars is changing. For the past years, car theft exposing weaknesses in the keyless entry system has known a huge increase in numbers. The most popular method is exposing cars equipped with Passive Keyless Entry (PKE) by performing a so-called relay attack. In PKE systems, the car will automatically unlock its doors when the key fob is detected in close proximity to the car (within 1 meter). By performing a relay attack, adversaries are able to extend this limited distance to tens or even hundreds of meters. Doing so, they are able to unlock cars without the physical key fob actually being in proximity of the car. Most cars equipped with PKE also replace the traditional mechanical key to start the engine (Passive Keyless Entry and Start). If the key fob is detected inside the car, the owner can push a button to start the engine. This makes the relay attack even more worrying as it not only unlocks the doors but also allows the adversary to drive of with the car.

The rise of these attacks has not gone unnoticed by the mainstream media. There are dozens of news articles and TV-reports about car thefts exposing the weaknesses of PKE(S) systems using such relay attacks [36] [57] [67] [7] [69]. According to Driver and Vehicle Licensing Agency (DVLA) in the United Kingdom, car thefts by performing a relay attack has known an impressive grown of 9.000 incidents in 2016 to 43.408 incidents in 2017 [58]. According to the Association of British Insurers, a record of €309 million in theft claims was recorded solely in the United Kingdom in the first nine months of 2018. Stichting Aanpak Voertuigcriminaliteit (STAVC) estimates that 40% of the 64.000 registered car burglaries in 2018 in the Netherlands are due to relay attacks [11]. Car thefts in the United States has known a decrease since 2004 (expect from 2012) [33]. However, since 2015 the amount of cars stolen began to increase again. Around this time PKE(S) systems started to commonly appear on lower-end cars instead of solely high-end luxury cars while the relay attack started to appear more frequently. As the reports don't give an official cause of this increase, the increase of cars equipped with PKE(S) and the rise of these relay attacks might be one of the reasons.

Although most media exposure and research is mainly focused on weaknesses in PKE(S) systems. The traditional Remote Keyless Entry (RKE) system is not completely secure either. In RKE systems, the car is (un)locked by pressing a button on the key fob. By executing a so-called replay & jamming attack, adversaries are able to gain access to victim's cars without physically having the key fob. In contrast to relay attacks, there are no official statistics regarding car thefts or burglaries by executing these replay & jamming attacks. Although the weaknesses of PKE(S) systems are much more worrying since adversaries can not only gain access to the car but also start it, the automobile industry have been using insecure RKE systems for decades.

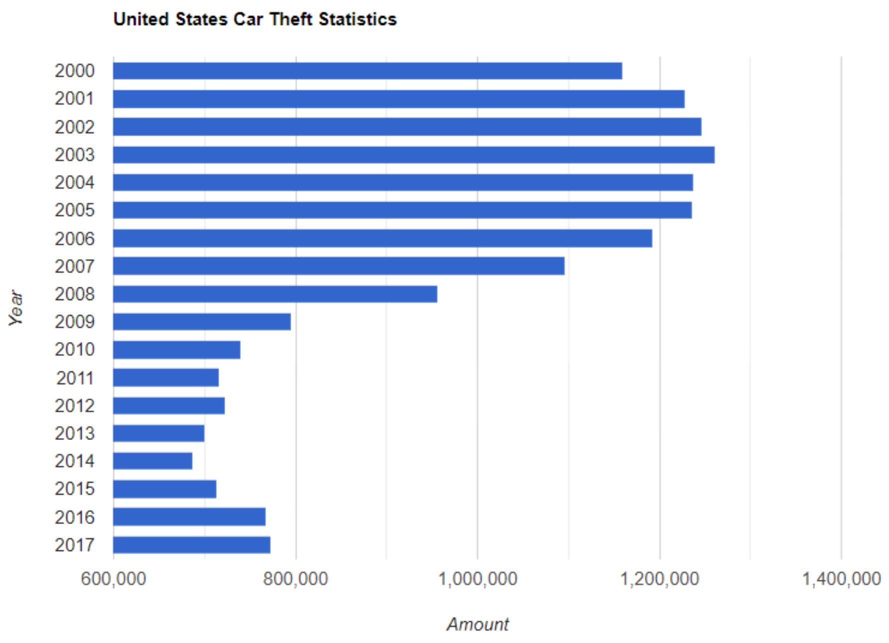


Figure 1.1: Car theft statistics from the United States (Numbers provided by [33]).

1.2 Research questions

The goal of this thesis is to provide an overview of the past and the present of keyless entry systems in the automobile industry. We'll implement a replay & jamming attack and relay attack against respectively RKE and PKE systems and discuss their possible countermeasures. As keyless entry protocols are not publicly published we'll also dig into the art of reverse engineering to obtain knowledge about these protocols used in RKE systems.

We can break up this thesis into 3 topics for which we present our research questions:

1. Exploitation of RKE systems (replay & jamming attack):
   (a) What is the feasibility (success rate, difficulty, cost, ...) of implementing a replay & jamming attack using software-defined radios? How can we reduce the cost of such implementation?
   (b) Is it possible to implement reactive jamming using software-defined radios taking into account the delay introduced by processing and transferring of data?
   (c) What are the possible countermeasures to prevent against these replay & jamming attacks? What is the current state of these solution and are they currently used by car manufacturers?
2. Exploitation of PKE(S) systems (relay attack):
   (a) Is the delay introduced by processing and transferring of data using a software-defined radio small enough to stay below the maximum delay threshold in order to successfully perform a relay attacks?
   (b) What are the possible countermeasures to prevent against these relay attacks? What is the current state of these solution and are they currently used by car manufacturers?
3. Obtaining knowledge about the protocols used in RKE systems by reverse engineering the radio frequency signals transmitted by the key fob:
   (a) What does the physical layer (modulation, data encoding, ...) and MAC layer (message format, data integrity, ...) of RKE protocols look like.

(b) Do these protocols vary between different car manufacturers and different car models from the same manufacturer?

In Chapter 2 we'll provide a background on various topics regarding keyless entry systems. We'll start with Section 2.1 in which we'll give a detailed explanation of how both RKE and PKE(S) systems work. In Section 2.2 we'll give a detailed explanation of how rolling code works, one of the most import security components of RKE systems. Various exploitations of keyless entry systems together with their possible countermeasures are explained in respectively Section 2.3 and Section 2.4. The research questions regarding countermeasures against both the replay & jamming and relay attack are partially answered in Section 2.4. Next, we'll extend our knowledge of how the different car components communicate with each other through the Internal Automobile Network in Section 2.5 followed by methods for manipulating the internal communication in Section 2.6. We'll end with section 2.7 in which we'll dig into the art of reverse engineering and explain how it is used to gain knowledge about the protocols used in keyless entry systems.

Before trying to gain knowledge about various keyless entry protocols and implement our own attacks, we'll need an introduction to digital signal processing. In Chapter 3 we'll be going over various modulation (Section 3.1) and data encoding (Section 3.2) techniques. In Section 3.3 we'll dig deeper in the theoretical part of digital signal processing followed by an introduction of various hardware and software components used in digital signal processing.

In Chapter 4 we'll implement and perform a replay & jamming attack against RKE systems. We'll implement 2 different methods: One using solely software-defined radios (Section 4.1) and one cost-efficient method in which we'll try to reduce the cost by replacing the transmitting software-defined radio (Section 4.2). During testing of our implementation we found a vulnerability in the RKE systems of a few cars which is explained in Section 4.3. In Section 4.4 we'll try to improve our implementation by using a technique called reactive jamming. In this chapter we'll try to find answers for our first set of research question.

In Chapter 5 we'll implement and perform a relay attack against PKE(S) systems. We'll start in Section 5.1 by discussing various methods for relaying data and comparing them to each other to find the most efficient method. In Section 5.2 we'll discuss the required hardware and software followed by an explanation of the implementation in Section 5.3. Last, we'll evaluate our implementation and discuss possible improvements in Section 5.4. In this chapter we'll try to find answers for our second set of research question.

In Chapter 6 we'll try to obtain knowledge (physical and MAC layer) of the underlying RKE protocols implemented by car manufacturers. In Section 6.1 we'll explain the general process of reverse engineering RKE protocols and use this process to reverse engineer 3 case studies found in Section 6.2, Section 6.3 and Section 6.4. Besides reverse engineering different case studies, we'll also write automated real-time demodulators for each case study. In this chapter we'll try to find answers for our third set of research question.

In Chapter 7 we'll conclude the thesis by summarising the answers of the research questions provided in this section. We'll also write a general conclusion of the current state of the automobile industry regarding their keyless entry systems and what we can expect in the next years.

Chapter 2

Background

2.1 Keyless entry systems

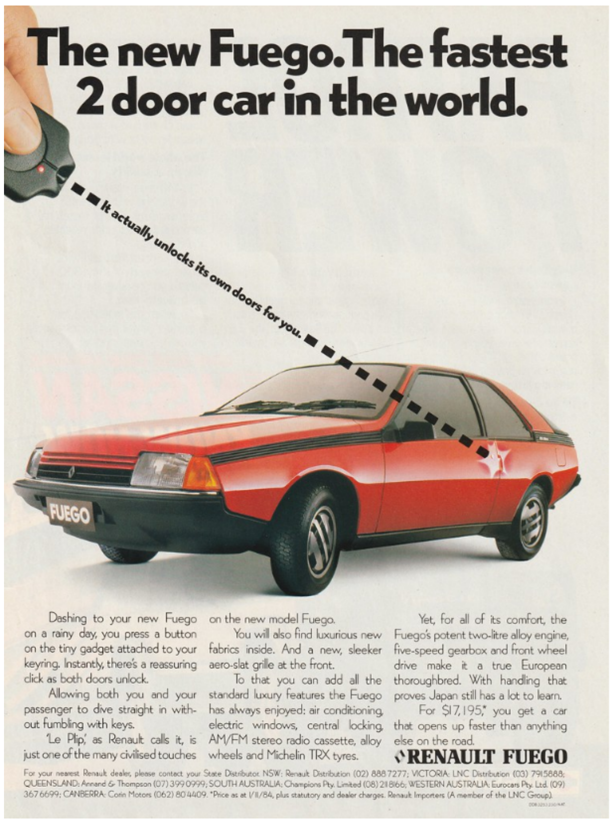
In the past, a mechanical key was required to lock or unlock the doors of a car. Nowadays, this is no longer the case due to the invention of the keyless entry systems. A keyless entry system replaces the mechanical lock with an electronic lock that is controlled remotely. This provides convenience for the owner as there is no longer any physical contact required with the car. On today's market there are two main types of keyless entry systems:

- The traditional **remote keyless entry** (RKE) in which the door will (un)lock upon pressing a button on a handheld device, called a key fob.
- A more modern and newer method called **passive keyless entry** (PKE) in which the receiver automatically unlocks the door when detecting the key fob within a limited range from the car, typically within 1 meter.

2.1.1 Remote keyless entry

The most common method of gaining access to a car is by pressing a button on the car's key fob. The RKE system using a handheld transmitter was first introduced by Renault in 1982 [56]. Renault used an in 1981 patented technology by Paul Lipschutz known as the "plip" or "plipper". A "coded pulse signal generator and battery-powered infra-red radiation emitter" was used along with an infra-red receiver. Renault soon started their advertising campaign advertising the smart key innovation on their new 1982 Renault Fuego (see Figure 2.1). The innovation was a huge success and drivers liked the convenience of the new RKE technology. However, the new introduced technology did not work as convenient as we know it today. As the keyless entry system was based on the infra-red technology, it requires a line-of-sight communication between transmitter and receiver (think of a typical television remote). Drivers owning the Renault Fuego called the inconvenience the "plip dance" as the driver had to "shuffle" around the car to try and correctly aim the transmitter at the receiver.

RKE systems we know today are no longer based on the infra-red technology but instead based on radio frequency communication. A RKE system consists of a short-range radio transmitter that produces a radio frequency (RF) signal used to communicate with the car. Pressing a button on the key fob wakes up a microcontroller inside which computes and sends an encoded binary data stream to the RF transmitter. The RF transmitter modulates the data and transmits it as a RF signal. When in range of the car, typically 10 to 100 meters, the RF signal is received by the receiver of the car. It demodulates the RF signal back to binary data and forwards it to the microcontroller which decodes the data and executes the desired command [34] (see Figure 2.2). The key fob is powered by a small coin cell battery while the receiver is powered by the car's


The new Fuego. The fastest 2 door car in the world.
It actually unlocks its own doors for you.
FUEGO

Dashing to your new Fuego on a rainy day, you press a button on the tiny gadget attached to your keyring. Instantly, there's a reassuring click as both doors unlock.

Allowing both you and your passenger to dive straight in without fumbling with keys.

'Le Plip,' as Renault calls it, is just one of the many civilised touches on the new model Fuego.

You will also find luxurious new fabrics inside. And a new, sleeker aero-slat grille at the front.

To that you can add all the standard luxury features the Fuego has always enjoyed: air conditioning, electric windows, central locking, AM/FM stereo radio cassette, alloy wheels and Michelin TRX tyres.

Yet, for all of its comfort, the Fuego's potent two-litre alloy engine, five-speed gearbox and front wheel drive make it a true European thoroughbred. With handling that proves Japan still has a lot to learn.

For $17,195,* you get a car that opens up faster than anything else on the road.

RENAULT FUEGO

For your nearest Renault dealer, please contact your State Distributor. NSW: Renault Distribution (02) 888 7277; VICTORIA: LNC Distribution (03) 791 5888; QUEENSLAND: Annand & Thompson (07) 399 0999; SOUTH AUSTRALIA: Champions Pty. Limited (08) 21 8166; WESTERN AUSTRALIA: Eurocars Pty. Ltd. (09) 367 6699; CANBERRA: Corin Motors (062) 80 4409. *Price as at 1/11/84, plus statutory and dealer charges. Renault Importers (A member of the LNC Group).

Figure 2.1: Advertisement for the Renault Fuego, the first ever model using keyless entry technology. [48]

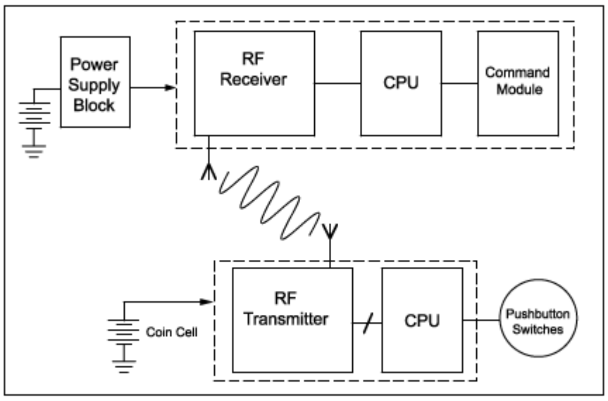


Figure 2.2: Block diagram illustrating the structure of a RKE system. [34]

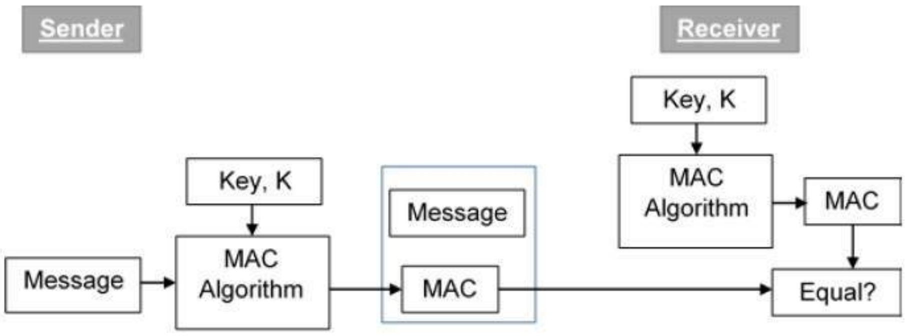


Figure 2.3: The process of generating a MAC. [66]

battery pack. For European car models the RKE system operates at a frequency of 433.92 MHz. North-American and Japanese car models operate at 315 MHz.

Needless to say it is important that a key fob only communicates with one specific car as otherwise anyone can (un)lock anyone's car. This is achieved by sending an identifier that is synchronised between the transmitter (key fob) and receiver (car). When the receiver's unit captures a signal with an unknown identifier, it ignores the signal. When the receiver's unit captures a signal with his specific synchronised identifier, it will execute the received command. The synchronised identifier is used in combination with a technique called rolling code or hopping code. In this thesis we'll be using rolling code to refer to this technique. In short, rolling code ensures that each transmitted signal by the key fob is unique and only valid once. This means that from the moment a car receives a valid key fob signal, it will no longer accept a key fob signal with the same rolling code. This prevents criminals from fraudulently gaining access to a victim's car by capturing a key fob's signal and retransmitting it. However, recently an attack emerged in which an adversary can prevent a signal from getting accepted by the car and at the same time capture and save the signal. As the car did not correctly receive the key fob signal, the rolling code of the captured signal is still valid. The adversary can later transmit the signal and unlock the victim's car. This specific attack is explained in a later stage of the thesis 2.2 together with an implementation of the attack 4. A more detailed explanation on how the rolling code technique works is explained in Section 2.2. Also in that section, a concrete example of a RKE protocol using rolling code is illustrated and explained.

Additionally, a RKE system typically provides data integrity to make sure the data did not change in transmit (=error detection). Data integrity is provided using a checksum. A checksum algorithm takes the actual transmitted data as input and generates a sequence of numbers and letters of a fixed length. There exist many different types of algorithms like a simple XOR, modulo or the two's complement algorithm. However, checksums can also be based on cryptographic hash function (SHA-1, SHA-256, MD5, ...), fingerprints or Cyclic Redundancy Checks (CRC). Upon receiving

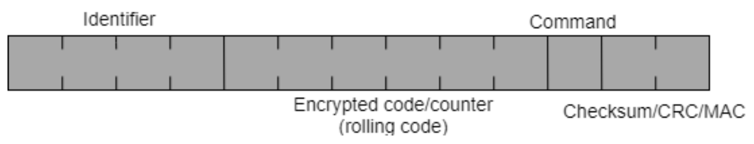


Figure 2.4: A typical message format in RKE protocols.

a message, the receiver calculates the checksum (using the same algorithm as the transmitter) and compares it with the received checksum. If both the values are identical, the receiver knows the data is correctly received and did not change during transit. When the values are different, it means that an error was detected and the original transmitted message is different from the received message. The primary disadvantage of a checksum is that is does not protect against intentional modification of the data. In other words, a checksum does not provide any message authenticity. An intruder can intercept a message, change the data and calculate the new checksum. The only requirement is that the intruder knows the algorithm used to calculate the checksum. However, when having access to the data by which the checksum is calculated, it is not that hard to derive the used algorithm (f.e. by bruteforce). For this reason, some implementation use a Message Authentication Code (MAC) instead of a checksum. Besides data integrity, a MAC also provides authentication of the message and therefore makes sure a message is not modified by a third party. A MAC algorithm generates a short tag based on the message (or part of the message) and a secret key [4] (see Figure 2.3). An example of a MAC algorithm is Keyed-hash Message Authentication Code (HMAC). HMAC creates a hash of the message and the secret key (the secret key is not used to encrypt the message or MAC). Because the adversary does not have access to the secret key, he is unable to calculate a correct MAC and therefore unable to correctly modify a message. The typical message format transmitted by a key fob is illustrated in Figure 2.4. However, each car manufacturer implement their own message format which is typically different for each specific model.

Typically, key fobs are also equipped with an immobiliser, known as the transponder key [21]. An immobiliser is an electronic security component that prevents the car from starting its engine when the original key fob is not detected. This means that when an adversary is able to bypass the lock (by f.e. physically breaking in, copying the key fob or re-using a transmitted signal), he is not able to start the engine even if he has an exact copy of the mechanical key. An immobiliser comes in the form of a RFID chip that is detected when the key is inside the ignition. Only the paired RFID tag is authorised to start the engine of the car. In RKE systems, the immobiliser is typically in passive mode. This means that even when the battery of the key fob is dead, the car will still be able to detect the RFID tag embedded inside the key fob. This is done via inductive coupling: The car creates an electromagnetic field that powers the RFID tag and wakes up the microcontroller inside the key fob. The microcontroller will compute the suited response and transmits it using only energy from the RFID reader. The response is verified by the car and when valid it knows the authorised key is used to start the engine. Since the key fob uses the energy of the car created by the electromagnetic field, it requires the key to be in close proximity of the car (f.e. the ignition). This method is also used in RKE systems equipped with a push to start button. By placing the key close to the designated area, the RFID tag can be read by the car.

2.1.2 Passive keyless entry

A car equipped with a PKE system allows the owner to gain access to their car without having to press a button on their key fob. A PKE system consists of two main components: the key fob and the base station. The key fob is located inside the pocket or purse of the authorised user while the base station is mounted inside a car. When the base station detects the authorised key fob in close proximity, by broadcasting a signal and waiting for a response of the key fob, it automatically unlocks the door. In addition to (un)locking a car without the need of pressing a button, most PKE systems also replace the traditional mechanical key to start the engine. Systems containing this feature are called Passive Keyless Entry and Start (PKES) systems. When the key fob is detected inside the car, the user is able to start (or stop) the engine without having to insert a mechanical key into the car's ignition. PKE(S) systems are no longer restricted to high-end luxury cars but are available as an optional feature in multiple standard car models of multiple different car manufacturers like Citroën [14], Hyundai [31], Ford [20], ...

PKE(S) systems work slightly different from RKE systems. In RKE systems the microcontroller and RF transmitter only consumes battery power upon pressing a button on the key fob. This is not the case in PKE(S) systems as the key fob is constantly listening for signals from the car's base station. This introduces a problem regarding the battery lifespan of the key fob. To minimise the power consumption of the key fob, PKE(S) systems introduces a technique in which the key fob is in a low-power listening mode until getting "woken up" by the car's base station. This "wake-up" message indicates that the base station will soon send a signal that requires computational power (and battery power) from the key fob. This technique works as follows: Upon touching the car's door handle, the car's base station broadcasts a low frequency (LF) "wake-up" message (125 kHz) which, as the name suggests, wakes up a microcontroller inside a key fob [21]. The "wake-up" message can be seen as a short, fixed and predefined data pattern, called a preamble. When the key fob detects the preamble, the microcontroller inside the key fob is "woken up" and sends an UHF (433 or 315 MHz) acknowledgement back to the car. In some PKE(S) systems, the car's base station generates a magnetic field which provides energy to the key fob (identical to the immobiliser). This energy is than used to wake up the microcontroller.

Once the base station received the acknowledgement from the key fob, a Challenge-response protocol is used to authenticate the key fob. In a Challenge-response protocol one party presents the other party with a question (=challenge) after which the receiver computes an answer (=response) to the question to prove its identity. The calculation of the response is based on a secret key that is known to the two communicating parties. Only the entity knowing the secret key can calculate the correct answer to the challenge. By issuing a different challenge for each button press, an adversary can't simply capture the response to the challenge and retransmit it as each different challenge expects a difference response. In PKE systems, the base station transmits a LF challenge presenting the key fob with a random number. The key fob encrypts the random number with its shared secret key and transmits is back to the car's base station. The base station encrypts the same random number with the shared secret key and compares the result with the response received from the key fob. Since the key fob and base station share the same secret key, the encrypted challenge should be identical. When the encrypted challenge is indeed the same, the key fob is authorised and the car unlocks the doors. To increase the security level some car manufactures implement a bidirectional authentication in which the authentication is performed in both directions. Namely, the key fob authenticates the car and the car authenticates the key fob. Two different PKE(S) system protocols are shown in Figure 2.5: (a) illustrates the method explained in this paragraph while (b) illustrated a method in which the challenge is transmitted together with the "wake-up" message. Additionally, Section 2.3.3 explains the concrete PKES protocol used by Tesla.

Besides the optimisation to maximise the battery lifespan, the base station has to track down if a key fob is located inside or outside the car. If the key fob is located inside of the car, the car will enable the push to start button to start the engine. The position detection is accomplished by the Received Signal Strength Indication (RSSI) measurement of the signals transmitted by the LF antennas. The RSSI is calculated by the receiver and indicates the power or strength of the received signal. Therefore, a higher RSSI value indicates a stronger signal. The power of a signal corresponds to the distance to the source. The RSSI value is higher when close to the source and lower when further away from the source of the transmitted signal. In PKE(S) systems, LF antennas are strategically distributed inside and outside of the car [40]. The LF antennas generate a magnetic field covering both the car's interior as well as the vicinity in which a key fob can be detected. The key fob measures the signal strength and calculates the RSSI value of the LF signal during the communication with the car's base station and transmits it back to the car. The resulting RSSI value can then be used to calculate the distance of the key fob to the transmitter (=car).

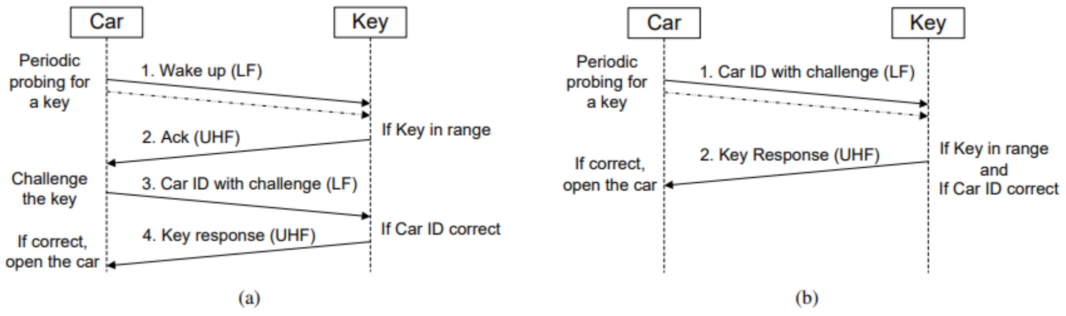


Figure 2.5: Two examples of PKE(S) system protocols. a) The car wakes up the key fob with a LF signal and receives a RF/UHF acknowledge response back from the key fob. The car sends a LF challenge with car ID. Key fob computes a response and sends a RF/UHF response back. b) Wake-up, car ID and challenge is send right away in each periodic probe. [21]

2.2 Rolling code security

The very first key fobs in the early nineties made use of a static code system. While the identifier transmitted by the key fob was unique for every car, the rest of the code was static. This means that when pressing the same button, an identical signal is transmitted. When using a static code system, it is easy to perform a replay attack. An adversary can simply capture the transmitted signal from a victim's key fob and unlock the doors by retransmitting the signal. To prevent these attacks from happening, car manufactures implement an extra security measurement called rolling code. Rolling code is a system whereby each transmitted signal is unique and only this specific unique message is accepted by the receiver. This means that signals transmitted by the key fob are only valid once, therefore previous transmitted signals can not be captured and retransmitted to (un)lock the doors of the car.

There are multiple different implementations of a rolling code system depending on the manufacturers. In Subsection 2.2.1, a generic implementation of such rolling code system is explained which means that it might vary depending on different manufacturers, but generally employ the same techniques in a similar way. In Subsection 2.2.2, a concrete implementation of a rolling system used by a vast amount of car manufacturers is explained.

2.2.1 Generic implementation

Typically, rolling code systems consist of a synchronised counter in both the transmitter (key fob) and receiver (car). Each time a button is pressed on the key fob or a valid message is captured by the car's receiver the synchronised counter is incremented. When transmitting a signal, the synchronised counter of the key fob is included inside the message. Upon receiving a signal, the synchronised counter is extracted from the message and compared to the counter of the receiver. Only when both the counters are equal, the car accepts and performs the captured command. This method of using a synchronised counter ensures that a transmitted message is only valid once because the synchronised counter of a retransmitted message differs from the synchronised counter of the receiver and therefore makes the message invalid. However, when someone accidentally presses a button on their key fob while out of range of the car, the key fob counter will increment. Since the button is pressed out of range of the car, and therefore not received, the counter of the car does not increment. Because the key fob and car no longer have the same counter, the car should ignores new signals received from the key fob. To solve this problem, a typical implementation will compare the next x (typically 128 or 256) counters with the received counter. When receiving a counter within this range, the car accepts the message and sets its counter equal to the received counter. When someone desynchronises or loses their key fob, the desynchronised or new key fob has to be resynchronised with the car's receiver. Luckily, most car manufacturers provide a way for the user to resynchronise their key fob with the car themselves. This is a sequence of actions that has to be performed by the user to switch a car to its programming mode and resynchronise with the key fob. A typical procedure is explained below [16] (example for certain Ford models):

1. Place the mechanical key inside the ignition.
2. Turn the ignition key on and off eight times in a row in less then 10 seconds to switch the car's system to programming mode. When successfully performed, the car should notify the user by beeping or locking the doors to notify the user it switched to programming mode.
3. Press a random button on the key fob(s) that has to be synchronised with the car (typically allows at most four key fobs).
4. Switch of the ignition.

Using a rolling code system based solely on a synchronised counter is not secure. An adversary can easily capture a message and extract the counter. Since it is an incremental counter, it is not hard to predict the next valid counter, namely the increment of the counter (or + a larger value as long as it is within the valid range of the car's synchronised counter). Therefore it is necessary to encrypt the synchronised counter so adversaries are unable to correctly extract the counter from a captured message. Using encryption, a secret key is shared between key fob and receiver. Only the receiver possessing the secret key can correctly decrypt the encrypted data. As the adversary has no access to this secret key (in case of secure cryptographic scheme), he is unable to correctly decrypt the captured message and therefore unable to extract the synchronised counter. Even if he knows the synchronised counter but does not know the secret key, he is unable to correctly encrypt the data which leads to the receiver being unable to correctly decrypt the message.

Some implementations might replace the incremental synchronised counter by a pseudo-random sequence of generated numbers [9]. A pseudo-random number generator (PRNG) generates a sequence of random numbers depending on the initial seed. If two PRNGs are initialised with the same seed, the PRNG will produce the same sequence of numbers. By using a shared secret seed between the car and key fob, the generated numbers are identical. As for the incremental counter, the car compares the received value with a certain amount of next pseudo-random generated numbers to avoid desynchronisation by pressing a button out of range of the car. It is important to initialise the PRNG with a secure seed. A commonly used technique for generating an initial seed is using the state of the computer system (such as time). However, generating an initial seed using only time is unsafe. Suppose an adversary knows at around which time the seed was initialised, he can brute force the initial seed. For example when someone knows the exact date a car left the manufacturer, he has a rough approximation of when the initial seed was generated (assuming the initial seed was generated using the current time). Therefore, high entropy (=randomness) is required to generate a secure seed. High entropy is often obtained from physical properties like noise signals, thermal noise, clock drift, ... However, even a PRNG initialised with a high entropy seed is not always cryptographically secure. A PRNG might be able to fool simple statistical tests in thinking the outputs are completely random. However, When someone knows the previous calculated random output values, it might be possible to find a correlation between the output values and therefore predict the next "random" value. Ideally a cryptographically secure pseudo-random number generator (CSPRNG) is used. a CSPRNG uses a high entropy seed and should make it impossible for an adversary to predict next outputs even if the previous generated outputs and used algorithm is known.

