Exploiting Software-Defined Radio

@code_byter

https://github.com/code-byter/Wireless-Master-Key

June 15, 2018

Contents

1 Introduction to Software-Defined Radio

Software defined radio is a wireless communication system, that is used in in a broad variety of applications. According to the Wireless Innovation Forum and the Institute of Electrical and Electronics Engineers (IEEE) Software-Defined Radio (SDR) is defined as

> "radio in which some or all of the physical layer functions are software defined" [7].

By using Software-Defined Radio, functionlity of devices can be modified without changing the physical layer as traditionally in the past. Thus different waveforms can be processed by a single device without changing the hardware. Functionality like multimode or multiband can be easily implemented using software. More detailed information about it can be found in [2] and [5].

The ideal receiver consists of an antenna connected to an analog-to-digital converter. The signal is then adapted to the requirements of the application by a digital signal processor. The ideal transmitter has a similar structure. The signal of the digital signal processor is is sent to a digital-to-analog converter and then transmitted by an antenna.
This technology is used in many ordinary devices like garage door openers, wireless outlets or car keys. But also more complex applications like the communications of smartphones are software-defined. Examples of such communication standards are Wi-Fi, GSM, WCDMA (3G) or Bluetooth. Only by using Software-Defined Radio those technologies can be implemented efficiently.

In the following paper the Software-Defined Radio communication of simple devices like garage door openers is investigated. The currently used technology is analyzed and its vulnerabilities are exploited in a device that serves as a proof of concept.

2 Analyzing Software-Defined Radio signals

2.1 Common Modulation Schemes

In order to transmit data wirelessly, a periodic signal called a carrier signal is modified. This modification contains the transmitted data [3]. The following modulation schemes always modify a single parameter of the carrier wave. The differently modulated signals for a certain data is presented in figure 1.

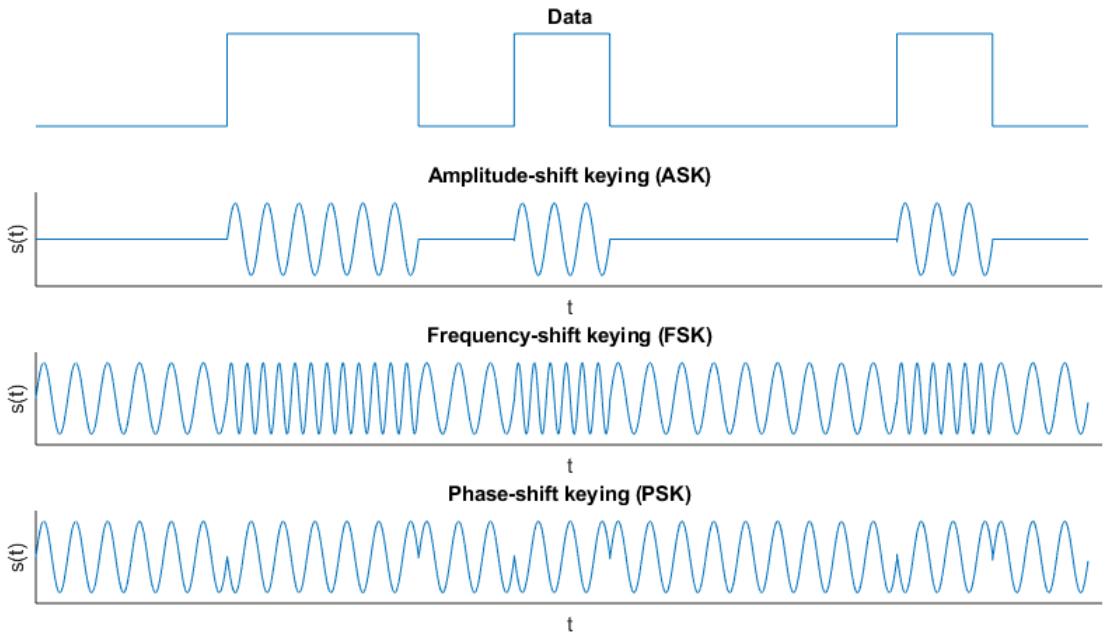


Figure 1: Transmitted data and its ASK, FSK and PSK modulated carrier signal

The most common modulation scheme is called Amplitude-shift keying (ASK). The signal is generated by modulating the amplitude of the carrier signal and can be described by the following formula.

$$s(t) = A(t) \cdot \sin(\omega t) \tag{1}$$

The signal shown in figure 1 contains a transmitted binary signal. The amplitude is the one of the carrier signal for a digital 1 and zero for a digital 0. This special type of modulation is called Binary Amplitude-shift keying (BASK).

