Karlsruher Institut für Technologie

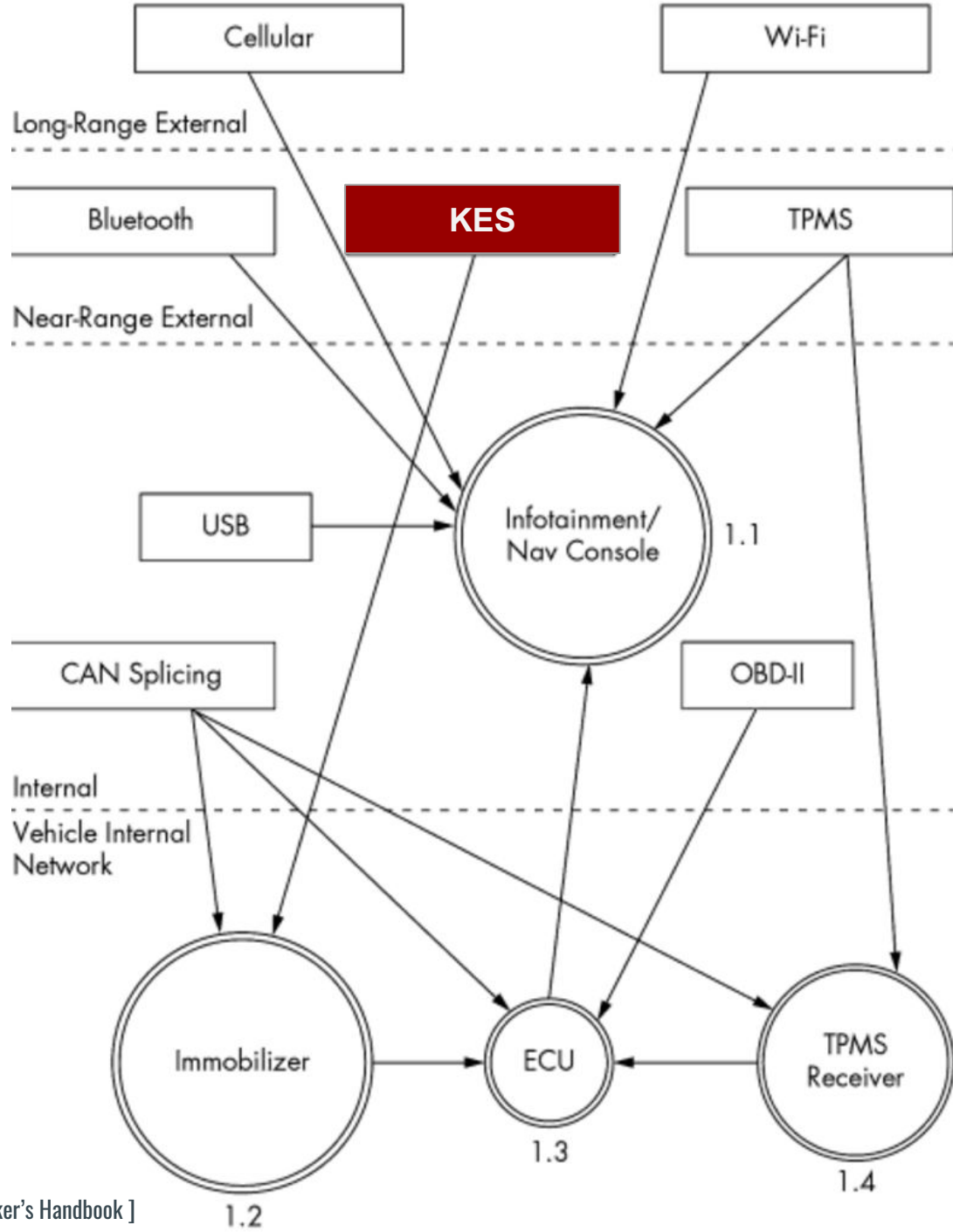
Keyless Entry Systems

KASTEL-Praktikum Sicherheit

Katharina Männle, Samuel Groß

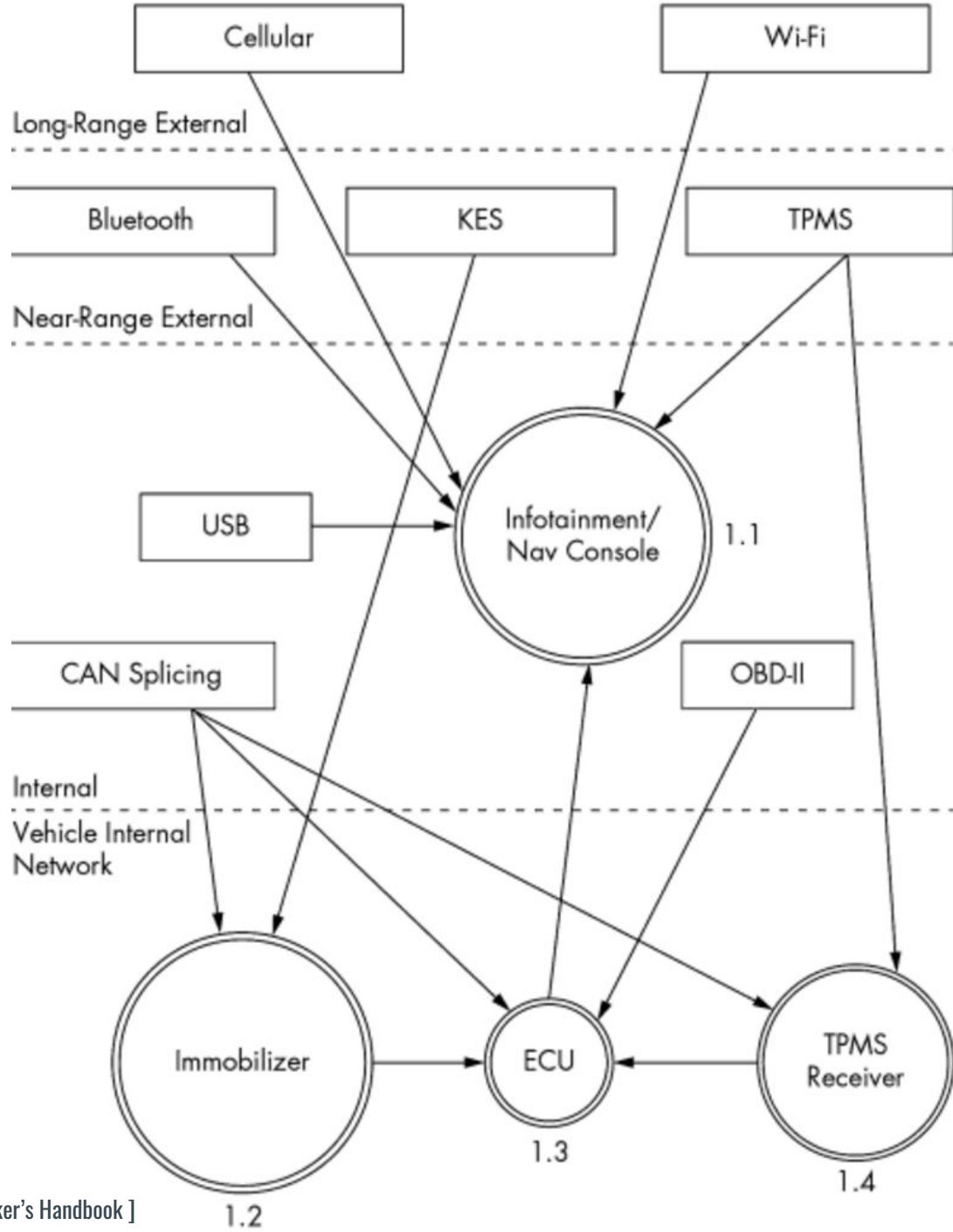
KOMPETENZZENTRUM FÜR ANGEWANDTE SICHERHEITSTECHNOLOGIE (KASTEL)

