Exfiltrating data over air gaps via hard drive LED modulation

Denver Cohen
Calvin Figuereo-Supraner

Background

- What are air gaps?
    - Physical disconnect between a secure network and public internet / LANs
- Regulations exist, like TEMPEST (an NSA public spec for COMSEC)
    - Level 1: Attacker has access one meter away from network
    - Level 2: 20m
    - Level 3: 100m (or equivalent free-space attenuation)
- After *infiltrating* an air gapped system, the data needs to leak back out somehow

Background

Lots of existing literature on *exfiltration* techniques.

	Examples	*Bandwidth*	*Distance*
Electromagnetic	GSMem	1000 bps	5m
Acoustic	Fansmitter, DiskFiltration	0.25-3 bps	15m
Thermal	BitWhisper	1-8 b/hr	0.4m
Optical	Keyboard LED, Screen backlight	20-150 bps	Line-of-sight

Project Objective

Get data out of an air gapped computer.

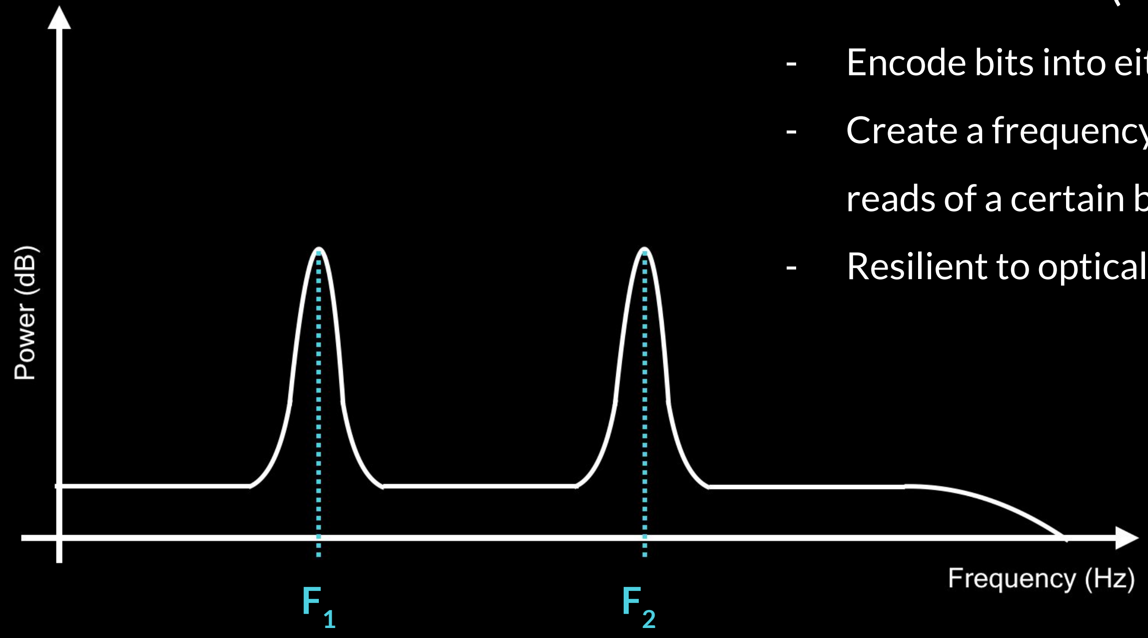
1. Rules
   - No physical access to the building
   - Realistic software and hardware operation
2. Weapon of choice
   - Optical, via hard drive LED
3. Justification
   - Line-of-sight is potentially infinite distance
   - Other optical methods are suspicious

Hard Drive LED

- When does the LED activate?
  - Whenever a read/write queue is present on the HDD
- Problem:
  - LED controlled by the motherboard’s BIOS, not directly by operating system!
- Solution:
  - Use ‘dd’ utility to trigger reads to drive from userspace (*iflag=direct will avoid cache*)
  - The motherboard lights the LED for us

LED Modulation Technique

Improvement on previous LED-IT-GO paper (OOK, Manchester, BFSK variant).

