Automatic Modulation Parameter Detection In Practice

Johannes Pohl and Andreas Noack

November 28, 2019

Proprietary wireless protocols everywhere

Example: Smart Home

- Increase comfort of users through wireless sockets, door locks, valve sensors ...
- Devices are designed under size and energy constraints
- Limited resources for cryptography

Risks of Smart Home

- Manufactures design custom *proprietary wireless protocols*
- Hackers may take over households and, e.g., break in without physical traces

How can we speed up the security investigation of proprietary wireless protocols?

Software Defined Radio

Why Software Defined Radios?

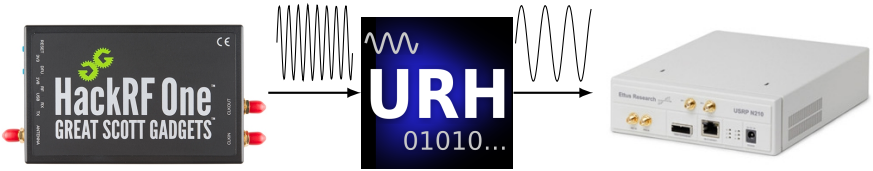
- Send and receive on nearly arbitrary frequencies[a]
- Flexibility and extendability with *custom software*

[a] e.g. HackRF: 1 MHz to 6 GHz

(a) USRP N210

(b) HackRF

Universal Radio Hacker

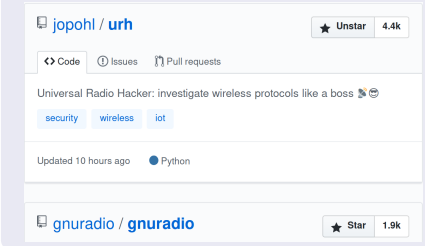


Universal Radio Hacker Popularity

Supported Platforms

Windows, Linux and OS X

Most starred repo on GitHub with #sdr tag



Available at official linux repositories

URH is available in official repositories of **Arch Linux**, **Gentoo**, **Void Linux**, **Fedora** and **openSUSE** (and **homebrew** for macOS).

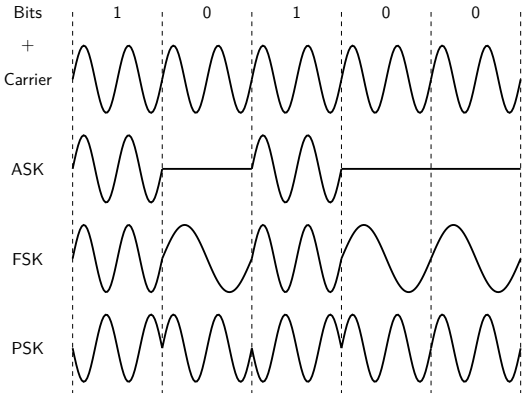
Publications

- DeepSec 2018 [1]
- Blackhat Arsenal USA 2017 [2]
- Blackhat Arsenal Europe 2018 [3]
- WOOT 2018 (USENIX Workshop) [5]
- IoT S&P 2017 (CCS Workshop) [6]

Digital Modulations

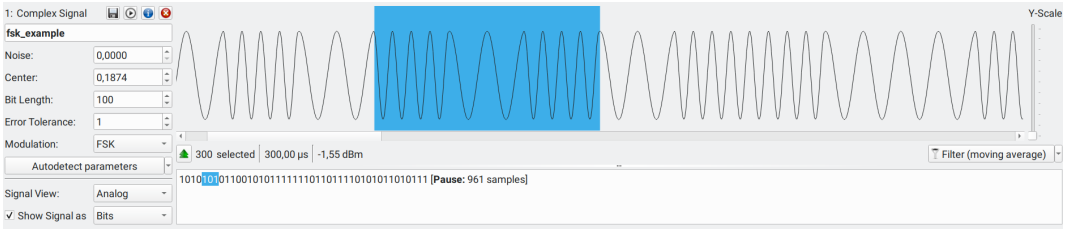
So what is a digital modulation?

- Mapping the binary data, i.e. **bits**, to a **analog carrier** to transport the signal over the air
- Analog signal has the form $A \cdot \sin(2\pi Ft + \varphi)$
- We can transport information in amplitude A, frequency F or phase φ
  - Amplitude Shift Keying (ASK)
  - Frequency Shift Keying (FSK)
  - Phase Shift Keying (PSK)