Another modulation scheme is called Frequency-shift keying (FSK). Instead of shifting the amplitude of the carrier wave as previously, the frequency is modulated. One everyday application of this technology is FM-radio. If binary data is transmitted the signal switches between two frequencies, called Binary Frequency-shift keying (BFSK), and can be described as two separate signals in the following formula.

$$\begin{aligned} s_1(t) &= A \cdot \sin(\omega_1 t) \\ s_2(t) &= A \cdot \sin(\omega_2 t) \end{aligned} \tag{2}$$

The third modulation scheme, called Phase-shift keying (PSK), modulates the phase of the carrier signal. This technique is used by the Wirless LAN standard IEEE 802.11. In case of a binary signal (Binary phase-shift Keying) the phase of the carrier signal is shifted by π as described in the following formula.

$$\begin{aligned} s_1(t) &= A \cdot \sin(\omega t) \\ s_2(t) &= A \cdot \sin(\omega t + \pi) \end{aligned} \tag{3}$$

The following research only analyzes ASK modulated signals because of its simplicity and widespread use throughout electronic communication.

2.2 Analyzing SDR Signals

In order to analyze a transmitted Software-defined Radio signal, a suitable receiver is needed. A commonly used one is the RTL-SDR, which is based on the RTL2832U chipset. It's designed to receive DVB-T TV signals, but because of its capability of receiving signals in the frequency range from 24 MHz to 1766 MHz, it's suitable for analyzing most SDR-signals. The raw I/Q-samples are transmitted via USB to the PC and can be analyzed via tools like GNU Radio or Gqrx.

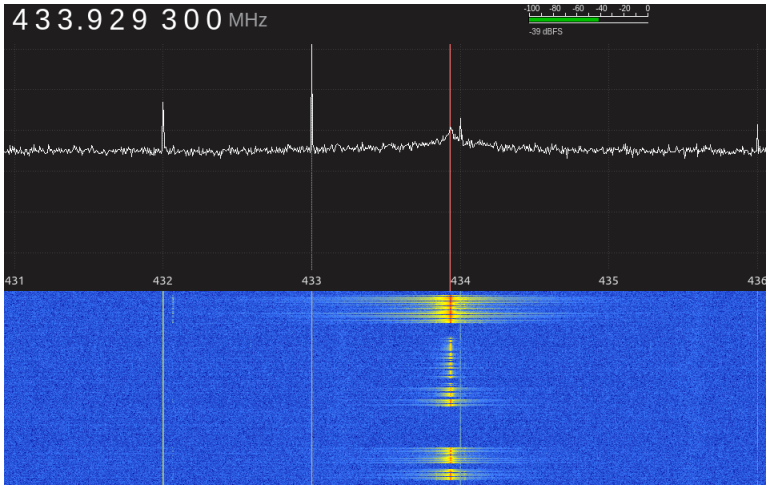


Figure 2: FFT-plot and spectrogram of a 433 MHz signal displayed in Gqrx

The received data is displayed in Gqrx as presented in figure 2. The upper panel of the application shows the amplitude of the signal with respect to the frequency. This data is displayed with respect to the time in the spectrogram beneath. Thus the amplitude and frequency of a signal can be observed over a longer period of time.

In order to analyze the emitted signal of a device, its frequency needs to be determined. This can easily be done by analyzing the spectrogram. The gradient of the amplitude can then be recorded and stored in an audio file. The waveform can be easily visualized and analyzed in audio-tools like Audacity.

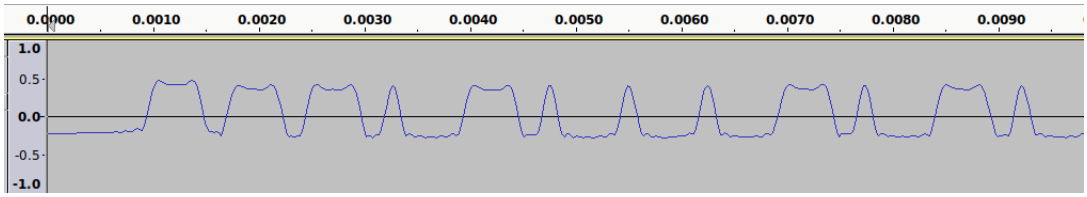


Figure 3: Amplitude of a recorded ASK-modulated signal

The recorded signal of a 433 MHz remote control is for instance shown in figure 3. It is a low pass filtered version of the actual record signal, which simplifies the analysis. It consists of a sequence of rectangular patterns of different lengths. A short transmitted pulse represents a 0 and a long pulse a 1.

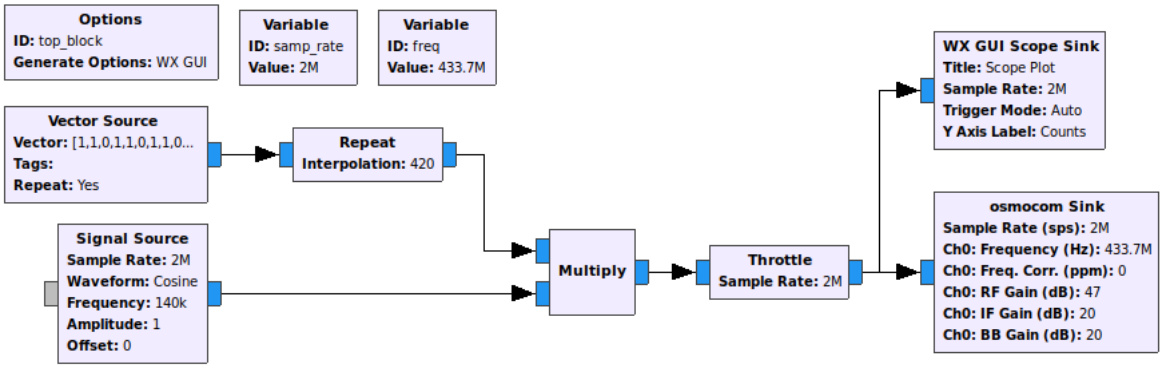


Figure 4: GNU Radio flow-graph for generating and transmitting ASK modulated signals

