Capturing and reversing wireless keyboard signal

SDR ISRAEL – MEETUP #6 – SEPTEMBER 1, 2016

ARIK YAVILEVICH

Introduction

- Goal: Learning more about radio by capturing and analyzing wireless transmissions of a wireless keyboard and mouse
- Has been done before but usually not with RTL-SDR
- Apparently not as easy as it might seem
- Fun
- Process: Identify, capture, get bytes/bits, reverse protocol

How to do reversing

- Based on EFF Reverse Engineering FAQ
  - Don't violate contracts (NDA, license agreement, etc.)
  - Own the hardware and software
  - Don't break encryption
  - If capturing data
    - Publicly accessible network
    - Or obtaining consent
- https://www.eff.org/issues/coders/reverse-engineering-faq

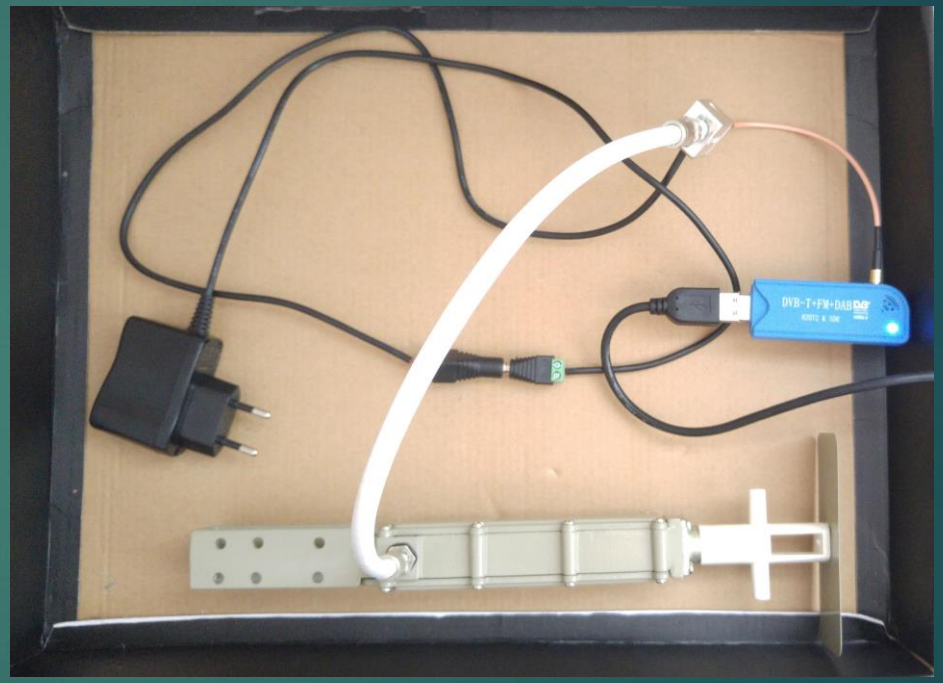
Identify and capture

2.4Ghz space (common ISM)