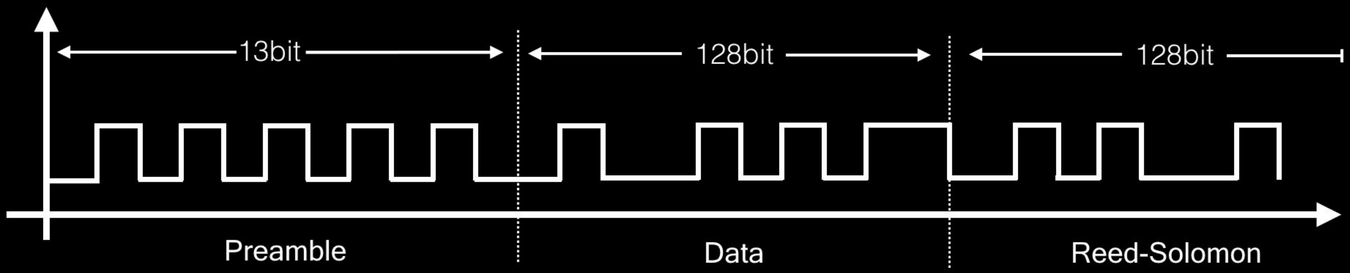


We use true FSK (frequency-shift keying):

- Encode bits into either a symbol of F_1 or F_2
- Create a frequency on the LED using repeated reads of a certain block size
- Resilient to optical interference = long range

Bit Framing

- Two types of packets transmitted on repeat
- HEADER packet
  - 13 bits: Barker code for preamble detection
  - 16 bits: Indicates plaintext length
  - 32 bits: Reed-Solomon code for FEC
- DATA packet(s)
  - 13 bits: Barker code, again
  - 0-128 bits: Plaintext
  - 128 bits: Reed-Solomon code