In order to transmit SDR-signals, a more advanced device is necessary. In this paper, a device called HackRF One is used . It is capable of receiving and transmitting signals with a frequency between 1 MHz and 6 GHz. To create and transmit ASK modulated signals, a software tool called GNU Radio is used. By using different "blocks" with a certain functionality, a signal can be generated and transmitted or received and modified/analyzed. A flow-graph to generate the ASK-modulated signal of the previously analyzed remote is shown in figure 4.

2.3 Basic Attacks

A wireless remote is only supposed to work with one device and not interfere with a different one. This is accomplished by assigning a unique or at least secure code to the transmitter and receiver. For wireless outlets or older garage doors this code is determined by the position of an array of Dual In-Line Package (DIP) Programming Switches within the device. This creates a nearly unique combination of usually five to twelve binary digits. In order to trigger that devices, this exact code has to be (a part of) the transmitted signal. As the number of different combinations is finite, the device is vulnerable to Brute-force Attacks.

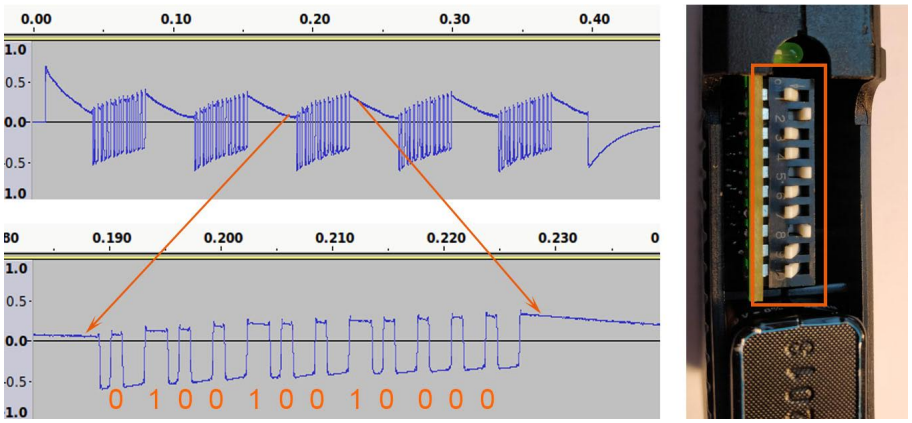


Figure 5: Recorded Signal and the physical DIP switches of a garage door remote

The recorded signal of a garage door remote is shown in figure 5. It is composed of about five repetitions of the exact same code. Between each of those repetitions, there's a small break which lasts about $t_b = 34\,\text{ms}$. Each bit is transmitted over a $t_p = 3\,\text{ms}$ long time period. The whole signal is 12-bit long and consists of the binary code of the DIP switches and two constant bits in the end.

The time required to Brute-force attack the device can be calculated and optimized easily. By brute forcing the code every single possibility needs to be transmitted. In case of 12-bit code 2^{12} combinations are possible. The whole attack time can be calculated by the following formula.

$$\begin{aligned} t_{attack} &= 2^{12} \cdot 5(t_b + 12 * t_p) = 2^{12} \cdot 5(34\,\text{ms} + 12 \cdot 3\,\text{ms}) \\ t_{attack} &= 2^{12} \cdot 0.35\,\text{s} = 1433,6s \approx 24\,\text{min} \end{aligned} \tag{4}$$

This time can be reduced easily. First the repetitions of the same code aren't necessary, as they are only used to increase reliability. Thus the transmission time is already decreased by the factor five. Secondly the breaks between the signal aren't necessary either. One transmission now consists of only one code without breaks, so the attack time can be reduced to the following time.

$$t_{attack} = 2^{12} \cdot (12 \cdot 3\,\text{ms}) \approx 150\,\text{s} \tag{5}$$

This period of time can be decreased by over half by using a more complex algorithm. The receiver uses a bit shift register to verify the code. Thus it can't determine when a signal starts or ends, which offers two main advantages. By testing all 12-bit codes, every code with a shorter bit length is tested at the same time. Moreover the number of tested possibilities can be reduced by the De Bruijn sequence. The functionality of this algorithm is explained with a three bit code in figure 6.