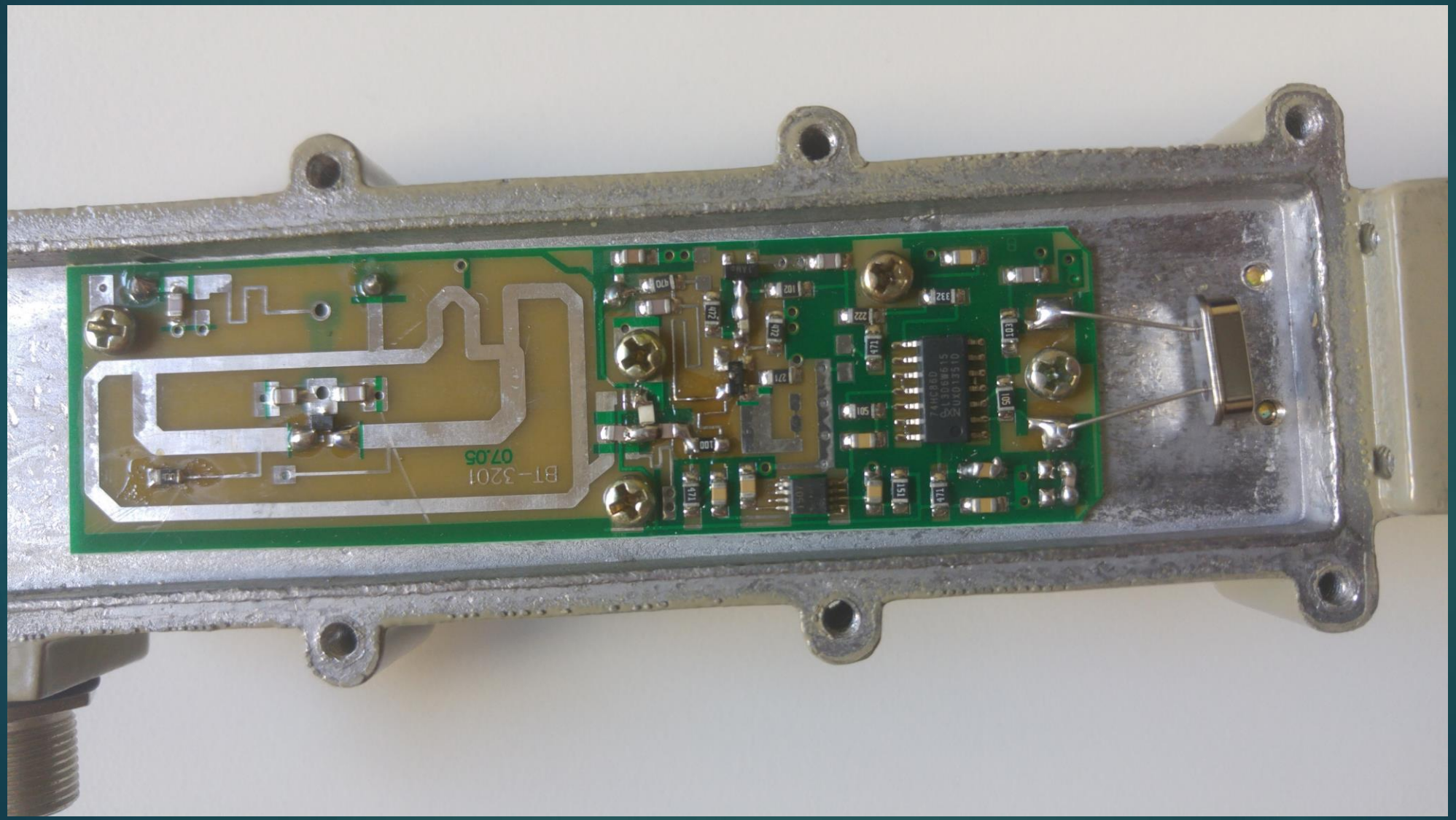
- Most of your Consumer Electronics will be in the 2.4Ghz range
  - Wi-Fi
  - Bluetooth
  - Proprietary protocols
- Almost any device I wanted to experiment with was in this range
- Not in range of RTL-SDR
- Wi-Fi and Bluetooth specifically are too complex to capture with RTL-SDR (especially for a beginner)
  - Wide
  - Hopping