Figure 2.6 illustrates how a message is created by a key fob upon pressing a button. A shared secret key is used by the encryption algorithm to encrypt the synchronisation counter. The size of the encrypted result depends on the used encryption algorithm. In this case a block cipher with a block length of 32 bits was used. The 32-bit encrypted message is transmitted together with the button information and the unique identifier or serial number. Figure 2.7 illustrates the operations of the generic receiver upon receiving a message. First the serial number is checked to ensure that the received message is transmitted by the paired transmitter. When the received serial number and the receiver's stored serial number are equal, the encrypted data is decrypted using the shared

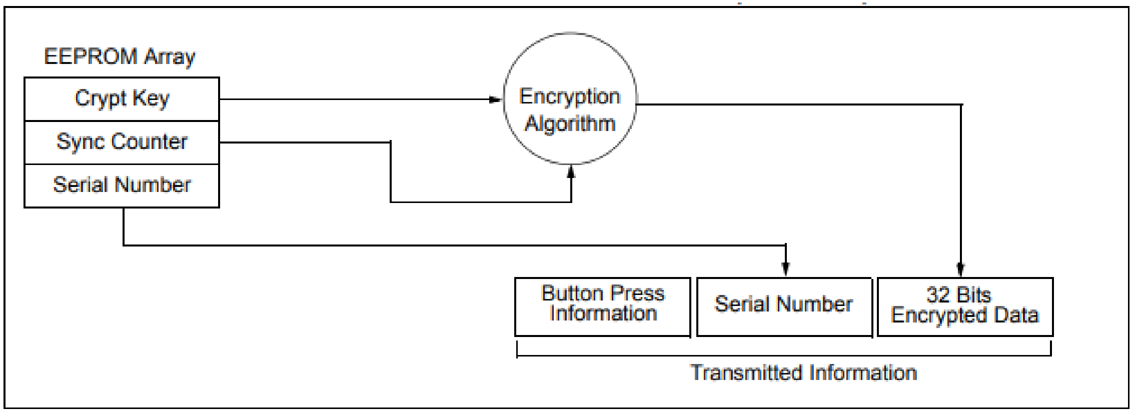


Figure 2.6: Creation of a transmitted message. [45]

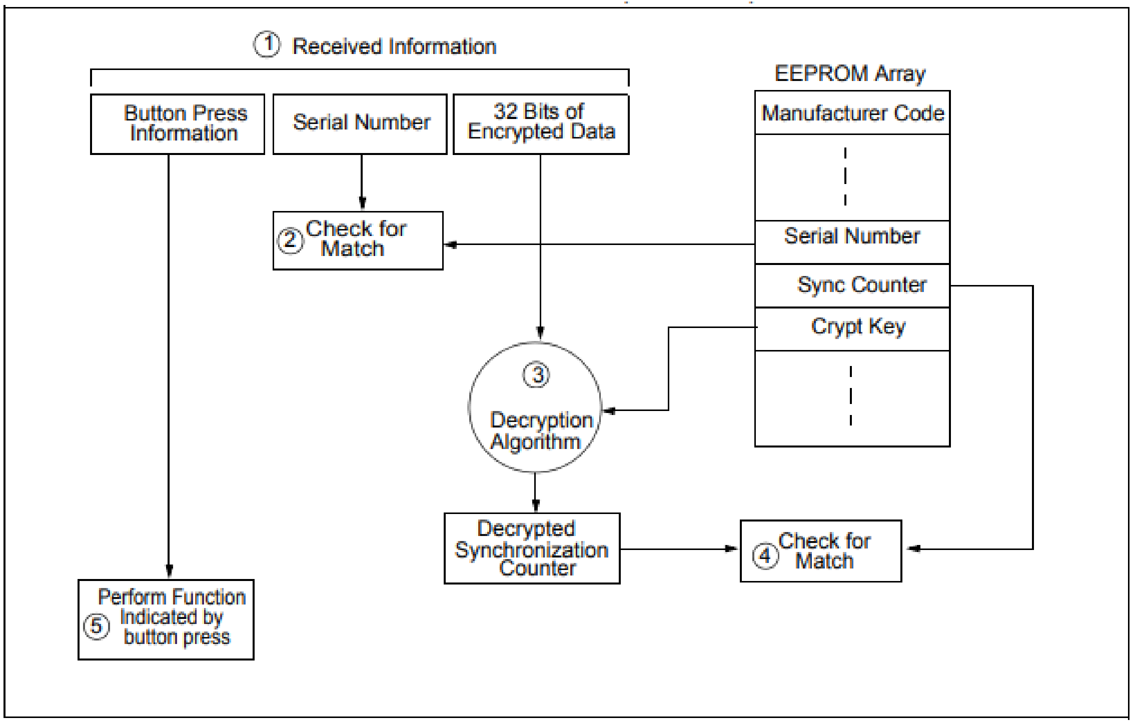


Figure 2.7: Basic operation of the receiver (decoder). [45]

Figure 2.8: Microchip HCS201 encoder utilising KeeLoq code hopping technology.

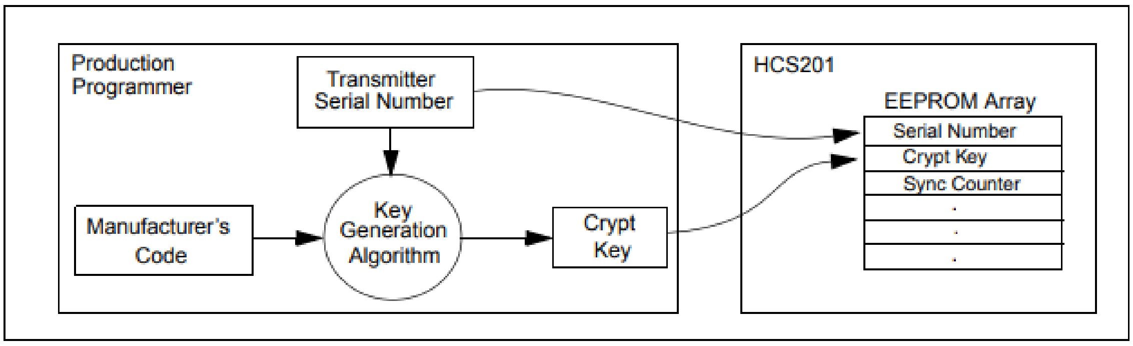


Figure 2.9: Creation and storage of secret key during production. [45]

secret key. The resulting decrypted synchronisation counter is then compared to the receiver's synchronisation counter.

In principle, such rolling code systems should provide a suitable security level for access control. However, rolling code systems are based on cryptographic schemes. Unfortunately, not all cryptographic schemes are secure, causing researchers and criminals to break the rolling code and clone a key fob. Besides breaking these cryptographic systems, there are other methods that don't break the rolling code cryptographic scheme but instead find a way to work around it. Various attacks on both remote and passive keyless entry and their rolling code systems are explained in Section 2.3.

2.2.2 KeeLoq by Microchip Technology

KeeLoq by Microchip Technology is a hardware-dedicated block cipher used in the generation of rolling code in keyless entry systems by various car manufacturers like Fiat, Honda, Jaguar, Toyota, Volvo, Volkswagen, ... Microchip Technology currently offers 13 different encoder chips (installed inside of a key fob) containing different additional features like low battery indicator, LED drive, programmable time-outs, ... They also offer 3 decoder chips (installed inside receiver). In this section the operations of the KeeLoq Code Hopping Encoder HCS201 will be explained in more detail (Figure 2.8). The HCS201 encoder utilises the KeeLoq code hopping technology, a block cipher based on a block length of 32 bits and a key length of 64 bits [45]. It is a cost-effective solution for producing rolling codes as it only costs €0,66 per unit.

Before use, a shared secret 64-bit cryptographic key used by the KeeLoq encryption and decryption algorithm has to be initialised. The secret key is generated by a specific 64-bit manufacturer's code and the transmitter serial number (unique for every encoder). The resulting secret key is stored in the EEPROM array of the encoder chip together with the transmitter serial number and synchronisation counter (Figure 2.9). Besides the generation of the 64-bit secret key, the receiver has to be synchronised with the transmitter before use. Synchronising includes calculation of the secret key and storing the transmitter serial number and synchronisation counter.

The message format created by the HCS201 consists of a 32 bits encrypted part and a 34 bits

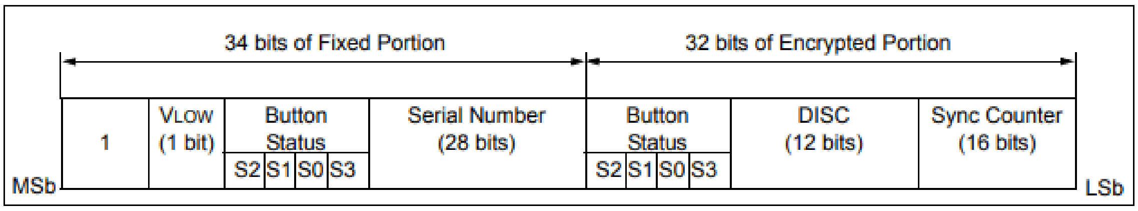


Figure 2.10: KeeLoq HCS201 generated message format. [45]

fixed code part. The encrypted data is generated from 4 button bits, 12 discrimination bits and the 16-bit synchronisation counter. The discrimination value helps the post-decryption validation check on the receivers side. Typically, the discrimination value is the 12 least significant bits of the serial number. When the encrypted part of the message is decrypted, the discrimination value is compared to the receiver's serial number to verify if the decryption process was valid. The fixed code data is made up by 2 status bits used to indicate if the battery is low, 4 button bits indicating which button was pressed and the 28-bit unique identifier or serial number (Figure 2.10). The combination of the encrypted data together with the fixed code data results in $7.38 * 10^{19}$ unique code combinations.

2.3 Exploitation of keyless entry systems

For a long time, users of keyless entry systems trusted in the security and liability of these systems. Unfortunately, most keyless entry systems have weaknesses that can be exploited by criminals to gain unauthorised access to others cars. These attacks can range from low-level cryptographic attacks to high-level attacks that find ways to bypass rolling code. In this section we'll explain different attacks on both RKE and PKE(S) systems.

2.3.1 Replay & jamming attack

In a replay attack, the transmitted signal by the authorised key fob is captured by an adversary and later retransmitted to fraudulently gain access to the victim's car. As noted before, car manufacturers use rolling code to make sure a signal transmitted by a key fob is unique and only accepted once by the car's receiver. This means that a retransmitted signal should be ignored in case it was already accepted previously, preventing criminals from performing such replay attack. For many years it seemed that secure rolling code systems solved the replay vulnerability until in 2014 Spencer Whyte developed a proof of concept attack called "Jam Intercept and Replay Attack" that was able to unlock virtually every car and garage door equipped with rolling code security [71].

The key idea in this attack is blocking the car's reception (=jamming) by transmitting random noise or carrier signals to "confuse" the receiver. If successful, the receiver is unable to correctly distinct the original transmitted signal from the jamming signal. Alternatively, a targeted carrier signal can be transmitted that causes bits in the transmitted message to change. For example changing bits in the data integrity field (checksum, MAC, ...) will cause the signal to be rejected by the receiver. Jamming is known concept for performing diverse attacks, for example, interrupting Wi-Fi or sensor network communication. In theory, all RF communication is prone to jamming. There exist different methods of jamming [73]:

- **Constant or continuous jamming**: In constant or continuous jamming the adversary continually transmits a jamming signal. In this method, the adversary needs little to no knowledge of the target communication except from the transmit frequency. A disadvantage of continually jamming is that it is easily detectable. It also causes all communication on the specific targeted operating frequency to be jammed which is not always preferred.

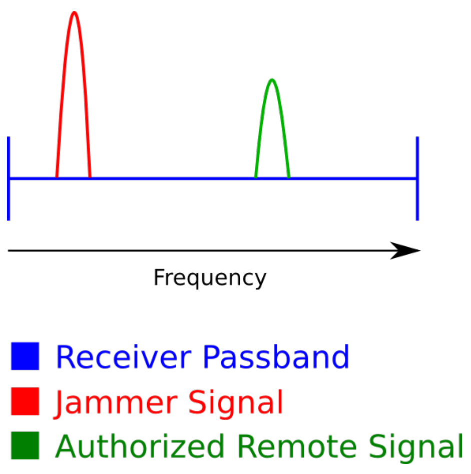


Figure 2.11: The car's frequency passband when jamming. [70]

- **Deceptive jamming**: In deceptive jamming legitimate packets are transmitted to prevent the targeted packet from getting received. An important application of deceptive jamming is found in the military context. For example interrupting the communication between aircraft and control post by providing the target with false information. A different case of deceptive jamming is in devices that have a separated receive and send mode. By constantly transmitting valid packets to the target, the device will stay in receive mode for as long as it is receiving packets from the adversary. This causes the target device to never switch to send mode and transmit any packets.

- **Random jamming**: Random jamming can behave like either a constant jammer or deceptive jammer. However, a random jammer will alternate between a jamming and "sleeping" mode in which no packets/signals are transmitted. This method is mainly used for energy conservation, especially for jammers without access to an unlimited power supply.

- **Reactive jamming**: In reactive jamming the target receiver is blocked for only a short amount of time (milliseconds) to prevent a specific signal that is already "on the air" from getting accepted. Alternatively, the adversary can transmit short bursts of data to make the signal that is already "on the air" invalid (for example by changing 0's to 1's) In contrast to the other methods, reactive jamming can be used to jam a specific target without interrupting different communication on the same channel. It is also very hard to detect a reactive jammer as it is only transmitting for a few milliseconds at a time. Later in this thesis we'll further touch upon the topic of reactive jamming (see Section 4.4).

For the following explanation, let's assume we are using a constant or continuous jamming to block the car's reception. In a replay & jamming attack, a small part of the frequency passband of the car's receiver is jammed while simultaneously recording the signal transmitted by the key fob. The frequency passband is a range delimited by a lower and upper frequency containing the range of frequencies the device can receive. The frequency passband of a car's receiver is typically larger than the operating frequency of the key fob. By jamming a part of the frequency passband close to the operating frequency of the key fob, the car's receiver is effectively jammed and unable to correctly receive the signal of a key fob (see Figure 2.11). By configuring a smaller frequency passband that does not include the jamming frequency on the adversary receiver, he can correctly capture the key fob signal without interference of the jamming signal. By doing this, the adversary now has a valid signal from the authorised key fob that is not yet received by the car. He can now transmit the signal using his own transmitter and unlock the victim's doors.

Most cars will notify the user if receiving a signal from the key fob by for example flashing it lights, closing or opening the mirrors, making a beeping sound, ... This means that the victim will most likely notice that the car did not correctly receive the key fob signal making the attack by Spencer Whyte unlikely to succeed in a realistic car theft. However, a year later Samy Kamkar,

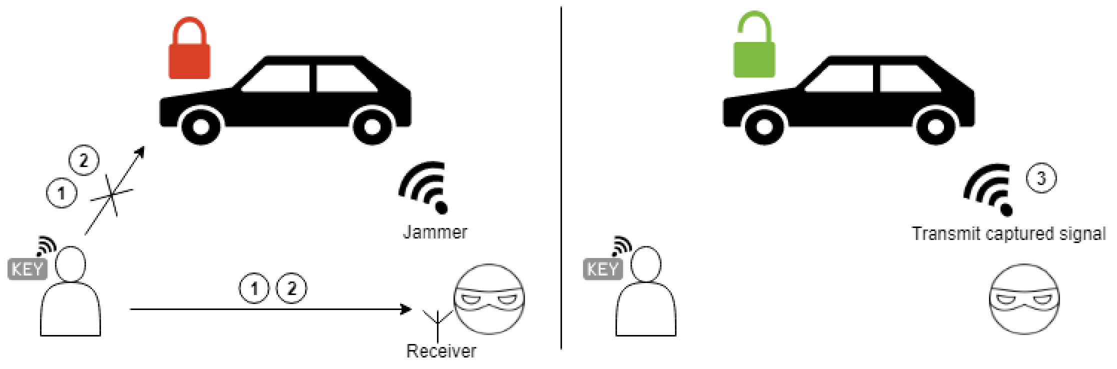


Figure 2.12: Diagram illustrating the replay & jamming attack.

a well known privacy and security researcher, created a $32 device that was based upon Spencer Whyte proof of concept but solved the inconveniences of his attack. Samy Kamkar came up with a scenario in which the victim won't notice an adversary is blocking the car's reception [38]. The scenario is illustrated in Figure 2.12 and goes as follows:

1. The adversary starts jamming the car's receiver while listening for the key fob signal outside of the jamming frequency. The victim presses the unlock button on their key fob and notices the doors did not unlock. The adversary at the same time recorded the signal and now has its first valid signal.
2. The natural reason of the victim would be pressing the unlock button a second time since the first try failed. The adversary keeps jamming and now records the second valid signal.
3. Immediately after receiving the second signal, the adversary stops jamming and transmits the first captured key fob signal resulting in successfully unlocking the car. The victim now thinks the second transmitted signal was correctly received by the car as it successfully unlocked the doors. However, in reality the car did not unlock because of the second transmitted signal but rather by the retransmitted first signal captured by the adversary. The adversary now has valid key fob signal that is not yet accepted by the car.

Samy Kamkar's explanation of the attack ends here. However, when performing the attack on the unlock signal, the victim will drive off and arrive at the destination. He will lock the car causing the rolling code to shift which makes the captured unlock signal invalid again. To prevent the rolling code from shifting, the adversary has several options:

- **Keep jamming after obtaining the second key fob signal**: To prevent the rolling code from shifting, the adversary has to keep jamming the victim's car even after obtaining the second key fob signal. There are a few possible options:
    1. The adversary can follow the victim to their destination and start jamming when the victim arrives forcing him to use the mechanical key to lock the car. This prevents the rolling code from shifting and therefore the adversary still has a valid unlock signal that he can use to gain access to the victim's car.
    2. Following the victim to their destination might be suspicious. Alternatively, the adversary can work with a companion. If the adversary knows where the victim is heading, the companion can go to the destination before the victim arrives and start jamming.
- **Modifying the lock signal into an unlock signal**: A more difficult way of performing the attack is to capture the lock signal instead of the unlock signal. As the victim parks and locks the car, the adversary no longer has to keep jamming to prevent the rolling code from shifting. However, the adversary can't unlock the car as he only captured the lock signal. However, most key fob signals have the data field separated from the rolling code field. Theoretically, if the adversary knows the next rolling code, he can change the lock command bits to the unlock command bits resulting in a valid unlock signal. This method is rather

complex since the adversary needs to capture, demodulate, decode and analyse the signal. Once he analysed the signal and determined which specific bits are used for the command, he has to modify the bits, encode and modulate the data back to the original format and transmit it. Additionally, most key fobs manufacturers implement some sort of data integrity used to validate that a signal is correctly received and/or did not get modified by a third-party. Modifying the signal will cause the data stream to have an invalid checksum or MAC and therefore get rejected by the car. Modifying the checksum is possible if the adversary can derive the used algorithm from the data. However, when the manufacturer uses a MAC instead of a simple checksum, the adversary is unable to change the MAC as he has no access to the secret key used to generate the MAC.

- **Placing battery powered device underneath car**: An alternative way of performing this replay & jamming attack is to attach a device underneath the car. This device continuously jams the car while recording the key fob signals. This way the adversary does not have to execute the attack himself. After placing the device on a parked car, he can come back a few days later to retrieve the device with multiple valid lock and unlock signals stored on it.

2.3.2 Signal amplification relay attack

A signal amplification relay attack is a form of man-in-the-middle attack where the communication range between transmitter and receiver is extended by amplification of the signal. Where the replay attack in previous section is more relevant to RKE systems, this attack is especially interesting for PKE(S) systems. As noted before, in PKE(S) systems a door is unlocked when the authorised key fob is detected in close proximity to the car. A signal amplification relay attack extends this limited range to, in theory, an unlimited range allowing the adversary to unlock and start the victim's car without having to physically carry the authorised key fob [21].

The attack is performed by two adversaries: The first adversary stands in close proximity to the car while the other stands in close proximity to the key fob, illustrated in Figure 2.13. The first adversary P1 triggers the LF "wake-up" message by pulling or touching the door handle of the car. P1 captures the message using his own receiver and sends it to his companion P2 who is standing in close proximity to the key fob. For example, this can be outside of the victim's house in case the key fob is located close to the outside walls or at a restaurant next to the victim. P2 carries a device that receives the signal transmitted by P1 and transmits it to the key fob. If the key fob detects the "wake-up" message, it wakes up the microcontroller, demodulates, decodes and interprets the message. The key fob computes a response to the message and transmits it over an UHF channel (typically 433.92 MHz). Since the key fob replies on an UHF channel (like a standard RKE key fob) instead of a LF channel, the signal typically reaches the car without the use of any relay. In case the UHF signal does not reach the car, the adversaries can opt for a second relay between P2 and P1. The UHF signal is received (either with or without relay) and verified by the car. In case of a valid response, the car will unlock its doors thinking the key fob is in close proximity. In case of a PKES system this routine is executed twice to start the car. The adversaries can now drive away in the victim's car. As the adversaries drive away, most cars will give a warning as they no longer detects the key fob. For safety reasons a car's engine won't just shut off while driving. The thieves can now drive the car until it is manually turned off by the adversary himself.

A proof of concept of such relay attack can already be achieved using 2 simple coils of wire. In a Belgian reportage, the researcher was able to perform a relay attack by connecting an end of each coil to each other [67]. One coil is placed close to the car's transmitter while to other coil is placed close to the key fob. The electromagnetic radiation transmitted by the car is captured by the first coil and travels through the wire to the second coil. The second coil outputs this electromagnetic radiation which is detected by the key fob. The key fob computes its response and transmit it to unlock the car.

Relay attacks go beyond PKE(S) systems and can also be performed in almost all Near Field Communication (NFC) systems for example contactless payment using NFC [42] [22]. An example is the Ping.Ping payment system used in Belgian universities (including University of Hasselt)

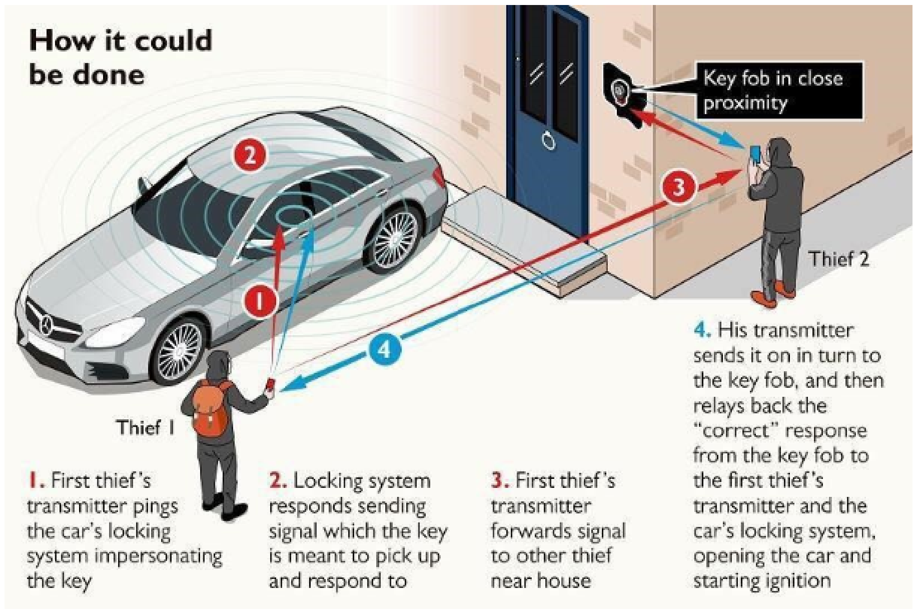

How it could be done
Key fob in close proximity
Thief 2
Thief 1
1. First thief's transmitter pings the car's locking system impersonating the key
2. Locking system responds sending signal which the key is meant to pick up and respond to
3. First thief's transmitter forwards signal to other thief near 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 2.13: Diagram of a signal amplification relay attack on a PKE(S) system. [55]

and businesses in which a NFC card has to be placed in close proximity to a NFC reader. The adversary holds a receiver close to the NFC reader that receives the LF signal transmitted by the NFC reader. He relays the signal to a second device that is in close proximity to the NFC card of the victim (f.e. the customer behind him in line). The device transmits the signal from the NFC reader and captures the response of the NFC card. This response is captured by the device and relayed back to the device located by the NFC reader. The NFC reader detects the NFC card of the victim as it would be physically close to the reader. The purchases of the adversary are now payed by the victim's card.

The signal amplification relay attack is one of the most used attacks in modern car thefts. Devices that can perform these relay attacks can be purchased online by anyone. However, due to the recent rise of these relay attacks together with the intense media exposure, researchers are fully engaged in finding solution to prevent relay attacks. One solution is using distance bounding mechanisms in which the time delay between sending a challenge and receiving the response is used to determine the distance between the car and key fob. Distance bounding mechanisms are explained in Section 2.4.3. However, a fully secure solution to prevent relay attacks need time to implement, test and deploy. Additionally, existing cars equipped with a PKE(S) system require an update which is not always trivial. Therefore we can conclude that relay attacks will probably remain a big problem for the next decade.

2.3.3 Time-Memory Trade-Off attack

A Time-Memory Trade-Off (TMTO) attack is an attack which improves running time by using more space (memory), commonly used in cryptographic attacks [60]. There are three parameters: Space (=available data storage), time and data (=data available to the adversary during the attack). As the name states it is a trade-off in which the adversary reduces one or two parameters in favour of the others. A TMTO attack consists of two main phases:

1. *Preprocessing phase*: This phase can be seen as the preliminary work. An adversary explores the cryptosystem by for example giving a random input and storing the output in a large table or data structure. By analysis of results, different input and output values can be grouped together. For example when multiple different input values result in the same output value, these input values can be grouped together. This phase typically takes a long time since the

adversary has to try and compute all possible values. The resulting data structure is later used to look up results by inputting real-time data and can be up to a few terabytes.

2. *Real-time phase*: In this phase an adversary is granted actual data from the device he is attacking which he uses to look up specific useful data in his preprocessed data structure created in the *preprocessing phase*. For example by looking up the output value in the data structure, a set of input values resulting in the same output can be found. Due to the large amounts of preprocessed data, the "real-time phase" typically takes only a short amount of time.

This attack was recently used by Wouters et al. in 2018 [43] to expose an insecure and outdated cryptographic algorithm used for authentication in the keyless entry system of Tesla. Tesla uses a PKES system similarly to the system explained in section 2.1.2. The Tesla PKES protocol is illustrated in Figure 2.14. The car periodically advertises a "wake-up" message including its 2-byte unique identifier. Once the key receives the "wake-up" message, it will transmit a response back to car, signaling it is ready to receive the authentication challenge (1) (Challenge-response protocol). The car verifies the response and transmits a 5-byte challenge to the key fob. The key fob uses a DST40 compression function (more information follows in the next paragraph) to produce a 24-bit response that is transmitted over an UHF channel (2). The car verifies the response and, if correct, unlocks the doors. Next, the car and key fob engage in a keep-alive protocol to check whether or not the key is still present in the car (3). This is accomplished by periodically advertising a "wake-up" message and waiting for a valid response. To start the car, the driver presses down on the brake pedal which initiates a second round of the Challenge-response protocol which uses the same DST40 technique as described above with a different challenge (4). However, the used Challenge-response protocol introduces some issues: Due to the lack of mutual authentication, anyone who knows the car's identifier, can get a response from a key fob. Because the identifier is periodically advertised, therefore broadcasted by the car itself, the identifier is publicly available and can be captured by anyone.

Besides the lack of mutual authentication, the system uses an insecure and outdated cryptography cipher called DST40. DST40 transforms a 40-bit challenge into a 24-bit response using a 40-bit secret key. Because the 24-bit response is smaller than the 40-bit challenge input, there will be multiple secret keys that produce the same response to a given challenge. This insecurity can be used in the preprocessing phase of the TMTO attack: Take a fixed 40-bit challenge and compute the DST40 response for every possible value of the 40-bit secret key. All 40-bit keys that compute the same 24-bit response are grouped in a single file and stored in a data structure. The lookup table consisting of 2^{24} files, each containing roughly $2^{1}6$ keys. (see Figure 2.15). The data structure is computed two times using two different fixed 40-bit challenge. The second challenge-response pair is required to determine the correct key out of the 2^{16} potential keys obtained from the first challenge-response pair. Once we have a complete data structures for all possible values of the 40-bit key, the preprocessing phase is complete.

Using the two data structures, we can now perform the interesting part of the attack and try and unlock a Tesla without carrying the authorised key fob. The goal is to track down the shared 40-bit secret key that is used by a specific key fob to compute the response on a given challenge. Since the car's identifier is publicly advertised we can impersonate the car and transmit any chosen challenge to a key fob and observe the response. We start with obtaining the identifier of the victim's car. This is done by capturing the broadcasted "wake-up" message and extracting the public identifier. Next, we impersonate the car using the captured identifier and transmit the fixed 40-bit challenge used to create the preprocessed data structure to the key fob. We observe the response of the key fob and can now recover the corresponding file containing 2^{16} candidate keys. By sending a second different 40-bit challenge, the response is used to find the specific 40-bit secret key among the 2^{16} candidate keys. Once we obtained the 40-bit secret key, we can now impersonate the key fob and unlock and start the car.

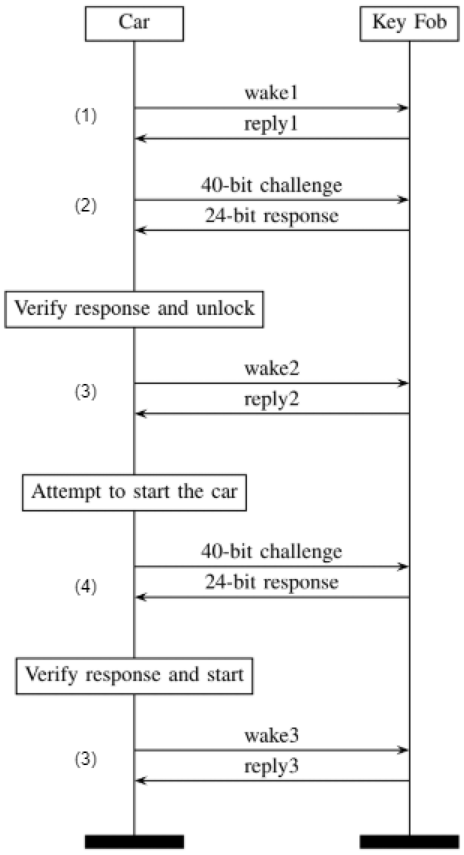


Figure 2.14: Tesla's PKES protocol used to unlock and start a car. [43]

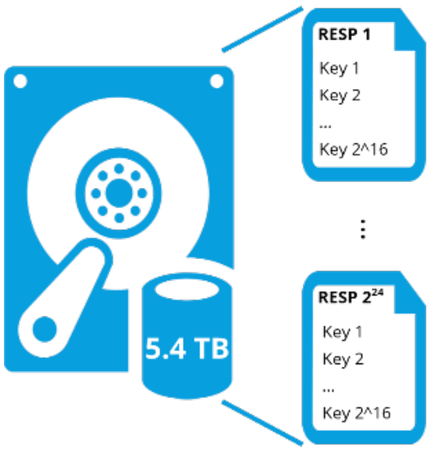


Figure 2.15: Data structure grouping the keys with the same response to one file. [43]

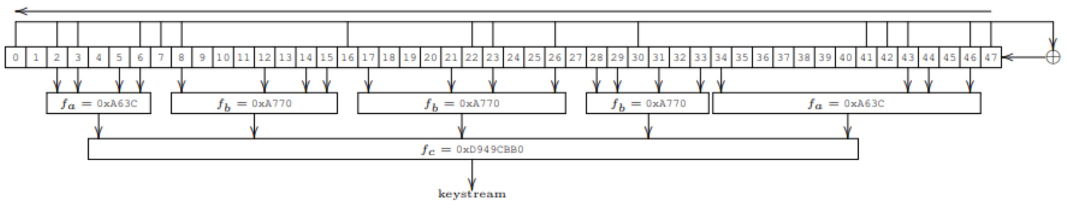


Figure 2.16: Hitag2 stream cipher structure consisting of a 48-bit LFSR and three non-linear filter functions f_a, f_b *and* f_c. [72]

2.3.4 Low-level cryptographic exploits against rolling code schemes

Since rolling code is generated using cryptographic schemes, it trusts on the reliability of these schemes. Since 2006, several keyless entry related algorithms have been exposed to cryptographic weaknesses resulting in breaking the rolling code. One such cryptographic breakable scheme is called Hitag2, a stream cipher used in producing rolling code. A so-called correlation-based attack performed by D.Garcia et al. in 2016 allows recovery of the secret key used to create the cipher. By obtaining the secret key they were able to clone the remote control with only four to eight captured rolling codes [24]. A correlation attack is used for breaking stream ciphers whose keystream is generated by combining the output of several binary linear feedback shift registers (LFSR) using a nonlinear boolean function [44]. These attacks are possible when there is a significant correlation between the output of the individual LFSRs in the keystream generator and the output of the nonlinear boolean function that combines the output of all LFSRs. The main idea of the attack is to limit the exhaustive or brute-force search, and therefore fasten the process by identifying good candidate keys with high correlation score. In Hitag2, the cipher consists of a 48-bit LFSR and a nonlinear filter function f (which consists of 3 different circuits f_a, f_b *and* f_c) (see Figure 2.16). For each clock cycle, 20 bits of the LFSR are put through the filter function f that generates 1 bit of the keystream. Next, the LFSR is shifted 1 bit to the left to introduce a new bit on the right. D.Garcia et al. successfully verified their findings in practice on car models from Alfa Romeo, Chevrolet, Citroen, Dacia, Fiat, Ford, Lancia, Mitsubishi, Nissan, Opel, Peugeot, Renault ranging from year 2004 to 2016.

In 2008 a paper was released that presented a practical key recovery attack against KeeLoq based on a slide attack and a novel approach to meet-in-the-middle attack [32]. KeeLoq is a block cipher alternative to Hitag2 used in producing rolling codes in RKE systems. A slide attack focuses on block ciphers consisting of multiple rounds of an identical key permutation. This attack deals with the idea that increasing the number of permutations increases the strength of the cipher. This results in the slide attack being independent of the amount of permutations of the cipher. A meet-in-the-middle space-time trade-off attack which tries to reduce the time complexity of an ordinary brute force attack in exchange of requiring memory storage (space). The attack requires the encryption scheme to rely on a sequence of encryptions (f.e. 3-DES) and requires the adversary to have access to both the plaintext and the ciphertext [41]. The adversary starts with calculating all possible intermediate ciphertexts of the known plaintext using all possible secret keys. Next, the adversary decrypts the known ciphertext with all possible secret keys resulting in the intermediate plaintexts. The adversary tries to find blocks in which the resulting ciphertexts (from encrypting the plaintext) and the resulting plaintexts (from decrypting the ciphertext) are identical (=meeting in the middle). The keys used to encrypt the plaintext and decrypt the ciphertext are most likely the encryption keys used for the block cipher. The attack requires 2^{16} plaintexts and has a time complexity of $2^{44.5}$ KeeLoq encryptions. The fully implemented attack requires 65 minutes to obtain the required data and 7.8 days of calculations on 64 CPU cores. However, an adversary can purchase a cluster of 50 dual core computers for €10.000 and find the secret key in about two days.