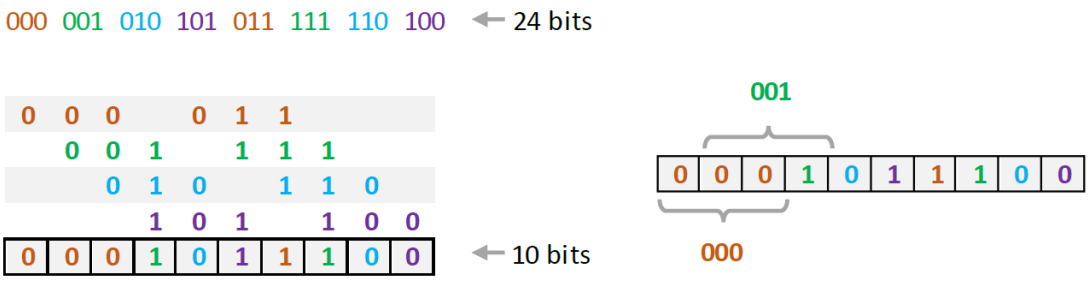
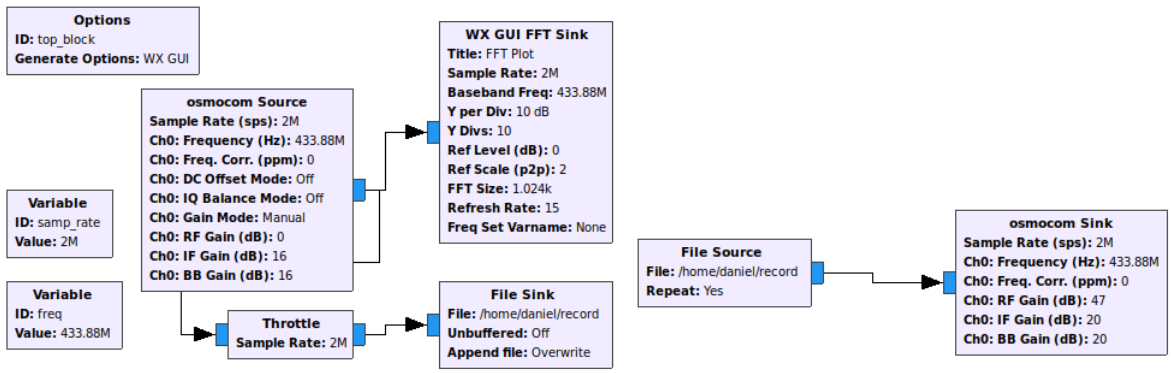


Figure 6: Recorded Signal and the physical DIP switches of a garage door remote

With a three bit code eight different combinations are possible which results in a 24 bit long code. By using the De Bruijn algorithm these possibilities are "overlapped". This means the code 000 and 001 can be combined to 0001. Thus the total bit length can be drastically reduced, by still testing every possible combination. The resulting code is now only 10 bits long. By using this improvement, a 12-bit secured device can now be triggered within under one minute.

Another easy attack is the replay-attack or spoofing. As more modern devices usually have codes with a bit length over 20-bit the Brute-force Attack is less relevant. In this case, the replay-attack is more suitable. By recording and emitting the signal at any later point the receiver can be triggered. The corresponding GNU Radio flow-graph is shown in figure 7. The user on the other hand doesn't notice that the signal is received by the spoofer.



(a) Recording (b) Replaying

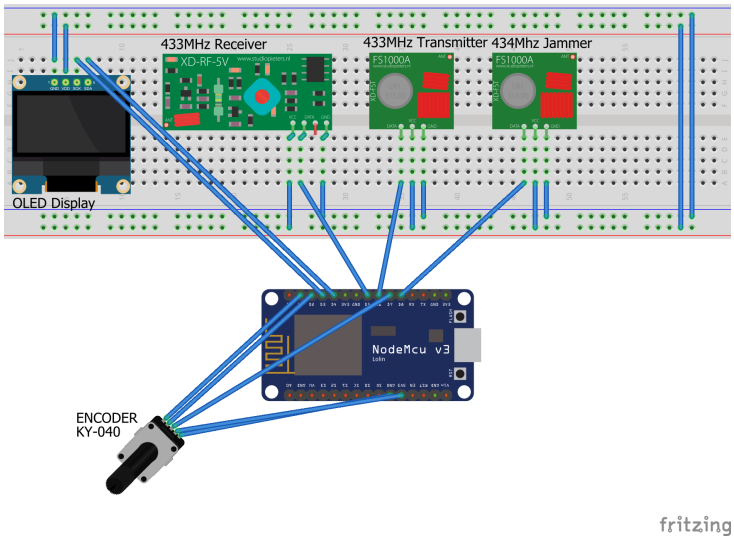
Figure 7: GNU Radio flowgraph for replay-attack

3 Proof of Concept

In order to verify the effectiveness of these attacks a low-cost, arduino-based device called the Wireless Master Key is built. The goal is to build an easy-to-use device to effectively show the vulnerabilities of today's devices.

3.1 Components of the Wireless Master Key

The key parts of this device are the receiver and transmitter. In order to keep the system simple their operating frequency is kept fixed at 433 MHz. Apart from some few old garage doors, which are using the 40 MHz frequency band, most devices use the 433 MHz one. To improve its flexibility not only a 433 MHz transmitter but also a 434 MHz one is used. Thus transmitting at 434 MHz and receiving at 433 MHz is simultaneously possible.



Component	Device-Pin
Reveiver	D5
Transmitter	D6
Jammer	D8
SCL	D3
SDA	D4
CLK	D1
DT	D2
SW	D7

Figure 8: Wiring of the Wireless Master Key

For enhanced usability the device cannot only be controlled over the PC's Serial Port, but has an additional interface consisting of an encoder and an OLED display. To control all these components the ESP8266 chipset on the NodeMCU platform is used. The wiring is shown is figure 8.