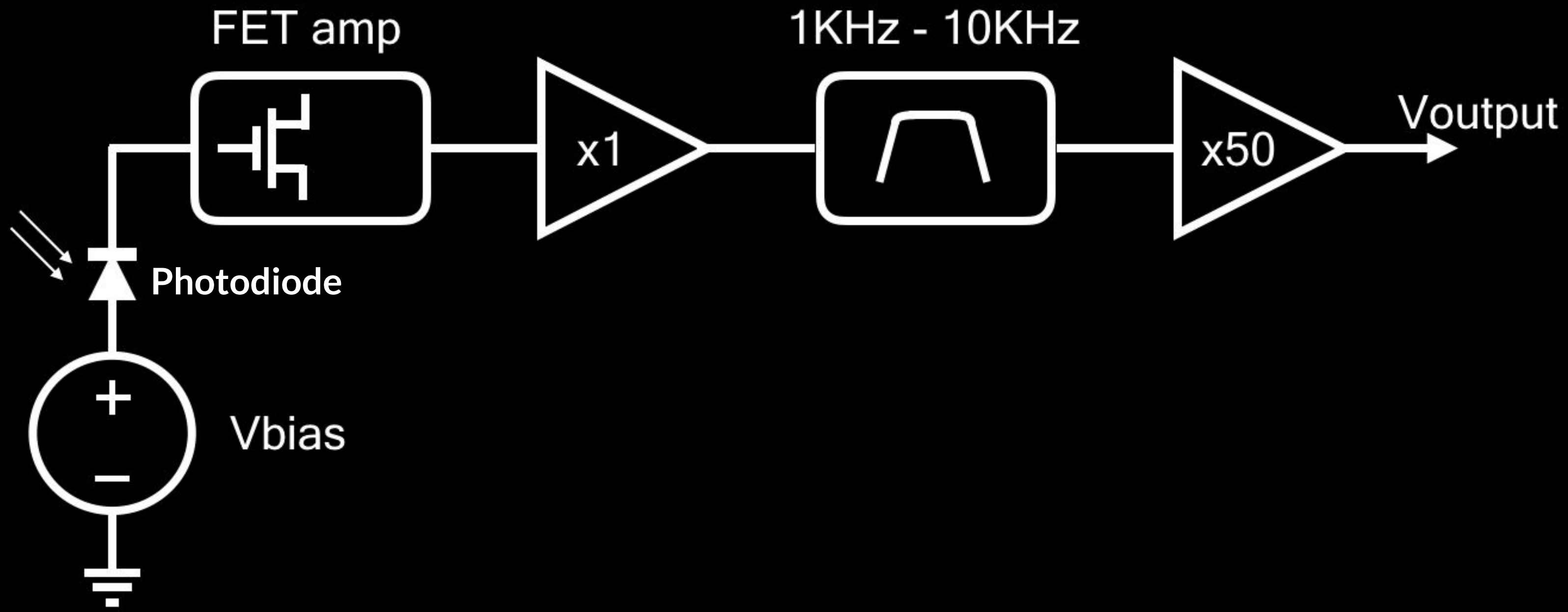


Transmitter Summary

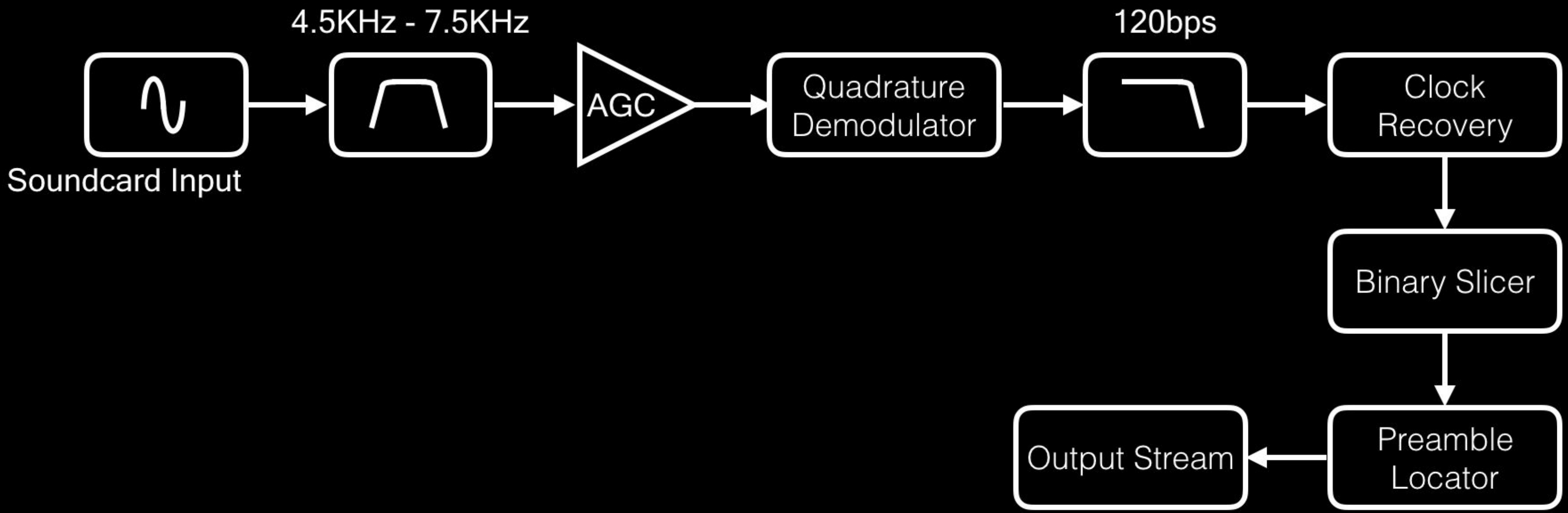
- Technique
  - Repeated ‘dd’ invocations (*read-only from userspace*)
- Throughput
  - We get about 120 bps (*50% overhead, 50% content*)
- Scheme
  - 0: 5 kHz LED toggling, for 8 ms
  - 1: 7 kHz LED toggling, for 8 ms
  - *(HDD goes up to 60 kHz, but LED isn’t fast enough)*

```
$ cat secretfile.txt | ./encoder.py | ./readbfsk.sh
```