Down converter

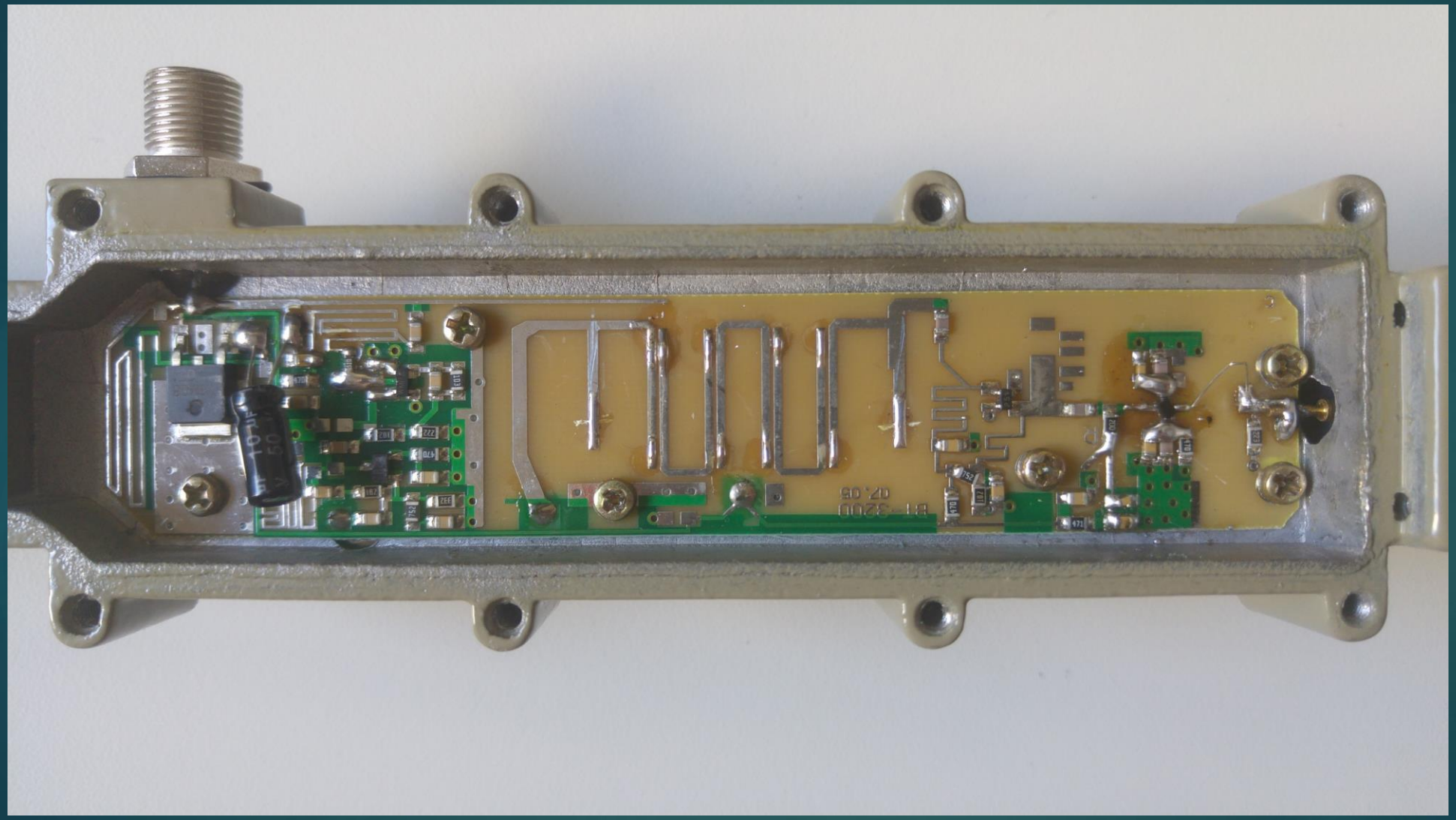
- L.O 1998MHZ MMDS down converter 2.2-2.4GHz
  - L.O Frequency: 1998MHz
  - L.O Stability: ≤±30KHz
- Aliexpress: 12$ device, 10$ shipping, 5$ power = 27$
- Came with a 18V power injector
- Can work with a battery
  - Vout = 8V if Vin>10.5V
- https://en.wikipedia.org/wiki/Digital_down_converter