3.2 Functionality and Implementation

The microcontroller is programmed by using the Arduino platform. For implementation the in appendix A shown libraries are used. The main features are the following attack-methods: jamming, spoofing, brute-force, etc.

The first option is jamming. This is accomplished by using the 434 MHz transmitter, and thus maintaining full functionality of the built-in 433 MHz receiver. By transmitting constantly a small code consisting of only digital 1 signals, the attacked device is jammed and doesn't trigger any action. As manufacturers want to build their devices as cheap

and reliable as possible, their used receiver is working at a broad frequency range. Thus the jammer's signal is received by the attacked device, but not by the Wireless Master Key. This is the easiest implemented attack, as it's only able to stop somebody from using his device, but can't trigger it. The emitted signal and its frequency range is shown in figure 9. The implementation can be found in appendix B.

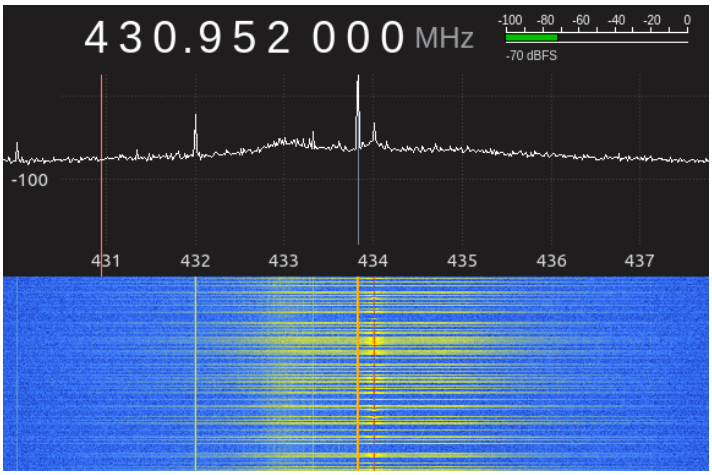


Figure 9: Transmitted jamming signal of the Wireless Master Key shown in Gqrx

The second feature is spoofing/sniffing (appendix C and D). The device is constantly listening for transmitted SDR signals. If one is received, it gets decoded and analyzed. The decimal value, bit length and pulse length are stored and displayed. Next the hacker can replay the signal. Thus he has full control over the device if the signal is received once.

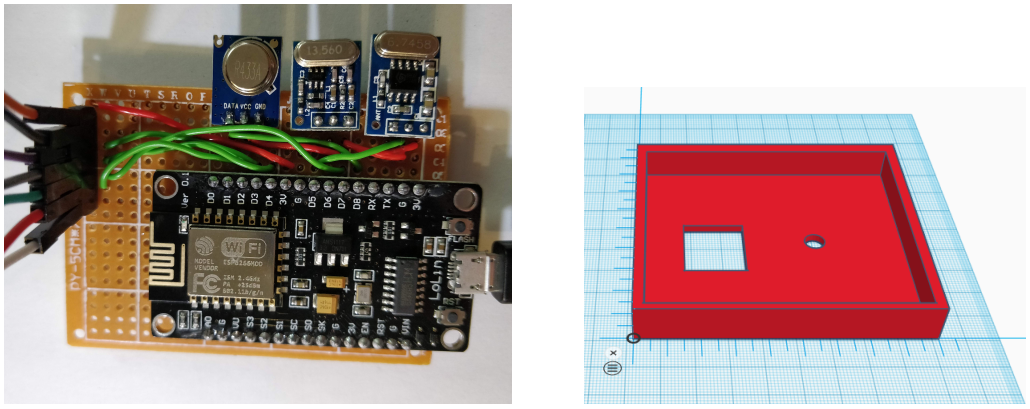
The next implemented option is called "Hacking" and performs a Brute-force Attack (appendix E). All possible combinations of 8-bit long codes are transmitted. Some of the previously explained tweaks to improve transmission time are implemented. By using this attack the hacker can control every wireless device that is working with codes of of bit length.

The last feature gives the user the option to transmit their own codes (appendix F). By the Serial Port communication between the wireless master key and the PC is enabled. The decimal code, the bit length and the pulse width of the signal can be individually set and the signal is transmitted.

The most effective implemented attack is spoofing. As most modern devices transmit singals with a bitlength over 20-bit, it's not practical to test every possibility. More information about these attacks can be found in [1], [8] and [8]