Receiver Architecture

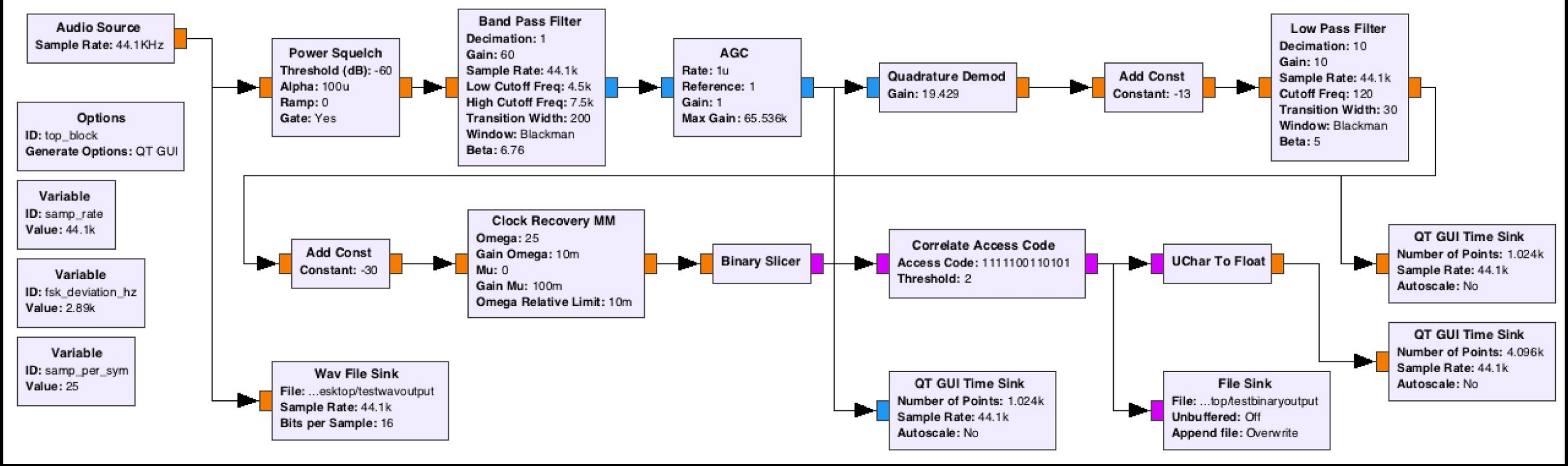


Receiver Software

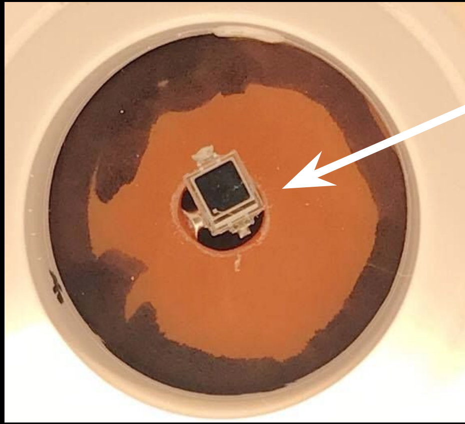


Receiver Software

- GNU Radio is awesome



Receiver Results



Photodiode,
telescope eyepiece mount

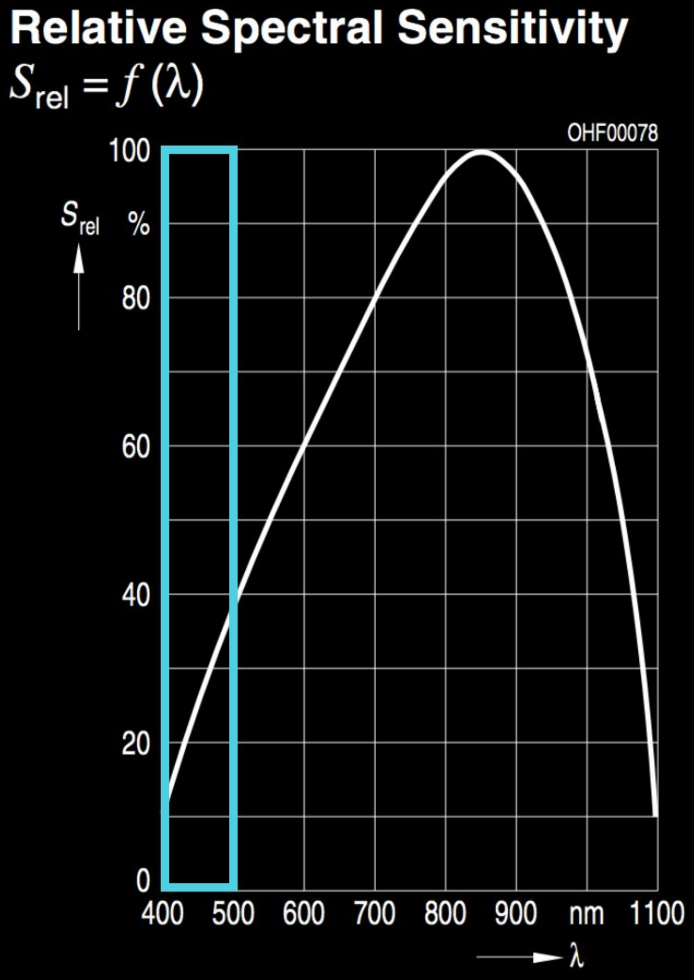
Photodiode on FET,
free-air mount

Overall construction:

- diecast aluminum box
- copper-clad PCB
- dead-bug style

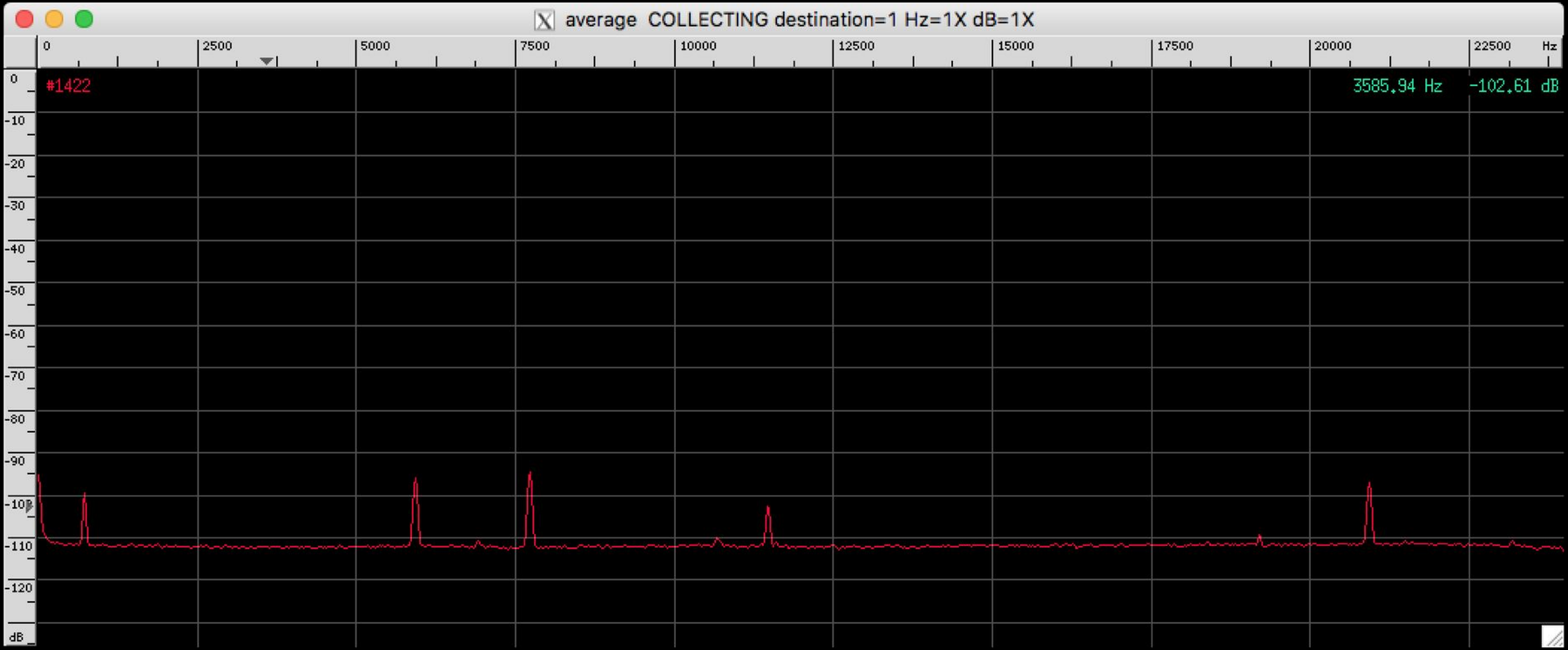
Non-Idealities

1. Photodiode not tuned for our wavelength
    - Hard drive LED is blue
2. Optical spectrum is noisy (building lighting, car headlights)
    - WPI campus is well-lit at night, for safety purposes
3. OS background processes are running
    - Causes burst errors