Down converter - inside



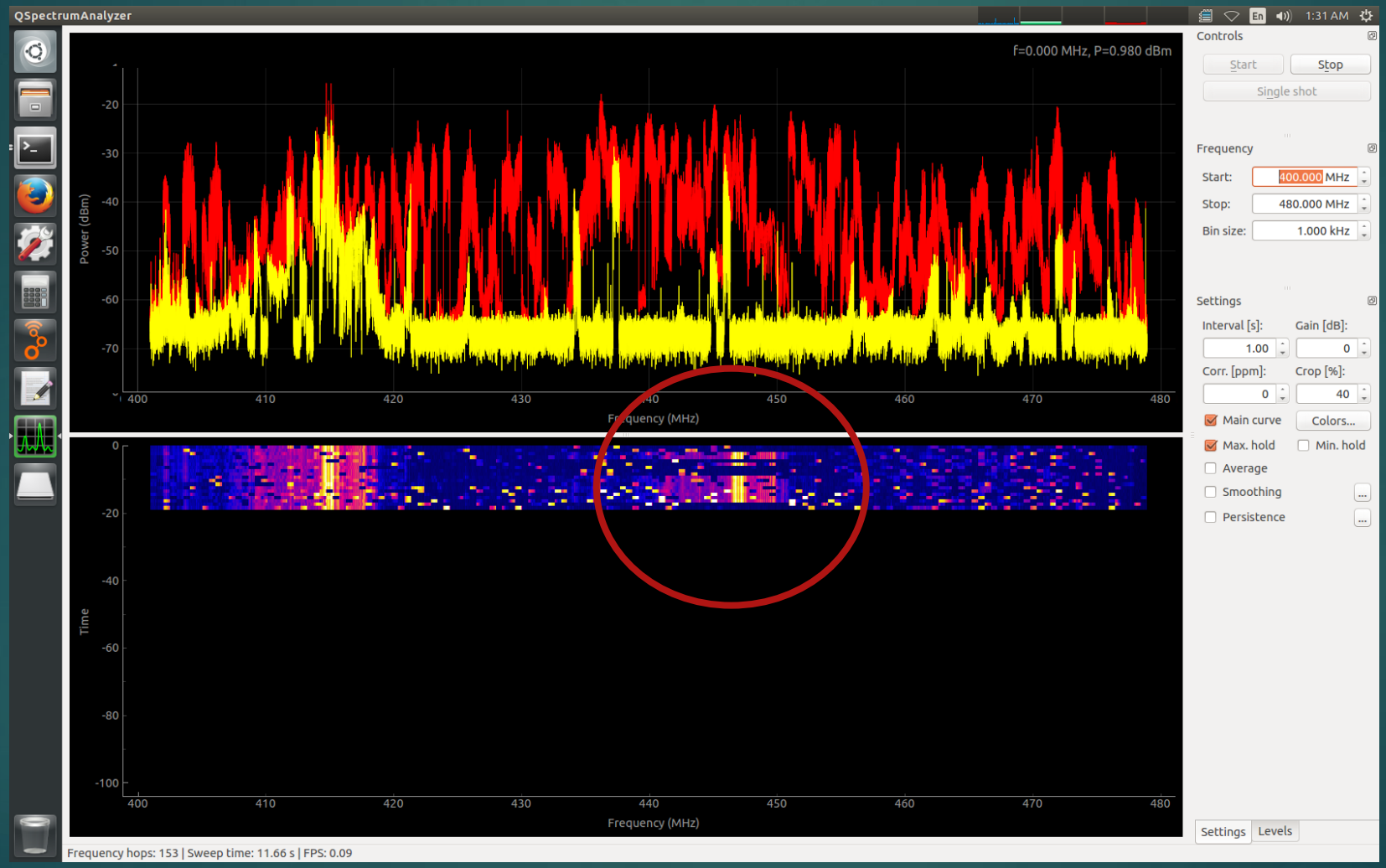
Down converter - inside 2



Scanning

- QSpectrumAnalyzer was mostly useless
  - Busy spectrum
  - Didn't see all devices
  - Was worse with rtl_power_fftw
- Decided to focus on one device that was showing in the scanner
  - Rapoo E2700 wireless keyboard and track pad
- Fosphor is awesome (osmocom_fft –F)
- DEMO
  - Try to identify visually