Keyless entry systems are extremely vulnerable to these cryptographic attacks. What makes these

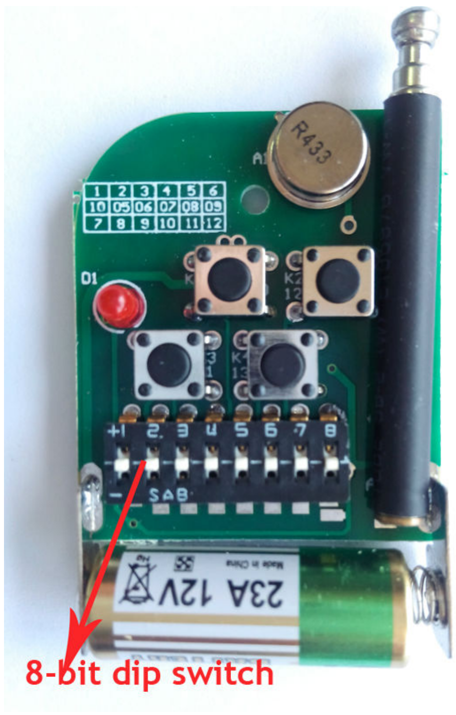


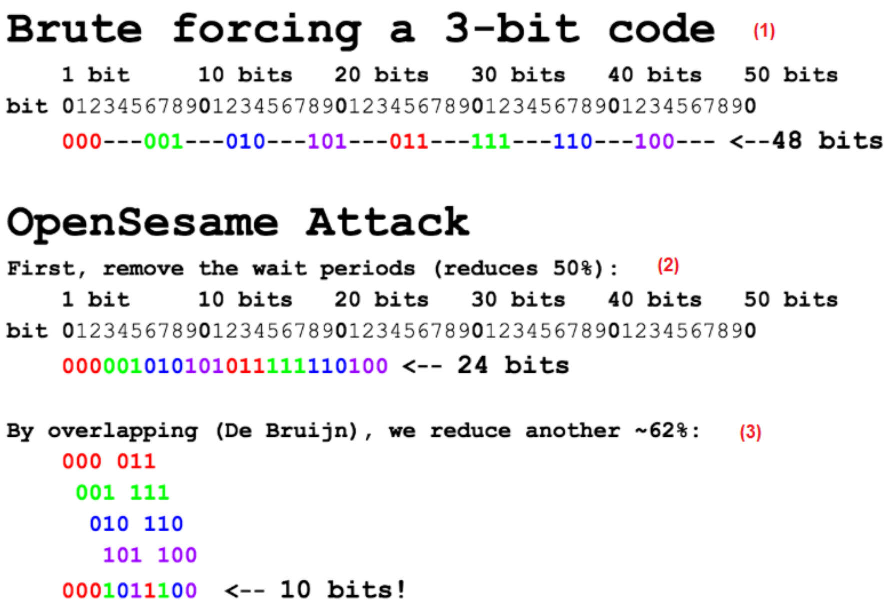
Figure 2.17: Dip switch remote control key fob.

attacks so efficient is the fact that breaking a single cryptographic scheme affects possible millions of cars of multiple different manufacturers as seen in the Hitag2 attack. Updating those millions of affected cars is a struggle (and most likely impossible) as the cryptographic schemes are integrated in physical hardware chips inside the key fob and car.

2.3.5 Intermezzo: OpenSesame remote control garage door brute-forcer

RKE systems are not limited to the automobile industry. The second most known application is remote control garage doors. Modern remote control garage doors utilise the same RKE technology as used in the automobile industry. However, the adaption from insecure static codes to a "secure" garage door system did not go as smoothly as in the automobile industry. On top of that, garage doors typically last multiple decades in contrast to cars. A precise number on how many percent of the garage doors still use static code is not available. However, because of the argument mentioned above, there are most likely still a lot of static code garage doors operating today.

The first remote controlled garage doors used static code which means the remote control of a garage door could open every other garage door from the same manufacturer. Soon arose the inevitable problem that a person's key fob could open the garage door of their neighbour. As this might be inconvenient, the main problem was people breaking into others homes by exploiting their static code garage doors. However, when the remote controlled garage doors became popular around the world, manufacturers changed to a different approach. Instead of a transmitting a static code, the garage door and key fob could now be synchronised to each other by matching so-called binary dip switches (see Figure 2.17). This binary dip switch is typically 8 to 12-bit long which means it provides respectively 256 (2^8) and 4096 (2^{12}) unique configurations. Let's assume we set a password that can contain upper- and lowercase letters, digits and 10 special characters. This means we can choose out of 72 different characters. By only choosing a 2-character password, we already have 5184 (72^2) different possibilities which is more than a 12-bit dip switch remote control. This means that a 2-character password is harder to brute-force than a 12-bit dip switch garage door system. As the dip switches solved the problem regarding neighbours controlling each others garage doors, it did not change the essence of the problem.



```
Brute forcing a 3-bit code  (1)
    1 bit     10 bits    20 bits    30 bits    40 bits    50 bits
bit 012345678901234567890123456789012345678901234567890
    000---001---010---101---011---111---110---100--- <--48 bits

OpenSesame Attack
First, remove the wait periods (reduces 50%):  (2)
    1 bit     10 bits    20 bits    30 bits    40 bits    50 bits
bit 012345678901234567890123456789012345678901234567890
    000001010101011111110100 <-- 24 bits

By overlapping (De Bruijn), we reduce another ~62%:  (3)
    000 011
     001 111
      010 110
       101 100
    0001011100  <-- 10 bits!
```

Figure 2.18: Overview of OpenSesame attack created by Samy Kamkar. [39]

In 2015, Samy Kamkar, created a device that could open almost every garage door using such 12-bit dip switches in less than 10 seconds [39]. To illustrate the attack, let's assume we are using a 3-bit dip switch remote control. A 3-bit dip switch has 8 unique codes. These 8 unique codes consists of 3 bits which mean 24 bits have to be transmitted to brute-force the garage door. Since the receiver does not know when a code is starting or ending, we have to introduce a wait period of 3-bit in order to let the receiver know the received code has ended. This means that instead of 24 bits, we now have to transmit 48 bits to brute-force the garage door (Figure 2.18 (1)). However, Kamkar found out that most garage doors use a bit shift register. This means that when receiving more than 3-bit, the most-significant bit is thrown away (=shifted) to make place for the next bit. This means that garage doors using such bit shift register do not need a wait period to distinguish different codes (Figure 2.18 (2)). On top of that, he used a De Bruijn sequence to significantly reduce the amount of required bits. De Bruijn found an algorithm to create an overlapping sequence of numbers that produces every possible permutation of these numbers. For the 3-bit dip switch, there exist a De Bruijn sequence of only 10 bits instead of 24 bits (Figure 2.18 (3)). Let's now assume we have a 12-bit dip switch garage door: There are 4096 12-bit unique possibilities with a 12-bit wait period between each code. This results in 98304 ($4096 * 12 * 2$) bits that has to be transmitted by the device to brute-force the garage door. Since most garage doors use a bit shift register, we can remove the wait period which results in "only" half the required bits. Finally, there exist a De Bruijn sequence of 4107 bits that represent the 4096 12-bit codes. This means that instead of transmitting 98304 bits, using the OpenSesame attack, it is possible to brute-force a 12-bit dip switch garage door by only transmitting 4107 bits.

2.4 Countermeasures

In this section, we'll discuss different types of countermeasures against different attacks on keyless entry systems. In the first 3 subsections, we'll be explaining countermeasures against attacks on PKE(S) system. we'll start with the immediate countermeasures which requires only simple actions from the user. In the second and third subsection, we'll discuss countermeasures which require software and/or hardware updates to the keyless entry system. In the final subsection we'll talk about a solution to prevent against replay & jamming attack on RKE systems

2.4.1 Immediate countermeasures

Signal blocking pouch (=Faraday pouch): The simplest method for protecting against attacks on PKE(S) systems is to block the communication to and from the key fob. This is achieved by putting the key fob inside of a signal blocking or Faraday pouch which prevent it from receiving and transmitting any signal. This prevents adversaries to perform a relay attack or any other attack in which communication with the key fob is required (like the Tesla TMTO-attack in Section 2.3.3). These cases are available on almost all popular online retail shops ranging from only €2 to €30. Another way of blocking a key fobs signal is by putting it inside of a metal tin box (f.e. a cookie box), wrap it in silver foil or put it inside the microwave or refrigerator. The metal contraption serves as a signal blocking case for the key fob and is recommended when the key fob is not used for example when at home.

Removing battery of key fob: A different countermeasure against attacks in which communication with the key fob is required is to fully disable the communication abilities of the key fob by removing the battery [21]. This causes the key fob to be deactivated and therefore forces the car owner to use the physical key (typically hidden inside the key fob). In PKES systems, this means the car owner is also unable to start the car. However, most manufactures provide a "dead battery" mode in which a car can still be started by placing the key fob very close to the predesignated location in the car (e.g. the start button). The car then communicates with the key fob's passive LF RFID tag (immobiliser) using short-range communication.

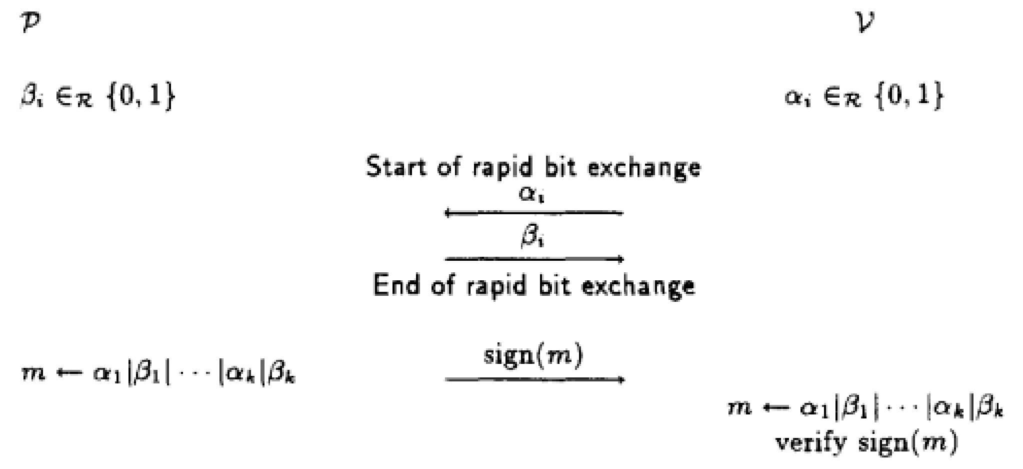
Disabling PKE: A self-evident countermeasure against relay attacks is disabling the PKE system of a car. In 2017 [26] and 2018 [49] (depending on the location), Tesla added the "Passive Entry" function in their latest software update following the reports of a successful relay attacks against Tesla. "Passive Entry" gives the owner the option to disable (or enable) the PKE system of the car. If the owner disables "Passive Entry", he can simply use his key fob as a traditional RKE key fob. However, not all cars equipped with PKE have the option to disable their PKE. Providing this option is up to the car manufacturer.

2.4.2 Motion sensor key fobs

One of the most recent countermeasures against relay attacks is the motion sensor key fob. A motion sensor key fob will go into sleep mode if there is no motion detected for a short period of time. Moving or picking up the key fob wakes it up and enables it to receive and transmit signals. As the vast majority of car thefts exploiting the passive keyless entry system takes place when the victim is not carrying their key fob, a motion sensor key fob seems to be valid countermeasure against relay attacks.

Thatcham Research, a company focused on car safety and security, recently revealed their latest car security ratings [51]: 10 out of 18 tested cars released in 2019 still failed to provide a valid solution for relay attacks (Volvo V60, Toyota RAV-4, Mazda 3, Citro"en DS3 Crossback, Ford Mondeo, Hyundai Nexo, Kia ProCeed, Lexus UX, Suzuki Jimny, Toyota Corolla Hybrid). As 8 of the 18 cars were no longer vulnerable to a relay attack (BMW 7 series, BMW X7, Porsche 911, Audi E-tron, Jaguar XE, Land Rover Evoque, Mercedes B-Class, Porsche Macan), it seems that the automobile industry is learning and trying to solve the concerned problem. As they did not reveal the exact countermeasure the car manufacturers implemented for all cars, they did however for the BMW 7 Series, BMW X7, Porsche 911. These 3 cars recently scored top marks on their security tests due to the fact they are using motion sensor key fobs. Also Ford announced that it will be using "theft-proof keyless fobs" equipped with a motion sensor later this year.

As motion sensor key fobs seems to prevent the vast majority of car thefts using a relay attack, it does not solve the essence of the problem. The problem occurs because passive keyless entry systems only use distance to verify the authenticity of a key fob. By utilising motion sensor key fobs, they provide a security measure on top of the underlying protocol while the protocol itself is still vulnerable to relay attacks. Criminals can still gain access to a victim's car while the key fob is not stationary (f.e. doing groceries, eating at restaurant, ...). In the next section, we'll



$\mathcal{P}$
$\mathcal{V}$
$\beta_i \in_{\mathcal{R}} \{0,1\}$
$\alpha_i \in_{\mathcal{R}} \{0,1\}$
Start of rapid bit exchange
α_i
β_i
End of rapid bit exchange
$m \leftarrow \alpha_1|\beta_1|\cdots|\alpha_k|\beta_k$
sign(m)
$m \leftarrow \alpha_1|\beta_1|\cdots|\alpha_k|\beta_k$
verify sign(m)

Figure 2.19: Distance-bounding protocol against mafia frauds. [10]

discuss distance-bounding protocols that provide a permanent solution for relay attacks against PKE systems.

2.4.3 Distance-bounding protocols

Distance-bounding is one of the most promising and effective countermeasures against relay attacks on PKE(S) systems. Distance-bounding protocols allow an upper-bound on the distance between the key fob and car by using the fact that signals travel at a fixed speed: The speed of light. By measuring the Round-Trip Time (RTT) between a verifier (=car) and prover (=key fob) and the assumption that signals travel at the speed of light, the distance between the two parties can be calculated. Implementing distance-bounding in physical hardware is not easy as the slightest processing delay will influence the calculated distance. A processing delay of just 1 nanosecond causes a distance error of 30 cm (speed of light travels 30 cm in 1 nanosecond).

The first distance-bounding protocol was introduced in 1994 by Brands and Chaum [10]. Brand and Chaum introduce 2 different type of protocols. The first protocol protects against so-called mafia fraud (a.k.a. relay attacks). The second protocol protects against distance fraud. In a distance fraud, the dishonest prover tries to mislead the verifier in thinking he is close in proximity by for example sending a response in advance. By combining both the protocols, they protect against both mafia and distance fraud. The first protocol works as follows (see Figure 2.19):

1. (**Initialization phase**) Verifier (V) generates uniformly at random k bits α_i, Prover (P) generates uniformly at random k bits β_i.
2. (**Rapid-bit or low-level distance-bounding exchange phase**) For i = 1, ..., k:
    - V sends bit α_i to P
    - P sends bit β_i to V (immediately after receiving α_i)
3. (**Signature phase**) When all bits are exchanged, P concatenates the 2k bits α_i and β_i and signs it with his secret key. The signature is send to V.
4. (**Verification phase**) V checks if the received signature from P is a correct signature (the message is signed with the corresponding secret key and the message of concatenated bits is correct). If it holds, V checks if the maximum delay times between sending a bit α_i and receiving a bit β_i from P is within an acceptable range (a.k.a in proximity of V).

However, this protocol does not protect against distance fraud as the bits send by P do not have to depend on the bits received from V. If P knows the rate at which V sends out bits, P can send out bits before actual receiving them from V. Because the RTT is fraudulently lowered, P can mislead V in thinking P is in close proximity. Brands and Chaum proposed two potential solutions: The first solution consists of V sending out bits with randomly chosen delay times. P is unable to predict the delay times and therefore is unable to send out bits in advanced (V will not accept bits before sending them out). The second solution proposed ensures that the bits transmitted by P depend on the received bits from V. One way of achieving this is by creating a concatenated bitstring $m_1|...|m_k$ of all bit values chosen by P (the m_i bits are random and independent from the received bits from V). Upon receiving a bit from V, P sends out the bit $\beta_i = \alpha_i \oplus m_i$ (XOR operation). When all bits are exchanged V verifies whether the bitstring $(\alpha_1 \oplus \beta_1|...|\alpha_k \oplus \beta_k$ equals the public bitstring of P. The concatenated bitstring $m_1|...|m_k$ has to be committed to V using a secure commitment scheme. A commitment scheme is a cryptographic scheme that allows P to send the bitstring to V while keeping it hidden from others (including V). When the exchange of bits is complete, P can reveal the values to V. This ensure that others cannot change the value of the bits before P reveals the values to V.

The combined protocols is shown in Figure 2.20 and goes as followed:

1. (**Initialization and commit phase**) V generates uniformly at random k bits α_i. P generates uniformly at random k bits m_i and commits the concatenated bitstring $m_1|...|m_k$ to V using a secure commitment scheme.
2. (**Rapid-bit or low-level distance-bounding exchange phase**) For i = 1, ..., k:
   - V sends bit α_i to P
   - P sends bit $\beta_i = \alpha_i \oplus m_i$ to V (immediately after receiving α_i)
3. (**Open commit and signature phase**) When all bits are exchanged, P opens the commitment by sending the appropriate information to V. P concatenates the 2k bits α_i and β_i and signs it with his secret key. The signature is send to V.
4. (**Verification phase**) V verifies whether the bitstring $(\alpha_1 \oplus \beta_1|...|\alpha_k \oplus \beta_k)$ equals those committed by P $(m_1|...|m_k)$. If this holds, V checks if the received signature from P is a correct signature. If this is satisfied, V checks if the maximum delay time is within an acceptable range (a.k.a in proximity of V).

Through the years more and more different distance-bounding protocols arose. Another popular distance-bounding protocol was proposed by Hanche and Kuhn [30] in 2005. This protocol is an optimised version of the protocol proposed by Brands and Chaum. The protocol does not require a final signature as the response bits in the rapid bit exchange are used as signature. The overall quantity of the messages is also decreased which improves time efficiency. The protocol also handles a certain amount of wrong response bits due to potential channel noise.

In 2016 A. Abidin et al. [2] proposed a quantum distance-bounding protocol. The advantage of quantum-based distance-bounding protocols is that unlike regular digital bits, quantum bits (=qubits) cannot be measured without modifying their state nor be decoded before fully receiving them. Due to the fact that adversaries do not have access to qubits, one can only guess the qubits that are being sent.

2.4.4 Time-based message protocol

RKE protocols do not provide any indication of the age of a message transmitted by the key fob. The receiver has no knowledge about when the received message was created and therefore can't know if a message came from the authorised key fob or was retransmitted by an adversary at a later moment in time. By implementing a time-based feature in the RKE protocol the receiver can reject messages that are out-dated and therefore prevent against replay & jamming attacks.

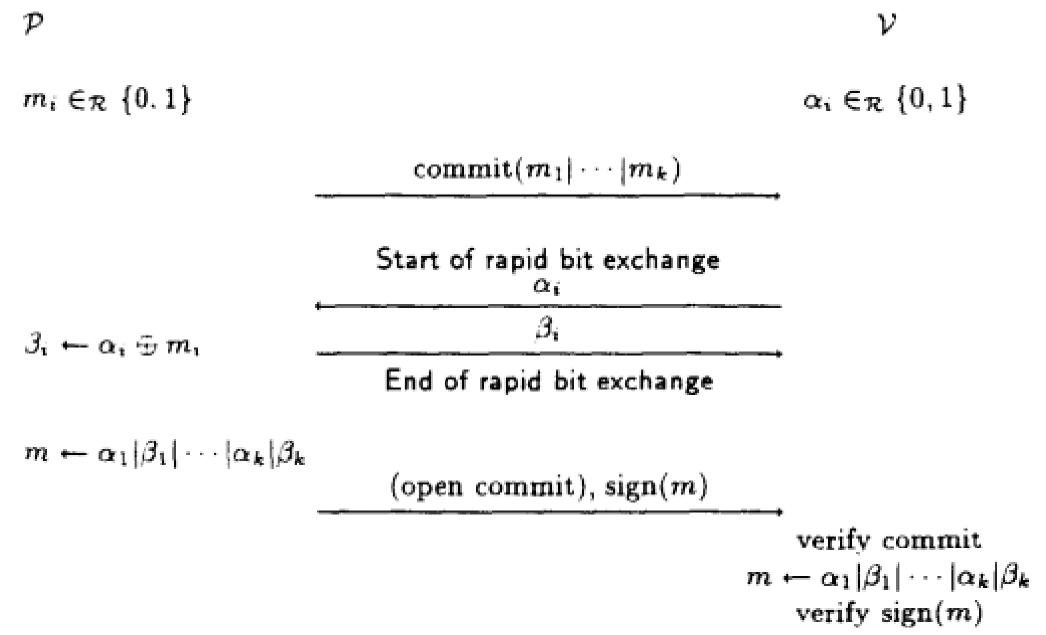

$\mathcal{P}$
$\mathcal{V}$
$m_i \in_{\mathcal{R}} \{0,1\}$
$\alpha_i \in_{\mathcal{R}} \{0,1\}$
$\text{commit}(m_1|\cdots|m_k)$
Start of rapid bit exchange
α_i
$\beta_i \leftarrow \alpha_i \oplus m_i$
β_i
End of rapid bit exchange
$m \leftarrow \alpha_1|\beta_1|\cdots|\alpha_k|\beta_k$
(open commit), $\text{sign}(m)$
verify commit
$m \leftarrow \alpha_1|\beta_1|\cdots|\alpha_k|\beta_k$
verify $\text{sign}(m)$

Figure 2.20: Combined distance-bounding protocols against mafia and distance fraud. [10]

As seen in Section 2.2.2, KeeLoq by Microchip Technology is one of the most popular hardware-dedicated block cipher used to generate rolling codes in RKE systems. Recently, Microchip Technology introduced their new “Ultimate KeeLoq” technology that provided time-driven messages as a counter-reaction against replay & jamming attacks [63]. In Ultimate KeeLoq, both the transmitter and receiver are equipped with a timer. This timer is either crystal-driven or a counter that counts the Watchdog Timer (WDT) periods. In a crystal-driven timer an electronic oscillator uses the mechanical resonance of a vibrating crystal to create an electrical signal with a precise frequency which provides a stable clock rate. A WDT is a timer that runs independent of the CPU and is used to trigger interrupts or resets the CPU. In case of the KeeLoq microchip, the WDT is mainly used to reduce power consumption. To reduce the power consumption the CPU will constantly switch to sleep mode. The WDT wakes up the CPU at a timed interval to let it perform some tasks before going back to sleep. The exact value of the time will differ between the transmitter and receiver because they started running on different times. However, both the timers will run at the same speed. When the receivers synchronises with the transmitter, he gets a snapshot of the time of the transmitter. By calculating the difference between its own timer value and the snapshot (=delta value) of the transmitter, he knows the different in time between its own timer and the transmitter’s timer. This calculated delta value is then stored in memory by the receiver.

Upon pressing a button on the key fob, the timer value of the transmitter is transmitted in the encrypted section of the message. Upon reception of the message, the receiver will extract the time-stamp and add the stored delta value. If the resulting value is inside the receiver’s acceptance window, he accepts the message. For every valid received signal, the receiver will update its delta value to stay synchronised with the transmitter (see Figure 2.21). However, it is impossible to synchronise the timers forever. Every slight tolerance of the timers will cause an error that accumulates over time. Therefore KeeLoq makes a distinction between short and long periods of time in which consecutive signals are received. In short periods of time, the acceptance window is very small as the timer errors are insignificant. However, in long periods of time (weeks or months) the errors accumulate and the time difference is significantly larger. Therefore the acceptance window is much larger than the acceptance window in short periods of time. The boundary between a short and long period is set by the system designer himself together with the

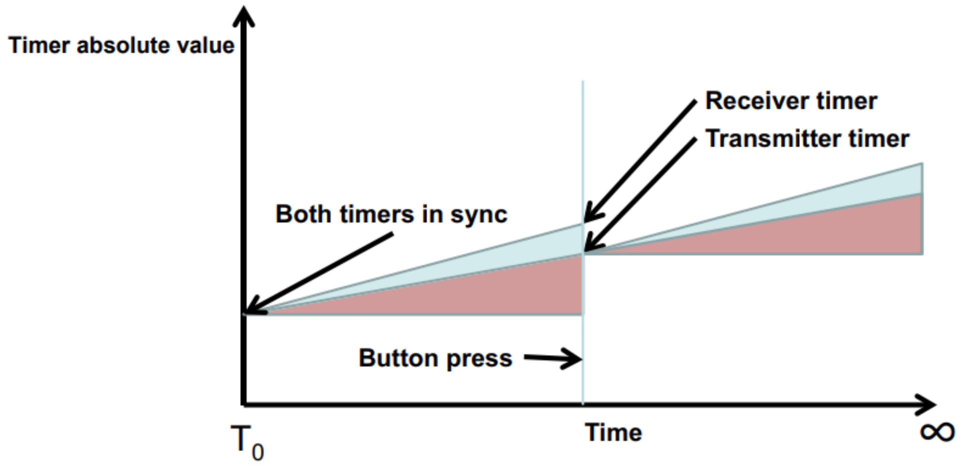


Figure 2.21: Graph illustrating the timer drift after synchronisation. [63]

maximum error limit.

2.5 Internal Automobile Network

2.5.1 Control Area Network

Nowadays, cars are no longer analog mechanical contraptions. This means that when stepping on the gas pedal, it is not directly controlling the throttle valve of the engine. In modern cars, a digital message is send to the engine module, telling it the driver wants to go faster. Every small to fundamental function in a car is computerised. Therefore modern cars can been seen as a complex network of computer systems. At the core of this network is the Control Area Network (CAN-bus), the central nervous system of a car which different functions can use to communicate. Each individual function is called an Electronic Control Unit (ECU) that can range from controlling the windows, doors or mirror adjustments to controlling the transmission, power steering or hand brakes [17]. The ECUs communicate with one another by broadcasting CAN packets on the CAN-bus. Since a CAN packet is broadcasted, all components on the bus receive the packet and decide whether it is intended for them or not.

The CAN-bus protocol was developed by Bosch in 1986 allowing ECUs to communicate with each other without complex dedicated wiring in between (see Figure 2.22). Besides avoiding complex dedicated wiring, the CAN-bus protocol has many other advantages [13]:

- **Low cost**: All ECUs communicate via the same CAN interface and are no longer sending direct analogue signals to each other. Since one-to-one connections are no longer required, it is a more cost efficient method for intercommunication between ECUs.
- **Centralised**: Since all communication is send over the same CAN-bus system, it allows for a central error diagnosis and configuration across all different ECUs
- **Robust**: The system is robust toward failure of individual ECUs and electromagnetic interference making it ideal for the use in cars.
- **Efficient**: Some CAN messages are more important than others, therefore they can be prioritised via IDs. The highest IDs are non-interrupted (f.e. the brake messages), while messages controlling the entertainment system are likely less important and therefore have a lower priority ID.
- **Flexible**: New ECUs can easily be added to the CAN-bus while existing ECUs can be modified.

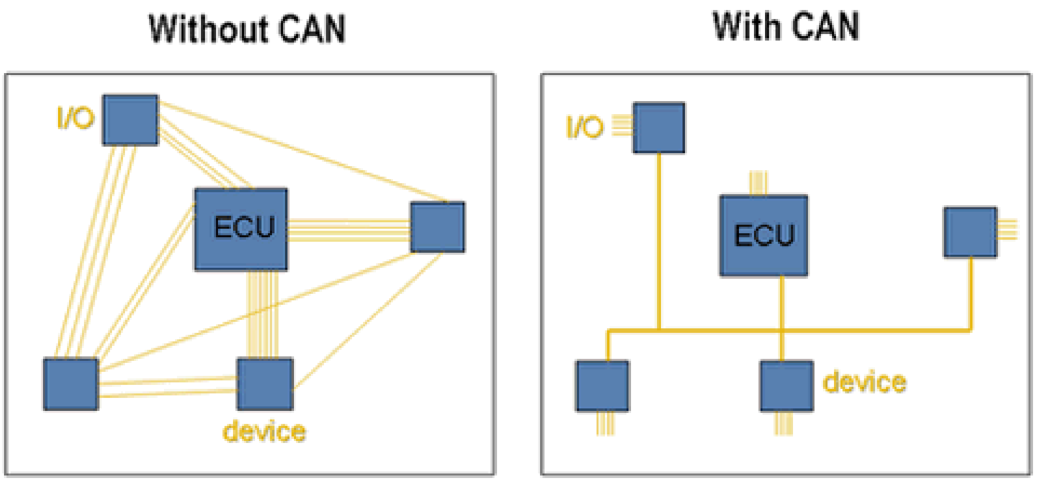


Figure 2.22: The internal network of a car with and without CAN-bus. [13]

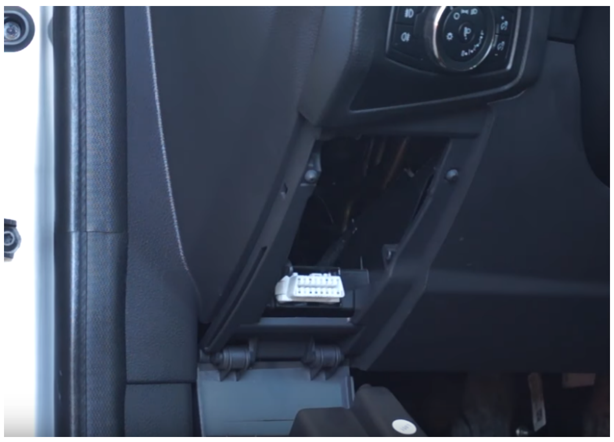


Figure 2.23: The OBD-II port in a Ford Focus.

2.5.2 On-board Diagnostics

Most cars have a direct interface to the CAN-bus called the Onboard Diagnostic (OBD-II) port [6]. In general, the OBD-II port is used for self-diagnostic and reporting of a car and gives the owner or technician access to the status of various subsystems. CAN-bus traffic can be read directly through the OBD-II port and enables the user to inject self-crafted CAN packets.

On most cars, the physical OBD-II port is located beneath the dashboard and can be access by a pre-built or self-made OBD-II interface adapter (see Figure 2.23.

2.6 Manipulation of the Internal Automobile Network

Since there is no source identifier or authentication built into the CAN packets, it is possible to both sniff the CAN network and inject self-crafted CAN packets. This enables an adversary to control a car through manipulation of the Internal Automobile Network [15]. There are different ways to gain access to a car's internal network or CAN-bus. The simplest one is through the physical OBD-II port as it is made to read CAN packets and inject diagnostic packets into the system. In 2013 Charlie Miller & Chris Valasek wrote a paper on how to manipulate and control a car through the physical OBD-II port [46]. They demonstrated this attack on a 2010 Ford Escape and 2010 Toyota Prius and were able to reverse engineer the CAN-bus communications of both cars and inject self-created custom CAN packets. The research duo used two types of CAN packets to manipulate the CAN-bus:

- **Normal CAN-bus packets**: Miller & Valasek were able to inject normal CAN-bus packets that controlled the speedometer and odometer of both cars which enabled them to display self-chosen values. It was possible to manipulate the on-board navigation but were unable to directly control the brakes of both cars. However, in the Toyota, they succeeded in manipulating the optional Pre-Collision System (PCS) used in Cruise Control mode. The PSC monitors the state of objects ahead of the car and will determine if a driver is going to collide with an object by analysing specific object distance data transmitted on the CAN-bus. By injecting CAN packets telling the PSC that an object is real close, they were able to slam the brakes of the car. They also found it is possible to overload the CAN network of the Ford causing a denial of service (DoS) on the CAN-bus. In this state, different ECUs act differently. For example, this causes the PSCM (Power Steering Control Module) ECU to completely shut down resulting in no longer providing steering assistance. The wheel becomes difficult to move and will not move further than 45% no matter how hard someone pulls the wheel which makes the driver unable to take sharp turns.

- **Diagnostic CAN-bus packets**: Diagnostic packets are packets transmitted by diagnostic tools used by mechanics to communicate with and interrogate an ECU. These packets can typically directly control the more critical ECUs of the car like the brakes or engine. Since it enables to control the critical ECUs, the user needs to authenticate. This is required so that not everyone can just use these diagnostic tools but only mechanics with a specific certificate. However, Miller & Valasek were able to extract the authentication keys by reverse engineering the diagnostic software tool itself. Once they were able to authenticate, they could disable the brakes and completely kill the engine while driving.

The OBD-II port is a physical port inside the car which makes the attack model rather unrealistic as the adversary has to be connected to the OBD-II port when performing an attack. However a wireless accessible OBD-II interface adapter can be connected to the port and later on be accessed remotely. For example an adversary can use a previously explained relay or replay attack to gain access to the car and connect his wireless OBD-II interface adapter. When succeeded the adversary can leave the car without the car owner noticing someone broke into their car while still having the OBD-II interface adapter connected to the port. An adversary also can break in physically by pick locking the door or smashing a window making the victim think it was just a regular car burglary. However, the adversary intentions are not to steal valuables but to connect the interface adapter to the car's OBD-II port. Since in most cars the port is located underneath the dashboard, the victim will unlikely notice the adapter.

Modern cars have a wide variety of on-board wireless features like Bluetooth, Wi-Fi, UConnect, DAB+, ... which makes it possible for someone to wireless connect to their car. Like other ECUs, these features are also connected directly to the CAN-bus to communicate with one another. Since these wireless features are connected to the CAN-bus, an adversary can expose potential vulnerabilities to manipulate the CAN-bus through such features. CAN-bus manipulation through OBD-II wasn't seen as a great danger to car owners because of the physical boundary (the adversary has to be connected to the OBD-II port or at one point has to break in the car to connect the wireless OBD-II interface adapter). However, by exposing these wireless features, it makes it possible for an adversary to remotely connect to a victim's internal automobile network and therefore remotely control a victim's car.