3.3 Assembling the Wireless Master Key

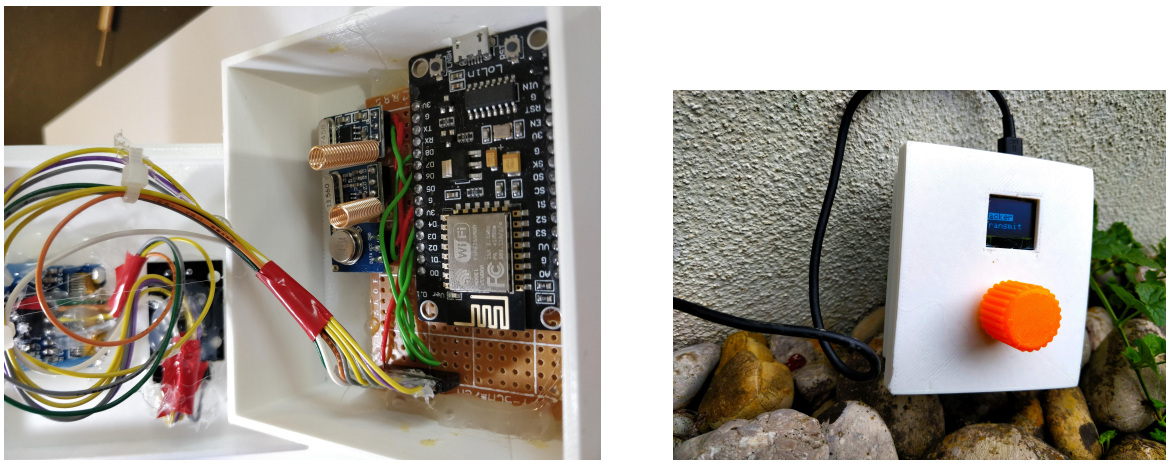
The building process starts with assembling the whole device on a breadboard. After sustained tests this prototype is soldered to a perfboard. This drastically improves reliability as the connections between the parts are permanently fixed. In order to mount all parts, an enclosure is necessary. Therefore a 3D-model is designed and printed. These parts are shown in figure 10.



(a) Soldered Perfboard (b) Enclosure model

Figure 10: Enclosure and electronics of the "Wireless master key"

The different components are assembled and hot-glued together. The finished device can be seen in figure 11. The navigation menu of the device is kept simple. The different options are listed vertically and can be navigated by turning or pushing the rotary encoder.



(a) Assembled interior (b) Full device

Figure 11: Assembling the device

4 Rolling codes

For more secure applications like locking cars, the risk of a replay-attack is too high. Therefore a technique called rolling codes is used to secure the communication. In the following part this technology is explained and possible attacks are presented.

4.1 Functionality

The replay-attack is for rolling-code systems is obsolete as their transmitted code isn't fixed, but changes with every interaction. This means the transmitted code is only valid once.

In order to generate and verify these codes, synced PRNGs (pseudorandom number generator) are built in every receiver and transmitter. Whenever the key-fob is pressed, the transmitter sends a code to the receiver. The receiver compares the received code to the calculated one. This algorithm, calculating the next code, is complex and unique, so it can't be hacked easily. If the code is accepted the next one is generated and the old one can't be used anymore.

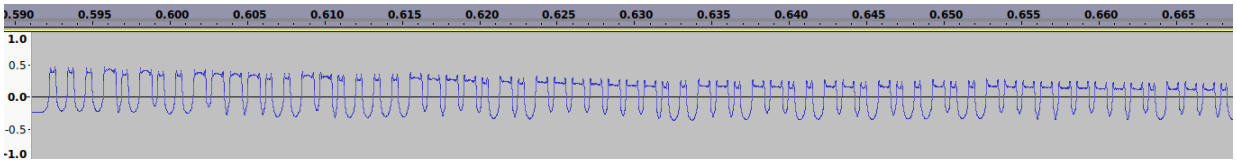


Figure 12: Recorded signal of a rolling-code secured garage door

In order to improve reliability a rolling code window is implemented. This means the receiver doesn't only accept the next code, but also a few following ones. This is necessary if the key is accidentally pressed when it's out of range. As this window is quite small (usually about 256 codes) attacks are more complicated. As shown in figure 12, the bit length of these codes is too long to be brute forced.
This technology is one of the most secure currently used ones. Nearly every car uses it to secure its unlock mechanism.

4.2 Vulnerability

As the receiver accepts each code just once, the replay-attack only works if the device is out of range. This can be easily verified by using the HackRF One. First the remote needs to be placed out of range and it's transmitted signal recorded. This can be archived in the console by the following command.

```
1 $ hackrf_transfer -r 390_data.raw -f 433800000
```

As the recorded signal is still valid, it can be transmitted again. If the receiver is in range, the device is triggered. This can be done by using the following command.

```
1 $ hackrf_transfer -t 390_data.raw -f 433800000
```

This works with nearly every current car, but so far this type of attack isn't practical in real life, as the remote has to be pressed while it's out of the receiver's range. This problem can be solved by preventing the receiver from being triggered, even if the remote is within the range. By jamming the receiver as already previously implemented in the Wireless Master Key, this state can be archived.