Scanning – screen shot



Gathering info from open sources

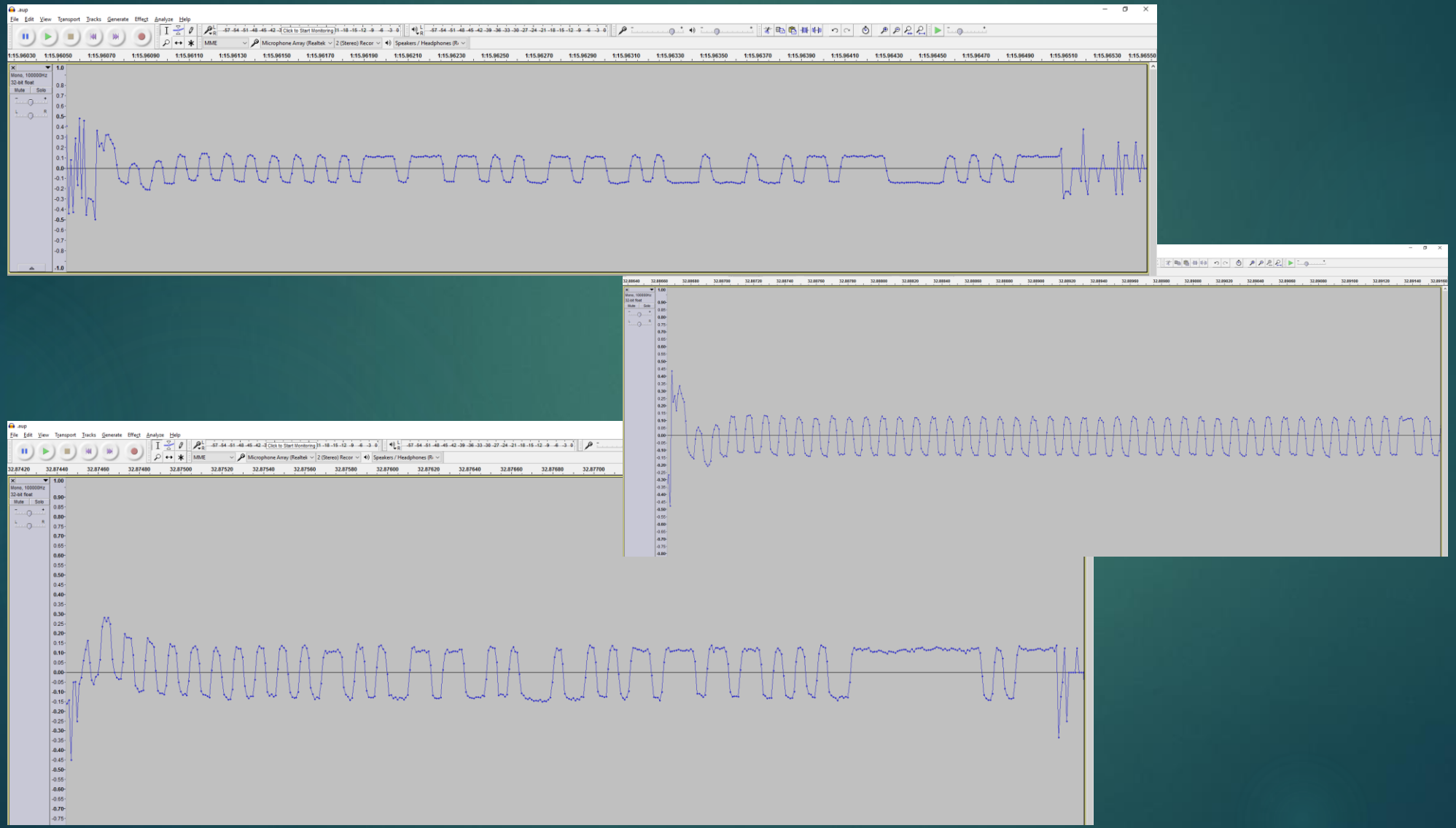
- Rapoo E2700
- FCC ID: PP2E2700
- https://fccid.io/PP2E2700
- What is important?
  - Frequencies
  - Modulation
  - Data rate