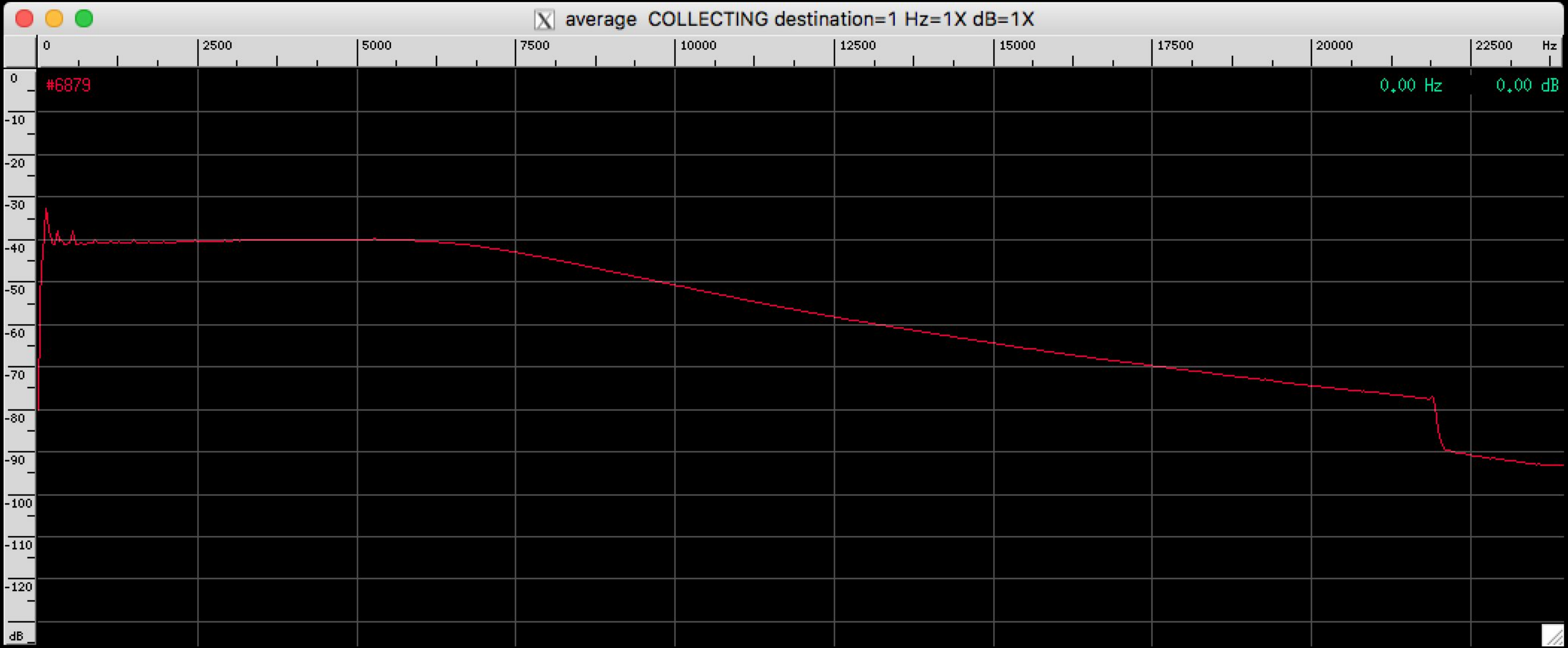


Check your components thoroughly before you build!