In 2015 Charlie Miller & Chris Valasek made headlines again after successfully remotely exploiting a 2014 Jeep Cherokee without the need for any physical access [47]. The duo used the on-board connectivity feature UConnect, a system that controls the car's infotainment, navigation, built-in apps, and cellular communications. Miller & Valasek found an open port on the UConnect micro-controller that allows any 3G device on the Sprint network (LTE Advanced) to communicate with any UConnect-enabled car. The pair flashed the CAN-connected microcontroller in the UConnect head unit with a new self-modified firmware version through the open port. The malicious firmware enabled them to send CAN messages to many different car components and control system. Now that they were able to communicate with the CAN-bus, they could manipulate it as like they were physically connected to the bus through the OBD-II port. During a press demonstration they were able to remotely control the air conditioning, radio, wiper fluid, and many more. More worryingly, they were able to disable the transmission and brakes of the car. The attack caused automobile manufacturers to recall 1.4 million cars for a security update.

2.7 Reverse engineering

Reverse engineering or back engineering is the process by which a created man-made object is taken apart to obtain knowledge from the object in order to duplicate, enhance or understand the object. The process of reverse engineering applies to multiple different topics [12]:

- **Software**: Reverse engineering software can be used to extract design and implementation information, find bugs, analyse the product's security, recover the working of the software in case the source code of the application has been lost, ...
- **Integrated circuits/hardware**: Reverse engineering an integrated circuit (or computer chip) can be used to obtain a schematic of the hardware, duplicate the hardware, understand the low-level working of the integrated circuit, ...
- **Military applications**: Reverse engineering for military purposes is mostly used in order to copy other nations technologies like weapons, rockets, planes, navigation systems, ... A different reason for reverse engineering military applications is discovering limitations and weaknesses of the technology which can be exploited.
- **Protocols**: The main objective of reverse engineering protocols is to get a better understanding of the protocol. An important sub-topic of reverse engineering protocols is the reverse engineering radio frequency signals. A radio frequency signal can be reverse engineered to identify protocol's message format, the source of the signal, duplicate the transmitter, discover utilised modulation and encoding techniques, ... Another application of reverse engineering protocols is seen in Section 2.3.4. In Section 2.3.4 we talked about breaking cryptographic schemes used for generating rolling codes. Before trying to find weaknesses in the schemes, they had to understand the protocol that is used by the particular cryptographic scheme. In this case reverse engineering is used as a preparatory step in breaking cryptographic schemes.

In this thesis we mainly focus on the reverse engineering of protocols, more specific, radio frequency signals transmitted by a car key fob. The main goal is to obtain the protocol's message format and extract the binary message from a captured radio frequency signal. Converting a radio frequency signal to its binary format requires knowledge of the utilised modulation and encoding technique. A common way of requiring this knowledge is by visually analysing a captured signal using dedicated software. More information about modulation (Section 3.1) and encoding (Section 3.2) techniques are explained in Chapter 3: Signal processing. Chapter 3 also touches upon the above mentioned dedicated software (Section 3.3.3).

Around a decade ago, capturing radio frequency signals was only possible through expensive special-purpose hardware that was only capable of capturing a limited bandwidth of the available spectrum. Because this hardware was so expensive, a common person interesting in signal processing could not afford this hardware. The release of cheap software-defined radios (SDR) caused a huge rise in the accessibility of knowledge regarding signal processing, more specifically reverse engineering radio frequency signal (more information regarding SDRs follows in Section 3.3.1). Besides a SDR being very cheap, it is also capable of capturing a wide range of frequencies (f.e. 30 MHz to 6 GHz). Reverse engineering radio frequency signals transmitted by a key fob is typically only found in the form of blog posts. An example is the blog post of Nihal Pasham who captured his car key fob with a cheap $40 SDR and manually analysed, demodulated and decoded it to find the corresponding binary message format [50]. A second example is from Bastian

Blössl who wrote an automated script that demodulates and decodes a radio frequency signal in real-time [8].

In Chapter 6, a process for reverse engineering radio frequency signals is explained. The chapter explains the complete process from start to end including: Capturing the radio frequency signal, discovering the modulation and encoding technique, obtaining the binary format of the radio frequency message and discovering the corresponding message format of the protocol. Thereafter we'll apply this method to multiple case studies in order to obtain information about their RKE protocol. In addition, automated demodulators and decoders were written for each specific key fob that converts the radio frequency signal to its binary format instantly upon receiving it.

Chapter 3

Signal processing

In the following chapter we'll give an introduction in the topic of digital signal processing. We'll start with explaining different modulation techniques (Section 3.1) and data encoding methods (Section 3.2) related to keyless entry protocols. In Section 3.3 we'll briefly touch upon the theoretical part of digital signal processing followed by hardware, software and useful frameworks related to digital signal processing.

3.1 Modulation

Modulation is the process of varying one or more properties of a carrier signal like amplitude, frequency or phase, with a modulating or baseband signal that typically contains information to be transmitted [62] (see Figure 3.1). The periodic waveform or carrier signal is typically generated using a radio frequency oscillator by converting a direct current (DC) supply voltage to an alternating output signal (AC). The carrier signal and modulating or baseband signal is combined into a new modulated signal and transmitted via an antenna or any other electrical cable that transmits radio frequency signals (f.e. co-axial cable). When transmitting the modulated signal via an antenna, the signal typically first passes through a radio frequency power amplifier to boost the power of the signal. There are two main types of modulation: Analog modulation and digital modulation.

3.1.1 Analog modulation

In analog modulation an analog baseband signal (f.e. voice or video) is transferred over a higher frequency signal such as a radio frequency band [59]. The most popular analog modulation techniques are frequency modulation (FM) and amplitude modulation (AM) used in radio broadcast, both FM and AM transmit information in the form of electromagnetic waves. As the name states, AM works by varying the amplitude of the carrier while the frequency remains constant. In FM, data is transmitted through the changes in frequency while the amplitude is kept constant. A comparison between AM and FM is given in Table 3.1 [61].

3.1.2 Digital modulation

Digital modulation is used to transfer digital data (f.e. bits) over an analog communication channel (f.e radio frequency band). For example, a telephone line is designed to transfer analog electrical signals. Computers, however, communicate with each other using binary data which can't be send directly over such telephone line. Therefore, the digital binary data has to be converted to analog signals. A typical modem (abbreviation for modulator-demodulator) is used to modulate digital

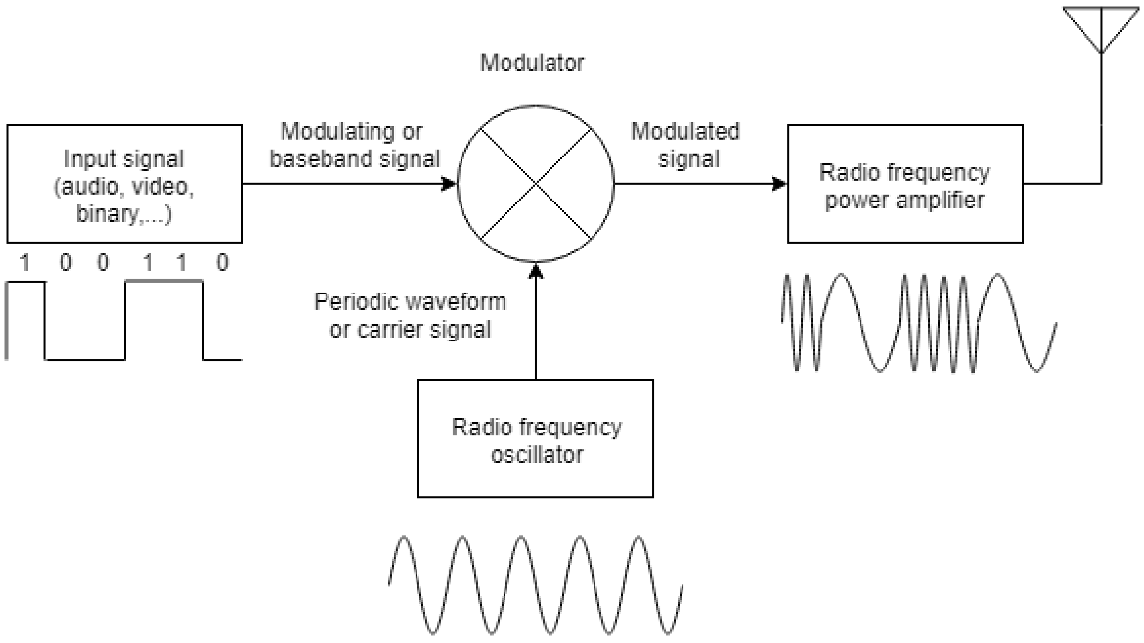

Modulator
Input signal (audio, video, binary,...)
1 0 0 1 1 0
Modulating or baseband signal
Modulated signal
Radio frequency power amplifier
Periodic waveform or carrier signal
Radio frequency oscillator

Figure 3.1: Modulation process.

	Amplitude Modulation	**Frequency Modulation**
Frequency allocation[a]	535 - 1705 KHz	88 - 108 MHz
Pros	Simple Cheap Long distance Easy to detect	Better (audio) quality Less prone to noise
Cons	Poor (audio) quality Prone to noise	Complex Expensive Limited distance Prone to physical objects

[a]The transmission of AM and FM signals is (physically) not limited to these frequency ranges. However, due to the huge demand on the frequency spectrum for various services the available frequency spectrum had to be regulated. Therefore, each specific application or service has a specific allocated range in the frequency spectrum. The regulation is controlled by various governmental and international organizations.

Table 3.1: An amplitude vs frequency modulation comparison table.

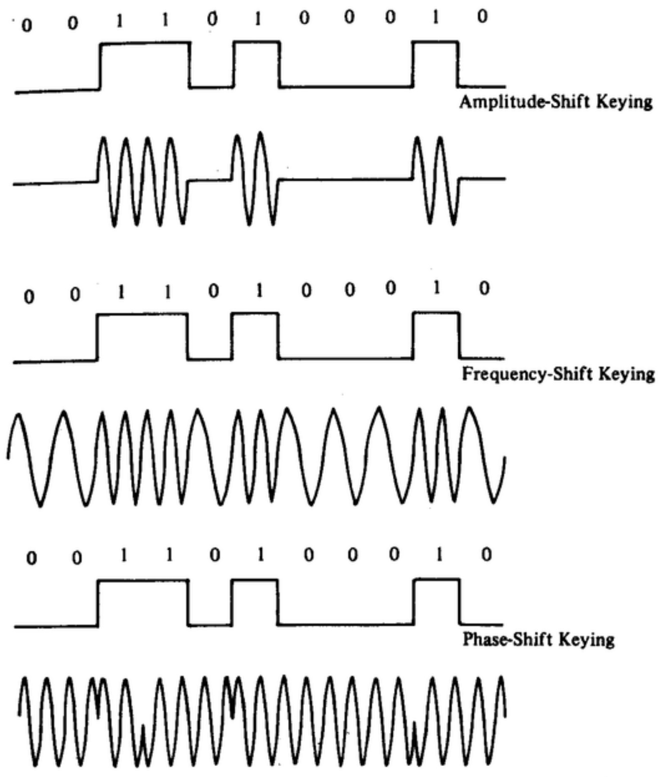


Figure 3.2: Digital modulation schemes for ASK, FSK and PSK. [25]

outgoing data to analog data and demodulating analog incoming data to digital data. Another example, more relevant to this thesis, is sending binary data through the air as radio frequency signals. Key fobs generate binary data that is understood by a receiver. However, it is impossible to transmit binary data through the air. Therefore, the binary data has to be first converted to an analog waveform using digital modulation techniques. There are multiple different digital modulation techniques but the most fundamental techniques are based on so-called "keying". The three most important modulation techniques which covers most of digital modulation techniques used in key fobs are Amplitude-Shift Keying (ASK), Frequency-Shift Keying (FSK) and Phase-Shift Keying (PSK) [65] (see Figure 3.2).

Amplitude-Shift Keying is a form of amplitude modulation in which the amplitude of the carrier signal is changed according to the digital input signal. The simplest and most common ASK technique is called On-off Keying (OOK) and represents a digital data at the presence or absence of a carrier wave. The binary value of 1 is represented by the presence of a carrier while the binary value 0 is presented by the absence of the carrier [37].

Frequency-Shift Keying is a form of frequency modulation in which the frequency of the carrier signal varies according to the digital signal values. When the binary input data is 1, the modulated wave is high in frequency while a binary 0 is low in frequency.

Phase-Shift Keying represents binary data by the change in phase of the carrier signal by varying the sine and cosine at a particular time [53]. The two most popular PSK techniques are Binary Phase-Shift keying (BPSK) and Quadrature Phase-Shift Keying (QPSK). BPSK is also called 2-phase PSK or 2-PSK in which it takes two phases separated by 180° by changing the sine wave carrier. BPSK is therefore able the transmit one bit per symbol (0, 1). QPSK or 4-PSK changes the sine wave carries as well as the cosine wave carrier resulting in 4 different phases. Therefore, QPSK can transmit two bits per symbol (00, 01, 10, 11) meaning it can double the data rate while still using the same bandwidth as BPSK.

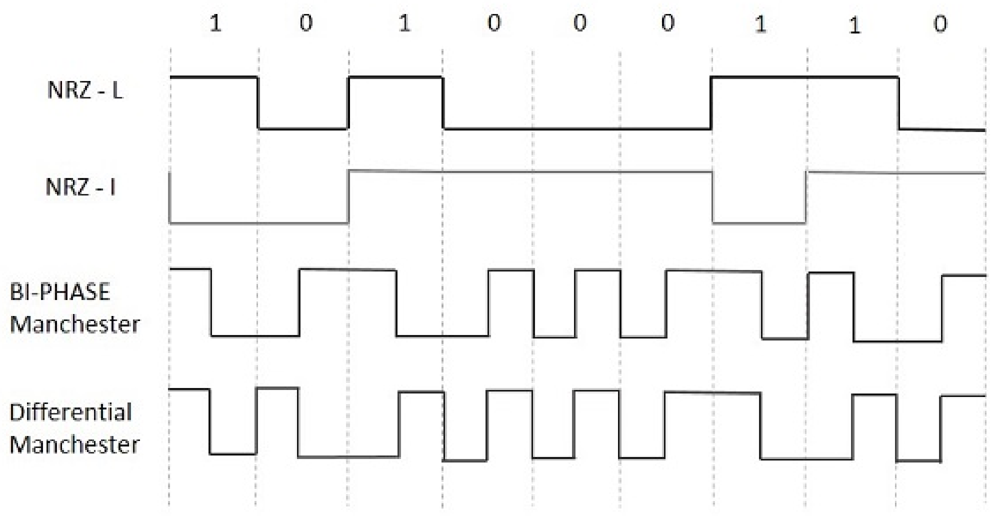


Figure 3.3: Different data encoding techniques. [64]

3.2 Data encoding

It might seems obvious that a high signal equals a binary 1 and a low signal equals a 0. However, most of the times, this is not the case. Typically data is encoded before being transmitted. Encoding is the process of converting data or a given sequence of characters, symbols, alphabets etc., into a specified format, for the transmission of data [64]. In data encoding, also called line coding, various patterns of voltage or current levels are used to represent 1's and 0's of a digital signal on the transmission link. The advantages of data encoding or line coding can be a simplification of bit clock recovery, error detection capabilities, the possibility to transmit data at higher rate, ... depending on which method is used. There exist multiple different data encoding methods, the most common are illustrated in Figure 3.3.

Non-return-to-zero level (NRZ-L) is the most intuitive data encoding technique in which a binary 1 is mapped to a logic-level high and a binary 0 is mapped to a logic-level low. A variant on NRZ-L is the **Non-return-to-zero inverted** (NRZ-I) technique. In NRZ-I a specific binary value forces a transition in state or level while the other binary value doesn't change state or level (keeps the previous level). NRZ-I refers to both NRZ-M, in which a binary 1 forces a transition, or NRZ-S, in which a binary 0 forces a transition.

Bi-phase Manchester is one of the most commonly used line coding techniques available. While transitions in NRZ techniques are done at the beginning or end of a bit-interval, Manchester coding techniques forces a transition at the middle of the bit-interval [5]. Due to the frequent line voltage transitions (at least once every bit), proportional to the clock rate, it allows the signal to be self-clocking. This allows a receiver to easily synchronise with the sender while no longer needing an external clock signal. A disadvantage of using Manchester coding over a NRZ coding is that twice the bandwidth is required to achieve the same data rate due to the forced transition in the middle of each bit-interval [35].

There are two variants of the Bi-phase Manchester coding: One in which a binary 1 is represented by a high to low transition, while a binary 0 is represented by a low to high transition. The other variant represents a binary 0 by a high to low transition, while a binary 1 is represented by a low to high transition.

Differential Manchester is an expansion on the Bi-phase Manchester coding. A binary 1 makes the first half of the bit-interval equal to the last half of the previous bit-interval. A binary 0 makes the first half of bit-interval opposite to the last half of the previous bit-interval [3]. Therefore, a binary 1 can be indicated by a transition at the beginning of the bit-interval. Unlike with Bi-phase Manchester coding, only the presence of a transition is important, not the polarity. This makes differential coding schemes immune to inversion (wire swapped) since it doesn't matter if the transition is from low to high or high to low.

3.3 Digital signal processing

Analog signals transmitted via radio waves are often referred to as continuous-time signals. Continuous-time signals can be defined and represented at any instant of time in the sequence. That is, the sequence or interval of the signal is infinite or uncountable. Analog or continuous-time signals observed "on the air" can be represented by a complex exponential using the Euler's identity: $e^{i\phi} = cos(\phi) + isin(\phi)$. When an analog signal is represented using complex values (using Euler's identity) it is called a phasor or quadrature signal [52].

Before a signals can be (digitally) processed, the analog or continuous-time signal has to be converted to a digital signal. A digital signal is often referred to as a discrete-time signal. A discrete-time signal is defined at certain (discrete) time instants. Therefore, these signals can be represented as a sequence of finite numbers. The Euler identity or quadrature signals are commonly used for representing digital signals and are often called I/Q signals as it represents the in-phase (I) and the quadrature (Q) components of the signal. The conversion from an analog to a digital signal is done by sampling the continuous analog signal. This is done by an analog-to-digital converter (ADC). By sampling a continuous analog signal we obtain a discrete-time digital signal. In the digital signal processing world, the ADC is contained in a software-defined radio explained in the next section. Once an analog signal is converted to a digital signal (f.e. I/Q signal), we can start processing the digital signal to obtain another digital signal (=digital signal processing). One of the goals of digital signal processing is to remove channel noise and demodulating and decoding the signal in order to obtain higher-layer data like its binary form.

3.3.1 Software-defined radio

A SDR is a radio communication device where components that are typically implemented as hardware components (mixers, filters, amplifiers, modulators, ...) are implemented in software. A SDR enables the use of a general-purpose processor for processing signals while formally this was done through special-purpose hardware (typically expensive). A common example of a SDR is a sound card that is installed in almost all computers. Modern sound cards use a DAC to convert digital audio into an analog format which can be listened to through headphones or speakers [74]. The SDR devices used for signal processing are devices that are capable of capturing and/or transmitting radio frequency signals. This device consists of a RF front end (generally defined as everything between the antenna and the digital baseband system) which can be seen as the ADC.

Before converting the analog signal to a digital signal using an ADC, the SDR tunes the received radio frequency signal to a baseband signal (signal centered around 0 Hz). This process is called downconverting. The main advantage of downconverting a radio frequency signal is that the required sample rate to correctly represent the analog signal is significantly reduced. The sample rate is the number of samples per seconds used to represent (or sample) an analog continuous signal. According to the the Nyquist Theorem, also known as the sampling theorem, the minimum sample rate to correctly represent an analog signal is twice the maximum frequency of the analog signal. This means, when for example capturing a 433.92 MHz key fob signal without downconverting, it would require a sample rate of 867.84 million samples per second (S/s). Capturing this many samples per second is impossible for a standard-purpose computer. When transmitting, the baseband signal is converted to a radio frequency signal (=upconverting).

Figure 3.4 illustrated how binary data is transmitted "on the air" from sender to receiver. A device has some binary data that he wants to transmit using a Wi-Fi dongle. First, the binary data is

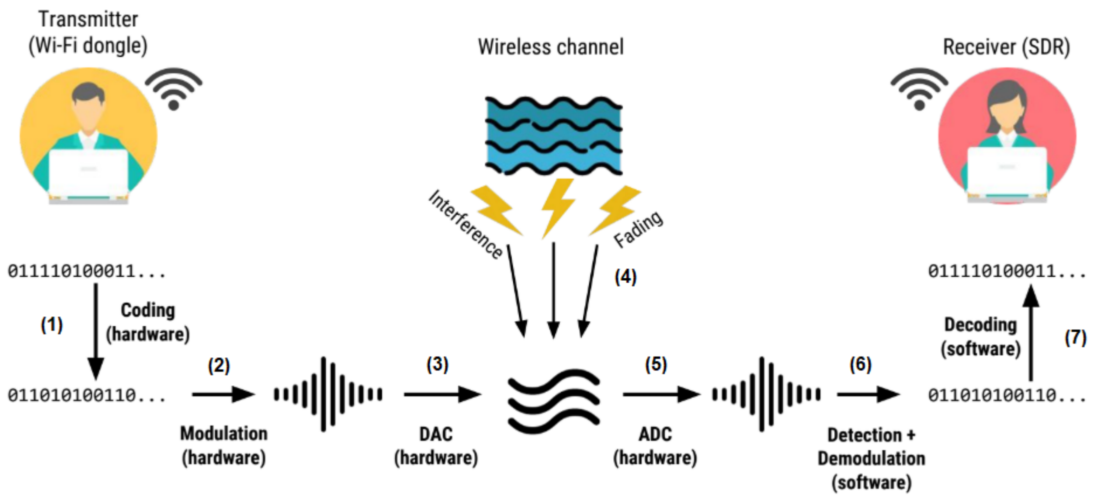


Figure 3.4: Overview of the transmission of binary data over a wireless channel received by a SDR. [52]

encoded (1) as explained in Section 3.2. Next, the encoded data is modulated (2) resulting in a digital signal representation of the binary data as explained in Section 3.1.2. The digital signal is converted to an analog signal using a digital-to-analog converter (DAC) and transmitted over the wireless channel (3). While the signal propagates “on the air”, it is exposed to numerous different factors causing interference (4). First, the wireless channel is shared by multiple transmitters of many network related devices. Non-network devices like a microwave ovens can also interfere with the radio frequency signals. Additionally, physical structures like concrete, brick or metal walls, trees or large rocks can disturb wireless signals. Even the weather (wind, rain, snow, air temperature, water content of the atmosphere, ...) can influence the propagation of signals through the air. Therefore, the reception of wireless signals is typically seen as more complex compared to transmitting signals due to interference when propagating “on the air”. Once the signal is captured by the receiver the analog signal is converted to a digital signal using the ADC (5). The receiver typically listens for a predefined sequence of bits (=preamble) present in the signal. A preamble is used to synchronise the transmission between sender and receiver and to indicate the start of a message. If the preamble is detected, the incoming signal is demodulated (6) and decoded (7) to obtain the binary data generated by the transmitter (7).

There are multiple different types of SDRs ranging from a few euro’s to hundred of euro’s on today’s market. One of the best known devices and utilised for the experiments in this thesis is called a “HackRF One” [23] (see Figure 3.5). HackRF One by Great Scott Gadgets is a USB open source hardware platform SDR capable of transmission and reception of radio signals from 1 MHz to 6 GHz. It is a half-duplex system which means it cannot transmit and receive at the same time. Therefore, a second SDR is used to transmit and receive RF signals simultaneously. This SDR is a RTL-SDR dongle (RTL2832U) [54] which is limited to capturing signals between 500 kHz and 1766 MHz and unable to transmit (see Figure 3.6.

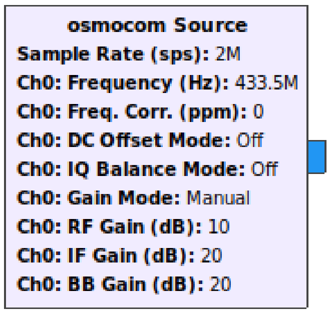
3.3.2 GNU Radio

GNU Radio is a free open-source software development toolkit that provides signal processing blocks to implement software radios [28]. It can be used with external RF hardware to create SDRs. Without any hardware it can be used in a simulation-like environment. By connecting signal processing blocks, a processing flow is defined. That is a predefined sequence in which a source signal is processed by multiple processing blocks. GNU Radio comes with a graphical interface (GNU Radio Companion) which makes implementing software radios easier. Processing

Figure 3.5: HackRF One by Great Scott Gadgets.

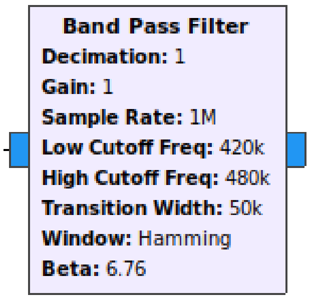
RTL-SDR.COM
QUICKSTART SETUP GUIDE: RTL-SDR.COM/QSG
DVB-T+DAB+FM+SDR
RTL2832U R820T2 TCXO+BIAS T+HF
V3

Figure 3.6: RTL2832U RTL-SDR dongle.



osmocom Source
Sample Rate (sps): 2M
Ch0: Frequency (Hz): 433.5M
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 10
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

Figure 3.7: "Osmocom Source" block.



Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 1M
Low Cutoff Freq: 420k
High Cutoff Freq: 480k
Transition Width: 50k
Window: Hamming
Beta: 6.76

Figure 3.8: "Band Pass Filter" block.

components (filters, mixers, ...) are represented by graphical block elements that can be connected to each other via directed arrows. The graphical sequence of processing blocks is called a flow graph. GNU Radio is compatible with most SDR hardware vendors including HackRF One and RTL-SDR dongle.

There are multiple predefined signal processing blocks with different functionalities available in GNU Radio. However, to configure and use most SDRs, an additional package *gr-osmosdr* has to be installed which comes with additional blocks for GNU Radio. What follows is an explanation of some of the most important blocks used in experiments related to the wireless security of keyless entry systems in this thesis:

- **Source block**: A source block is used to generate a source signal or capture radio frequency signals via a SDR. The "Osmocom Source" block (see Figure 3.7) uses a SDR to capture a specific frequency spectrum and includes a wide range of configurable parameters like sample rate, frequency (Hz), RF gain (dB), IF gain (dB), BB gain (dB), ... Besides source blocks that use a SDR to capture signals, there are plenty other source blocks available in GNU Radio like a "Signal Source" which generates a variety of signal types (Sine, Cosine, Square, ...) or a "Noise Source" that generates noise without the need of a SDR.

- **Sink block**: Sink blocks are output blocks that can be used to transmit signals ("Osmocom Sink"), visualise signals in real-time, listen to FM radio stations ("Audio Sink") and many more. There are multiple types of visualisation available in GNU Radio like waterfall, frequency, histogram, ... (see Figure 3.9). With the use of visualisation blocks the user can observe the configured frequency spectrum in real-time.

  Another important sink block is a "File Sink" that writes a digital signal, captured with a SDR or generated by a block, to a file. The saved signal can later be used as a source or can be loaded by different programs.

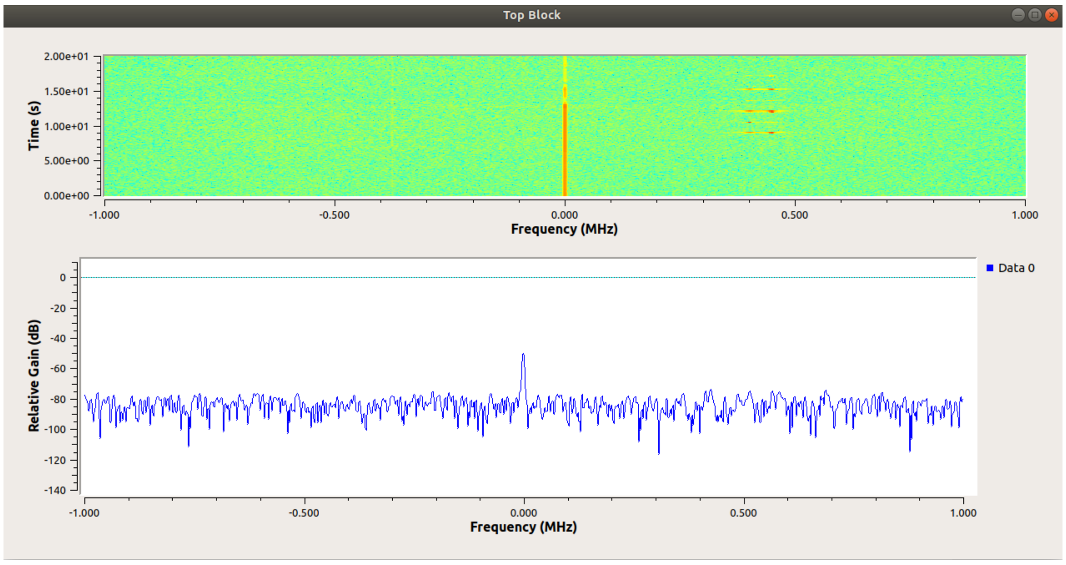


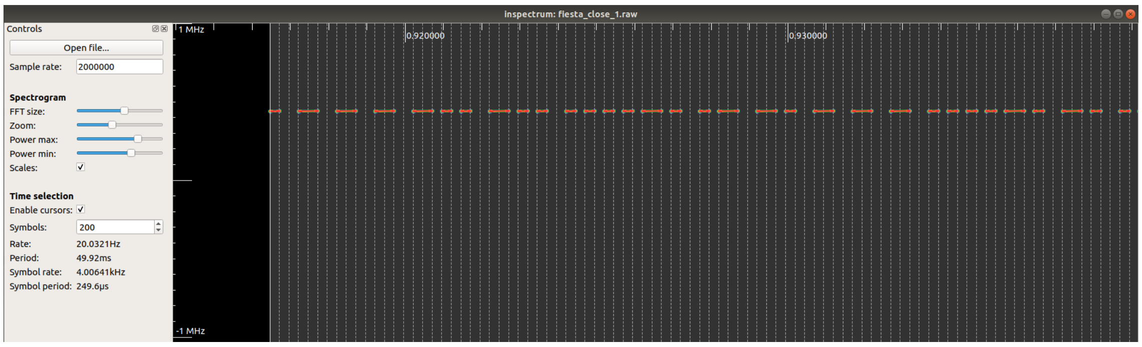
Figure 3.9: A waterfall and frequency sink block.

An import input parameter for source and sink blocks that capture or transmit signals using a SDR is the sample rate. The higher the sample rate, the better the analog signal is represented. The sample rate depends on the processing power of the computer and the capabilities of the SDR. When the sample rate is too high and the CPU is unable to process the samples in real-time, GNU Radio notifies the user that an overflow is detected (GNU Radio outputs 'O' to console) which results in a lost of samples. Each SDR is limited to a minimum and maximum sample rate when capturing or transmitting signals. The HackRF One is officially limited to a minimum of 2 million S/s (however, it will also work correctly with a lower sample rate) and a maximum of 20 million S/s. Since signals are downconverted, sample rates between 1 and 2 million S/s should be more than sufficient.

- **Filter block**: A filter block is used to pass signals with a frequency lower/higher than a specific cutoff frequency and attenuates signals with a frequency higher/lower than the cutoff frequency. Therefore they can be used to attenuates specific frequency ranges the user is not interested in. GNU Radio provides 3 types of filter blocks: A "Low Pass Filter" in which low frequencies are passed, a "High Pass Filter" in which high frequencies are passed and a "Band Pass Filter" in which frequencies within a configurable range are passed (see Figure 3.8). To illustrate, an "Osmocom Source" block at 2 million S/s captures a frequency range of [-1;1] MHz. Meaning that if a source has a center frequency of 433.5 MHz, it will capture RF signals between 432.5 and 434.5 MHz. Key fob signals are typically transmitted at 433.92 MHz with typically a few kHz deviation, so there is no point for utilising a 2 MHz frequency bandwidth. The unnecessary frequency ranges can be attenuated using a band pass filter.
- **Custom blocks**: Users can implement their own blocks or download blocks created by other users in addition to the predefined blocks. The blocks are written in C++ or Python.

3.3.3 Inspectrum

Inspectrum is a Linux based signal analyser tool in which signals are visualised using a configurable spectrogram [27]. The spectrogram visualises the signal in which the x-axis indicates the time and y-axis indicates the frequency. Using Inspectrum, the user can obtain knowledge of the captured signal like the operating frequency, symbol rate, modulation technique, data encoding method, ... Users can manually configure the window size, zoom, ... of the spectrogram for better visualisation. By "enabling cursors" the user can drag a grid over the signal to measure rate, period, symbol period and symbol rate (see figure 3.10) which is typically used for demodulating and decoding the



Sample rate: 2000000

Spectrogram
FFT size:
Zoom:
Power max:
Power min:
Scales: ✓

Time selection
Enable cursors: ✓
Symbols: 200
Rate: 20.0321Hz
Period: 49.92ms
Symbol rate: 4.00641kHz
Symbol period: 249.6µs

Figure 3.10: Signal analysis tool Inspectrum.

signal. In addition, the user can consult a sample, amplitude, frequency and phase plot depending on what he needs. Besides Inspectrum, there exists multiple alternatives to visualise and analyse RF signals. E.g Gqrx which is a Linux only SDR receiver powered by the GNU Radio framework and the Qt graphical toolkit. Other examples include CubicSDR, SDR#, HDSDR, ...