Equipment	Wireless Multi-media Touchpad Keyboard
Brand Name	RAPOO
Model Name.	E2700
OEM Brand/Model Name	N/A
Model Difference	N/A
Product Description	The EUT is a Wireless Multi-media Touchpad Keyboard. Product Type: Low Power Communication Device Operation Frequency: 2408~2474 MHz Modulation Type: FSK Date rate: 375Kbps Number of Channel: 67CH .Please see Note 2. Antenna Designation: Integral antenna Antenna Gain(Peak): 3.49 dBi Output Power: 66.75 dBuV/m (AV Max.) Based on the application, features, or specification exhibited in User's Manual, the EUT is considered as an ITE/Computing Device. More details of EUT technical specification, please refer to the User's Manual.
Channel List	Please refer to the Note 2.
Power Source	DC Voltage supplied from 2*AAA Battery
Power Rating	DC 1.5V
Connecting I/O Port(s)	Please refer to the User's Manual

Demodulation

- One approach, rtl_fm
  - rtl_fm -f 447m -s 2000k -g0 > rtl_fm_dump
  - You get binary: 16bit shorts, little endian
- Another approach, GnuRadio
  - DEMO
  - You get binary: 32bit floats, little endian
- Open in Audacity or Baudline as raw
  - Baudline, can't zoom in enough – best for non-burst signals
  - Audacity, needs the signal scaled [-1,1], shows wrong sampling rate
  - DEMO

Demodulation - signals



Digital and binary

Binary

- Perform clock recovery (reduce sample rate)
- Binary slicer
- Correlate access code
  - Also consider “Correlate access code - Tag” + “Tag Debug”
- No good documented modules that help you process the data from that point
- Now what? Write to file?
  - Sucks to work with, takes you out of GRC
  - Not real time
- Followed GRC expectation to write my own module

Writing your own module

- Can be done in Python or C++
- Helper utility called "gr_modtool" – generates a template
- Module contains Blocks
- Follow http://gnuradio.org/redmine/projects/gnuradio/wiki/Guided_Tutorial_GNU_Radio_in_Python#32-Where-Do-Blocks-Come-From
- Expect Python to be slower than C++
- DEMO (code structure)

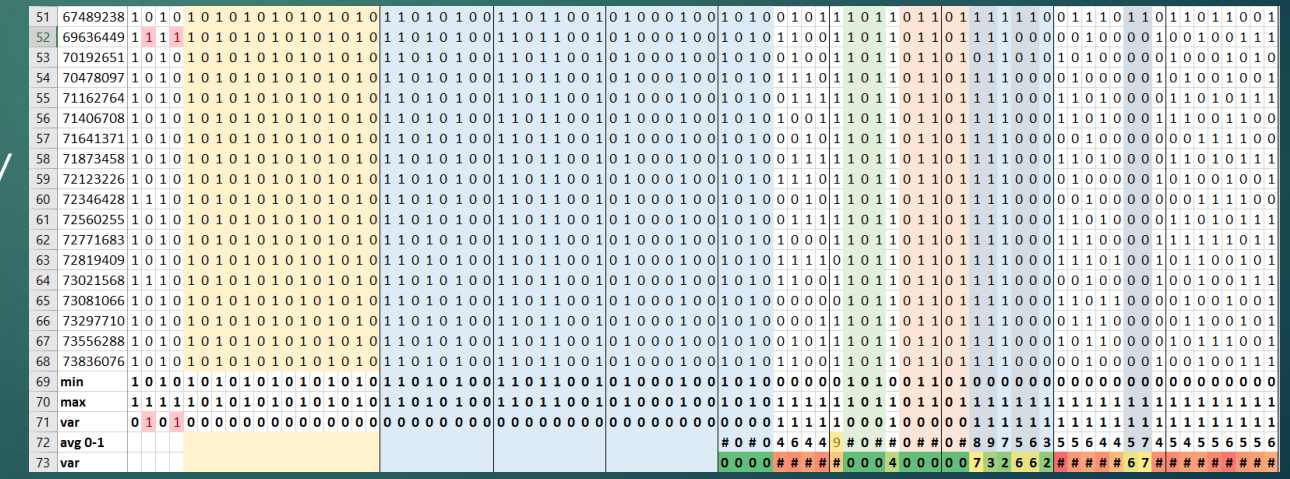
- https://github.com/ayavilevich/gr-aygarage