Car (wireless) attack surface



[Image source: Car Hacker's Handbook]

Car (wireless) attack surface



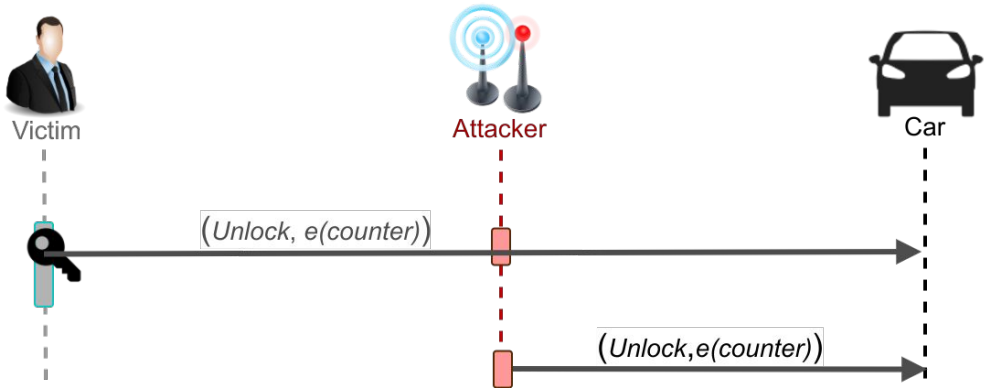
[Image source: Car Hacker's Handbook]

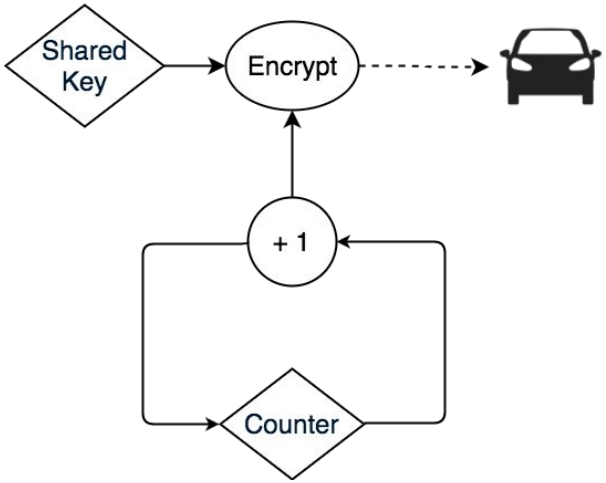
Motivation

- Most cars can be opened remotely from a short distance
- Interesting for attackers
  - Allows access to vehicle
- But: not much documentation available
  ➡ Need to reverse engineer
- What are possible attacks?

Brainstorming Attacks: Replay

- Capture unlock signal
- Replay later



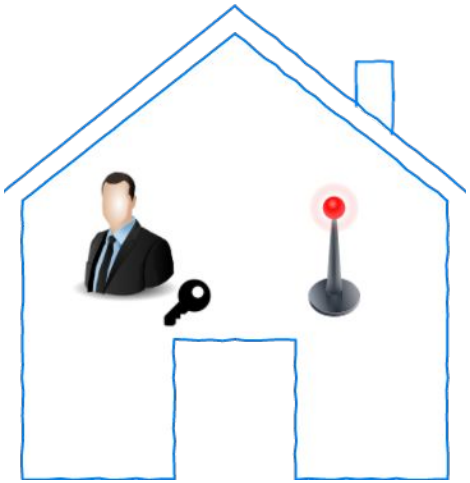


- Mitigated by rolling code?
  - Counter, encrypted with preshared key
  - Incremented after each use
  - Car has own counter, synchronized
  - Car accepts future codes (e.g. up to 256), **but not previous ones**

Brainstorming Attacks: Relay

Capture unlock signal indoors, use outdoors

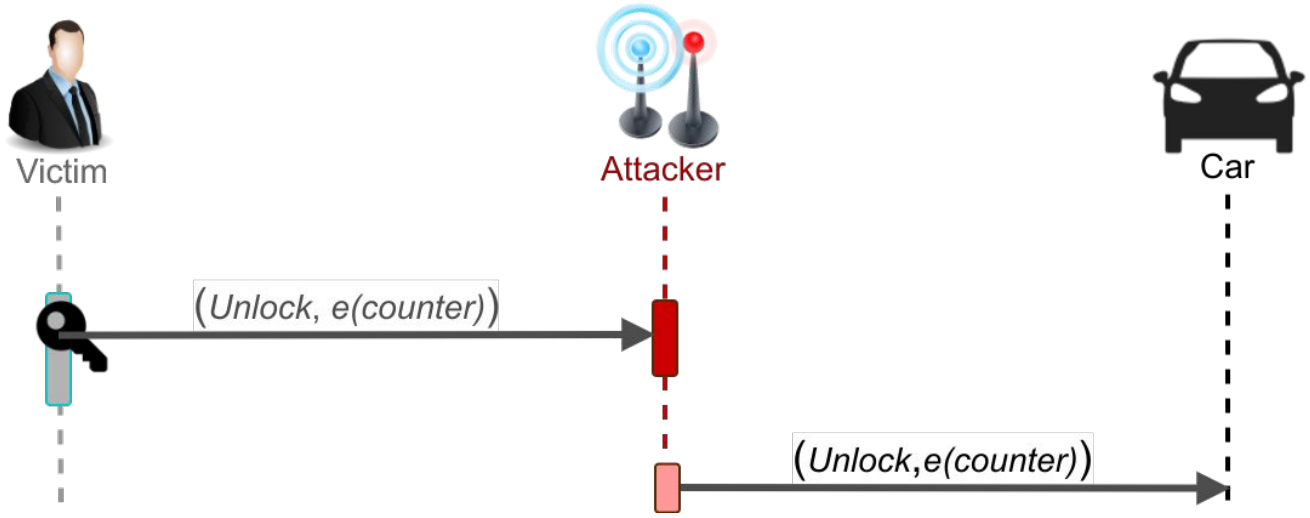
- Assumption: unidirectional communication
- Limited practicality, need access to key
- (Actually, need “unused” signals)



Brainstorming Attacks: Jamming + Relay

Prevent "use" of signal with jammer, use signal later

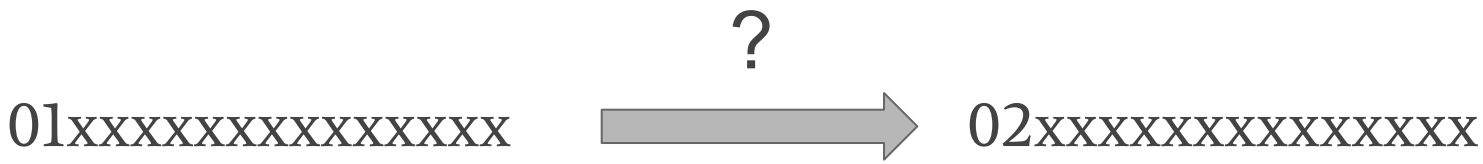
- "Victim" would notice this, can we avoid it?



Brainstorming Attacks: Packet Modification

Turn lock signal into unlock signal?