Chapter 4

Experiment: Replay & jamming attack

In the following chapter we'll be implementing a fully automated replay & jamming attack (explained in Section 2.3.1). We provide two different methods: The first method requires 2 SDRs, one for transmitting signals, one for simultaneously capturing signals. The second method is an implementation specifically for the Raspberry Pi. The main purpose of the second method is to reduce the cost by replacing the transmitting SDR by the General Purpose Clock pin (GPIO4) of a Raspberry Pi.

An advantage of implementing a replay & jamming using only SDRs is the fact that it is typically easier than using dedicated hardware modules. Figure 4.1 shows the device called RollJam that Samy Kamkar created to execute his replay & jamming attack. Executing the attack using dedicated hardware modules requires knowledge about hardware (f.e. connecting them to each other), low-level programming knowledge to control the microcontroller and modules and low-level signal processing knowledge. Using SDRs, we can simply plug in the device into a computer via USB and control it using the GNU Radio framework. The only requirements for this method is a limited knowledge regarding digital signal processing and Python (or C++).

During testing of our implementation, we found a security vulnerability in the 2015 Ford Fiesta and 2008 Mazda MX5 keyless entry system. This vulnerability is explained in Section 4.3 together with methods for exploiting it. Last, we try to improve our implementation by using reactive jamming which can be found in Section 4.4.

4.1 Method 1: HackRF One & RTL-SDR

For the first method we'll be implementing the replay & jamming attack using SDRs. First, we'll talk about the required hardware and software followed by the problems that occurred during implementation and testing (Section 4.1.1). Next, we'll give a detailed explanation of the implementation (Section 4.1.2). Last, we evaluate the success rate of our implementation and see which cars are vulnerable and which are not (Section 4.1.3).

4.1.1 Hardware & software

As noted before, this implementation requires 2 SDRs: One for transmitting the jamming signal and transmitting the captured key fob signal, one for capturing the key fob signals. In this experiment the HackRF One is used as the transmitter while the RTL-SDR serves as the receiver. Alternatively, the attack can also be executed using a single SDR provided that it is a full-duplex SDR capable of transmitting and receiving simultaneously.

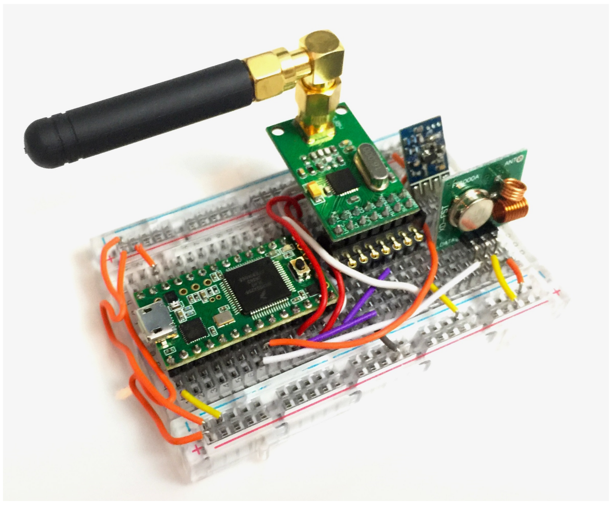
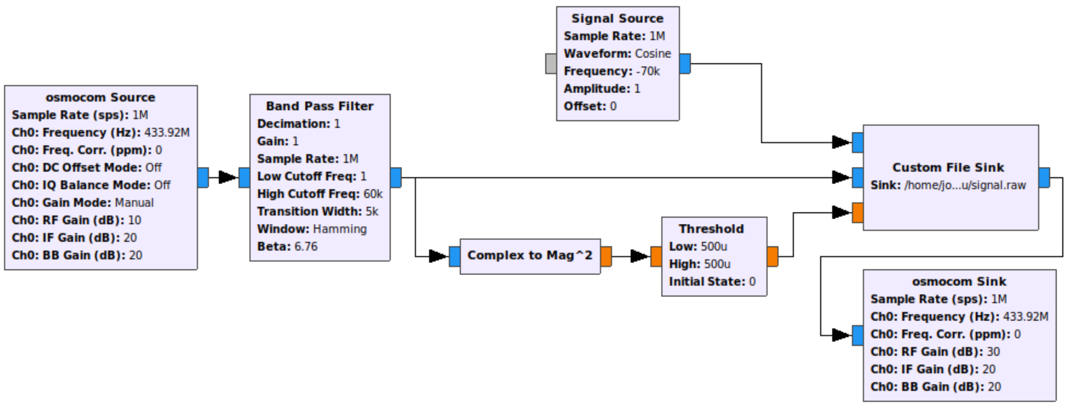


Figure 4.1: RollJam device created by Samy Kamkar used for performing replay & jamming attack. [29]

The attack is implemented using GNU Radio Companion (Section 3.3.2). However, it is also possible to execute the attack outside GNU Radio's graphical interface by simply executing the script using Python. Besides GNU Radio and Python, the *gr-osmosdr* package has to be installed. This packages includes the "osmocom" blocks that allow GNU Radio to control and use different SDRs including the HackRF One and RTL-SDR. The attack can be executed using a regular laptop (preferably running a Linux distribution). However, it is also possible to perform the attack using a Raspberry Pi. For testing purposes, running the attack using a laptop would be sufficient. However, performing such attack in a real crime scenario is inconvenient and might draw attention. Therefore, the Raspberry Pi can come in handy as it is a small, compact and cheap computer.

Installing the required packages on Raspbian (Raspberry Pi's operating system) and executing the attack introduced some unexpected problems: As mentioned before, the "Osmocom Source" and "Osmocom Sink" blocks are used to control the SDRs in GNU Radio. However, using these blocks resulted in Python corrupted memory errors when initialising the block. Multiple different type of errors occurred like "segmentation faults", "corrupted double-linked list" and "memory corruption". The error was not related to invalid initialisation parameters as the script worked perfectly fine on a computer running Ubuntu. After searching on multiple forums on the internet, it seemed that all Raspberry Pi's running Raspbian suffer from this exact error when using blocks provided by the *gr-osmosdr* package. The problem was already mentioned 2 years ago on multiple different forums. However, till this day there is not a single (working) solution provided on these forums. After analysing the error using "coredumpctl" and "gdb" we were unable to find a solution ourselves. Since it seems that the error only occurred on the Raspbian operating system, we decided to install a different operating system on the Raspberry Pi. After installing Ubuntu Mate, an Ubuntu distribution most suitable for the Raspberry Pi, the error disappeared and we were able to use blocks provided by the *gr-osmosdr* package in our GNU Radio implementation.

The second problem that occurred was less related to the replay & jamming attack itself but will be mentioned as one might experience the same problem. At random moments on boot the SD card corrupted for no particular reason. Each time this happened, the SD card had to be flashed again and the required packages had to be installed which can take a couple of hours. After searching on online forums it seems that Ubuntu Mate had difficulties with specific SD cards. The SD card used for our Raspberry Pi (Lexar High-Speed 8GB) was not listed as a "bad" SD card. However, after changing to a different SD card (SanDisk Extreme 32GB), the problem was solved. A complete installation guide for the required packages on Ubuntu Mate can be found in Appendix A.



osmocom Source
Sample Rate (sps): 1M
Ch0: Frequency (Hz): 433.92M
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 10
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 1M
Low Cutoff Freq: 1
High Cutoff Freq: 60k
Transition Width: 5k
Window: Hamming
Beta: 6.76

Signal Source
Sample Rate: 1M
Waveform: Cosine
Frequency: -70k
Amplitude: 1
Offset: 0

Complex to Mag^2

Threshold
Low: 500u
High: 500u
Initial State: 0

Custom File Sink
Sink: /home/jo...u/signal.raw

osmocom Sink
Sample Rate (sps): 1M
Ch0: Frequency (Hz): 433.92M
Ch0: Freq. Corr. (ppm): 0
Ch0: RF Gain (dB): 30
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

Figure 4.2: Replay & jamming attack flow graph in GNU Radio Companion.

4.1.2 Implementation

The resulting flow graph is shown in Figure 4.2. It starts with an "Osmocom Source" block that is used to control the receiving SDR and capture signals. As key fob signals are transmitted at a transmit frequency of around 433.92 MHz, the SDR is configured to capture at a center frequency of 433.92 MHz. Next, The "Osmocom Source" block is connected to a "Band Pass Filter" block. Using the band pass filter, the jamming signal is filtered by attenuating the frequency ranges that contain the jamming signal (433.85 MHz). The frequencies passed by the band pass filter are 433.86 MHz to 433.98 MHz which contains the default transmit frequency of a key fob. The outgoing signal from the "Band Pass Filter" block is duplicated (2 outgoing lines). One signal goes to the signal detection blocks while the other goes to the "Custom File Sink" block.

We'll start with the signal detection blocks: The "Complex to Magˆ2" block and the "Threshold" block. The signal detection is based on the amplitude of each individual sample. The incoming complex values from the "Band Pass Filter" block are converted using the function Re(in)ˆ2 + Im(in)ˆ2 by the "Complex to Magˆ2" block. The result can be interpreted as the amplitude of a sample. Samples containing a carrier frequency have a higher amplitude than samples without. The converted values enter the "Threshold" block. The "Threshold" block outputs a 0 or 1 depending on the input value. The output transitions from 0 to 1 when the power value of the input signal transitions from below to above the "High" parameter. The output transitions from 1 to 0 when the power value of the input signal transitions from above to below the "Low" parameter. This means that the amplitude of a sample containing a carrier should output 1 while the amplitude of a sample lacking a carrier should output a 0. The value of the threshold (100μ) is experimentally chosen. The value is a consideration between detecting the key fob signal as far as possible while not triggering the signal detection by random noise.

Next, we'll explain our custom self-implemented file sink, the "Custom File Sink" block. The block requires one parameter, the "sink" parameter. This parameter defines the location at which the second captured key fob signal is saved. The second key fob signal can later be transmitted to gain access to the victim's car. The block requires 3 input streams. Starting from top to bottom: The first input stream is the signal used for jamming the target. In our case we use a "Signal Source" block that generates a carrier at a constant frequency specified by the "frequency" parameter. In our case the frequency parameter is -70k (-70.000). Since we'll transmit the key fob signal at a frequency of 433.92 MHz, we set the frequency to -70k which results in the jamming signal being transmitted at a frequency of 433.85 MHz. The next input stream is the captured signal filtered by the "Band Pass Filter". The input signal includes the key fob signals which are saved to transmit later. The last input stream is provided by the signal detection blocks. When the "Custom File Sink" block receives a 1 from the signal detection block, it will start saving the incoming signal from the "Band Pass Filter" for 1 second. Initially, the implementation kept saving an incoming signal until it received a 0 (no signal detected) from the signal detection blocks. However, some key fob implementations (f.e. key fob of 2015 Ford Fiesta) transmit multiple burst of data separated with a short pause in where no carrier is transmitted. The short pauses between bursts causes the signal detection blocks to output 0's which falsely indicates the end of a signal. To make the implementation work with as many different key fob protocols as possible without having knowledge of the internal working of the protocol, the implementation will save incoming signals for 1 second upon detecting a signal.

The "Custom File Sink" block has 1 output stream that goes the "Osmocom Sink" block. The "Custom File Sink" block will provide the transmitting SDR with the jamming signal from the "Signal Source" until it receives the second key fob signal. When the second key fob signal is detected, the transmitting SDR has to stop jamming and instantly transmit the first captured key fob signal. Initially we thought that the we had to stop jamming and transmit the key fob signal after fully capturing the second key fob signal (after 1 second). This seems logical as if we stop jamming to soon, the car will still correctly receive the second key fob signal. However, transferring data over USB introduces a delay of around 900 milliseconds between giving the SDR the task of transmitting the first captured key fob signal and actually transmitting the key fob signal. This causes a total delay of almost 2 seconds (1s for capturing the signal + 0,9s transfer delay) between the victim pressing the button on their key fob and the SDR actually transmitting the first key fob signal. Therefore, the "Custom File Sink" block will provide the "Osmocom Sink" block with the first captured key fob signal upon detecting the second key fob signal instead of upon receiving the complete second key fob signal. More details about the delay caused by processing and transferring the data follows in Section 4.4.

4.1.3 Evaluation

We were able to successfully perform the attack against 4 out of the 7 test cases (2015 Ford Fiesta, 2008 Mazda MX5, 2008 Mercedes-Benz B-Class, 2008 Mazda 2). 2 out of 3 test cases that were resistant to our attack were two Ford models that used an almost identical implementation of the RKE protocol. While the rest of the test cases utilise a single transmit frequency (or two in case of a FSK modulation), both the 2018 Ford Focus and 2012 Ford C-MAX utilise 3 different frequencies for each transmitted signal. The success of replay & jamming attacks lies in the fact that most car's receivers allocate a larger frequency passband than necessary. However, both the key fobs of the Ford Focus and Ford C-MAX utilise the complete frequency passband to transmit a signal (see Section 6.4). In this case both the key fobs transmit a burst at the lower boundary of the passband, the center of the passband and the upper boundary of the passband. Because they use the complete frequency passband, there is no available frequency to transmit the jamming signal.

It seems that the replay & jamming attack can successfully bypass rolling code in most RKE system that use a single transmit frequency. However for the 2018 Mazda CX5 we were unable to successfully jam the receiver. Only when jamming on exactly 433.92 MHz, the car was unable to successfully receive the signal from the key fob. But since the key fob transmits a 2-FSK modulated signal that contains the 433.92 MHz frequency band, this implementation is unable to filter out the jamming signal and therefore capture a valid key fob signal. Why exactly it is so hard to jam the 2018 Mazda CX5 is unclear. It might be possible that the receiver allocates a frequency passband that is just large enough to capture the key fob signal. Alternatively, it might implement some sort of noise filtering to cancel out the jamming signal.

As seen in this evaluation, it is not necessary to implement specific countermeasures like time-based message protocols to prevent against most replay & jamming attack. One method is by using the complete frequency passband of the car's receiver (Ford Focus & C-MAX) or by allocating a as small as possible frequency passband that just contains the operating frequency of the key fob. Doing so, the adversary is unable to disturb the reception of the car while capturing a valid key fob signal.

4.2 Method 2: Rpitx & RTL-SDR

In this section we'll demonstrate an alternative method specifically for the Raspberry Pi. As mentioned before, the main purpose of this method is to reduce the cost by replacing the transmitting SDR by the General Purpose Clock pin of the Raspberry Pi. Since a SDR capable of transmitting radio frequency signals are sold starting from around €200 - €300, the cost can be reduced significantly. Identical to the first method we'll talk about the required hardware and software followed by the problems that occurred during implementation and testing (Section 4.2.1). Next, we'll give a detailed explanation of the implementation (Section 4.2.2). Last, we evaluate the success rate of our implementation and compare it to the first method (Section 4.2.3).

4.2.1 Hardware & software

Recently *rpitx* was released, a general radio frequency transmitter for the Raspberry Pi which doesn't require any other hardware to transmit radio frequency signals. *Rpitx* is compatible with all Raspberry Pis (PiZero, PiA+, Pi2B, Pi3B ...) expect from the PiB which is partially supported. As it turns out, the "General Purpose Clock" pin (GPIO4) of the Raspberry Pi which serves as the GPU clock pin can be transformed into a transmitter that is capable of handling frequencies from 5 kHz to 1.5 GHz. To amplify the transmitted signal, a simple wire, serving as an antenna, can be connected to the GPIO pin. The transmit power is proportional to the length of the wire. For capturing signals, we'll use the RTL-SDR.

Implementing the automated replay & jamming attack did not go as smoothly as intended. During implementation a lot of problems occurred. Additionally, the two experienced problems regarding the Raspberry Pi described in Section 4.1.1 also apply to this method. However, using *rpitx* introduced some additional problems:

- After installing *rpitx* on the Raspberry Pi (Ubuntu Mate), it did not transmit any signals. Unfortunately, it neither displayed any error messages. As *rpitx* worked perfectly on Raspbian, it seemed that this problem was related to Ubuntu Mate. After contacting the author of *rpitx*, he hinted that the problem might be related to the GPU frequency value, which indeed was the case. *Rpitx* comes with a *install.sh* script that automated the installation of *rpitx* and other required packages. The script also changes the GPU frequency ("gpu_freq") value in "/boot/config.txt" to 250. This is required in order to run *rpitx* properly. However, The script failed to change the GPU frequency on Ubuntu Mate. Manually changing the GPU frequency in "/boot/config.txt" to 250 fixed the problem.
- After capturing a signal using the RTL-SDR and transmitting it using *rpitx*, it seemed that the car did not accept the transmitted signal. This problem was caused by the supported sample rate ranges of the RTL-SDR and *rpitx*. As the max sample rate of *rpitx* is 200.000 S/s, the sample rate of the RTL-SDR was tuned to 200.000 S/s. As the official documentation of the RTL-SDR only states that the maximum sample rate is 3,2 million S/s, a sample rate of 200.000 S/s should not be a problem. However, it was noted on unofficial RTL-SDR forums that the RTL-SDR has two possible sample rate ranges: 225.001 S/s to 300.000 S/s and 900.001 S/s to 3.200.000 S/s. When tuning the RTL-SDR to an invalid sample rate (f.e. 200.000 S/s) it will automatically change the sample rate to the default value of 1.000.000 S/s. As *rpitx* was transmitting a signal at a rate of 200.000 S/s that was captured at a rate of 1.000.000 S/s, the signal was invalid. However, as the minimum sample rate of the RTL-SDR is *225.001 S/s*, there is still a difference of 25.001 S/s. To overcome this gap of 25.001 S/s, the RTL-SDR is tuned to 1.000.000 S/s and downsampled using a decimation value of 5 which results in 200.000 S/s ($\frac{1.000.000}{5}$). After transmitting the downsampled signal, the car accepted the signal.
- The replay & jamming attack implementation includes an automated signal detection identical to the one in method 1. However, we have to take in consideration that the output of the GPIO pin is more or less a square wave. A square wave is the sum of (theoretically) infinite sine waves which are called harmonics. Lets assume we want to transmit a signal

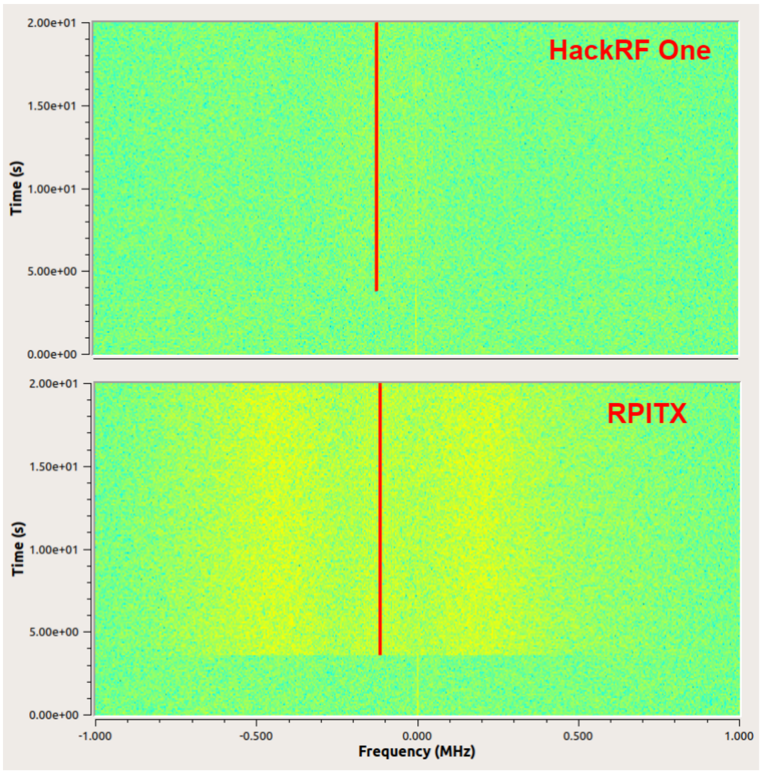


Figure 4.3: Noise comparison between transmitting carrier using HackRF One and Rpitx.

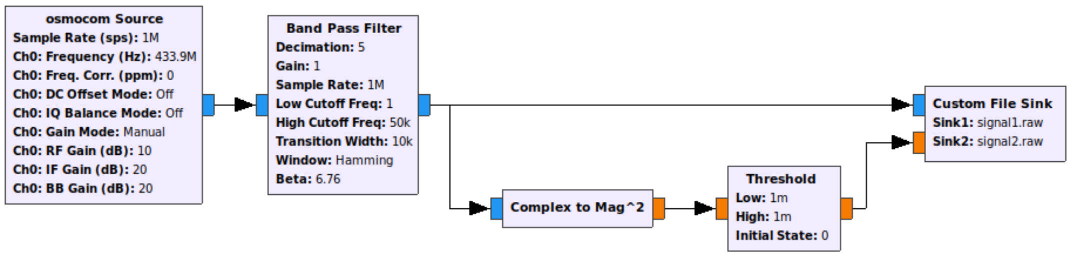
on a 10 MHz frequency: Ideally, we would only like a sine wave at 10 MHz. However, a 10 MHz square wave is the sum of sine waves at 10, 20, 30, 40, ... MHz. Therefore we'll also be transmitting signals at the other frequencies even though it is not intended. Additionally *rpitx* transmits a lot of unintentional noise as the GPIO4 pin was not created with the purpose of transmitting signals (see Figure 4.3). These harmonics and unintentional noise are detected by the automated signal detection and therefore starts capturing a none existing key fob signal. To reduce the noise and intensity of the harmonics we changed the length of the attached wire (=antenna). Initially, a 20 cm wire was used. By cutting the wire to approximately 5 cm, the signal strength reduced dramatically while still being able to jam the car's receiver. However, this did not completely solve the problem as sometimes the noise or harmonics still triggered the signal detection. To solve this, we implemented some thresholding to make the signal detection trigger less quickly. How exactly the thresholding is implemented is explained in the following section.

4.2.2 Implementation

The main difference with the implementation compared to the first method is that we can't directly control *rpitx* in GNU Radio as it is a command-line tool. To use *rpitx* together with GNU Radio, a separated *RPITX* class was implemented to simplify the use of *rpitx* in Python. We use "subprocess.Popen" to execute a child process, in our case transmitting a signal using *rpitx*. The class provides 3 different functions: A function to start jamming (*start_jammer()*), a function to stop jamming (*stop_jammer()*) and a function that transmits a signal from an input file passed in the function parameter (*transmit(file)*). The *start_jammer()* function calls the following command via "subprocess.Popen":

```
./sendiq -s 200000 -f 433.8e6 -t float -i jammer.raw -l
```

"-s" defines the sample rate, "-f" defines the transmit frequency, "-t" defines the IQ type, "-i"



osmocom Source
Sample Rate (sps): 1M
Ch0: Frequency (Hz): 433.9M
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 10
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

Band Pass Filter
Decimation: 5
Gain: 1
Sample Rate: 1M
Low Cutoff Freq: 1
High Cutoff Freq: 50k
Transition Width: 10k
Window: Hamming
Beta: 6.76

Complex to Mag^2

Threshold
Low: 1m
High: 1m
Initial State: 0

Custom File Sink
Sink1: signal1.raw
Sink2: signal2.raw

Figure 4.4: Replay & jamming attack flow graph in GNU Radio Companion.

defines the input file and “-l” tells *rpitx* to loop the input file. “subprocess.Popen” returns the process object which is stored in a variable. The *stop_jammer()* function kills the process using “os.system” in Python with the *pid* provided by the stored process object.

```
os.system(‘‘sudo kill −9 %s’’ % (pid, ))
```

The third function *transmit(file)* is similar to the *start_jammer()* function. The function transmits a signal by passing the following command to “subprocess.Popen”:

```
./sendiq −s 200000 −f 433.9e6 −t float −i file
```

The resulting flow graph (see Figure 4.4) is simplified version of the flow graph in the first method. The first difference is the decimation parameter in the “Band Pass Filter” block. Besides attenuating specific frequency ranges, the “Band Pass Filter” block is also used to downsample the incoming signal. As noted before, the maximum sample rate of *rpitx* is 200.000 S/s while the minimum sample rate of the RTL-SDR is 225.001 S/s which causes a difference of 25.000 S/s. By capturing samples at a rate of 1.000.000 S/s and decimating the sample rate by 5, the signal is dopwnsampled to 200.000 S/s ($\frac{1.000.000}{5}$).

The second difference is the simplified version of the “Custom File Sink” block. Since we no longer directly transmit signals via GNU Radio, the “Signal Source” and “Osmocom Sink” blocks are removed. Due to the problem regarding the detection of noise and harmonics transmitted by *rpitx*, a signal detection thresholding is implemented in the “Custom File Sink” block. in GNU Radio, signals are passed to the next block in packets of multiple samples (max 4096 samples). The size of a packet depends on the process capabilities of the block. Blocks that are computationally inexpensive are capable of processing larger packets than computationally expensive blocks. Initially, the signal detection was triggered upon receiving a single “1” from the “Threshold” block which introduced many false positives. The slightest noise or harmonics could trigger the automated signal detection and start capturing a none existing key fob signal. By setting a minimum on the amount of 1’s detected in a packet before triggering the detection of a signal, we could lower the false positives drastically. However, after implementing the threshold the problem was not yet fully solved and there were still some false positives making it through. Instead of implementing thresholding limited to a single packet, an additional threshold was implemented covering multiple packets. Instead of triggering the signal detection upon detecting a single valid packet (=surpassing the minimum amount of 1’s), the signal detection is only triggered when 3 valid packets in a row are detected. The combination of these 2 threshold ensures that the noise and harmonics did not trigger the automated signal detection in most cases.

4.2.3 Evaluation

The success rate of the replay & jamming attack using *rpitx* seems to be very random. We can successfully perform the attack 3 times in a row, however, the 4th time the jammer is unable to jam the receiver. There is no logical reason for this phenomena since we are not moving the setup between different runs.

Like noted before, *rpitx* transmit a lot of unintentional noise and harmonics when transmitting the jamming signal. We significantly reduced the false positives by implementing thresholding but we were unable to completely eliminate them. Additionally, *rpitx* only has a fixed transmit power which can not be configured. In method 1, we could increase the transmit power of the SDR in case the jamming signal was not powerful enough to jam the car. In this method, we are restricted to changing the length of the wire to increase or decrease the transmit power. However, increasing the transmit power also increases the strength of the noise and harmonics. The antenna is real sensitive as the position of the Raspberry Pi and antenna influences the strength of the transmitted signal. When for exampling touching the antenna during transmission, the body acts as a transmitter. By using the body as a transmitter, the signal is amplified as well as the unintentional noise and harmonics.

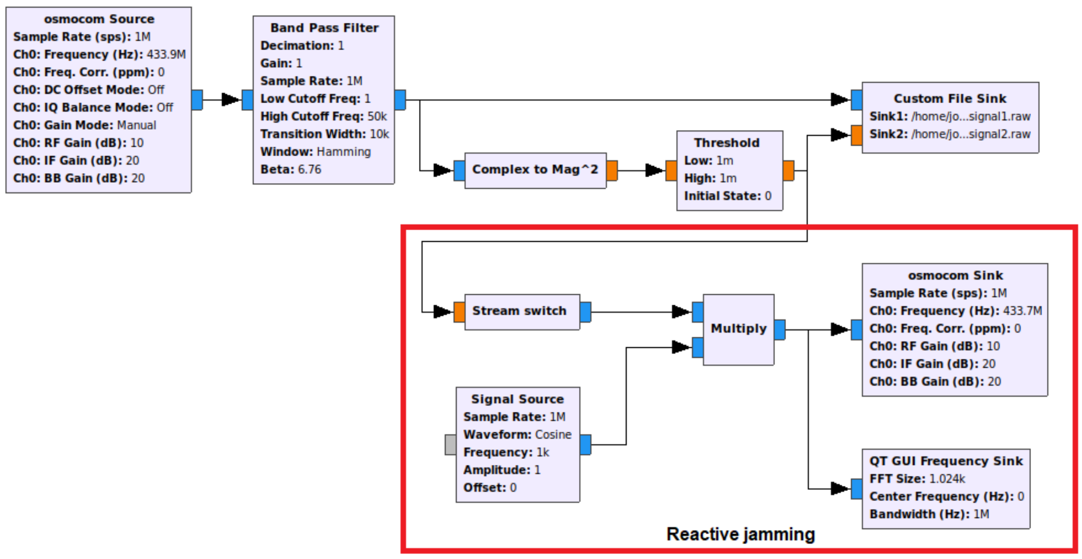
While we were able to successfully perform the replay & jamming attack using *rpitx* on some cars, the success rate is much lower compared to the first method. By changing the configurations (antenna length, frequency passband, jam frequency, ...) for each specific car, we were able to successfully perform the attack against 1 of the 7 cars (2015 Ford Fiesta). We were also able to perform the attack against the 2008 Mazda MX5 and 2008 Mazda 2 after trying many times changing the configuration and placement of the Raspberry Pi. However, performing the attack a second time did fail again. As we were only able to perform the attack a few times against these cars, we will not include them as successful test cases. The success rate of the attack is highly dependent on the distance between the jamming frequency and the boundary of the frequency passband of the adversary's receiver. By jamming closer to the operating frequency of the key fob, it is easier to jam the car's receiver. Because of the noise we can not directly jam along side the passband of the adversary as it triggers the automated signal detection.

To perform this attack in a real crime scenario, the adversary has to configure their setup (antenna length, frequency passband, jam frequency, ...) according to the victim's car to successfully perform the attack. Manually configuring the setup for each individual car is in most cases impossible for the adversary as he has no access to the key fob. We can conclude that using this method in a real crime scenario is not recommended and in most cases even impossible.

A potential solution for reducing the noise and harmonics permanently is using a low pass filter to ensure *rpitx* does not transmit signals above a certain boundary. Alternatively, we can change the signal detection to detecting a specific preamble instead based upon the amplitude of a sample. This way the noise and/or harmonics won't trigger the signal detection. However, if we base the signal detection upon a specific preamble, the attack will only work on the specific RKE protocol that uses this exact preamble.

4.3 Findings: "Time-out" vulnerability

During testing of the replay & jamming attack, a security vulnerability was found in the RKE system of both the 2015 Ford Fiesta and the 2008 Mazda MX5. After receiving a valid signal from the key fob, the Ford Fiesta's receiver ignores every rolling code for 30 seconds. In other words, out-dated signals captured in the past are accepted by the car for 30 seconds upon receiving a valid rolling code. A possible explanation for this vulnerability is the fact that the Ford Fiesta's key fob transmits 4 to 50 identical messages depending on the how long the button on the key fob is pressed (see Section 6.2.1 for more information). Instead of verifying the rolling code for each burst. The receiver will most likely only verify the first received burst. If the rolling code of that burst is valid, it will ignore all rolling codes of the next bursts received within 30 seconds. However this method makes no distinction between different individual signals. Therefore, when transmitting a previous captured signal (with an expired rolling code), it handles the received bursts as messages with a valid rolling code. This vulnerability can be exploited in various ways to gain access to the car. For example: A victim parks their car (at home, work or at the mall) and locks the door. The adversary now has 30 seconds to transmit a previous captured unlock signal to unlock the door. A victim is most likely out of sight of their car in less than 30 seconds which means the adversary can unlock the car without getting noticed.



osmocom Source
Sample Rate (sps): 1M
Ch0: Frequency (Hz): 433.9M
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 10
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 1M
Low Cutoff Freq: 1
High Cutoff Freq: 50k
Transition Width: 10k
Window: Hamming
Beta: 6.76

Custom File Sink
Sink1: /home/jo...signal1.raw
Sink2: /home/jo...signal2.raw

Complex to Mag^2

Threshold
Low: 1m
High: 1m
Initial State: 0

Stream switch

Multiply

osmocom Sink
Sample Rate (sps): 1M
Ch0: Frequency (Hz): 433.7M
Ch0: Freq. Corr. (ppm): 0
Ch0: RF Gain (dB): 10
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

Signal Source
Sample Rate: 1M
Waveform: Cosine
Frequency: 1k
Amplitude: 1
Offset: 0

QT GUI Frequency Sink
FFT Size: 1.024k
Center Frequency (Hz): 0
Bandwidth (Hz): 1M

Figure 4.5: Reactive jamming flow graph.

This behaviour is also found in the Mazda MX5. However, for some unknown reason, the car will only accept the 3rd transmitted signal. Also instead of 30 seconds, the car will ignore the rolling code for 60 seconds. Another difference compared to the Ford Fiesta vulnerability is that the 60 second “time-out” can be extended. When receiving a previous captured signal the 60 second “time-out” will renew. this “time-out” renewal introduces different attack possibilities. For example, a transmitter can be installed beneath the car that keeps transmitting a signal. The 60 second “time-out” will keep getting extended which makes the car ignore rolling code completely for as long the transmitter is alive.

The vulnerability was also tested on a 2018 Ford Focus, 2012 Ford C-MAX, 2018 Mazda CX5 and a 2008 Mercedes-Benz B-Class. However, it seems that these models did not suffer from the vulnerability.