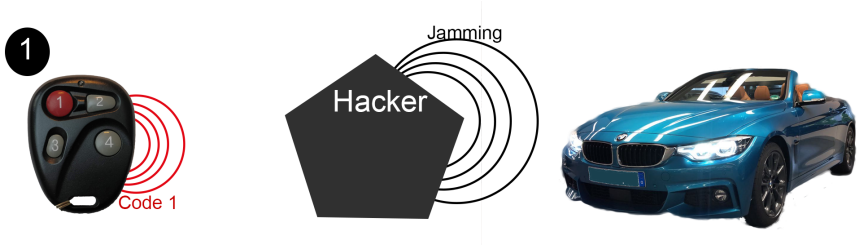


Figure 13: First step of the attack: Jamming and listening

In the first step, the receiver of the attacked device is being jammed, thus no code can be detected. A scheme of it is shown in figure 13 . The frequency of the jamming signal is partly above the frequency of the key or partly below, but the key's emitted frequency can't be jammed. The victim presses the remote, thus a signal is transmitted. The hacker receives this code, as he uses a better antenna than the attacked device, and stores it. The device isn't triggered so the victim presses the remote again.

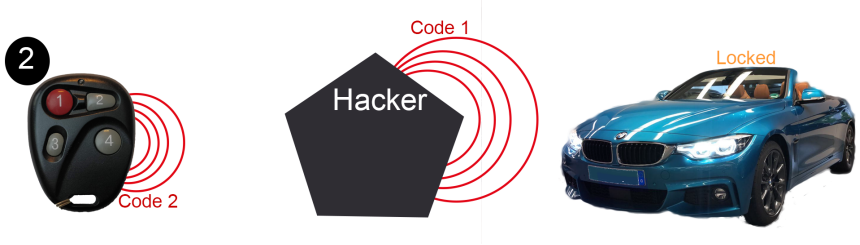


Figure 14: Second step of the attack: Recording the second code and transmitting the first one

Secondly the hacker then receives and stores the second code and the device isn't triggered. The jammer is stopped and the first code is emitted. This process is visualized in figure 14. The car/garage is unlocked. Thus the victim isn't suspicious as the device acts, apart from the two key-presses, normally.

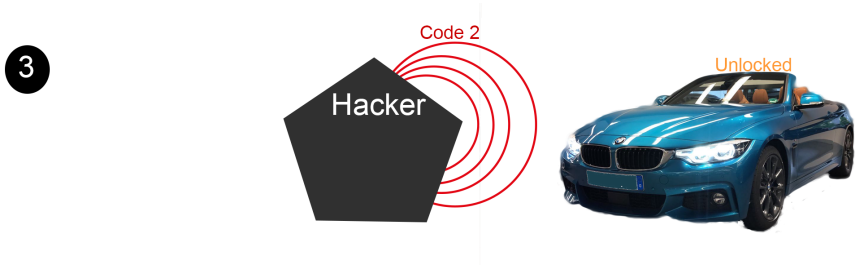


Figure 15: Second step of the attack: Recording the second code and transmitting the first one

As the hacker only transmitted the first code, he has still a spare one. He can use it whenever he wants, as shown in figure 15. It's only disabled if the victim presses the remote again.

Car keys have different signals for locking and unlocking. By analyzing these signals you notice, that those signals have apart from one single bit the same structure. By flipping this bit, a code used to lock the car can be used to unlock it.

4.3 Countermeasures

There are many different possibilities to secure devices against this type of attack. Some simple countermeasures are introduced in the following paragraph.

First of all, bit flipping to change between a lock and unlock signal needs to be disabled. This is archived by decryption the transmitted signal. Thus the hacker only receives a lock signal, which is useless to him. A sniffed unlock signal can't be used as the victim locks the car after using it, disabling the stored key.
A second possible, probably most effective countermeasure is using time-based algorithms. Thus the transmitted key is only accepted for a certain period of time, which makes attacks like these nearly impossible. Some few car manufacturers like Cadillac already implemented this feature, called "Dual KeeLoq".
A third option is replacing the one-way-communication. By using a transceiver in both devices, a more secure communication protocol can be used.

5 Conclusion

In conclusion it is obvious that current Software-Defined Radio solutions can be exploited easily. No matter if it's a wireless outlet, a wireless temperature sensor, the car with rolling code technology or Passive-Keyless-Entry Systems. But most users aren't even aware of this type of attacks. The previously presented attacks are only the most simple ones, but researchers discovered a lot more vulnerabilities of nearly every device. Thus the manufacturers are already informed about those problems.

Now it's their turn to implement solutions. Most of these vulnerabilities can be fixed without significant effort, as most solutions software and hardware solutions are already available. For example time based algorithms are used in RSA tokens, so they can easily be implemented in car keys, too. Moreover, some keys already use transceivers instead of transmitter, but the feature just isn't implemented yet. But sometimes the creativity of the users is asked to secure Software-Defined Radio solutions.

References

[1] Kamkar Samy. *Drive It Like You Hacked It: New Attacks and Tools to Wirelessly Steal Cars*. DEF CON (2015).
http://samy.pl/defcon2015/2015-defcon.pdf

[2] Varde, J.H. & Gohil N.B. & Shah J.H.. *A General System Design & Implementation of Software Defined Radio System. Journal of Information, Knowledge and Research in Electronics and Communication Engineering*. Volume 02, Nov 12- Oct 13

[3] Kumar Lunagariya, Jay & Gokhruwala, Kenil & Vachhani, Khyati. *Design Analysis of Digital Modulation Schemes with GNU Radio*. (2015).