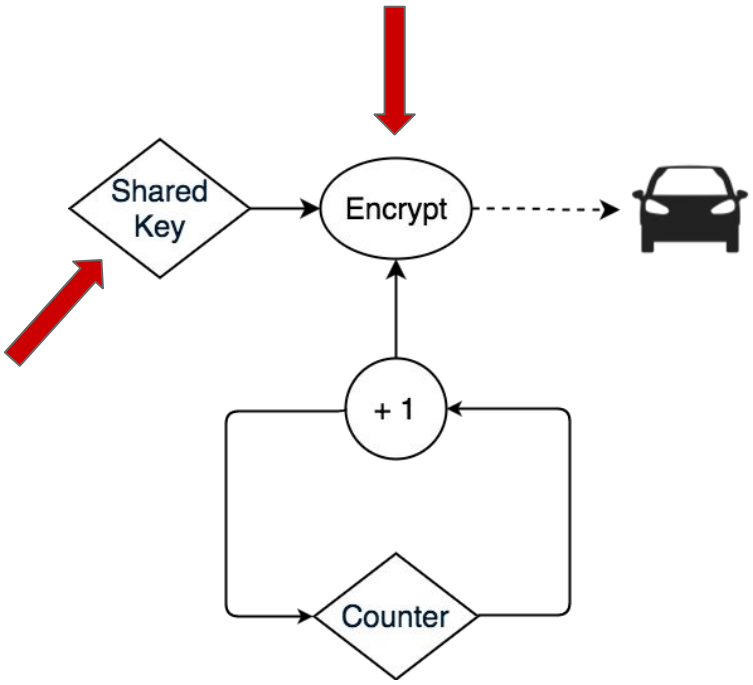
- Does this enable any interesting attacks?
- Need knowledge of packet format



Brainstorming Attacks: Cryptography

Attack cryptography (rolling code)

- Find flaw in encryption algorithm
- Recover shared key
  - Hardware attack



Brainstorming Attacks - Summary

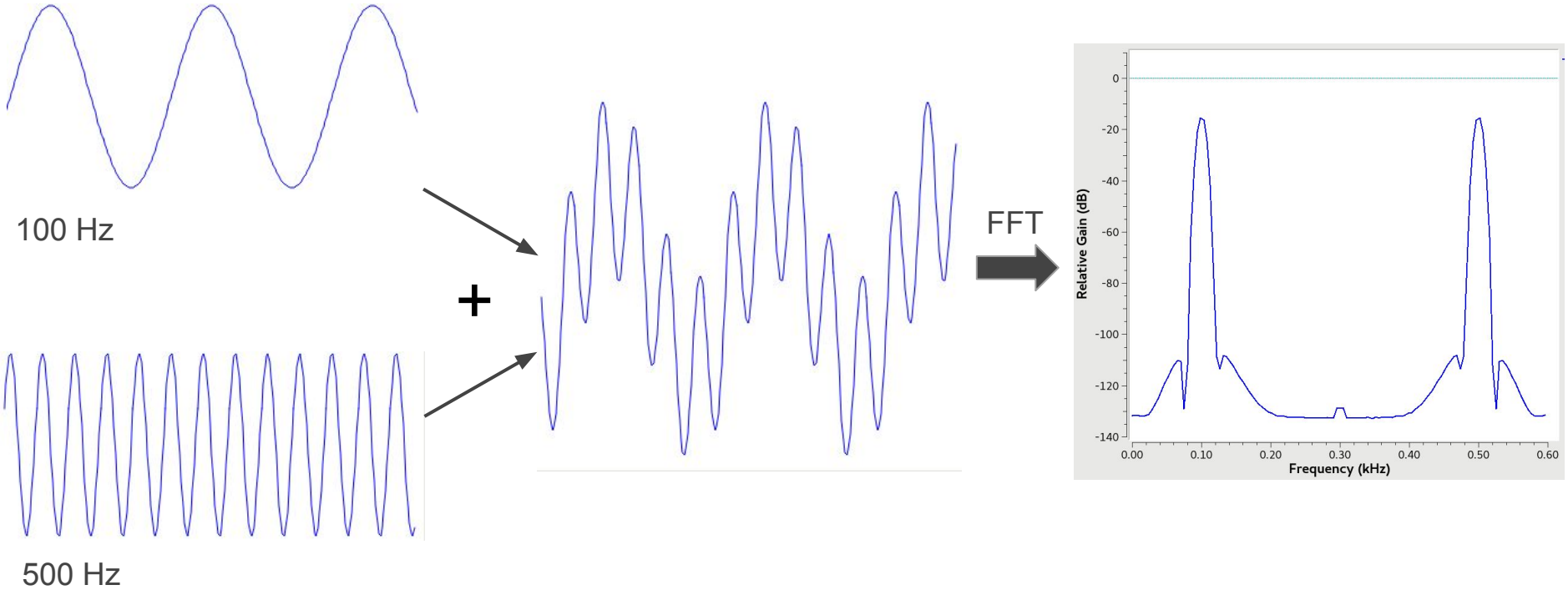
1. **Replay:** capture unlock signal, replay later on
2. **Relay:** capture unlock signal indoors, use outdoors
3. **Jamming:** prevent “use” of signal with jammer, use signal later
4. **Modify packets:** turn lock signal into unlock signal?
5. Break cryptography

Capturing the Signals

A short introduction to SDR and Gnuradio

Some Radio Theory

- Electromagnetic waves
- Typical frequencies: 433MHz, 2.4GHz, 5GHz, ...
  - so called unlicensed bands
- Waves overlap additively: superposition
- “Recover” frequencies with fourier transformation

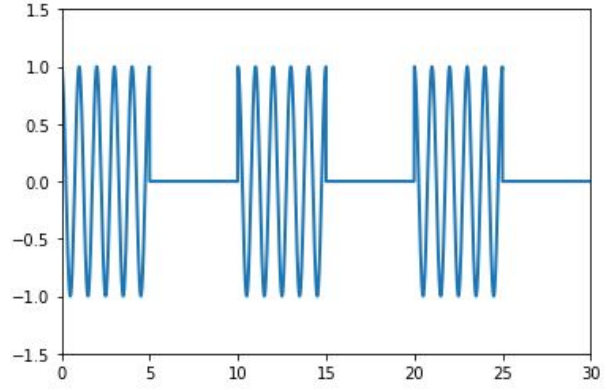


Modulation

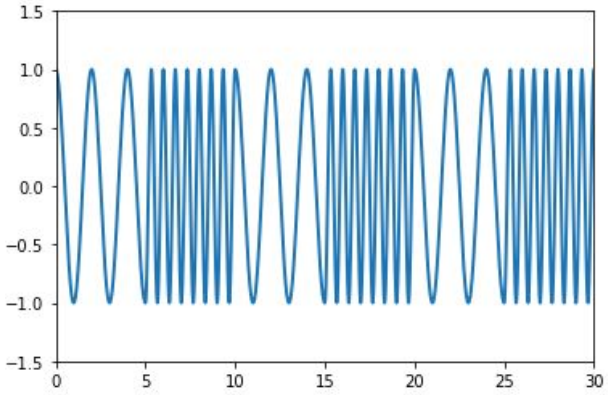
- How to transmit a binary signal through radio waves?
  ⇒ Modulation

- Different (basic) types of modulation:

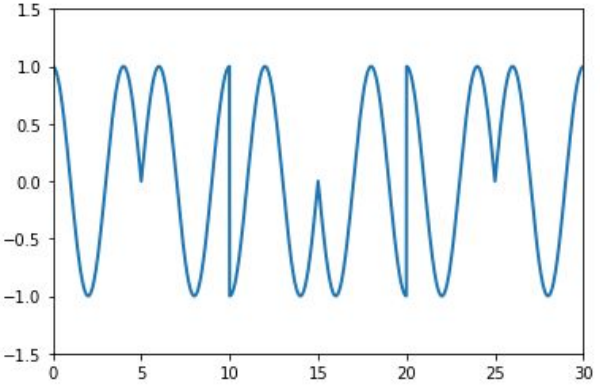
Amplitude modulation



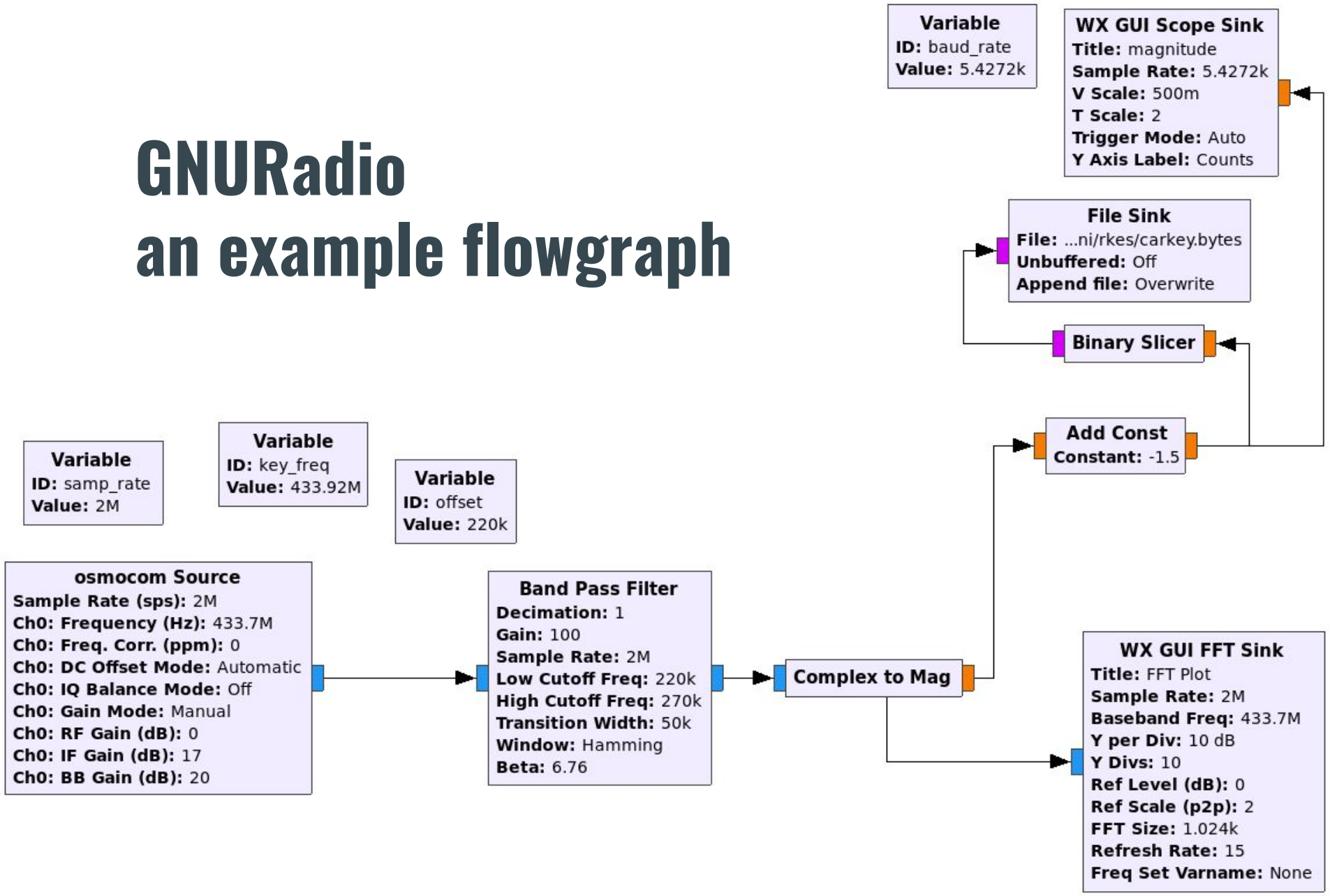
Frequency modulation



Phase modulation



GNURadio
an example flowgraph



Hardware

- **Software-defined Radio:** control a radio device with software
- Can control frequency, modulation, amplitude, ...
- Can filter, convert, ...
- HackRF produces sequence of samples (complex number) with fixed sample rate (e.g. 2MHz)
- All other processing done in software (e.g. GNURadio)

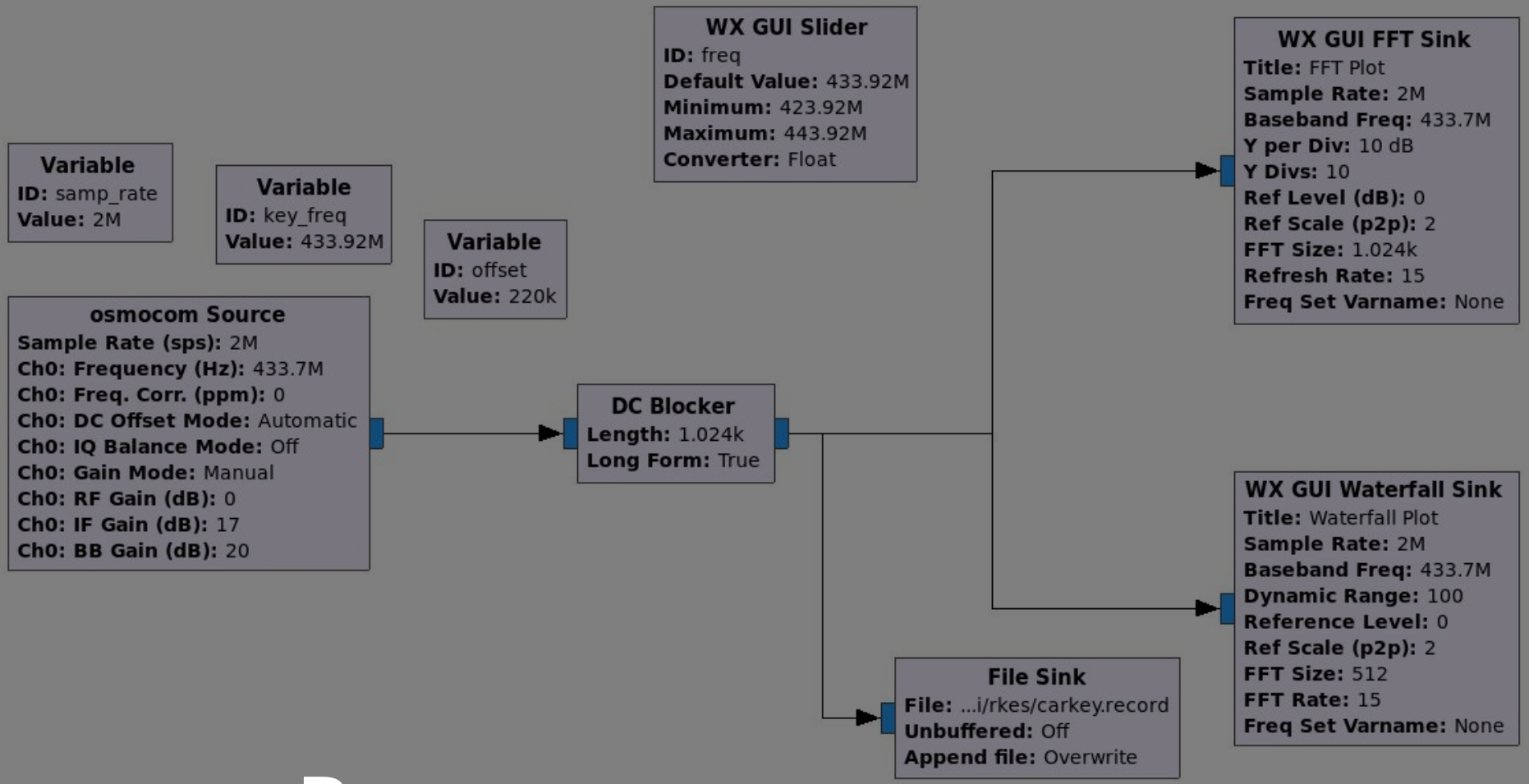
Capturing the key signal

1. **Tune HackRF to specific frequency** (433.92MHz)