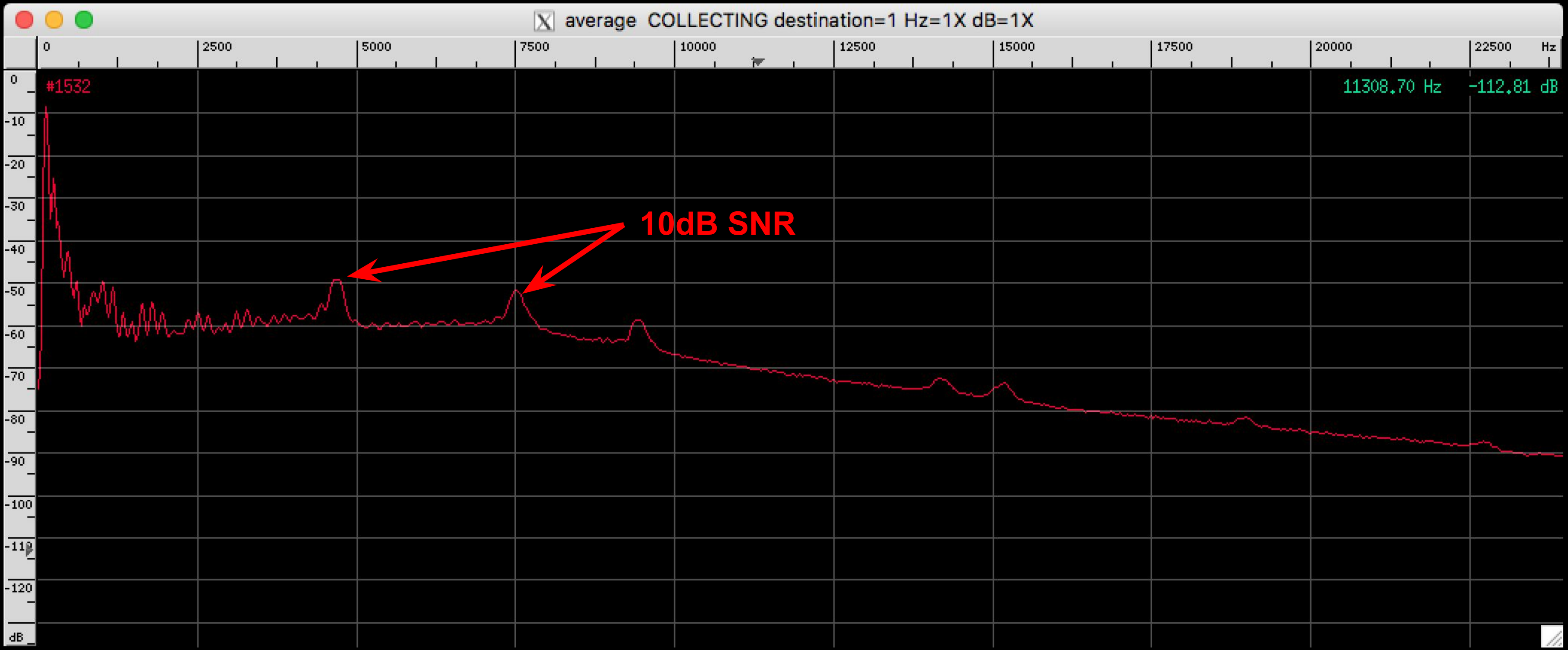
Blue LED
λ = ~400-500 nm



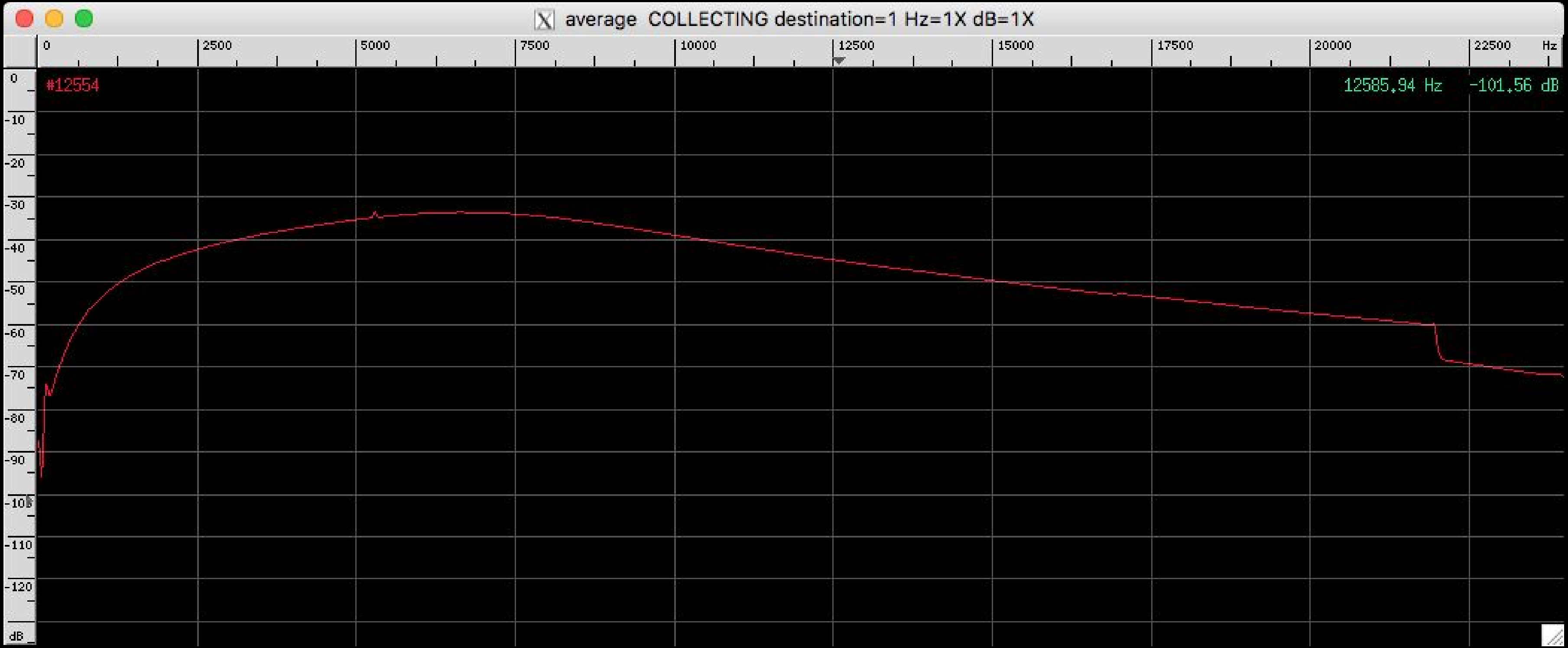
Sound Card noise floor: -110dB



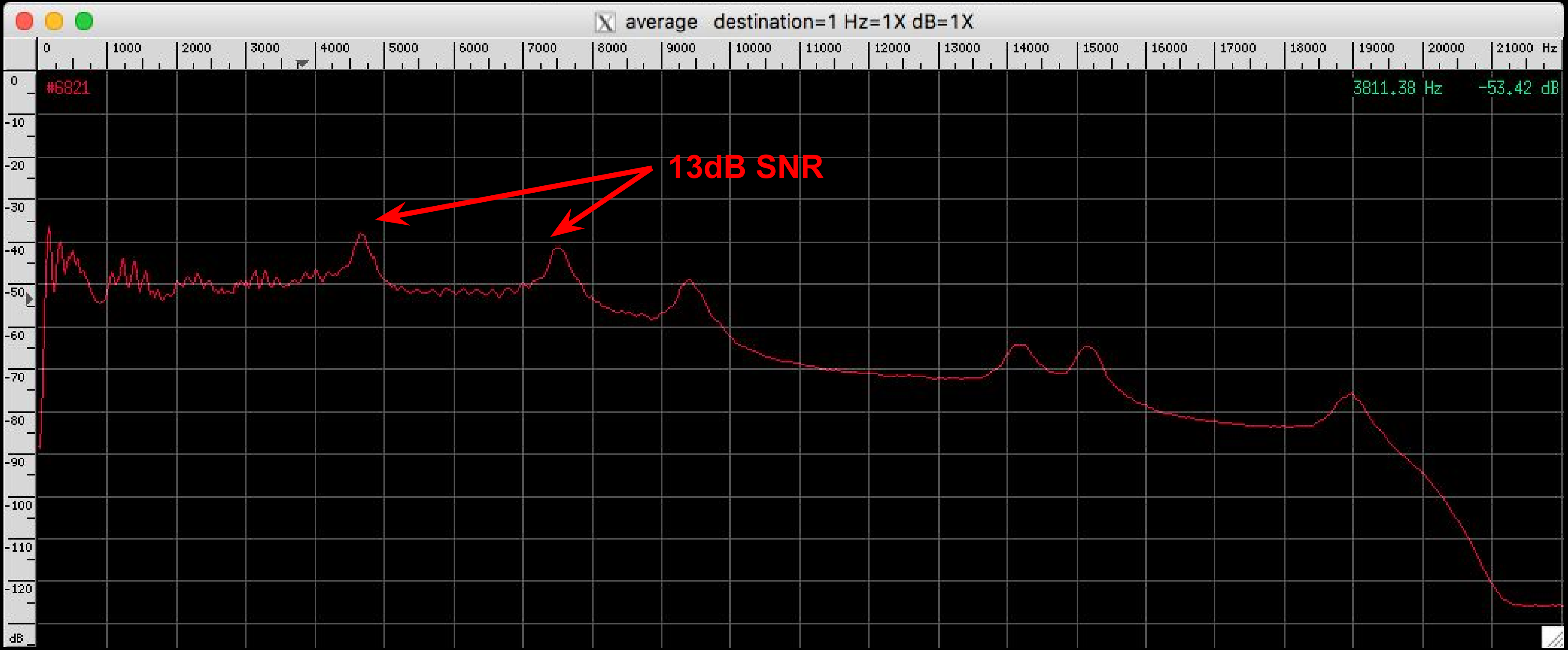
Detector frequency response, before modification



Telescope Run 1: Notice 120Hz spike



Modified Detect Frequency Response: Note low F roll-off



Telescope Run 2: Notice reduced 120Hz spike

Final Results

Final Results

Transferred payload

- Still in the process of decoder algorithm design

Take a majority vote after preamble

```
111110011010101001000
```

Check ASCII validity in a sliding window

```
55157    b'ard drivE LED\x8c\x01...'
```

Thanks to:

- Thomas Eisenbarth, our advisor
- The Applied Crypto and Secure Embedded Systems Laboratory, for the use of their space
- MITRE Collaboratory, for the use of their space