[4] Grayver, Eugene. *Implementing Software Defined Radio.*, Springer-Verlag New York. 2013

[5] Richardson, Anna. *SECURITY OF VEHICLE KEY FOBS AND IMMOBILIZERS*.
http://www.cs.tufts.edu/comp/116/archive/fall2015/arichardson.pdf

[6] SDR Forum. *SDRF Cognitive Radio Definitions Working Document SDRF-06-R-0011-V1.0.0*. 2007

[7] Rascagnères, Paul. *433MHz ASK signal analysis Wireless door bell adventure*. 2015

[8] Seeber, Balint. *Hacking the Wireless World with Software Defined Radio - 2.0*.
https://www.blackhat.com/docs/asia-15/materials/asia-15-Seeber-Hacking-the-Wireless-World-With-Software-Defined-Radio-2.0.pdf

Appendices

A Libraries and Initialization

Listing 1: Included libraries

```
#include <Arduino.h>
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <RCSwitch.h>
#include <Encoder.h>
```

B Jammer Implementation

```
void initJammer(){
  display.clearDisplay();
  display.setTextSize(2);
  display.setTextColor(WHITE);
  display.setCursor(0, 0);
  display.println("Jammer");

  display.setTextSize(2);
  display.setTextColor(BLACK, WHITE);
  display.setCursor(115, 0);
  display.println("X");

  display.setTextSize(2);
  display.setTextColor(WHITE);
  display.setCursor(0, 25);
  display.println("Jamming ...");
  display.display();

  while (!exitOption()){ //Start jamming
    delay(30);
  }
}
```

C Sniffer Implementation

```
1  void initSniffer(){
2    displaySniffer();
3
4    while (!exitOption()){
5      if (receiver.available()){
6        display.clearDisplay();
7        displaySniffer();
8
9        display.setTextSize(2);
10       display.setTextColor(WHITE);
11       display.setCursor(0, 22);
12       display.println(receiver.getReceivedValue());
13
14       display.setTextSize(1);
15       display.setTextColor(WHITE);
16       display.setCursor(0, 42);
17       display.println(receiver.getReceivedBitlength());
18       display.setCursor(12, 42);
19       display.println("bit");
20       display.setCursor(40, 42);
21       display.println(receiver.getReceivedDelay());
22       display.setCursor(58, 42);
23       display.println("us");
24       display.display();
25       receiver.resetAvailable();
26     }
27     delay(5);
28   }
29 }
```

Figure 16: Visualization of the received signal

D Spoofer Implementation

Listing 2: Implementation of the spoofer's core-functionality

```
1   long receivedValue = 0;
2   int receivedBitlength = 0;
3   int receivedDelay = 0;
4   int receivedProtocol = 0;
5
6   while (scanning){
7     if (receiver.available()){
8       receivedValue = receiver.getReceivedValue();
9       receivedBitlength = receiver.getReceivedBitlength();
10      receivedDelay = receiver.getReceivedDelay();
11      receivedProtocol = receiver.getReceivedProtocol();
12      receiver.resetAvailable();
13      scanning = false; //Signal is received and stored
14    }
15    delay(10);
16  }
17
18  //Transnmit signal
19  if (transmitting){
20    while (!exitOption()){
21      jammer.setPulseLength(receivedDelay);
22      jammer.setProtocol(receivedProtocol);
23      delay(50);
24    }
25  }
26  displayMenu();
27 }
```

E Brute-Force Attack Implementation

```
1  void initHacker(){
2    isButtonPressed = false;
3    display.clearDisplay();
4    display.setTextSize(2);
5    display.setTextColor(WHITE);
6    display.setCursor(0, 0);
7    display.println("Hacker");
8
9    display.setTextSize(2);
10   display.setTextColor(BLACK, WHITE);
11   display.setCursor(115, 0);
12   display.println("X");
13
14   display.setTextSize(2);
15   display.setTextColor(WHITE);
16   display.setCursor(0, 25);
17   display.println("Hacking...");
18   display.display();
19
20   jammer.setRepeatTransmit(2);
21   int bitlength = 8;
22   int maxlength = 256; //2^8
23   for (int i = 0; i <= maxlength; i++){
24     delay(10);
25     Serial.println(i);
26     if (exitOption())
27       break;
28   }
29   displayMenu();
30 }
```

F Transmitter Implementation

```
1  void initTransmit()
2  {
3    Serial.println("Welcome to the wmk433");
4    Serial.println("Please enter the binary code");
5
6    unsigned long code = 0;
7    int pulseLength = 0;
8    int protocol = 0;
9    int bitlength = 0;
10   Serial.println("\nCode in dezimal: ");
11   code = readSerialInt();
12   if (!(code))
13     return;
14   Serial.println("\nNumber of bits: ");
15   bitlength = readSerialInt();
16   if (!(bitlength))
17     return;
18   Serial.println("\nDelay per bit: ");
19   pulseLength = readSerialInt();
20   if (!(pulseLength))
21     return;
22   Serial.println("\nProtocol: ");
23   protocol = readSerialInt();
24   if (!(protocol))
25     return;
26   jammer.setProtocol(protocol);
27   jammer.setPulseLength(pulseLength);
28   jammer.setRepeatTransmit(60);
29   Serial.println("\nTransmitting ...");
30 }
```