2. *Optional:*
   Add filter to remove unwanted signals

3. *Optional:*
   Add GUI sinks for visualization

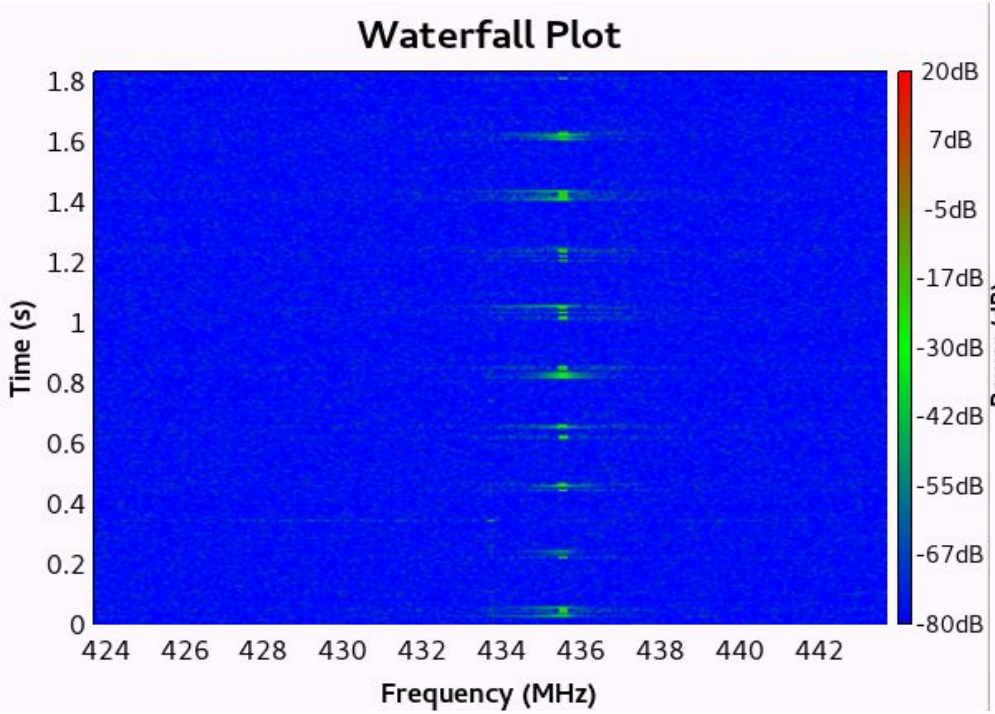
4. **Write samples into a file for offline processing**

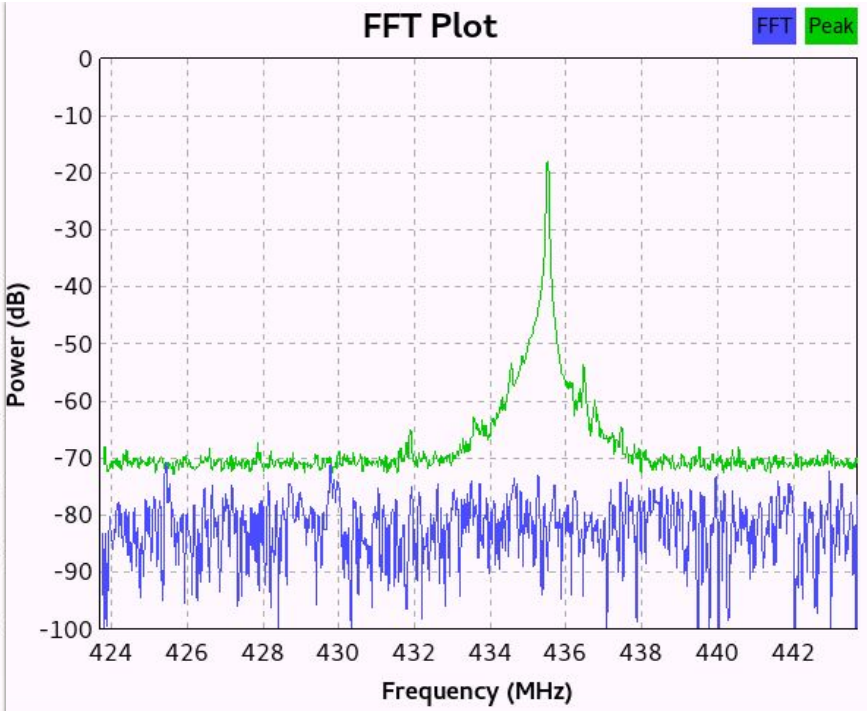

WX GUI Slider
ID: freq
Default Value: 433.92M
Minimum: 423.92M
Maximum: 443.92M
Converter: Float
WX GUI FFT Sink
Title: FFT Plot
Sample Rate: 2M
Baseband Freq: 433.7M
Y per Div: 10 dB
Y Divs: 10
Ref Level (dB): 0
Ref Scale (p2p): 2
FFT Size: 1.024k
Refresh Rate: 15
Freq Set Varname: None
Variable
ID: samp_rate
Value: 2M
Variable
ID: key_freq
Value: 433.92M
Variable
ID: offset
Value: 220k
osmocom Source
Sample Rate (sps): 2M
Ch0: Frequency (Hz): 433.7M
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Automatic
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 0
Ch0: IF Gain (dB): 17
Ch0: BB Gain (dB): 20
DC Blocker
Length: 1.024k
Long Form: True
WX GUI Waterfall Sink
Title: Waterfall Plot
Sample Rate: 2M
Baseband Freq: 433.7M
Dynamic Range: 100
Reference Level: 0
Ref Scale (p2p): 2
FFT Size: 512
FFT Rate: 15
Freq Set Varname: None
File Sink
File: ...i/rkes/carkey.record
Unbuffered: Off
Append file: Overwrite

Demo

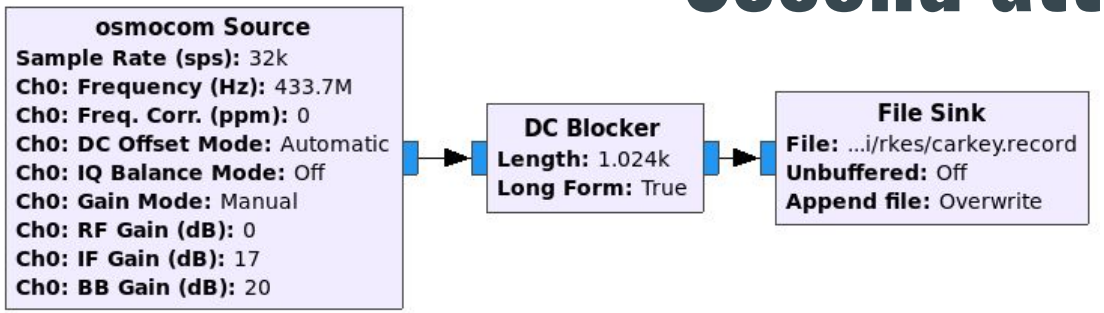
Checklist

1. Capture signal ✔





~~First attack: Replay~~
Second attack: Relay



1. Capture signal (inside)
2. Replay signal (outside)
3. :)

Possible attack scenario: car owner inside, can get access to key for a short while