4.4 Reactive jamming

Both implementations use a naïve method of jamming in which the jamming signal is continuously transmitted. Instead of continuous jamming, reactive jamming would be a better option. As noted before, in reactive jamming the target is jammed for only a short amount of time to prevent a specific key fob signal that is already “on the air” from getting accepted. This prevents the adversary from continuously jamming all other cars surrounding the target and minimising the risk of getting caught. When using reactive jamming, the time between pushing a button on the key fob and transmitting the jamming signal should be as close to zero as possible since the slightest delay could cause the key fob signal to be accepted by the car before the jamming signal is transmitted.

Figure 4.5 shows an implementation of our attack using reactive jamming. The part of the flow graph surrounded by the red rectangle handles the reactive jamming. Upon receiving a 1 from the “Threshold” block (signal detection), the custom written “Stream switch” block will output 1’s for a short amount of time. This output is received by the “Multiply” block together with the output of the “Signal Source” block used to generate the jamming signal. Multiplying an input signal by 1 gives the same exact value, namely, the generated signal by the “Signal Source” block. These values are passed to the “Osmocom Sink” block that transmits the jamming signal. When no signal is detected the “Stream switch” will output 0’s. Multiplying the generated signal by 0 results in a signal made up of complex values with both the real and imaginary value 0. Therefore,

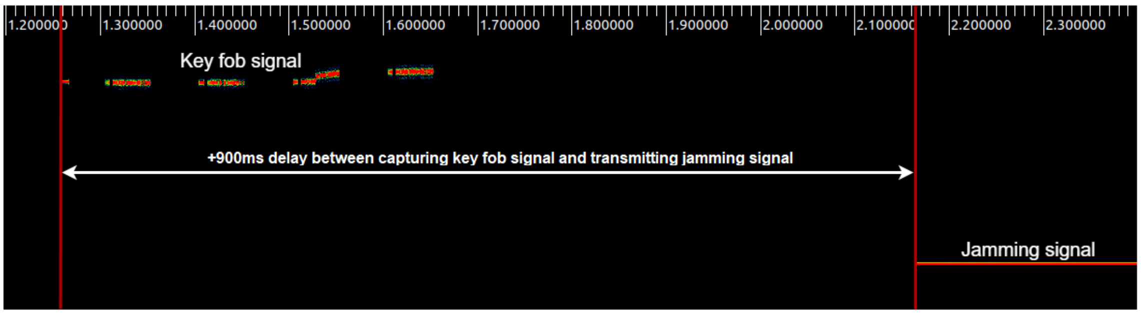


Figure 4.6: Inspectrum spectrogram illustrating delay between capturing the key fob signal and transmitting the jamming signal.

when no key fob signal is detected, no jamming signal is transmitted.

When testing our implementation using reactive jamming, we noticed that we were unable to start jamming in time due to the delay caused by the processing and transferring of the signal. First, the analog signal is captured by the receiver, converted to a digital signal and transferred to the computer via USB 2.0. Next, the digital signal is processed by the flow graph in order to detect the key fob signal. Upon detecting a key fob signal, the digital jamming signal is transferred to the HackRF via USB 2.0, converted back to an analog signal and transmitted. The conversion from analog to digital and vice versa, the actual processing of the digital signal and the transferring of samples between the SDR and computer introduces a delay of almost 900 milliseconds between capturing the key fob signal and transmitting the jamming signal (see Figure 4.6). This delay causes the key fob signal that is "on the air" to be accepted by the car before a jamming signal can be transmitted.

In 2016, Vo-Huu et al. stumbled upon the same problem when trying to implement jamming in Wi-Fi networks [68]. Because of the delay introduces the SDR they were unable to jam the network in time. To overcome this problem, Vo-Huu et al. modified the firmware of the HackRF such that the required processing is performed by the HackRF's microcontroller itself. This means that it is no longer required to transfer any data between the HackRF and the computer. For additional performance reasons, the jamming signal is not generated on the fly but instead a pre-generated jamming signal is stored in the HackRF's memory. By modifying the firmware of the HackRF they were able to successfully interfere with the Wi-Fi communication using reactive jamming.

Chapter 5

Experiment: Signal amplification relay attack

In the following chapter we'll perform a self-implemented signal amplification relay attack (Section 2.3.2) against PKE(S) systems. In Section 5.1 we'll explain the general concept of the attack and compare different method for relaying data. Next, we'll explain the hardware and software required to perform the attack in Section 5.2. In Section 5.3 we'll show and explain the implementation of our attack. We end with an evaluation of our implementation and see if we were able to successfully perform the attack against cars equipped with PKE(S) (Section 5.4).

5.1 Concept

Even though most PKE(S) systems do not yet use any distance-bounding protocols, the success rate of the attack is highly dependant of the delay introduced by relaying the data. In 2011, Francillon et al. experimented with relay attacks against various PKE(S) systems and measured the maximum acceptance delay of 10 different car models [21]. As seen in Table 5.1 the maximum acceptance delays are measured between 35 microseconds to 20 milliseconds. Francillon et al. were able to achieve a minimum delay of 4 microseconds by modifying their USRP FPGA (field-programmable gate array) to bypass communication with the computer. Doing so, they were able to unlock all 10 different car models. Using distance-bounding protocols these maximum acceptance delays can be reduce to a few nanoseconds which makes it impossible to relay data in time.

There exist different methods for relaying the LF signal to the other relay device. We'll list some of the options below and compare them:

Car model	**Max. Delay**
Model 1	500 μs
Model 2	5 ms
Model 3	-
Model 4	500 μs
Model 5	1 ms
Model 6	10-20 ms
Model 7	620 μs
Model 8	620 μs
Model 9	2 ms
Model 10	35 μs

Table 5.1: Maximum acceptance delay of 10 different car models [22].

1. **Relay-over-WiFi**: Our initial idea was to relay the LF signal using a Wi-Fi network. The simplest way of doing this is to connect the devices to an existing Wi-Fi network. However, this requires the adversary to have access to the Wi-Fi network (in other words, know the password). Since cars are generally located outside, it is likely that one or both devices are not in range of any existing Wi-Fi networks. A more convenient way of ensuring that 2 devices can communicate with each other is by setting up a standalone Wi-Fi network using one of the 2 devices as an access point. For this experiment our 2 devices are Raspberry Pi's as we'll be using *rpitx* to transmit signals. It is possible to create an access point from a Raspberry Pi's without requiring any additional hardware. By connecting the other Raspberry Pi to the access point, the 2 devices can communicate with each other without having to rely on any existing networks. A short guide on how to setup a wireless access point on a Raspberry Pi running Ubuntu Mate is found in Appendix B.

   When testing our implementation, we experienced some problems. Sampling analog signals to obtain digital signals is a memory (and CPU) intensive process. Capturing signals at a sample rate of 1 million S/s can take up to 8 Mb/s which has to be transmitted over a TCP socket. After decimation by 5 (200.000 S/s) the data-transfer rate is reduced to approximate 1.6 Mb/s. When testing the data throughput from Pi to Pi over our standalone Wi-Fi network the max throughput we could achieve is 1.5 Mb/s. When running the implementation, the TCP socket was unable to cope with these amounts of data which caused overflow in the GNU Radio blocks.

2. **Relay-over-cable**: An alternative way of relaying data is via an Ethernet cable. The speed of transmitting data over an Ethernet cable is typically faster and more reliable than over Wi-Fi. By using this method, there is also no need for setting up a wireless access point. This method can use the same implementation as the Relay-over-WiFi method as it independent of the physical layer of the OSI-Model. However, using a physical Ethernet cable can be inconvenient and might cause suspicion. Also, the presence of a wall or door can make a relay attack over a physical cable impossible.

3. **Relay-over-UHF-channel**: In the two previous methods, we digitised the analog signal and relayed the digital data to the other relay device over a higher-level transport protocol. In this method we'll be relaying the analog signal via an UHF channel without the need of any other transport protocol. First, the LF signal is received by the SDR and downconverted before obtaining the digital signal (see Section 3.3.1). This creates a baseband signal centered around 0 Hz and therefore independent of the actual transmit frequency of the original signal. We can now simply configure the transmitting SDR to the desired frequency (f.e. 2.5 GHz) and transmit the signal. By amplifying the signal, the signal can travel significantly further compared to the original LF signal transmitted by the car. The second adversary who is in proximity of the key fob receives the signal, downconverts it to a baseband signal and upconverts it back to the original LF signal (125 kHz). The signal is received by the key fob and transmits its response at a frequency of 433.92 MHz.

   Relaying messages over an UHF channel is significantly less complex than the two previous methods. First, setting up a standalone Wi-Fi network is no longer required. Second, we don't have to use any higher-level transport protocols like TCP and therefore don't have to setup any connections/sockets which introduce extra delay, overhead and complexity. Relay-over-UHF-channel should therefore have a smaller delay regarding relaying of the LF signal.

Since delay introduces by relaying is such an important factor for the success rate of a relay attack, we'll be implementing a relay-over-UHF-channel attack as it should have the least amount of processing delay.

5.2 Hardware & software

A signal amplification relay is attack is performed by two adversaries and therefore requires a separate device for each adversary. For this experiment we'll be using 2 Raspberry Pi's, 2 receiving

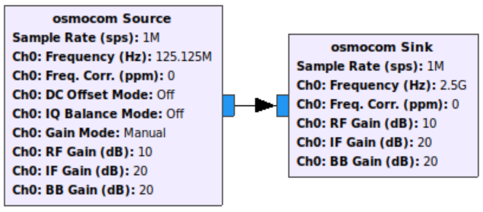


Figure 5.1: GNU Radio flow graph for relay device 1.

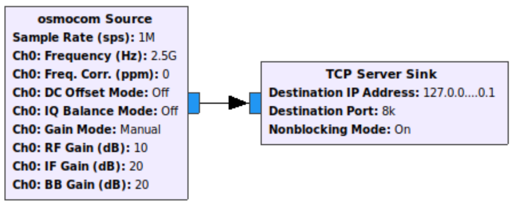


Figure 5.2: GNU Radio flow graph for relay device 2.

SDRs (RTL-SDR), 1 transmitting SDR (HackRF One) and a single upconverter (Ham It Up v1.3). Since we are capturing LF signals (125 kHz), our regular whip or telescopic antenna (suited for capturing HF, VHF and UHF radio bands) was no longer able to optimally receive the LF signal. Therefore we changed to a loop antenna specially tuned for receiving 125 kHz signals (ANT125K-45). We also have to use an upconverter (Ham It Up v1.3) to lower the minimum receive frequency of the RTL-SDR to 125 kHz (default minimum is 500 kHz). We'll be using the GNU Radio framework to control our SDRs and implement the attack. For transmitting the LF signal to the key fob, we pipe the received data to *rpitx*.

5.3 Implementation

The GNU Radio implementation for performing the signal amplification relay attack is extremely easy. The implementation for the device that captures the LF signals from the car consists of only 2 blocks. First, the 125 kHz signal is captured using a "Osmocom Source" block. Since the RTL-SDR is unable to capture a 125 kHz signal, we utilise an upconverter that uses a 125 MHz oscillator. Therefore, we have to configure our "Osmocom Source" block at a receive frequency of 125.125 MHz to obtain 125 kHz signals. The source block is directly connected to a "Osmocom Sink" block that transmits the signal using a 2.5 GHz frequency (see Figure 5.1). At the receiving side we'll be using a "Osmocom Source" block to capture the UHF signal. The source block is connected to a "TCP Server Sink" block that sends the data to a local netcat client that pipes the data to *rpitx* (see Figure 5.2). *Rpitx* transmits the signal at the original frequency of 125 kHz. To open a netcat client that pipes the received data to *rpitx*, we have to run the following command in the terminal of the device:

```
nc 127.0.0.1 8000 | sudo ./sendiq -s 200000 -f 125e3 -t float -i-
```

5.4 Evaluation

Due to problems regarding the shipping of the required 125 kHz loop antenna, we were unable to test our implementation against cars equipped with PKE(S). However as noted before, the most important factor in the success of this attack is the delay introduced by relaying the data. By relaying a regular 433.92 MHz signal instead of the LF signal transmitted by the car, we can measure the delay and conclude if the attack would successfully work in a real situation using a PKE car by comparing it to the maximum delay thresholds seen in Table 5.1.

The delay introduced for this implementation is identical to the delay introduced when implementing reactive jamming in Section 4.4. When doing the same experiment for this implementation we measured a delay of 900 milliseconds. This is only the delay introduced by the first device that captures the signal and transmits it over a 2.5 GHz channel. The second device will also need to capture the UHF signal and downconvert it back to the original 125 kHz frequency (433.92 MHz in our case). This will introduce an additional delay of 900 milliseconds. Therefore the time between transmitting the LF signal by the car and receiving a response from the authorised key fob will take around 2 seconds (900ms + 900ms + transferring time of signals over the air). As the maximum acceptance delay in Table 5.1 is 20 milliseconds, it impossible to successfully implement a signal amplification relay attack using out-of-the-box SDRs.

As mentioned in the beginning of this chapter, Francillon et al. were able to successfully perform the attack against all 10 test models. However, they initially experienced the same problem in which the delay introduced by transferring the data between the USRP and computer was too high. By modifying the USRP FPGA, they were able to bypass the transferring of data between the device and computer. Doing so, they were able to successfully perform the attack. As seen in Section 4.4, Vo-Huu et al. experienced an identical problem when trying the manipulate a Wi-Fi network by using reactive jamming. The introduced data transferring delay was too high in order to jam the target in time. By modifying the firmware of SDR, they were able to eliminate the transfer between the SDR and computer and successfully manipulate the network. As it is impossible to perform the relay attack using an out-of-the-box SDR (or USRP), it is possible by modifying the firmware of the device.

Chapter 6

Experiment: Reverse engineering radio frequency signals

In the following chapter we'll reverse engineer radio frequency signals transmitted by a key fob. We introduce 3 case studies: The signal of a 2008 Mazda MX5, 2015 Ford Fiesta and 2018 Ford Focus key fob. For each key fob, we'll determine the modulation and data encoding technique utilised to form the transmitted signal. Next, we'll use these findings to demodulate and decode the signals resulting in the corresponding binary data. For the demodulation and decoding of the signals, custom real-time automated demodulators are written. Finally, the resulting binary data is analysed to find the message format (f.e. unique identifier, command, sequence number, rolling code, ...) used by the RKE protocol of each key fob.

6.1 The process of reverse engineering

More or less the same process can be used to reverse engineer any signal transmitted by a key fob. This process contains of 6 steps: Capturing the key fob signal, signal analysis to determine the signal properties (carrier frequency, symbol rate, ...), determining the modulation technique, determining the data encoding method, demodulation and decoding of the signal and finally, analysis of the resulting binary data. Using these 6 steps, it should be possible to reverse engineer most RKE key fobs. For each step, we'll be giving a detailed explanation:

1. **Capture the key fob signal**: Before we can analyse a radio frequency signal, we have to capture it. We can capture a radio frequency signal by using a SDR and a program capable of utilising the SDR f.e. GNU Radio, SDR#, Gqrx, ... Figure 6.1 shows a simple GNU Radio flow graph that uses an "Osmocom Source" block to capture signals utilising through the connected SDR. Since European car key fobs operate at a frequency of 433.92 MHz (or 315 MHz in North-America and Japan), the SDR has to be configured to capture a frequency range including the frequency of 433.92 MHz. The SDR is configured at a rate of 2 million S/s and a center frequency of 433.5 MHz. This means that the frequency range of 432.5 MHz to 434.5 MHz is captured which includes the 433.92 MHz transmit frequency of a key fob. The "Osmocom Source" block is connected to a "File Sink" block used to save the captured signal. Optionally, a "Frequency Sink" or "Waterfall Sink" can be added to visualise the frequency spectrum.

2. **Signal analysis**: In the signal analysis step we determine the signal properties by analysing the transmitted signal using visualisation tools like Inspectrum. These properties are necessary to help demodulating, decoding and analysing the binary data at a later stage. Examples of such properties are:

    - Symbol rate: The rate at which each individual symbol occurs

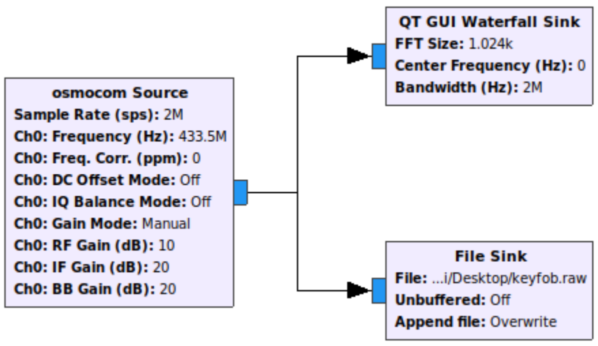


Figure 6.1: GNU Radio flow graph that captures and saves a key fob signal.

- Carrier (or transmit) frequency: The frequency at which the signal is transmitted
- Signal length: The length of the signal (in symbols)
- ...

The symbol rate and signal length can be determined by using the "enabling cursor" functionality of Inspectrum. When activating this functionality, a grid overlay divided in smaller blocks will appear on top of the spectogram. The user can chose the amount of blocks ("Symbols" parameter) and change the size of the blocks by increasing/decreasing the size of the grid. When each individual block of the grid matches and covers a single symbol of the signal, the symbol rate can be read directly from the interface 3.10). The signal length can be determined by matching the amount of blocks with the symbols of the signal. The carrier or transmit frequency can be read directly from the vertical axis in Inspectrum.

3. **Determine modulation technique**: In most cases the modulation technique can be identified by solely visual inspection of the signal. How exactly different modulation techniques can be visually identified is explained in the case studies themselves. However, it is also possible to automate the identification of a modulation technique. An example is *DSpectrumGUI*, a reverse engineering tool built on top of Inspectrum that aims to make it trivial to demodulate radio frequency signals. This includes an automated detection of the modulation and data encoding technique followed by automated demodulation and decoding of the input signal. Unfortunately, *DSpectrumGUI* supports only a limited set of modulation techniques: Pulse-Width Modulation, On-off Keying and Frequency-Shift Keying (2-FSK) which is treated as an On-off Keying modulation. It is also limited to the detection and decoding of the Manchester data encoding method. However, these are the most used modulation and data encoding techniques and should cover a large amount of key fob signals.

4. **Determine data encoding method**: Similar to determining the modulation technique, Inspectrum can help to resolve the data encoding method. However, determining the exact data encoding method by solely visual observation of the spectrogram is not enough. Inspectrum can help in narrowing down possible data encoding methods. For example, variants of the Manchester data encoding methods (explained in Section 3.2) can easily be recognised visually. Manchester data encoding methods force a transition in the middle of each bit which means there can't be more than 2 high (or low) level symbols in a row (see Figure 3.3). Therefore, certain data encoding methods can be excluded and the possible used data encoding methods can be narrowed down. However, there are different variant for the Manchester coding method that can't be distinguished visually. Finding the correct variant of the used data encoding is a matter of trial-and-error and analysis the resulting binary data. The correct data encoding variant can be identified by finding patterns in the binary

data. An example of such patterns is a counter field that increases each time the key fob button is pressed.

Alternatively, as mentioned in step 3, *DSpectrumGUI* can be used to automate the identification of the used data encoding method. However, *DSpectrumGUI* is very limited as it can only identify the Manchester data encoding method and is unable to distinguish different variants of the Manchester data encoding method.

5. **Demodulating and decoding the signal**: When both modulation technique and data encoding method are identified, the captured signal can be converted to actual raw binary data by demodulating and decoding it. This can be done manually but is not recommended as it is time consuming and the error rate is rather high as a single wrong demodulated and/or decoded bit affects the entire binary data. A better way of demodulating and decoding a signal is by automating it using software. Besides identification of the modulation and data encoding method, *DSpectrumGUI* can demodulate and decode the input signal (if the utilised modulation technique and data encoding method is supported). However, reverse engineering a signal using *DSpectrumGUI* can still be time consuming as the input signal has to be captured using a different program and signal properties has to be identified manually. Additionally, even when both the modulation technique and data encoding method are supported, it is not always possible to correctly demodulate and decode the signal as we'll see in a case study.

   A better and quicker way of demodulating and decoding a radio frequency signal is using a real-time demodulator and decoder. This means that a signal is demodulated and decoded instantly upon capturing it. In these case studies, the real-time demodulators/decoders were created using GNU Radio.

6. **Analysis of the raw binary data**: After demodulating and decoding the signal, the raw binary data is obtained. Since there is no standard message format for data transmitted by a key fob, car manufacturers use a self-chosen format. In most cases, the message format evens differs for each different model of the same car manufacturer. This means that for almost every different car model available, a different message format is used. On top of that, the used format is not publicly published by the manufacturers and therefore unknown to the public. It is also not guaranteed that a result is found since some signals are fully encrypted and therefore not possible to analyse.

   A convenient method for analysing data is by capturing multiple consecutive signals from a single command (f.e. unlock) and comparing them to each other to see which bits are static and which vary. Static bits might indicate the unique identifier of the car or the pressed command. The command bits can be confirmed by capturing signals from a different command (f.e lock). If the assumed command bits of the lock signal are different from the unlock signals but static for the same command, the bits are most likely used to indicate the pressed command. Capturing consecutive signals can also identify potential counter bits. Counter bits can be identified by searching bits that change their value (0/1) for every consecutive transmitted signal. Bits which contain no noticeable pattern and seems random for each transmitted signal can indicate the rolling code or potential data integrity bits (checksum, CRC or MAC).

Alternatively, it is possible to obtain information about a specific radio frequency transmitter by browsing the internet. In the United States it is obligated for the manufacturers to publish information (user manual, specification, hardware, ...) for every radio transceivers in a public database. The database is maintained by the United States Federal Communications Commission and found on their website: `https://fccid.io/`. Each radio transceivers is identified by a FCC ID written on the device (typically inside the casing). This ID can be used to lookup information about the corresponding device. this information typically contains operating frequency, modulation, bit encoding, ...

Unfortunately, Europe does not impose those obligation to manufacturers selling radio transceivers in Europe. This means that there is no public database available in Europe. However, some

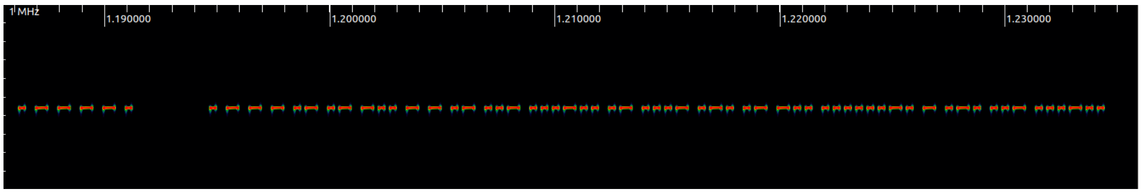


Figure 6.2: A single burst of the Ford Fiesta's key fob.

European key fobs contain a US FCC ID that can be looked up on `https://fccid.io/` as see in the first case study.

6.2 Case study: 2015 Ford Fiesta

6.2.1 Reverse engineering

The user manual of the 2015 Ford Fiesta is available at [18] (FCC ID: KR55WK47899). As the manual states, the radio frequency transmitter was created by Siemens VDO Automotive AG. Additional information written in the user manual in described in the corresponding reverse engineering steps themselves.

1. **Signal analysis**: Figure 6.2 shows a single burst transmitted by the key fob of the 2015 Ford Fiesta. As we analyse the complete signal we see that 4 such bursts are transmitted for each button press. After comparing the four bursts, we conclude that all four bursts are identical. Each burst has the same random structure: A short predefined sequence of symbols, a short pause in which no carrier is transmitted followed by a longer sequence of symbols. The first short sequence of symbols is equal for each individual transmitted signal while the longer sequence seems to vary. The first short sequence is therefore most likely the preamble while the longer sequence is the actual data (command, rolling code, ...).

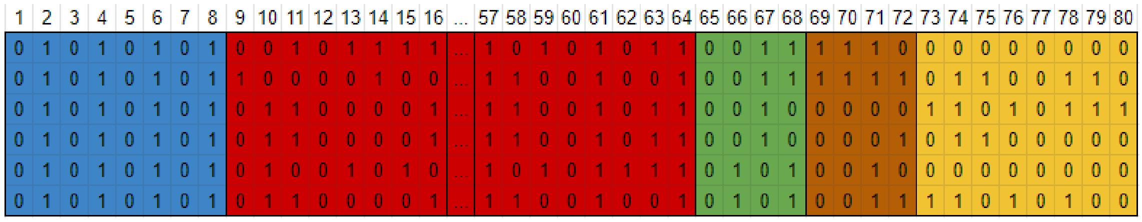
   The signal properties of the 2015 Ford Fiesta key fob are shown below:

   - Preamble length: 20 symbols
   - Data Length: 160 symbols
   - Transmit/carrier frequency: 433.937 MHz
   - Symbol rate: 4 kHz

   The user manual confirms the transmission of 4 identical bursts (named "telegrams"). However, it also states that the key fob can transmit up to 50 bursts depending on how long the button is pressed. Additional information obtained from the user manual is that a single burst (or telegram) consists of 52ms of data and 48ms of a pause.

2. **Determine modulation technique**: As seen in Figure 6.2, the frequency of the signal is constant while the amplitude changes over time indicating that the signal is modulated using the Amplitude-Shift Keying (ASK) modulation. The data is represented by the presence or absence of a carrier wave which means the **On-off Keying modulation** (variant of ASK modulation) technique is used. This is one of the simplest and most intuitive forms of modulation in which the presence of a carrier wave represents a binary 1 while the absence of the carrier wave represents a binary 0. The user manual found on `https://fccid.io/` [18] confirmed that the used modulation technique was indeed the On-off Keying modulation.

3. **Determine data encoding method**: The signal is built up from two possible lengths of symbols: a short symbol and a longer symbol which length is twice the length of the short symbol. As mention before, Manchester coding forces a transition in the middle of each bit which means there can't be more than 2 high level symbols in a row. This is exactly



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
0	1	0	1	0	1	0	1	0	0	1	0	1	1	1	1	...	1	0	1	0	1	0	1	1	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	1	1	0	0	0	0	1	0	0	...	1	1	0	0	1	0	0	1	0	0	1	1	1	1	1	1	0	1	1	0	0	1	1	0
0	1	0	1	0	1	0	1	0	1	1	0	0	0	0	1	...	1	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	1	1	0	1	0	1	1	1
0	1	0	1	0	1	0	1	0	1	1	0	0	0	0	1	...	1	1	0	0	1	0	1	1	0	0	1	0	0	0	0	1	0	1	1	0	0	0	0	0
0	1	0	1	0	1	0	1	0	1	0	0	1	0	1	0	...	1	0	1	0	1	1	1	1	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	1	0	1	1	0	0	0	0	1	...	1	1	0	0	1	0	0	1	0	1	0	1	0	0	1	1	1	1	0	1	0	1	0	0

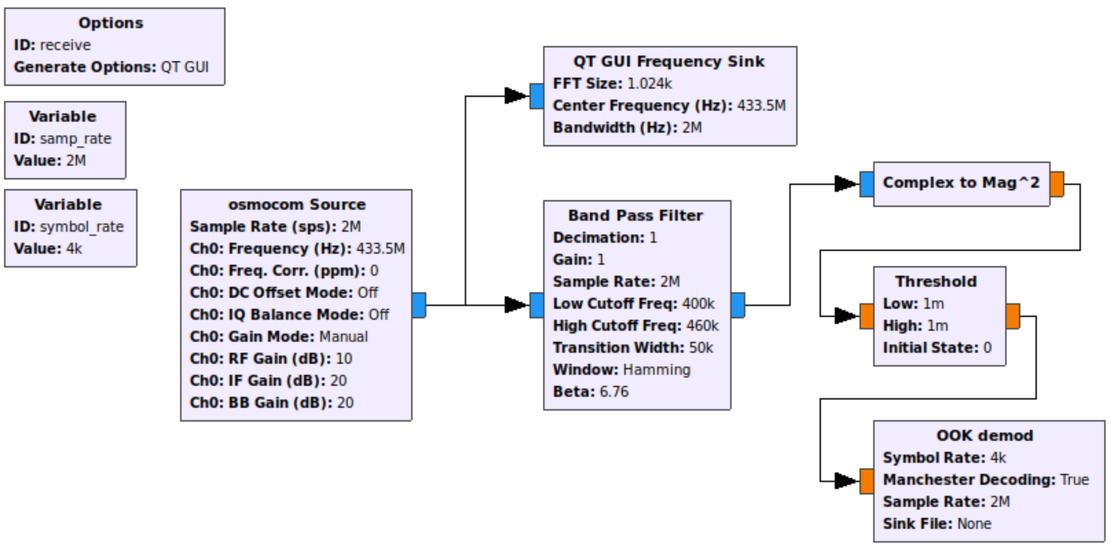
Figure 6.3: Message format for the Ford Fiesta key fob.

the case for our signal as the maximum length of a symbol is twice the length of a single short symbol. This means that the data encoding method used must be a variant of the Manchester data encoding method. However, we cannot yet conclude which exact variant of the Manchester data encoding is used as this can only be determined by analysing the demodulated and decoded raw binary data. Since we have access to the user manual we could confirm that indeed a Manchester coding was used to encode the data. The manual [18] also states that the Bi-phase Manchester coding was used. This eliminates the potential use of a Differential Manchester coding method, however, it is still unknown which of the two Bi-phase Manchester coding methods is used: One in which a binary 1 is represented by a high to low transition, while a binary 0 is represented by a low to high transition. Or, one in which a binary 0 by a high to low transition, while a binary 1 is represented by a low to high transition.

4. **Demodulating and decoding the signal**: After determining the modulation technique (On-off Keying) and data encoding method (Bi-phase Manchester coding), we can now create a real-time demodulator/decoder using GNU Radio. The resulting On-off Keying demodulator/decoder flow graph is shown and explained in Section 6.2.2.

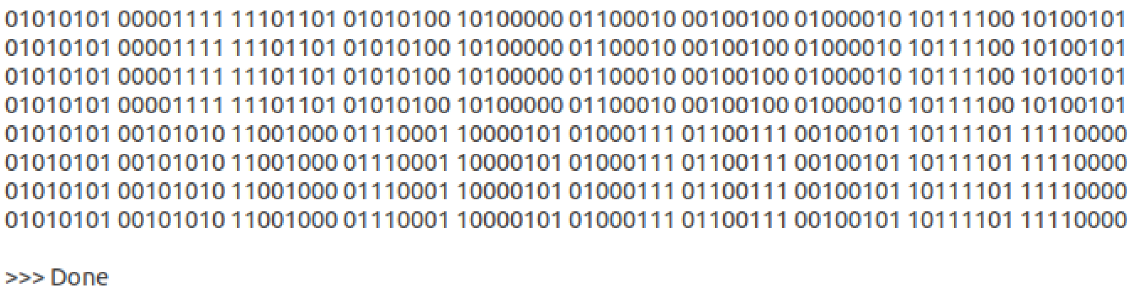
5. **analyse demodulated and decoded binary data**: Figure 6.3 shows single bursts of 6 consecutive captured signals demodulated and decoded using our custom On-off Keying demodulator/decoder shown in Section 6.2.2. We excluded the preamble since it is the same for each transmitted signal. The first and second signal represents the unlock command, the third and fourth signal represents the lock command and the last two signals represents the trunk command. The different colors represents different fields in the message format. In total, a single burst of data contains of 10 bytes (excluding the 10 bit preamble). Besides the 10 bit preamble, there is a secondary 1 byte preamble (blue) that is the same for every transmission. The next 7 bytes (red) is most likely the rolling code since they seem random and there is no correlation between successive signals. The next 4 bits (green) corresponds to the pressed button on the key fob. "0011" represents the unlock command, "0010" represents the lock command and "0101" represents the trunk command. The following 4 bits (orange) is used as a counter. As seen in the figure, the counter is incremented each time a button is pressed. However, there seems to be a correlation between the 4 command bits and the 4 counter bits. The 4 command bits are actually also part of the counter. The 4 least-significant bits (orange) are continuous and shared between different commands while the 4 most-significant bits (green) are unique for each command. When the 4 least-significant bits (orange) reach the "1111" state, each individual command value (green) will increase on the next button press. This makes the command bits variable and part of a 1 byte counter. Finally, the last byte (yellow) seems to be random for every transmission. Since we already identified the rolling code the last byte most likely serves as data integrity. After trying to find the exact checksum/CRC algorithm by running brute-force scripts, no solution was found. This is a strong indication that a Message Authentication Code (MAC) is used to validate the message.