Frame extractor block

- Generic helper follows “Correlate access code”
- Parameters: “pre length” and “post length”
- Prints to the console a “frame” (sequence of bits) that are surrounding the “access code” detection point
- Later addition: “bits to bytes” and “byte offset”
- Prints sample # and hex representation of the frame
- DEMO (execution)

Reversing - analysis

- Feed the bits to Excel
- Different types of frames
  - Frames that are the same and repeat a lot – assume an “ack”
  - Frames that “start” a segment and are different – assume “data”
- Statistical analysis of data
- Can still see differences,
  - even for frames of same key
  - later learned sometimes
    - key = 0



Reversing - encoding

- Why encoding?
- Ruled out differential encoding because there are runs of 0 and of 1
  - So no Manchester, etc.
- Also tried Gray code, Delay encoding, etc.
- Learned about whitening and different algorithms for that
- Was stuck!
- Read an article that mentioned that a fixed value of 0x5a is being used for whitening. 0x5a = 0101 1010
  - It matched and data started to make sense
- Brute-forced CRC algorithm with CRC RevEng
  - http://reveng.sourceforge.net/

Reversing - protocol

Events

CRC is on the green, data part

0x5A whitening applied on data and CRC part

Keyboard

AA	AA	D4	D9	44	F[seq]	[80]1	[keycode]	[wx]	[yz]
Preamble		Address or sync			Data			CRC 16bit, XMODEM	
					Sequence	Retransmit	Key / 0		

Mouse

AA	AA	D4	D9	44	C[seq]	[x1]	[y1]	[x2]	[y2]	[x3]	[y3]	[x4]	[y4]	[wx]	[yz]
Preamble		Address or sync			Data									CRC 16bit, XMODEM	
					Sequence	Mouse moves, origin is lower left, signed bytes									

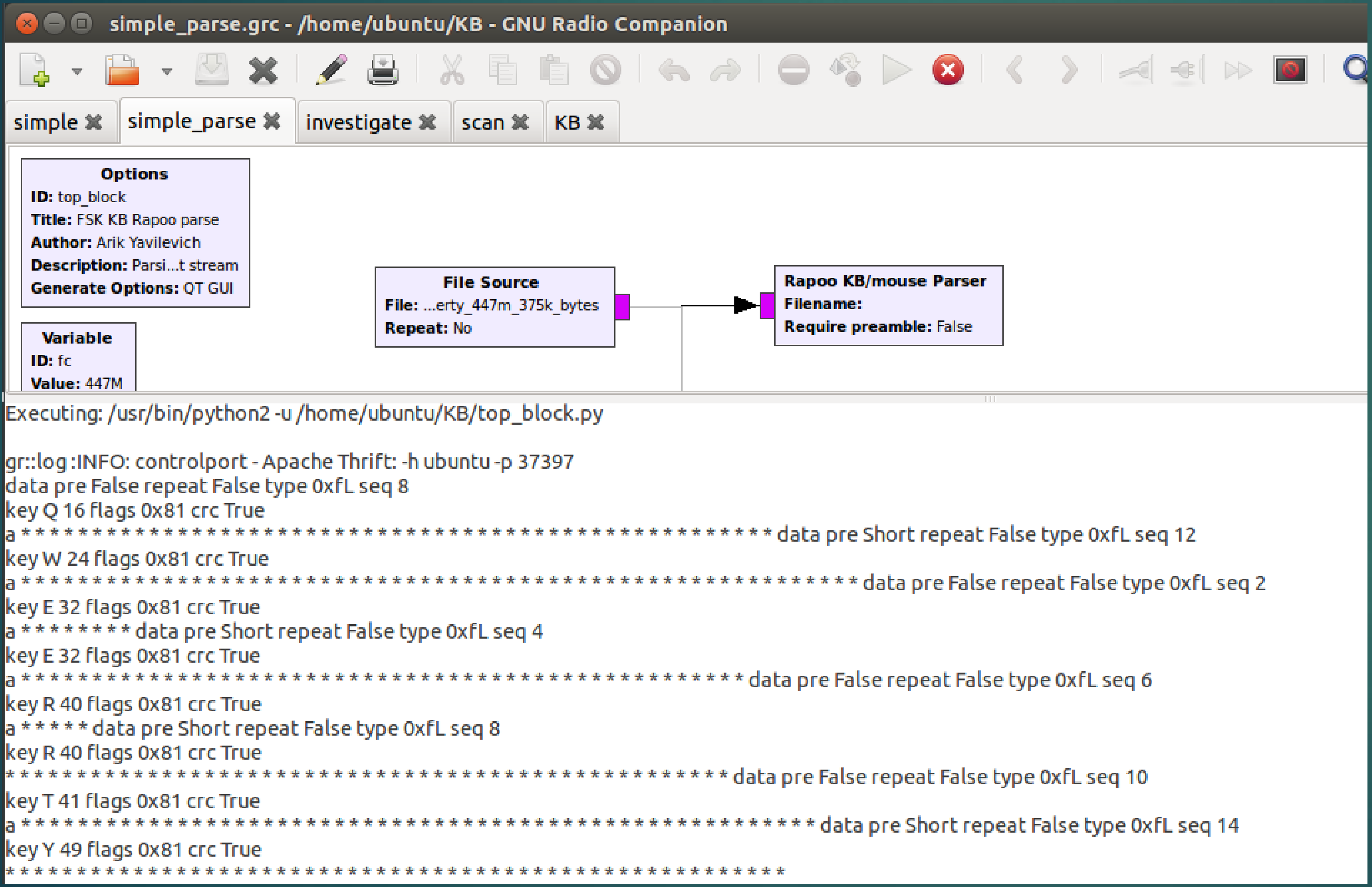
Ack (from dongle)