Checklist

- Capture signal ✔
  - ~~Attack: Replay~~
  - Attack: Relay

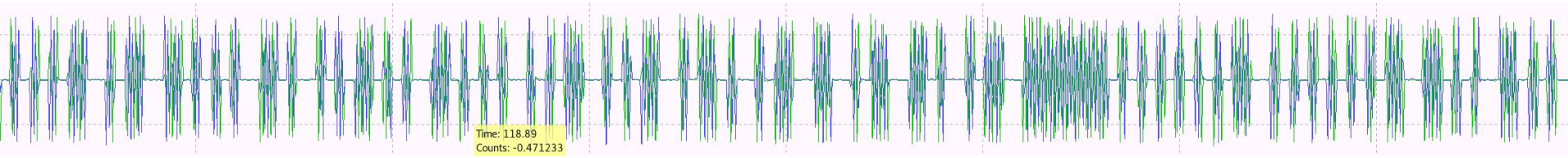
Signal Analysis

Understanding the signal

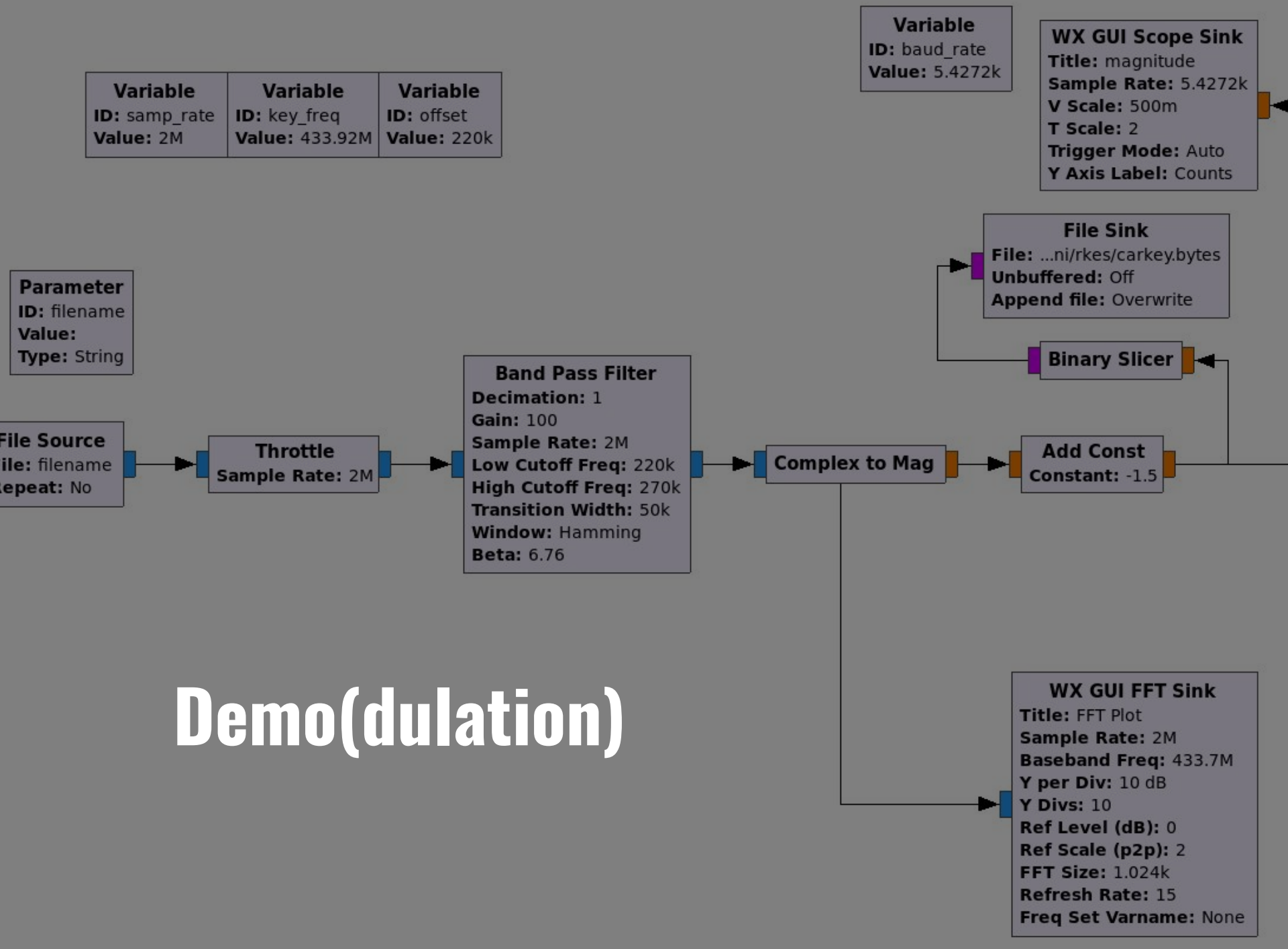
- “Waterfall” graph indicates amplitude modulation
- Clearly visible ones and zeroes

[Image produced with “Inspectrum”]

Decoding the signal



- Clearly amplitude modulation is used
- In GNURadio: Complex-To-Magnitude block
  - Replaces each sample with the magnitude of the wave at that time
- Then use binary slicer to obtain sequence of ones and zeroes

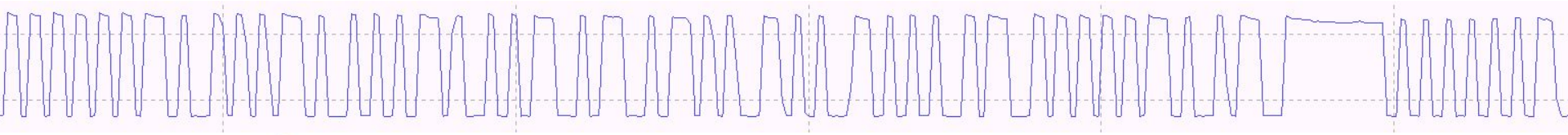

Variable
ID: samp_rate
Value: 2M
Variable
ID: key_freq
Value: 433.92M
Variable
ID: offset
Value: 220k
Variable
ID: baud_rate
Value: 5.4272k
WX GUI Scope Sink
Title: magnitude
Sample Rate: 5.4272k
V Scale: 500m
T Scale: 2
Trigger Mode: Auto
Y Axis Label: Counts
File Sink
File: ...ni/rkes/carkey.bytes
Unbuffered: Off
Append file: Overwrite
Parameter
ID: filename
Value:
Type: String
Binary Slicer
Band Pass Filter
Decimation: 1
Gain: 100
Sample Rate: 2M
Low Cutoff Freq: 220k
High Cutoff Freq: 270k
Transition Width: 50k
Window: Hamming
Beta: 6.76
File Source
ile: filename
epeat: No
Throttle
Sample Rate: 2M
Complex to Mag
Add Const
Constant: -1.5
Demo(dulation)
WX GUI FFT Sink
Title: FFT Plot
Sample Rate: 2M
Baseband Freq: 433.7M
Y per Div: 10 dB
Y Divs: 10
Ref Level (dB): 0
Ref Scale (p2p): 2
FFT Size: 1.024k
Refresh Rate: 15
Freq Set Varname: None