We did not yet identify the used variant of the (Bi-phase) Manchester coding method. The above explained message format was decoded using the IEEE 802.3 Manchester coding in which a low to high transition results in a binary 0 while a high to low transition results in a binary 1. The identification of the counter strongly suggests that the used **IEEE 802.3**



```
Options
ID: receive
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 2M

Variable
ID: symbol_rate
Value: 4k

osmocom Source
Sample Rate (sps): 2M
Ch0: Frequency (Hz): 433.5M
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 10
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

QT GUI Frequency Sink
FFT Size: 1.024k
Center Frequency (Hz): 433.5M
Bandwidth (Hz): 2M

Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 2M
Low Cutoff Freq: 400k
High Cutoff Freq: 460k
Transition Width: 50k
Window: Hamming
Beta: 6.76

Complex to Mag^2

Threshold
Low: 1m
High: 1m
Initial State: 0

OOK demod
Symbol Rate: 4k
Manchester Decoding: True
Sample Rate: 2M
Sink File: None
```

Figure 6.4: On-off Keying demodulator flow graph.



```
01010101 00001111 11101101 01010100 10100000 01100010 00100100 01000010 10111100 10100101
01010101 00001111 11101101 01010100 10100000 01100010 00100100 01000010 10111100 10100101
01010101 00001111 11101101 01010100 10100000 01100010 00100100 01000010 10111100 10100101
01010101 00001111 11101101 01010100 10100000 01100010 00100100 01000010 10111100 10100101
01010101 00101010 11001000 01110001 10000101 01000111 01100111 00100101 10111101 11110000
01010101 00101010 11001000 01110001 10000101 01000111 01100111 00100101 10111101 11110000
01010101 00101010 11001000 01110001 10000101 01000111 01100111 00100101 10111101 11110000
01010101 00101010 11001000 01110001 10000101 01000111 01100111 00100101 10111101 11110000

>>> Done
```

Figure 6.5: On-off Keying demodulator output of two received signals.

Manchester encoding method is in fact correct.

6.2.2 Implementation: On-off Keying demodulator

The custom written On-off Keying demodulator demodulates a signal based on the power value of each individual sample. It uses the square of magnitude (results in the power value) of each sample to determine if a carrier was detected or not. The presence of a carrier indicates the presence of a high level signal which results in a higher power value. Therefore, the absence of a carrier results in a lower power value.

First, we have to define some variables: Besides defining the number of samples taken per second to represent the analog signal, the sample rate is later used to calculate the amount of samples per symbol. By calculating the amount of samples per symbol, we can count them and distinguish high and low-level symbols. Last, we have to define the symbol rate (symbol_rate). The symbol rate was already determined in step 1 (4 kHz) when we analysed the signal using Inspectrum.

The flow graph shown in see Figure 6.4 starts with an "Osmocom Source" block that listens for signal at the center frequency of 433.5 MHz at a rate of 2 million S/s (the sample rate can be chosen freely). Next, the captured signal is filtered by attenuating useless frequency ranges using a "Band Pass Filter". In this flow graph we pass the frequencies between 433.90 MHz and 433.96 MHz as the transmit frequency of the key fob is 433.937 MHz. For each sample we convert the complex value to the square of the magnitude using a "Complex to Magˆ2" block. The resulting value is served to the "Threshold" block. Identical to our replay & jamming attack, the threshold

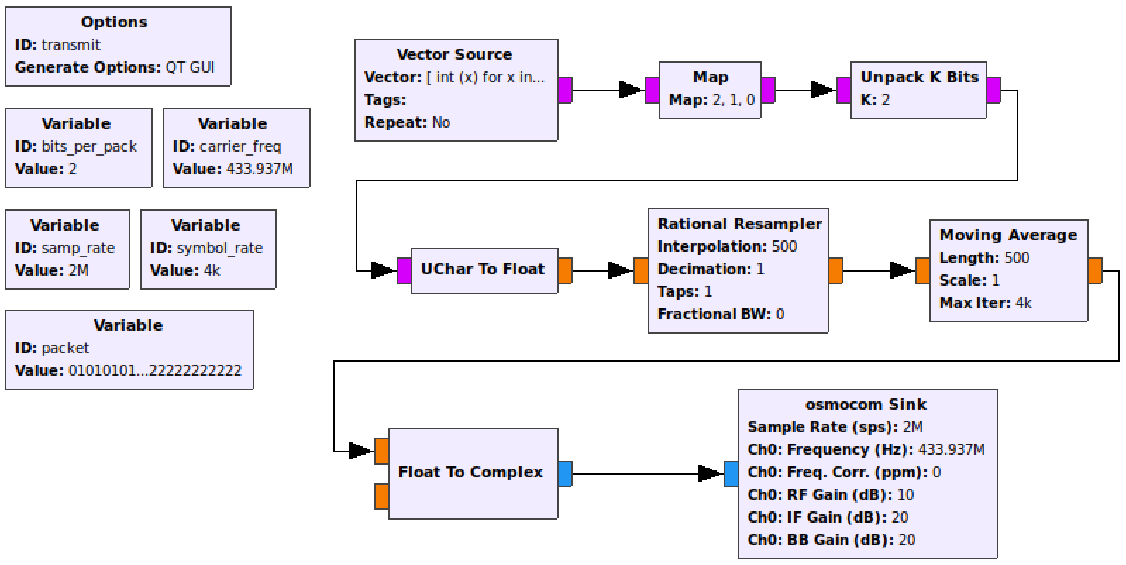


Figure 6.6: On-off Keying transmitter flow graph.

values are experimentally chosen. The value depends on the strength of the signal which is different for each key fob. The strength of the received signal is also influenced by the distance between the key fob and the SDR receiver. Last, we have our the custom written “OOK demod” block that performs the actual demodulation and decoding of the signal. Dividing the sample rate by the symbol rate gives us the amount of samples used to represent a single symbol. For example, at a rate of 2 million per second and a symbol rate of 4 kHz, each symbol contains of 500 samples ($\frac{2.000.000}{4.000}$). This means that a binary 1 is represented by 500 consecutive 1’s while a binary 0 is represented by 500 consecutive 0’s.

Besides demodulating the incoming signal, the “OOK demod” block also contains the Manchester decoder. The decoder converts a high to low transition (10) to a binary 0 and a low to high transition (01) to a binary 1. After decoding, the resulting binary data is printed to the console. Alternatively, an output file (Sink File) can be specified in the block’s variables that outputs the resulting binary data to a file. As mentioned before, a single button press transmits 4 identical bursts. Therefore, the result of the GNU Radio demodulator outputs 4 identical binary strings representing the 4 bursts. Since the first short sequence of symbols (preamble) is the same for every transmission, it is not printed (see Figure 6.5).

6.2.3 Implementation: On-off Keying transmitter

In addition to the On-off Keying demodulator, an On-off Keying transmitter (+ Manchester encoder) was written for the Ford Fiesta. The On-off Keying transmitter synthesises (modulate and encodes) an On-off Keying modulated signal from a binary input and transmit is using the connected SDR. After serving the binary output data obtained from the demodulator to the On-off Keying transmitter, we were able to successfully unlock the car without using the actual key fob or a previous captured signal (after exploiting the 30 second security vulnerability explained in Section 4.3).

The On-off Keying transmitter seen in Figure 6.6 is based on [19]. We begin with declaring some variables: Like with the demodulator, the sample and symbol rate have to be defined. The sample rate can be chosen freely depending on the processing power and supported sample rates of the SDR. The symbol rate (4 kHz) is already determined in earlier stages. Next, we define the carrier or transmit frequency at which the resulting signal should be transmitted. In case of the Ford Fiesta, a carrier frequency of 433.937 MHz is defined. The bits_per_packet variable is related to the data encoding method, in this case Manchester data encoding. Since Manchester coding forces a bit transition in the middle of each bit period, we can represent this by using 2 bits. In our case, a binary 0 is represented by a high to low transition (10) and a binary 1 by a low to high transition (01). The value of bits_per_packet is therefore 2. The last and most import variable is the binary data itself. Besides 0's and 1's, the "packet" variable also allows 2's to represent a period in which no signal is transmitted. This is used to represent the short pause between the preamble and the actual data (see Figure 6.2) as well as the pauses between the individual bursts.

We begin with a "Vector Source" block that iterates over each value in the "packet" variable. Each value gets converted to a hexadecimal value using the "Map" block. 0 is converted to 0x02 (0000 0010), 1 is converted to 0x01 (0000 0001) and 2 is converted to 0x00 (0000 0000). Using the "Unpack K Bits" block we take the K least-significant bits of the input value. In our case, K is 2 (bits_per_pack) as we only need 10, 01 and 00 to represent our symbols. Next, we have to upsample to our desired sample rate using the "Rational Resampler" block. As mention before in Section 6.2.2, the amount of samples per symbol is calculated by dividing the sample rate with the symbol rate. In this case we have a sample rate of 2 million S/s and a symbol rate of 4 kHz, which means each symbol is represented by 500 samples (interpolation value). Next, we use the "Moving Average" block to smooth out the signal. Finally, we transmit the signal at the correct carrier frequency (433.937 MHz) using an "osmocom Sink".

As we can transmit custom On-off Keying signals, we can use the transmitter to "play around" with the key fob's signal and observe the behaviour of the car. For example, how does the car react when modifying the message? By taking the output of our On-off Keying demodulator, changing a single bit in each burst (except from the last 8 data integrity bits) and transmitting it using the On-off Keying transmitter, we noticed that the car still accepted the signal. From the moment we change 2 data bits, the car ignores the received signal. However, when changing bits in the data integrity field of the message, a single flipped bit causes the signal to be invalid. We also validated if the 4 identical bursts transmitted for every button press were mandatory to assure a valid signal. In other words: Are we able to create a valid signal by transmitting less than 4 bursts. After transmitting different amounts of bursts, we concluded that the minimum amount of bursts is 3. When we transmitted 4 bursts but make a single burst invalid (by changing 2 data bits or 1 integrity bit), the signal was still valid. This means that the car ignores the invalid burst and still performs the command using 3 valid bursts. However, when we make 2 invalid bursts the signal was no longer valid as expected. The fourth transmitted burst is probably present for redundancy reasons in case a single burst is received with an error.

6.3 Case study: 2008 Mazda MX5

6.3.1 Reverse engineering

We were unable to find any documentation on `https://fccid.io/`. Therefore all statements made in the following section are based solely on the analysis of the transmitted signal by the key fob since we cannot compare them to any official documentation of the manufacturers.

1. **Signal analysis**: The key fob of the Mazda MX5 transmits one incessant burst for each button press. For each transmitted signal, the same structure is used: A noticeable long preamble followed by 3 successive identical sequence of symbols separated by a short preamble (see Figure 6.7). As expected, the long and short preambles are identical for each transmitted signal. The 3 successive identical sequence of symbols differ from each transmitted signals which indicates this sequence is the actual data (command, rolling code, data integrity, ...). We also notice that two different frequencies are used to transmit the signal. This is an indication of a specific modulation technique used (Frequency-Shift Keying) which is explained in the next step.

   After thoroughly analysing the signal, we notice that key fob uses 3 difference symbol rates: One for the preamble, one for some unknown symbols in front of every preamble and one for

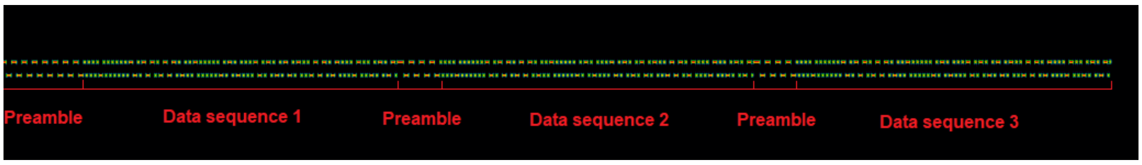


Figure 6.7: Signal structure for the Mazda MX5 key fob: A long preamble (only small part visible in figure), 3 identical sequence of symbols separated by a short preamble.

the data symbols. The properties of the different type of symbols are shown below:

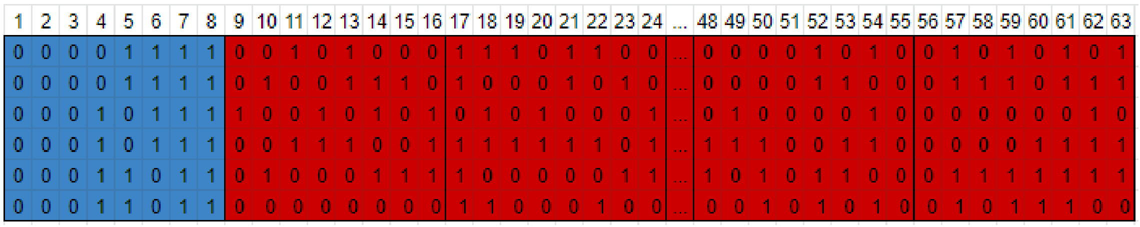
- Center frequency: 433.925 MHz (433.90 MHz & 433.95 MHz)
- First preamble:
  - Symbol rate: 3.332 kHz
  - Length: 697 symbols
- Preamble between data sequence:
  - Symbol rate: 3.332 kHz
  - Length: 8 symbols
- Data symbols:
  - Symbol rate: 7.987 kHz
  - Length: 144 symbols
- Unknown symbols in front of preamble:
  - Symbol rate: 8.741 kHz
  - Length: 3 symbols

2. **Determine modulation technique**: As mentioned in the previous step, the signal is represented by 2 different frequencies that alter each other. This means that the transmitted data is represented by the change of the carrier frequency and therefore means a Frequency-Shift Keying (FSK) modulation is used. A symbol at the higher carrier frequency (433.95 MHz) represents a binary 1 while a symbol at the lower carrier frequency (433.90 MHz) represents a binary 0.

3. **Determine data encoding method**: The signal transmitted by the 2008 Mazda MX5 key fob seems to use two different data encoding methods. For the actual data symbols, a variant of the **Manchester coding** method is used. Using the same argument as for the Ford Fiesta's key fob, there are only two different lengths of symbols (for the data symbols): A short symbol and a longer symbol which is twice the size of the shorter symbol. This indicates that there is a forced transition in the middle of each bit causing there can never be more than 2 high level symbols in a row.

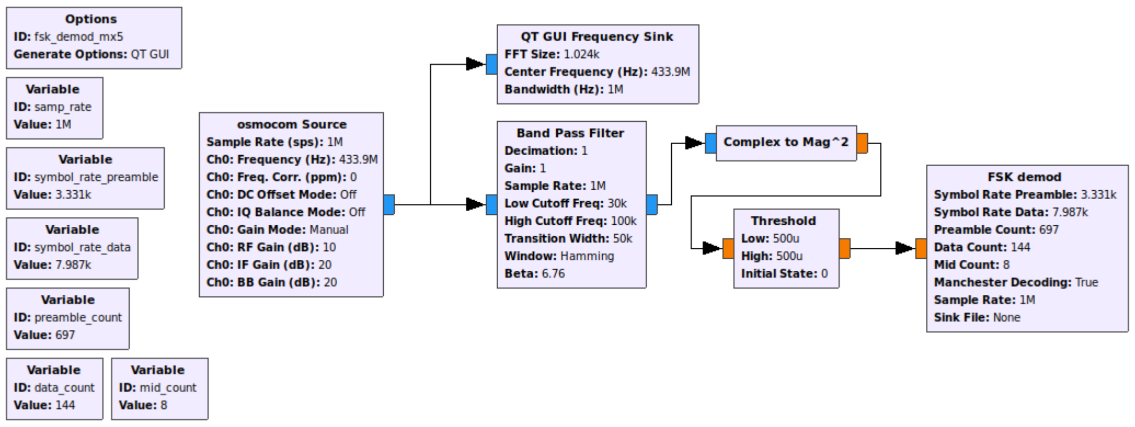
   Due to the fact that a single bit is represented by 2 different symbols (high to low or low to high) in a Manchester data encoding, the length of symbols should always be even. However, the length of the first preamble is 697 which is odd. This means that it is impossible that a Manchester data encoding method is used to encode the preamble. Therefore, we believe that a regular **NRZ-L** encoding is used (logic-level high represents 1, logic-level low represents 0) to represent the preamble.

4. **Demodulating and decoding the signal**: Due to multiple different symbol rates and 2 data encoding methods, the demodulation and decoding of the Mazda MX5's key fob is not as trivial as for the Ford Fiesta's key fob. Most Frequency-Shift Keying demodulators like the "Quadrature Demod" block of GNU Radio or *DSpectrumGUI* expect a constant symbol rate which is not the case for the Mazda MX5. It is also not common that two different data



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	...	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0	0	0	0	1	1	1	1	0	0	1	0	1	0	0	0	1	1	1	0	1	1	0	0	...	0	0	0	0	1	0	1	0	0	1	0	1	0	1	0	1
0	0	0	0	1	1	1	1	0	1	0	0	1	1	1	0	1	0	0	0	1	0	1	0	...	0	0	0	0	1	1	0	0	0	1	1	1	0	1	1	1
0	0	0	1	0	1	1	1	1	0	0	1	0	1	0	1	0	1	0	1	0	0	0	1	...	0	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0
0	0	0	1	0	1	1	1	0	0	1	1	1	0	0	1	1	1	1	1	1	1	0	1	...	1	1	1	0	0	1	1	0	0	0	0	0	1	1	1	1
0	0	0	1	1	0	1	1	0	1	0	0	0	1	1	1	1	0	0	0	0	0	1	1	...	1	0	1	0	1	1	0	0	0	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	...	0	0	1	0	1	0	1	0	0	1	0	1	1	1	0	0

Figure 6.8: Message format for the Mazda MX5 key fob.



Options
ID: fsk_demod_mx5
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 1M

Variable
ID: symbol_rate_preamble
Value: 3.331k

Variable
ID: symbol_rate_data
Value: 7.987k

Variable
ID: preamble_count
Value: 697

Variable
ID: data_count
Value: 144

Variable
ID: mid_count
Value: 8

osmocom Source
Sample Rate (sps): 1M
Ch0: Frequency (Hz): 433.9M
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 10
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

QT GUI Frequency Sink
FFT Size: 1.024k
Center Frequency (Hz): 433.9M
Bandwidth (Hz): 1M

Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 1M
Low Cutoff Freq: 30k
High Cutoff Freq: 100k
Transition Width: 50k
Window: Hamming
Beta: 6.76

Complex to Mag^2

Threshold
Low: 500u
High: 500u
Initial State: 0

FSK demod
Symbol Rate Preamble: 3.331k
Symbol Rate Data: 7.987k
Preamble Count: 697
Data Count: 144
Mid Count: 8
Manchester Decoding: True
Sample Rate: 1M
Sink File: None

Figure 6.9: Frequency-Shift Keying demodulator for the Mazda MX5.

encoding methods are used in one signal, which makes the automated decoding way harder. However, we were able to write a working real-time Frequency-Shift Keying demodulator and decoder for the Mazda MX5 (explained in Section 6.3.2).

5. **analyse demodulated and decoded binary data**: After demodulating and decoding the signal we obtain the raw binary data shown in Figure 6.8. Similar to the Ford Fiesta case study, the first and second signal is the open command, the third and fourth signal is the close command and the fifth and sixth signal is the trunk command. A single data sequence consists of 9 bytes. The first byte (blue) in the data sequence indicates the pressed command. "00001111" represents the open command, "00010111" represents the close command and "00011011" represents the trunk command. For the next 8 bytes (red), we did not find any correlation between consecutive transmitted signals as the bits seem to change randomly. There are two possible explanations: The 8 bytes are rolling code or are just encrypted and might contain different data fields like a counter, data integrity bits, rolling code, ... It is also possible that the last 1 or 2 bytes are data integrity bits. However, due to the fact that all 8 bytes seem to change randomly, we cannot separate different data fields.

   Due to the lack of data fields like a counter, it is hard to define the used variant of the Manchester data encoding method. Therefore we are unable to determine the exact variant of the Manchester data encoding method. The signals are decoded using the IEEE 802.3 Manchester coding, G.E. Thomas Manchester coding and differential Manchester coding. However, for none of the three variants a correlating data field was found (expect for the command). Figure 6.8 shows the resulting binary data decoded using the **IEEE 802.3 Manchester coding**. However, it is possible that a different variant of the Manchester data encoding was used.

6.3.2 Implementation: Frequency-Shift Keying demodulator

As mentioned before, the demodulation and decoding of the Mazda MX5's key fob is not trivial due to the variable symbol rates and different data encoding method used. However, we were able to create a Frequency-Shift Keying demodulator that can successfully demodulate and decode the transmitted signal. Due to the variable symbol rates and different data encoding methods used the demodulator will only work for signals with an identical structure. The means that it is most likely only useful for this specific Mazda MX5 model.

The demodulator interprets the Frequency-Shift Keying modulated signal as an On-off Keying modulated signal. The signal can be converted to an On-off Keying signal by ignoring the lower frequency symbols. This can be achieved by a “Band Pass Filter” block used to attenuate the lower carrier frequency (see Figure 6.9). As the signal is interpreted as an On-off Keying modulated signal, it uses the same demodulation technique as in 6.2.2. However, we cannot completely re-use the Ford Fiesta On-off Keying demodulator due to the variable symbol rates.

To solve the problem of the variable symbol rates, the symbol counts has to be included as an input parameter to the demodulator block. The counts are used so the demodulator knows the current symbol rate. The “FSK to bin” block first searches for 697 preamble symbols with a symbol rate of 3.331 kHz. When all 697 preamble symbols are detected, the demodulator expects 144 data symbols with a symbol rate of 7.987 kHz. Next, the demodulator waits for 8 preamble symbols with a symbol rate of 3.331 KHz. The 3 unknown symbols before each preamble are ignored as it does not serve any purpose. This sequence is repeated until the complete signal is detected (144 data symbols at 7.987 kHz $\rightarrow$ 8 preamble symbols at 3.333 kHz $\rightarrow$ 144 data symbols at 7.987 kHz).

6.4 Case study: 2018 Ford Focus

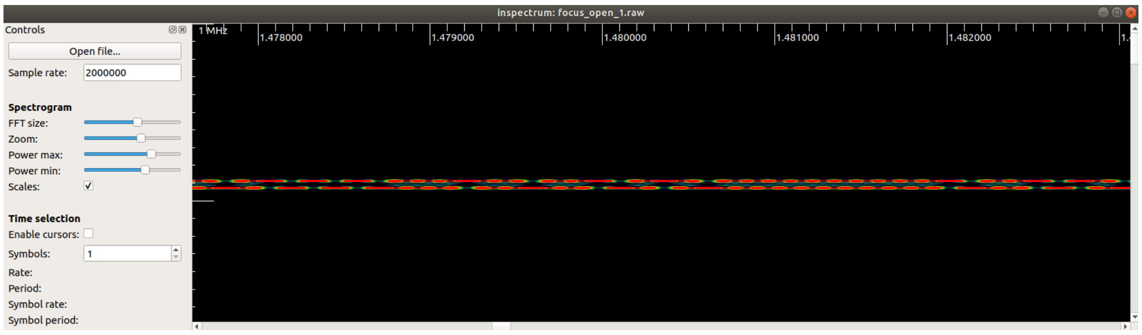
6.4.1 Reverse engineering

We were unable to find any documentation on `https://fccid.io/`. Therefore all statements made in the following section are based solely on the analysis of the transmitted signal by the key fob since we cannot compare them to any official documentation of the manufacturers.

1. **Signal analysis**: The first thing that immediately caught the eye when capturing the signal and visualizing it using a “Frequency Sink” was the fact that each button press seems to generate 3 different peaks in the frequency spectrum. After analysing the signal in Inspectrum, we notice that the key fob transmits 6 bursts on 3 different carrier frequencies. The 3 different carrier frequencies correspond to the 3 different peaks in the frequency spectrum. The first and fourth burst is transmitted at a carrier frequency of 433.589 MHz, the second and fifth burst is transmitted at a carrier frequency of 433.92 MHz while the third and sixth burst is transmitted at a carrier frequency of 434.251 MHz. However, When we zoom in to a single burst we notice that a single burst is transmitted using 2 different carrier frequencies which alter each other. As for the Mazda MX5, this indicates the signal is modulated using a Frequency-Shift Keying modulation.

   Furthermore, we notice that the 6 bursts are not identical in length. The first 3 transmitted bursts are noticeably longer than the last 3 transmitted bursts. When visually comparing the long bursts with the short bursts, it seems that the short bursts contain the same amount of data symbols as the long bursts. The only difference is the length of the preamble. The preamble length of the long bursts is 1614 symbols while the short bursts only have a preamble length of 64 symbols. Below a summary of the signal properties:

   - Symbol rate: 15.72 kHz
   - Long bursts (first 3 bursts):
     - Preamble length: 1614 symbols
     - Data length: 272 symbols
     - Center frequency burst 1: 433.589 MHz (433.57 MHz & 433.61 MHz)
     - Center frequency burst 2: 433.920 MHz (433.90 MHz & 433.94 MHz)



Controls

Open file...

Sample rate: 2000000

Spectrogram

FFT size:

Zoom:

Power max:

Power min:

Scales: ✓

Time selection

Enable cursors:

Symbols: 1

Rate:

Period:

Symbol rate:

Symbol period:

Figure 6.10: Single burst of the 2018 Ford Focus key fob visualised in Inspectrum.

  - Center frequency burst 3: 434.251 MHz (434.23 MHz & 434.27 MHz)

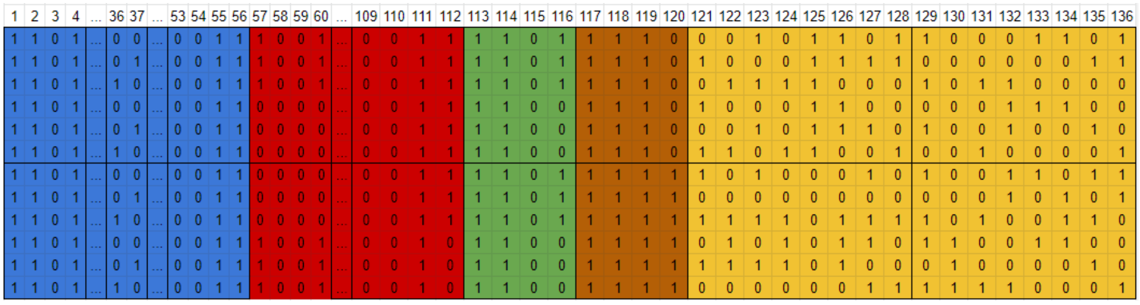
- Short bursts (last 3 bursts):
  - Preamble length: 64 symbols
  - Data length: 272 symbols
  - Center frequency burst 4: 433.589 MHz (433.57 MHz & 433.61 MHz)
  - Center frequency burst 5: 433.920 MHz (433.90 MHz & 433.94 MHz)
  - Center frequency burst 6: 434.251 MHz (434.23 MHz & 434.27 MHz)

2. **Determine modulation technique**: As mentioned in the previous step, for each burst, 2 different frequencies alter each other. This means that the transmitted data is represented by the change of the carrier frequency and therefore means a Frequency-Shift Keying modulation is used (see Figure 6.10).

3. **Determine data encoding method**: Just like in the 2 previous case studies, the data encoding method is a variant of the Manchester data coding. Using the same argument as for the Ford Fiesta's and Mazda MX5's key fob, there are only two different lengths of symbols: A short symbol and a longer symbol which is twice the size of the shorter symbol. This indicates that there is a forced transition in the middle of each bit causing there can never be more than 2 high level symbols in a row.

4. **Demodulating and decoding the signal**: A custom Frequency-Shift Keying demodulator and Manchester decoder is written to demodulate and decode the Ford Focus 2018 key fob signal. In contrast to the previous case study, the symbol rate is constant which makes it easier to create a demodulator/decoder. The Frequency-Shift Keying modulated signal is interpret as an On-off Keying modulated signal by ignoring the lower frequency symbols. Unlike the previous case study, we can re-use the On-off Keying demodulator block from the Ford Fiesta demodulator 6.2.2. The complete demodulator including Manchester decoder is explained in Section 6.4.2.

5. **analyse demodulated and decoded binary data**: Figure 6.11 shows the 6 bursts of 2 consecutive button presses. As mentioned before, the Ford Focus key fob transmits 6 bursts (3 long bursts followed by 3 short bursts) on 3 different carrier frequencies. However, the length of the bursts is only affected by the length of the preamble. When excluding the preamble, all bursts consists of 17 bytes of data. The first 7 bytes are equal for all transmitted bursts and for each transmitted signal expect for 2 bits (blue). The fourth and fifth bit of the fifth byte seems to change for each burst (bit 36 & 37). The fifth byte of the first, second and third transmitted long burst are respectively "00000000", "00001000" and "00010000". This sequence is repeated for the next 3 short bursts and the same for each transmitted signal. In other words, the fifth byte seems to correspond to the carrier frequency used to transmit the burst. As the first 7 bytes are static expect from 2 bits, it is most likely the unique car identifier.



1	2	3	4	...	36	37	...	53	54	55	56	57	58	59	60	...	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136
1	1	0	1	...	0	0	...	0	0	1	1	1	0	0	1	...	0	0	1	1	1	1	0	1	1	1	1	0	0	0	1	0	1	1	0	1	1	0	0	0	1	1	0	1
1	1	0	1	...	0	1	...	0	0	1	1	1	0	0	1	...	0	0	1	1	1	1	0	1	1	1	1	0	1	0	0	0	1	1	1	1	0	0	0	0	0	0	1	1
1	1	0	1	...	1	0	...	0	0	1	1	1	0	0	1	...	0	0	1	1	1	1	0	1	1	1	1	0	0	1	1	1	1	0	0	0	1	0	1	1	0	0	0	0
1	1	0	1	...	0	0	...	0	0	1	1	0	0	0	0	...	0	0	1	1	1	1	0	0	1	1	1	0	1	0	0	0	1	1	0	0	0	0	0	1	1	1	0	0
1	1	0	1	...	0	1	...	0	0	1	1	0	0	0	0	...	0	0	1	1	1	1	0	0	1	1	1	0	0	0	1	0	1	1	1	0	1	0	0	1	0	0	1	0
1	1	0	1	...	1	0	...	0	0	1	1	0	0	0	0	...	0	0	1	1	1	1	0	0	1	1	1	0	1	1	0	1	1	0	0	1	0	0	1	0	0	0	0	1
1	1	0	1	...	0	0	...	0	0	1	1	0	0	0	0	...	0	0	1	1	1	1	0	1	1	1	1	1	1	0	1	0	0	0	1	0	1	0	0	1	1	0	1	1
1	1	0	1	...	0	1	...	0	0	1	1	0	0	0	0	...	0	0	1	1	1	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1
1	1	0	1	...	1	0	...	0	0	1	1	0	0	0	0	...	0	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	0	1	1	1	1	0	1	0	0	1	1	0
1	1	0	1	...	0	0	...	0	0	1	1	1	0	0	1	...	0	0	1	0	1	1	0	0	1	1	1	1	0	1	0	1	0	1	1	0	1	1	0	0	1	1	0	0
1	1	0	1	...	0	1	...	0	0	1	1	1	0	0	1	...	0	0	1	0	1	1	0	0	1	1	1	1	1	1	1	1	0	1	0	0	0	1	0	0	0	0	1	0
1	1	0	1	...	1	0	...	0	0	1	1	1	0	0	1	...	0	0	1	0	1	1	0	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	1

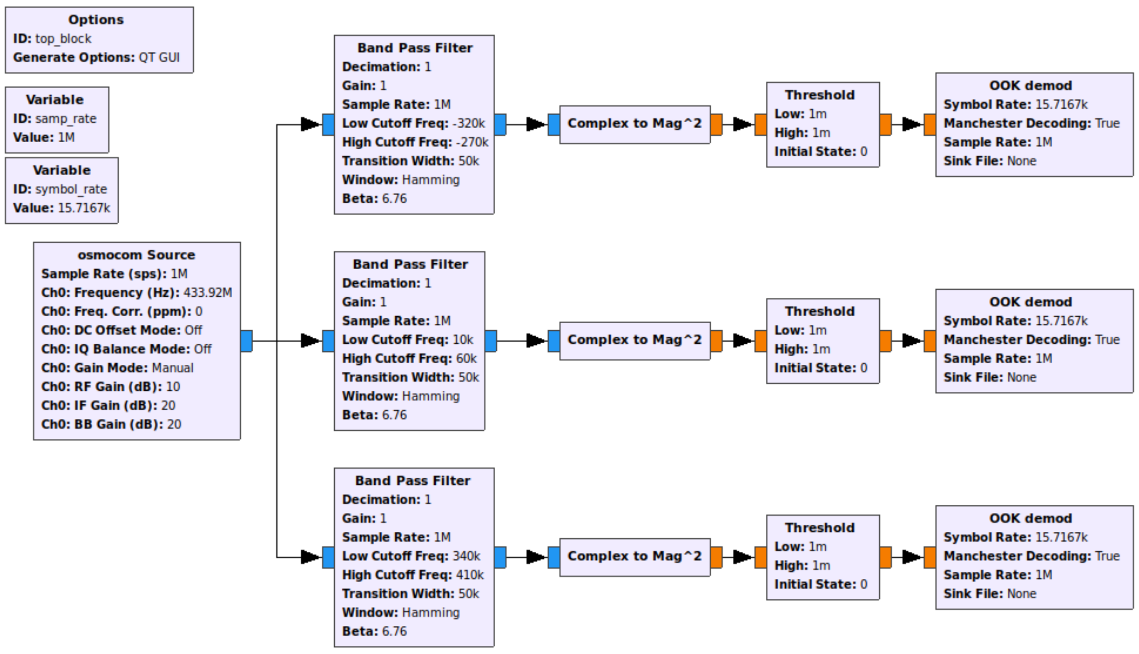
Figure 6.11: Message format for the Ford Focus key fob.