AA	AA	D4	D9	44	4B	78
Preamble		Address or sync			Const	

Repeat

AA	AA	EE	D4	D9	44	4B	78	2A
Preamble		Const	Address or sync			Const		

Reversing - block



Summary

Challenge - frame misses

- System successfully captures only some of the packets
  - Can be seen in real-time
  - Sequence value skips
  - Repeat bit
  - Timing skips
- Troubleshooting (through manual verification) points to clock-recovery in GRC as the cause of misses and bad bit detection
  - Might not be designed for “burst”
  - Possible to make a different implementation, but no plans to pursue
- Decided not to require the preamble in the parser, rely on address

Challenge - bursts

- Finding a needle in a haystack
- Sending 20 bytes at 2Mbs is just 80 micro second
- Difficult to detect devices
  - Especially when sampling rate unknown
  - Consider over-sampling and statistical approach to detect frames
- Squelch blocks in GRC too slow to respond – designed for speech
- Fosphor makes life easier

Conclusions

- Rapoo communications not encrypted
  - As expected
- Filtering didn't result in improvements – wide signal
- GNU Radio Companion very convenient for prototyping
- Writing your own GRC blocks is totally possible
- Python GRC blocks might be too slow for some uses

References

KeySniffer, 2016

- http://www.keysniffer.net/technical-details/
  - Non Nordic chips, cheap keyboards. QFSK
- https://conference.hitb.org/hitbsecconf2016ams/materials/D1%20COMMSEC%20-%20Marc%20Newlin%20-%20Applying%20Regulatory%20Data%20to%20IoT%20RF%20Reverse%20Engineering.pdf
  - Tech details

KeySweeper, 2015

- http://samy.pl/keysweeper/
- https://github.com/samyk/keysweeper
- http://arstechnica.com/security/2015/01/meet-keysweeper-the-10-usb-charger-that-steals-ms-keyboard-strokes/

Cyber Explorer, 2014

- http://blog.cyberexplorer.me/2014/01/sniffing-and-decoding-nrf24l01-and.html

KeyKeriki, 2008-2010

- http://www.remote-exploit.org/articles/keykeriki_v1_0_-_27mhz/
- http://www.remote-exploit.org/articles/keykeriki_v2_0__8211_2_4ghz/
- http://www.remote-exploit.org/content/keykeriki_ph7d9.pdf
- http://www.remote-exploit.org/content/keykeriki_v2_cansec_v1.1.pdf
  - NRF24x, [preamble, address, flags, payload, CRC]