Checklist

- Capture signal ✔

  ~~Attack: Replay~~

  Attack: Relay
- Demodulation ✔



```
*
001fcff0  01 01 01 01 01 01 01 01  01 01 01 00 00 00 00 00  |................|
001fd000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
001fd420  00 00 00 00 00 00 00 00  01 01 01 01 01 01 01 01  |................|
001fd430  01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01  |................|
*
001fd830  01 01 01 01 01 01 01 00  00 00 00 00 00 00 00 00  |................|
001fd840  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
001fe070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 01  |................|
001fe080  01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01  |................|
```

Clock Recovery

~ 1000 ones

```
001fd420  00 00 00 00 00 00 00 00  01 01 01 01 01 01 01 01
001fd430  01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01
*
001fd830  01 01 01 01 01 01 01 00  00 00 00 00 00 00 00 00
001fd840  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
*
001fe070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 01
001fe080  01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01
*
001fe490  01 01 01 01 00 00 00 00  00 00 00 00 00 00 00 00
001fe4a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
*
001fe8c0  00 00 01 01 01 01 01 01  01 01 01 01 01 01 01 01
001fe8d0  01 01 01 01 01 01 01 01  01 01 01 01 01 01 01 01
*
```

- Problem: **sample rate != signal bit rate** (baudrate)
- => one bit in signal == ~1000 symbols in demodulated signal
- GNURadio has Clock Recovery block, but produces bad results
  - Possible reason: zeroes are ~10% longer than ones for some reason...
- Instead: custom python script

```
01010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101
10101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010
01010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101
10110011010100110101001010101100101100101010101010110010110011010101001101011001101010011010100101010110011010100110101001010
10010101101010011010101001010110010101100101101001011001100101100110101001011001100101100110101001011001100101100110101001010
11001111111111010101010101011001010101010110011010100110101001010101100110101001101010010101011001101010011010100101010110011
```

Decoding

Preamble

```
010101010101010101010101010101010101010101
0110011001011001101001011001111111111
0110011001011001101001011001111111111
0110011001011001101001011001111111111
0110011001011001101001011001111111111111
```

Packet repeats 4 times

Packet data

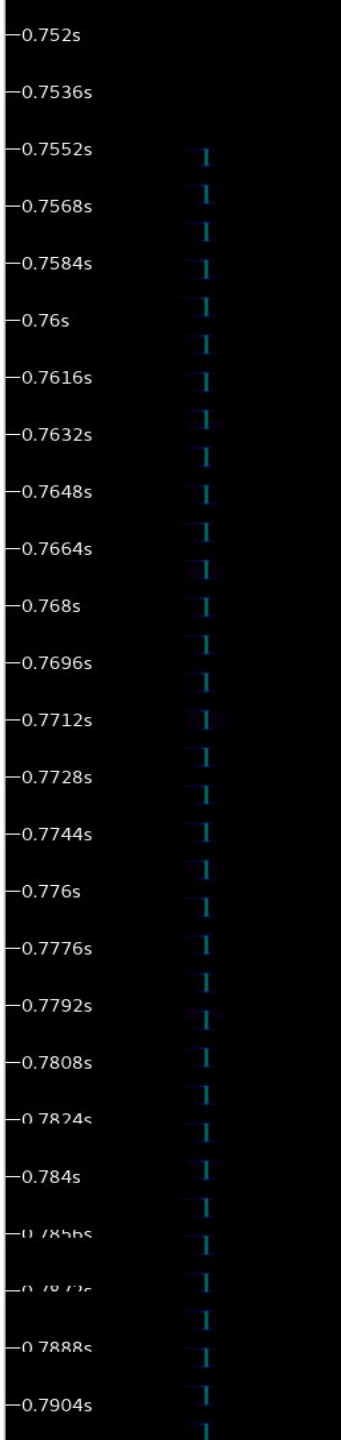
End of Packet marker?

```
010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101
0101010101010110010101010101011001101010011010100101010110010110010101010101011001011001101010100110100101011010100110101010100101011001010110010110100101100110010110011010010110011111111111
0101010101010110010101010101011001101010011010100101010110010110010101010101011001011001101010100110100101011010100110101010100101011001010110010110100101100110010110011010010110011111111111
0101010101010110010101010101011001101010011010100101010110010110010101010101011001011001101010100110100101011010100110101010100101011001010110010110100101100110010110011010010110011111111111
0101010101010110010101010101011001101010011010100101010110010110010101010101011001011001101010100110100101011010100110101010100101011001010110010110100101100110010110011010010110011111111111111
```

Decoding

- Looking at bits: never 3 ones or 3 zeroes in a row
- Possible Manchester Encoding?
  - 10 => 1, 01 => 0 (or vice versa)
- Preamble of zeros (0101010101…)
- Long series of ones => Packet boundary
- Next steps:
  - decode manchester encoding
  - convert bits to bytes
    (176 manchester-bits = 88 bits = 11 bytes)

01017709012f63bc44ca59



Checklist

- Capture signal ✔
  - ~~Attack: Replay~~
  - Attack: Relay
- Demodulation ✔
  - => Bitstream
- Decoding ✔
  - Clock recovery
  - Manchester decoding
  - Bits to bytes

```
00000000000000000000000000000000000000000000000000000000000000000000000000000000
...  00000000000000000000000000000001000000010000001001110111000010001
...  00000001111110101000111011000001001001011010111000000100 1
```

01027709 01f51d824b5c09

Understanding the packets

- Approach: capture different packets and compare them
- Also: guessing and trial + error

```
Lock:
0x1 0x1 0x77 0x9 0x1 0x86 0x18 0x93 0xbb 0xc8 0x19
0x1 0x1 0x77 0x9 0x1 0x4b 0x43 0x79 0x4a 0x45 0xe9
0x1 0x1 0x77 0x9 0x1 0x2f 0x63 0xbc 0x44 0xca 0x59
Unlock:
0x1 0x2 0x77 0x9 0x1 0xf8 0x4c 0xb3 0xd9 0xa3 0xf9
0x1 0x2 0x77 0x9 0x1 0x2b 0xb6 0x82 0x90 0xf2 0x29
0x1 0x2 0x77 0x9 0x1 0x8a 0x46 0x9a 0xff 0xd4 0x39
```