The remaining 10 bytes are actually quite similar to the Ford Fiesta's transmitted data. The actual rolling code is represented by the first 7 bytes (red). Strangely, a signal consists of 2 different rolling codes. The rolling code represented in the first 3 long bursts is different from the rolling code represented in the 3 short bursts. Why exactly the Ford Focus key fob uses 2 different rolling codes per transmitted signal is unknown. The next byte represents both the command and a counter, just like for the Ford Fiesta. The 4 most-significant bits (green) indicates the command but are at the same time part of the counter represented by the 4 least-significant bits (orange). In other words, the 4 most-significant bits are part of the counter but are different for each command while the 4 least-significant bits are shared between different commands and incremented for each button press independent of the pressed button. When the 4 least-significant bits reach the "1111" state, the next button press will increment all command bits by 1 while the 4 least-significant bits go back to state "0000". On top of that, the 4th most-significant bit flips between the 3 long bursts and 3 short bursts (bit 116). The last 2 bytes are different for each transmitted signal and each burst (yellow). We were unable to find any correlation between the 2 remaining bytes. Therefore they are most likely used for data integrity in the form of a Message Authentication Code (MAC).

Last, we tried to exploit our discovered vulnerability in which the car ignores all rolling code for a certain amount of time upon receiving a valid signal (see Section 4.3). After testing the exploit, it seems that the Ford Focus is not vulnerable. In addition to the car not being vulnerable to the exploit, it seems there is a built-in defense against replay attacks. Upon receiving a retransmitted signal with an invalid rolling code, the car will disable its RKE system for 5 minutes. This means that new valid signals from the original key fob are also ignored by the receiver's unit and the only way to unlock the door is to use the actual mechanical key. Also, it seems that the Ford Focus is not the only car in which the RKE system temporally disables. After performing the exploit on a 2012 Ford C-MAX, it also disabled the RKE system. However, after waiting for hours, it seemed that the car kept its RKE system disabled. Only after unlocking the door with the mechanical spare key (not the actual mechanical key of the primary key fob) the RKE system of the car was re-enabled.

6.4.2 Implementation: Frequency-Shift Keying demodulator

As mentioned before, we can demodulate the Frequency-Shift Keying modulated signal using an On-off Keying demodulator. In contrast to the Mazda MX5, we can re-use the "OOK demod" block from the Ford Fiesta case study since we have a constant symbol rate. However, we cannot re-use the complete On-off Keying demodulator as the key fob transmits the signal on 3 different carrier frequencies. A solution to this problem is using a separate block sequences for each carrier frequency as seen in Figure 6.12. The "Band Pass Filter" block filters each individual signal transmitted on a different carrier frequency and transforms the Frequency-Shift Keying modulated signal to an On-off Keying modulated signal by attenuating the lower transmit frequency. Each frequency range containing one of the 3 carrier frequencies passes through a separated demodulator/decoder block that demodulates/decodes the signal and outputs the binary result. As the flow graph has to do 3 times the amount of work compared to the other demodulates/decoders, it might be possible



Options
ID: top_block
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 1M

Variable
ID: symbol_rate
Value: 15.7167k

osmocom Source
Sample Rate (sps): 1M
Ch0: Frequency (Hz): 433.92M
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 10
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 1M
Low Cutoff Freq: -320k
High Cutoff Freq: -270k
Transition Width: 50k
Window: Hamming
Beta: 6.76

Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 1M
Low Cutoff Freq: 10k
High Cutoff Freq: 60k
Transition Width: 50k
Window: Hamming
Beta: 6.76

Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 1M
Low Cutoff Freq: 340k
High Cutoff Freq: 410k
Transition Width: 50k
Window: Hamming
Beta: 6.76

Complex to Mag^2

Threshold
Low: 1m
High: 1m
Initial State: 0

OOK demod
Symbol Rate: 15.7167k
Manchester Decoding: True
Sample Rate: 1M
Sink File: None

Figure 6.12: Frequency-Shift Keying demodulator for the Ford Focus.

that the CPU cannot handle the work load. In case of the Intel Core i5-4210M CPU, it could not handle 2 million S/s. After lowering the sample rate, the CPU was able to cope with the work load and was able to successfully demodulate and decode incoming signals. For a complete explanation of all the used blocks and defined variables, see Section 6.2.2.

Chapter 7

Conclusion

In this thesis we provided an overview of the past and the present of keyless entry systems in the automobile industry. The main focus lies on various exploitations of both RKE and PKE(S) systems. For decades it seemed that RKE systems were secure, until in 2014 the replay & jamming attack was invented that made it possible to bypass rolling code. As seen in Section 4.2, we were able to successfully implement a fully automated replay & jamming attack using software-defined radios. The implementation was able to successfully bypass rolling code in 4 out of the 7 test cars. We were unable to exploit 3 cars due to the implementation of their RKE system. In these 3 cases, the car's receiver did not leave any available spectrum for adversaries to transmit their jamming signal on. Therefore, adversaries are unable to disturb the car's receiver while capturing a valid signal themselves. This can be achieved by transmitting a signal that occupies the complete frequency passband of the receiver (2018 Ford Focus & 2012 Ford C-MAX) or by allocating a very small frequency passband that is just large enough to receive the signal of the key fob (2018 Mazda CX5).

Implementing a replay & jamming attack using software-defined radios is typically more expensive than using small dedicated hardware modules (f.e. the $32 device created by Samy Kamkar). Therefore, we looked into ways for reducing the cost of the attack. The main cost comes from the transmitting software-defined radio as they typically cost over €200. However it is possible to replace this software-defined radio by utilising the General Purpose Clock pin (GPIO4) of the Raspberry Pi using *rpitx*. As seen in Section 4.4, we were able to implement a replay & jamming attack using *rpitx*. However, the success rate was significantly lower compared to using a dedicated transmitting software-defined radio. Because the GPIO4 pin of the Raspberry Pi was not created with the intention of transmitting radio frequency signals, it is not optimised for signal transmission and therefore transmits a lot of unintentional noise. Due to transmission of the noise together with harmonics caused by square waves, it is hard to correctly jam the car's receiver while not triggering the implementation's automated signal detection. After implementing thresholding and manually configuring the frequency passband, jamming frequency and length of the antenna, we were able to successfully execute the attack against 1 out of 7 test cars. However, performing this replay & jamming attack in a real crime scenario is not recommended and most likely impossible as the adversary can not experimentally configure their implementation (frequency passband, jamming frequency, length of antenna, ...) as he has no access to the key fob of the victim.

To improve our implementation we tried implementing reactive jamming. A requirement to successfully perform reactive jamming is that the delay between detecting the target signal and transmitting the jamming signal is as close to zero as possible. However, a disadvantage of using software-defined radios is the fact that processing and transferring of data between the device and computer causes a significant delay. In our implementation this delay could reach up to 900 milliseconds which made it impossible to transmit the jamming signal in time. However, it is possible to reduce this delay significantly by modifying the firmware of the software-defined radio. By performing the signal processing on the software-defined radio's microcontroller itself, the transfer of data between the device and computer is eliminated.

The main reason why most RKE protocols are vulnerable to replay & jamming attacks is because they do not provide any information regarding the age of a message. However, at around the same time the attack was invented, KeeLoq Technology, one of the most used hardware-dedicated block ciphers in RKE systems released their new technology called "Ultimate KeeLoq". Ultimate KeeLoq is a time-driven keyless entry protocol that utilises time-stamps to provide the receiver with information regarding the age of a message. Doing so, the receiver can reject out-dated messages and therefore prevent against replay & jamming attacks. We tested 6 cars including 2 cars that were released after 2014 and came to the conclusion that none of the cars implemented such time-driven protocol yet. Therefore, the vast majority of today's cars on the road are most likely still vulnerable. However as seen in the first paragraph, it is not necessary to implement time-based messages in order to defend against replay & jamming attacks.

Next, we tried to exploit cars equipped with PKE(S) systems by implementing a signal amplification relay attack. The main concern is the delay introduced by processing and transferring of data between the software-defined radio and computer. As we compared different methods for relaying data, we came to the conclusion that relaying the LF signal over an UHF channel will be the most time efficient method as relaying it over Wi-Fi or an Ethernet cable introduced an additional transport layer (TCP or UDP) causing extra delay. As noted in Section 5.4, we were unable to test our implementation due to problems regarding the shipping of a required hardware component. However, as the introduced delay by relaying the data is the most important factor in performing a successful attack, we can measure the delay of our implementation using a regular 433.92 MHz RKE key fob signal. The delay introduced by the first device (capturing the LF signal and transmitting it over an UHF channel) is already 900 milliseconds. As the second relay device (located close to the key fob) has to do the same process (capturing the 2.5 GHz signal and transmitting it over a LF channel), an additional delay of 900 milliseconds is introduced. Taking the time into consideration that is required to relay the signal "over the air", we have a total delay of around 2 seconds. As Francillon et al. [21] tested 10 different car models in which the highest acceptance delay was 20 milliseconds, it is impossible to perform our signal amplification relay attack using out-of-the-box software-defined radios. However as Francillon experienced the same problem, they were able to successfully perform the attack by modifying the FPGA of their USRP. This problem is also related to the problem seen in Section 4.4 (Reactive jamming). By modifying the firmware of the SDR, it is possible to eliminate the transfer of data between the SDR and computer and therefore reduce the delay to under 20 milliseconds.

For the past years, media has covered dozens of stories regarding the vulnerabilities of PKE(S) systems. Since then, car manufacturers are fully engaged in finding solutions and improving their security. As noted in Section 2.4.2, 8 out of 18 tested cars released in 2019 implement extra security measures that defend against relay attacks. As it is not revealed for all car models what exact countermeasures they implement, 3 out of those 8 cars utilise a key fob equipped with a motion sensor (the other car models might do as well but this was not disclosed). Motion sensor key fobs are not a permanent solution against relay attacks as it does not solve the essence of the problem. Nevertheless, it is a step in the right direction.

When relay attacks were first introduced to the automobile industry, the main proposal for solving the problem was the implementation of distance-bounding protocols. Using distance-bounding protocols, it is possible to precisely calculate the distance between the key fob and car using the RTT and the transfer speed (speed of light) of the signal. However, we we're unable to find any information or statistics regarding the implementation of distance-bounding protocols in PKE(S) systems.

Besides creating our own implementation for these attack, we tried obtaining a better knowledge of how RKE protocols work. We tried to obtain knowledge of the physical (modulation, data encoding, ...) and MAC (message format) layer of RKE protocols by reverse engineering radio frequency signals transmitted by the key fob. We reverse engineered 3 case studies: A 2008 Mazda MX5, 2015 Ford Fiesta and a 2018 Ford Focus. Additional to these 3 case studies we also visually observed multiple other key fob signals (2012 Ford C-MAX, 2018 Mazda CX5, 2008 Mercedes-Benz B-class, 2008 Mazda 2) to obtain knowledge about their physical layer.

All 7 different car models expect from the 2008 Mazda MX5 and Mazda 2 use a different approach to structure their signals. Both the 2012 Ford C-MAX and 2018 Ford Focus utilise the same technique of transmitting signals at 3 different carrier frequency but their internal message structure is different. Remarkable is the fact that the 2015 Ford Fiesta which was released between the Ford C-MAX and Ford Focus did use a complete different modulation and structure compared to the 2 other cars. All key fobs modulated their data using Frequency-Shift Keying expect from the 2015 Ford Fiesta that uses an Amplitude-Shift keying modulation called On-off Keying. For encoding their date, they all used a Manchester coding expect from some preambles using a different encoding (typically a regular NRZ-L). The message format of the RKE protocols in our 3 case studies all differed from each other. However for the 2015 Ford Fiesta and 2018 Ford Focus, we could see some reoccurring data fields. For example the combined counter and command bits and the message ending with a data integrity field.

The automobile industry is taking steps in the right direction to improve the security of their PKE(S) systems. The intense media exposure forced the automobile industry to solve the concerned problems as quickly as possible. As every year more and more cars are becoming resistant against these relay attacks, it is most likely only a matter of time before all cars equipped with PKE(S) are secure. However, because there is little to no media exposure regarding replay & jamming attacks, there is little to no pressure on the automobile industry to improve their security of their RKE systems. As there are already solution on the market (f.e. Ultimate KeeLoq), new security protocols do not have to be invented from scratch. However, it seems that the automobile industry is ignoring the fact that criminals can fraudulently gain access to cars by exposing weaknesses in their RKE protocols.

Finding new exploitations in keyless entry systems and solving them will always be “a game of cat and mouse” between hackers or security researchers and the automobile industry. There is little chance that after solving the problem regarding these attack, both RKE and PKE(S) systems will be secure forever. Hackers and security researchers will always actively search for vulnerabilities in keyless entry systems in order to exploit them. However, this is not a bad thing as it puts pressure on the automobile industry to keep doing research and improving their security.

Chapter 8

Future work

As for the replay & jamming attack, we were unable to execute the attack using reactive jamming due to delay introduced by processing and transferring of the data between the SDR and computer. In order to reduce this delay, we have to modify the firmware of the SDR to perform the signal processing on the device's microcontroller itself. Doing so, the SDR and computer are no longer required to transfer data between each other as the processing is done on the SDR instead of the computer. Due to the limited time, we were unable implement a custom firmware for the HackRF One.

We were able to reduce the cost of the replay & jamming attack by eliminating the transmitting SDR by the General Purpose Clock pin of a Raspberry Pi using *rpitx*. However doing so, the success rate dropped significantly compared to using a dedicated SDR for transmitting. This was due to the noise and harmonics transmitted by *rpitx*. However, it is possible to attenuate the noise and harmonics by using low pass filters. Alternatively, we can change the implementation of the signal detection. Instead of triggering based upon the amplitude of a sample, we can base it upon detecting a specific preamble. This way the noise and/or harmonics won't trigger the signal detection. However, the attack will only work on RKE protocols that use the exact preamble.

We were unable to successfully perform a signal amplification relay attack against PKE(S) systems. PKE(S) do not yet implement any distance-bounding protocols. However, there is still a minimum delay required in order to successfully perform the attack. Identical to reactive jamming in the replay & jamming attack, the delay caused by processing and transferring of data between the SDR and computer exceeded the maximum delay in order to successfully perform the relay attack. This delay can significantly be reduced by modifying the firmware of the SDR to eliminate the transfer of data between the SDR and computer.

Appendices

Appendix A

Raspberry Pi and GNU Radio installation guide

This guide shows the installation of required packages on the Raspberry Pi in order to perform the replay & jamming attack in Section 4 and signal amplification relay attack in 5. First we'll start with installing the Ubuntu Mate operating systems on the Raspberry Pi. Due to the Python corrupted memory errors when using the "Osmocom" blocks on Raspbian cited in Section 4.1.1, we'll be using Ubuntu Mate as the operating system running on our Raspberry Pi's. Ubuntu Mate provides a specific version suited for the Raspberry Pi which can be downloaded on there website `https://ubuntu-mate.org/download/`. Once the image is downloaded, we have to flash the SD card. For flashing the Ubuntu Mate image to our SD card, we use Balena Etcher `https://www.balena.io/etcher/`. Balena Etcher is an open source tool which enables you to flash a SD card with a single click and automatically validates the flashing.

We'll be using *apt-get* for installing GNU Radio and other required packages. First start with updating apt-get to get the latest version via `sudo apt-get update`. First we'll be installing GNU Radio via `sudo apt-get install gnuradio` (this can take some time). After installing GNU Radio we have to install the required packages to enable the use of the RTL-SDR, HackRF and the "osmocom" blocks (gr-osmosdr). Before building gr-osmosdr make sure that all the dependencies you are intended to work with are properly installed. The build system of gr-osmosdr will recognise them and enable specific source/sink components thereafter. In our case this is the RTL-SDR and HackRF dependencies. Therefore, make sure to first install these dependencies before building gr-osmosdr. Before installing we'll first have to acquire some additional packages. We'll need git and cmake to download and build the RTL-SDR, hackRF One and gr-osmosdr git packages. We also need some additional packages in order to correctly install gr-osmosdr. Note that these packages are not listed as required in the official installation guide of gr-osmosdr. However, we were unable to correctly build gr-osmosdr without them. Install all required packages via `sudo apt-get install git cmake pkg-config libusb-1.0-0-dev libosmosdr-dev swig doxygen` (libusb-1.0-0-dev is required for RTL-SDR). First we'll be installing RTL-SDR. Clone the RTL-SDR git repository via `git clone git://git.osmocom.org/rtl-sdr.git` and build it via cmake:

```
cd rtl-sdr/
mkdir build
cd build
cmake ../
make
sudo make install
sudo ldconfig
```

Next, we'll be installing the required HackRF package. Clone the HackRF git repository via `git clone https://github.com/mossmann/hackrf.git` and build it via cmake in the same exact manner shown above (`cd hackrf/host` instead of `cd rtl-sdr/`).

When both the RTL-SDR and HackRF packages are installed, we can install the gr-osmosdr package. Clone gr-osmosdr via `git clone git://git.osmocom.org/gr-osmosdr` and build it using cmake as shown above (`cd gr-osmosdr/` instead of `cd rtl-sdr/`). When building gr-osmosdr, it will output the enabled and disabled components. Make sure the RTL-SDR and HackRF are listed as enabled components. If not, the build system was unable to detect the RTL-SDR and HackRF dependencies.

Appendix B

Setting up wireless AP on Raspberry Pi (Ubuntu Mate)

Setting up the access point is mainly based on [1]. Setting up and configuring an access point on Ubuntu Mate is almost identical to Raspbian. The only difference is that Ubuntu version 18.04 started using Netplan as its network configurator instead of the static interfaces file (/etc/network/interfaces). Netplan uses a YAML description for configuring the network interfaces. The configuration file used to set up the access point is shown below:

```
# /etc/netplan/01-network-manager-all.yaml
network:
    version: 2
    renderer: networkd
    ethernets:
        wlan0:
            dhcp4: no
            addresses: [192.168.10.1/24]
```

This YAML file replaces the static network configuration originally used in Ubuntu Mate and Raspbian:

```
# /etc/network/interfaces
allow-hotplug wlan0
iface wlan0 inet static
    address 192.168.10.1
    netmask 255.255.255.0
    broadcast 192.168.10.255
    network 192.168.10.0
```

We use “hostapd” to enable the Raspberry Pi’s network interface card to act as an access point. Last, we use “isc-dhcp-server” which automatically assigns an IP address to the connecting devices. For the complete configuration guide see [1].

Bibliography

[1] Kyle Abad. Install wi-fi hotspot on raspberry pi. https://www.diyhobi.com/install-wifi-hotspot-raspberry-pi-ubuntu-mate/. [Online; accessed 23-July-2019].

[2] Aysajan Abidin, Eduard Marin, Dave Singelée, and Bart Preneel. Towards quantum distance bounding protocols. In *International Workshop on Radio Frequency Identification: Security and Privacy Issues*, pages 151–162. Springer, 2016.

[3] Vivek Acharya. *TCP/IP and distributed systems.* Laxmi Publications Pvt. Ltd, 1 edition, 2006.

[4] Atmel. Avr411: Secure rolling code algorithm for wireless link. http://ww1.microchip.com/downloads/en/AppNotes/Atmel-2600-AVR411-Secure-Rolling-Code-Algorithm-for-Wireless-Link_Application-Note.pdf, 2015. [Online; accessed 13-February-2019].

[5] Atmel. Manchester coding basics. http://ww1.microchip.com/downloads/en/AppNotes/Atmel-9164-Manchester-Coding-Basics_Application-Note.pdf, 2015. [Online; accessed 8-January-2019].

[6] OBDII AutoTap. Dedicated to helping the home and independent technician understand and use obd-ii technology. http://www.obdii.com/, 2013. [Online; accessed 11-December-2018].

[7] Loutfi Belghmidi. Dieventruc overgewaaid uit de vs: autosleutels hacken blijkt kinderspel. https://www.vrt.be/vrtnws/nl/2018/03/29/dieven-hacken-sleutels-en-stelen-auto-s-/, 2018. [Online; accessed 03-August-2019].

[8] Bastian Blössl. gr-keyfob. https://github.com/bastibl/gr-keyfob. [Online; accessed 26-July-2019].

[9] Marshall Brain. How remote entry works. https://auto.howstuffworks.com/remote-entry2.htm. [Online; accessed 08-February-2019].

[10] Stefan Brands and David Chaum. Distance-bounding protocols. In *Workshop on the Theory and Application of Cryptographic Techniques on Advances in Cryptology*, EUROCRYPT '93, pages 344–359, Berlin, Heidelberg, 1994. Springer-Verlag.

[11] Redactie Carros. Stichting Aanpak Voertuigcriminaliteit: circa 25.000 auto-inbraken per jaar via keyless entry. https://www.carros.nl/videos/stichting-aanpak-voertuigcriminaliteit-circa-25-000-autodiefstallen/per-jaar-via-keyless-entry, 2018. [Online; accessed 03-August-2019].

[12] Elliot J. Chikofsky and James H Cross. Reverse engineering and design recovery: A taxonomy. *IEEE software*, 7(1):13–17, 1990.

[13] Chipkin. Can bus benefits and applications. https://store.chipkin.com/articles/can-bus-benefits-and-applications. [Online; accessed 06-February-2019].

[14] Citroën. Keyless entry and start. https://www.citroen.co.uk/about-citroen/technology/keyless-entry-and-start, 2019. [Online; accessed 04-June-2019].

[15] R Currie. Hacking the can bus: basic manipulation of a modern automobile through can bus reverse engineering. *SANS Institute*, 2017.

[16] eKeyfobs. Ford remote key fob programming instructions - how to. `https://www.youtube.com/watch?v=mG3R58-24QU`, August 2013. [Online; accessed 12-August-2019].

[17] CSS Electronics. Can bus explained - a simple intro (2018). `https://www.csselectronics.com/screen/page/simple-intro-to-can-bus#CAN-Bus-Intro-Dummies-Basics`, 2018. [Online; accessed 11-December-2018].

[18] Mark A. Finlayson. *User Manual / Functional Description of the Siemens VDO Radio Frequency Transmitter*. Siemens VDO Automotive AG.

[19] Foo-Manroot. Impersonating a remote using sdr and gnuradio. `https://foo-manroot.github.io/post/gnuradio/sdr/2018/01/15/gnuradio-ook-transmit.html`. [Online; accessed 22-February-2019].

[20] Ford. Ford keyfree-systeem met ford power-startknop. `https://www.nl.ford.be/shop/ontdekken/technologie/comfort/ford-keyfree-systeem`, 2019. [Online; accessed 04-June-2019].

[21] Aurélien Francillon, Boris Danev, and Srdjan Capkun. Relay attacks on passive keyless entry and start systems in modern cars. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Eidgenössische Technische Hochschule Zürich, Department of Computer Science, 2011.

[22] Lishoy Francis, Gerhard Hancke, Keith Mayes, and Konstantinos Markantonakis. Practical nfc peer-to-peer relay attack using mobile phones. In *International Workshop on Radio Frequency Identification: Security and Privacy Issues*, pages 35–49. Springer, 2010.

[23] Great Scott Gadgets. Hackrf one. `https://greatscottgadgets.com/hackrf/`, 2014. [Online; accessed 11-December-2018].

[24] Flavio D Garcia, David Oswald, Timo Kasper, and Pierre Pavlidès. Lock it and still lose it—on the (in) security of automotive remote keyless entry systems. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016. [Online; accessed 11-August-2019].

[25] Research Gate. Digital signal processing. `https://www.researchgate.net/figure/Digital-modulation-schemes-ASK-FSK-and-PSK_fig3_303471153`, 2004. [Online; accessed 13-December-2018].

[26] Gene. Tesla adds 'passive entry' to combat vehicle theft by keyless-entry exploit. `https://www.teslarati.com/tesla-passive-entry-ota-update-prevent-keyless-entry-theft/`, August 2017. [Online; accessed 12-August-2019].

[27] Miek GitHub. Inspectrum. `https://github.com/miek/inspectrum`. [Online; accessed 11-December-2018].

[28] GNURadio. Gnuradio. `https://wiki.gnuradio.org/index.php/Main_Page`. [Online; accessed 11-December-2018].

[29] Andy Greenberg. This Hacker's Tiny Device Unlocks Cars And Opens Garages. `https://www.wired.com/2015/08/hackers-tiny-device-unlocks-cars-opens-garages/`, 2015. [Online; accessed 11-December-2018].

[30] Gerhard P Hancke and Markus G Kuhn. An rfid distance bounding protocol. In *First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM'05)*, pages 67–73. IEEE, 2005.

[31] Hyundai. 2018 hyundai accent specs features. `https://www.cocoahyundai.com/blog/2018-hyundai-accent-specs-features/`, 2019. [Online; accessed 04-June-2019].

[32] Sebastiaan Indesteege, Nathan Keller, Orr Dunkelman, Eli Biham, and Bart Preneel. A practical attack on keeloq. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–18. Springer, 2008.

[33] Insurance Information Institute. Facts + Statistics: Auto theft. `https://www.iii.org/table-archive/21263`, 2017. [Online; accessed 03-August-2019].

[34] Maxim integrated. Designing Remote Keyless Entry (RKE) Systems. `https://www.maximintegrated.com/en/app-notes/index.mvp/id/1773`, 2002. [Online; accessed 10-December-2018].

[35] Luma W Jameel. Manchester coding and decoding generation theortical and expermental design. *American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS)*, 42(1):130–138, 2018.

[36] Jasper Jolly. Most of uk's top-selling keyless cars at risk of theft, says which? `https://www.theguardian.com/money/2019/jan/28/uk-keyless-cars-theft-which-ford-fiesta-vw-golf-nissan-qashqai-focus`, 2019. [Online; accessed 30-May-2019].

[37] Allan Jones, David Chapman, Helen Donelan, Laurence Dooley, , and Adrian Poulton. Exploring communications technology. `https://www.open.edu/openlearn/science-maths-technology/exploring-communications-technology/content-section-1.4`. [Online; accessed 8-January-2019].

[38] Samy Kamkar. Drive it like you hacked it. `https://samy.pl/defcon2015/`, 2015. [Online; accessed 11-December-2018].

[39] Samy Kamkar. OpenSesame. `https://samy.pl/opensesame/`, 2015. [Online; accessed 06-August-2019].

[40] Dr. Jedidi Kamouaa. Turn-key passive entry/passive start solution, 2008. [Online; accessed 11-August-2019].

[41] Chris Kowalczyk. Meet-in-the-middle attack. `http://www.crypto-it.net/eng/attacks/meet-in-the-middle.html`. [Online; accessed 30-May-2019].

[42] Max Maass, Uwe Müller, Tom Schons, Daniel Wegemer, and Matthias Schulz. Nfcgate: An nfc relay application for android. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec '15, New York, NY, USA, 2015. ACM.

[43] Eduard Marin, Lennert Wouters, and Tomer Ashur. Fast, Furious And Insecure: Passive Keyless Entry And Start In Supercars. `https://www.esat.kuleuven.be/cosic/fast-furious-and-insecure-passive-keyless-entry-and-start-in-modern-supercars`, 2018. [Online; accessed 11-December-2018].

[44] Willi Meier and Othmar Staffelbach. Fast correlation attacks on stream ciphers. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 301–314. Springer, 1988.

[45] Microchip. KEELOQ® Code Hopping Encoder: HCS201. `http://ww1.microchip.com/downloads/en/DeviceDoc/41098D.pdf`, 2011. [Online; accessed 14-February-2019].

[46] Charlie Miller and Chris Valasek. Adventures in automotive networks and control units. `https://ioactive.com/pdfs/IOActive_Adventures_in_Automotive_Networks_and_Control_Units.pdf`, 2013.

[47] Charlie Miller and Chris Valasek. Remote exploitation of an unaltered passenger vehicle. `http://illmatics.com/Remote%20Car%20Hacking.pdf`, 2015.

[48] James Mills. Keyless wonder: How did we end up with 'smart(wireless keys for our cars? `https://www.driving.co.uk/news/features/keyless-wonder-how-did-we-end-up-with-smart-keys-for-our-cars/`, 2014. [Online; accessed 03-August-2019].

[49] Tesla Notes. Tesla v8.1 2018.4.1 change log. http://teslanotes.com/docs/V81/2018-1-21-2018.4.1/, January 2018. [Online; accessed 12-August-2019].

[50] Nihal Pasham. Rf reverse engineering has become trivial — thanks to the 'opensource sdr' movement. https://medium.com/@nihal.pasham/rf-reverse-engineering-has-become-trivial-thanks-to-the-opensource-sdr-movement-d1f9216f2f04. [Online; accessed 26-July-2019].

[51] Thatcham Research. Closing down car security loopholes. https://www.thatcham.org/what-we-do/security/consumer-rating/, 2019. [Online; accessed 12-August-2019].

[52] Pieter Robyns. Network design and application, lecture notes: The wireless physical (phy) layer, December 2018.

[53] Margaret Rouse. phase-shift keying (psk). https://searchnetworking.techtarget.com/definition/phase-shift-keying, 2005. [Online; accessed 8-January-2019].

[54] RTL-SDR.com. About rtl-sdr. https://www.rtl-sdr.com/about-rtl-sdr/. [Online; accessed 11-August-2019].

[55] Taylor Simmons. Auto theft on the rise in Toronto area, and a security expert thinks he knows why. https://www.cbc.ca/news/canada/toronto/car-thefts-rising-1.4930890?mc_cid=57dc5c4db9&mc_eid=bea9ac3767, 2018. [Online; accessed 11-December-2018].

[56] Craig Smith. *The Car Hacker's Handbook: A Guide for the Penetration Tester*. No Starch Press, San Francisco, CA, USA, 1st edition, 2016.

[57] Luke John Smith. Car theft crisis: Number of keyless entry cars stolen rise - is your car at risk? https://www.express.co.uk/life-style/cars/1122322/car-theft-stolen-keyless-entry-hack-your-cars-risks, 2019. [Online; accessed 30-May-2019].

[58] Luke John Smithj. Car theft crimewave epidemic in the UK - Your keyless car could be stolen in 23 seconds. https://www.express.co.uk/life-style/cars/1048780/car-theft-keyless-entry-hack-stolen-seconds-UK-epidemic, 2018. [Online; accessed 13-January-2019].

[59] SpinCore. Implementation of analog modulation on spincore pulseblasterdds and radioprocessor boards. https://www.spincore.com/products/PulseBlasterDDS-300/Modulation_Techniques/analog_modulation.shtml. [Online; accessed 3-June-2019].

[60] Francois-Xavier Standaert and Jean-Jacques Quisquater. Time-memory tradeoffs. https://perso.uclouvain.be/fstandae/PUBLIS/74.pdf. [Online; accessed 9-January-2019].

[61] Kate T., Pooja Sehgal, Nikhilesh Jasuja, Mohamed Saabd, and Umer Majeed. Am vs. fm. https://www.diffen.com/difference/AM_vs_FM, 2018. [Online; accessed 13-December-2018].

[62] Techopedia. Modulation. https://www.techopedia.com/definition/8409/modulation. [Online; accessed 9-January-2019].

[63] Cristian Toma. Introduction to Ultimate KEELOQ® Technology. http://ww1.microchip.com/downloads/en/AppNotes/00001683A.pdf, 2014. [Online; accessed 08-August-2019].

[64] Tutorialspoint. Data encoding techniques. https://www.tutorialspoint.com/digital_communication/digital_communication_data_encoding_techniques.htm. [Online; accessed 13-December-2018].

[65] Tutorialspoint. Digital communication tutorial. https://www.tutorialspoint.com/digital_communication/index.htm. [Online; accessed 13-December-2018].

[66] Tutorialspoint. Message authentication. https://www.tutorialspoint.com/cryptography/message_authentication.htm. [Online; accessed 29-May-2019].

[67] Coen van den Braber. Hacksystemen om ”keyless auto” te stelen snel en gemakkelijk online verkrijgbaar. `https://www.vrt.be/vrtnws/nl/2019/03/13/hacksystemen-om-keyless-auto-te-stelen-snel-en-gemakkelijk-onlin/`, 2019. [Online; accessed 03-August-2019].

[68] Triet Dang Vo-Huu, Tien Dang Vo-Huu, and Guevara Noubir. Interleaving jamming in wi-fi networks. In *Proceedings of the 9th ACM Conference on Security &#38; Privacy in Wireless and Mobile Networks*, WiSec ’16, pages 31–42, New York, NY, USA, 2016. ACM.

[69] VTM. Kijk hoe dieven ook bij ons draadloos luxewagens stelen en wat je ertegen kan doen. `https://nieuws.vtm.be/vtm-nieuws/binnenland/kijk-hoe-dieven-ook-bij-ons-draadloos-luxewagens-stelen-en-wat-je-ertegen-kan`, 2019. [Online; accessed 03-August-2019].

[70] Spencer Whyte. Jam intercept and replay attack against rolling code key fob entry system using rtl-sdr. `http://spencerwhyte.blogspot.com/2014/03/delay-attack-jam-intercept-and-replay.html`, 2014. [Online; accessed 11-December-2018].

[71] Spencer Whyte. Jam Intercept and Replay Attack against Rolling Code Key Fob Entry Systems using RTL-SDR. `http://spencerwhyte.blogspot.com/2014/03/delay-attack-jam-intercept-and-replay.html?m=1`, 2014. [Online; accessed 08-August-2019].

[72] I.C. Wiener. Philips/nxp hitag2 pcf7936/46/47/52 stream cipher reference implementation. `http://cryptolib.com/ciphers/hitag2/`, 2007.

[73] Wenyuan Xu, Ke Ma, Wade Trappe, and Yanyong Zhang. Jamming sensor networks: attack and defense strategies. *IEEE network*, 20(3):41–47, 2006.

[74] Paul Zandbergen. What is a sound card? - definition, function types. `https://study.com/academy/lesson/what-is-a-sound-card-definition-function-types.html`. [Online; accessed 8-January-2019].