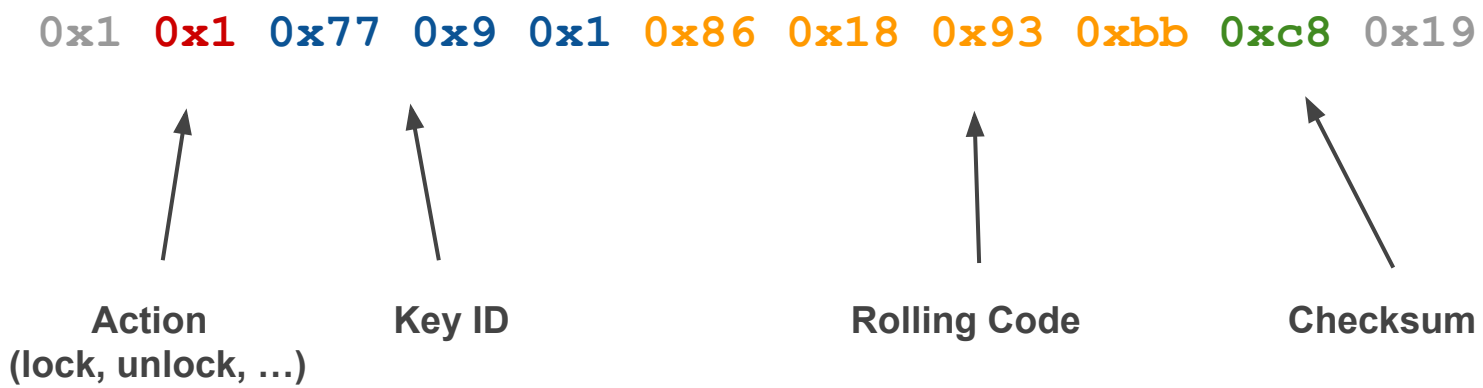
Understanding the packets

- Approach: capture different packets and compare them
- Also: guessing and trial + error

```
Lock:
0x1 0x1 0x77 0x9 0x1 0x86 0x18 0x93 0xbb 0xc8 0x19
0x1 0x1 0x77 0x9 0x1 0x4b 0x43 0x79 0x4a 0x45 0xe9
0x1 0x1 0x77 0x9 0x1 0x2f 0x63 0xbc 0x44 0xca 0x59
Unlock:
0x1 0x2 0x77 0x9 0x1 0xf8 0x4c 0xb3 0xd9 0xa3 0xf9
0x1 0x2 0x77 0x9 0x1 0x2b 0xb6 0x82 0x90 0xf2 0x29
0x1 0x2 0x77 0x9 0x1 0x8a 0x46 0x9a 0xff 0xd4 0x39
```

Understanding the Packets

```
chksum = reduce(operator.xor, a[1:9])
```



Modifying packets

- Question: can we turn a “lock” signal into an “unlock” signal and vice versa?
- Obviously would need to recompute checksum …
- … , but would only work if same rolling code counter is used for both
- Need to test out ⇒ Works!

`0101`7709012f63bc44`ca`59  `0102`7709012f63bc44`c9`59

Checklist

- Capture signal ✔

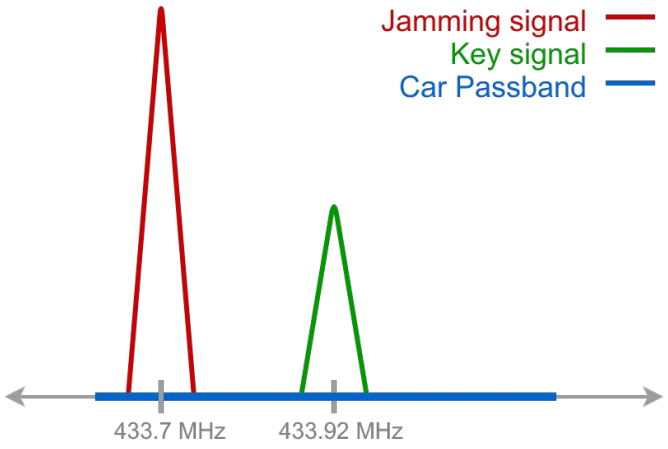
  ~~Attack: Replay~~

  Attack: Relay
- Demodulation ✔
  - => Bitstream
- Decoding ✔
  - Clock recovery
  - Manchester decoding
  - Bits to bytes

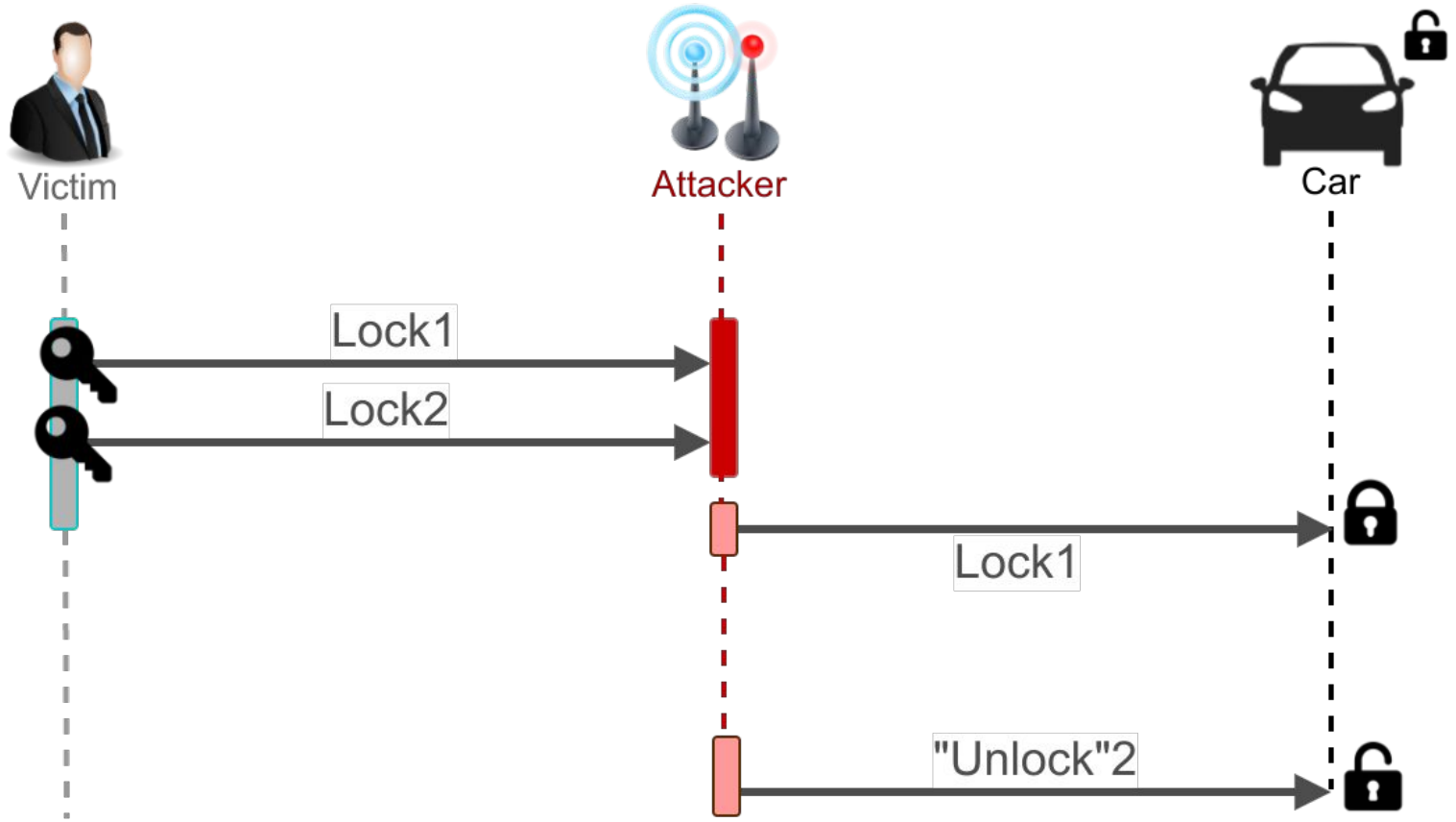
  Attack: Transform “Lock signal” to “Unlock signal”

Final Attack

- Idea:
  - Jam a valid signal and reuse later
  - Record signal while jamming
    - How?
  - Use jammed signal later
- Scenario:
  - Owner parks car
  - Wants to lock it
  - Jam lock signal
  - Turn into unlock signal
  - Use signal when owner is gone
- But: victim will notice?



Final Attack



Summary

- Capture signal ✔
  - ~~Attack: Replay~~
  - Attack: Relay
- Demodulation ✔
  - => Bitstream
- Decoding ✔
  - Clock recovery
  - Manchester decoding
  - Bits to bytes
    - Attack: Transform “Lock signal” to “Unlock signal”
    - Attack: Jamming + Packet Modification