Interpretation in URH

Demodulating signals made easy



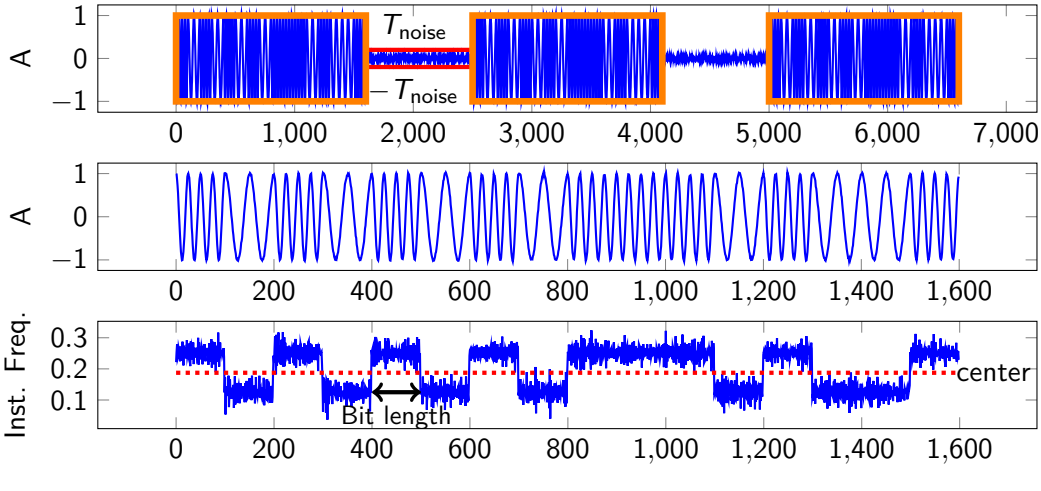
Interpretation Phase Features (apart from demodulation)

- **Synchronized selection** between demodulated and raw signal
- **Signal Editor**, that is, copy, paste, crop, mute signal selections
- Configurable moving average and bandpass **filters**

How can we make this even simpler? **Automatically** detect modulation parameters!

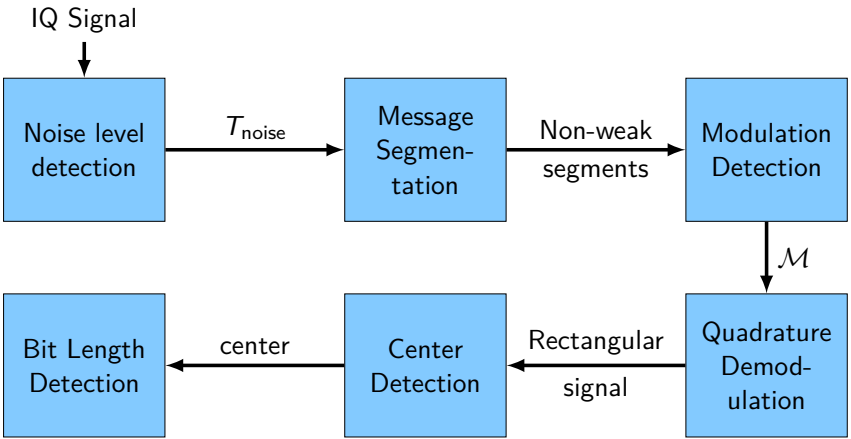
Visualization of Parameters

For all plots: x axis represents current sample



Detecting Modulation Parameters

Automatic detection of modulation type and parameters in Interpretation



Noise Level Detection

Finding the noise level T_{noise} of a signal is the basis for message segmentation and works the following way:

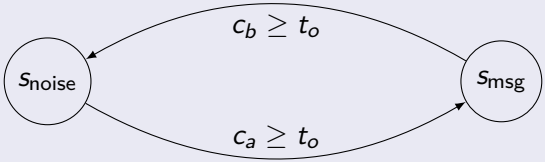
1. Divide the signal into equal sized chunks C_i.
2. For each chunk, calculate the mean magnitude $\overline{m}_i = \overline{|C_i|}$.
3. Get minimum mean magnitude $m_{\text{min}} = \min\{\overline{m}_i : \forall i\}$.
4. Pick magnitudes of chunks those mean magnitudes do not exceed m_{min} by 10%:
$$M_{\text{noise}} = \{|C_j|, \overline{m}_j < 1.1 \cdot m_{\text{min}}\}$$

Finally, the noise level T_{noise} is returned as the maximum of M_{noise}, to cover the full noise range.

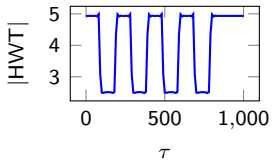
Message Segmentation: Separate Messages from Noise

Message Segmentation Algorithm

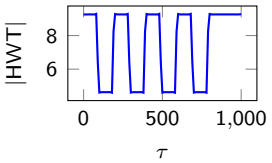
- Based on noise level T_{noise} from previous step
- Must be robust against outliers
- Use two internal states: s_{noise} – reading noise, s_{msg} – reading message.
- Switch states only if consequent samples above/below noise (c_a/c_b) surpass a threshold t_o (=**outlier tolerance**). In practice, $t_o = 10$ samples performs well.



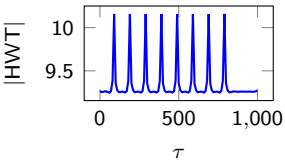
Modulation Detection with help of Wavelet Transform



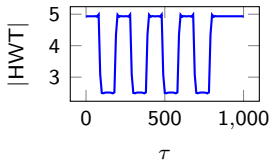
(a) 2-FSK



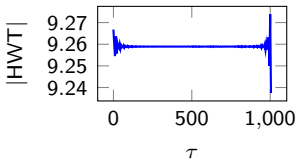
(b) 2-ASK



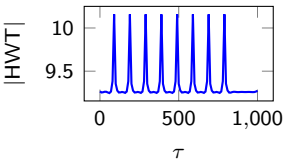
(c) 2-PSK



(d) Normalized 2-FSK



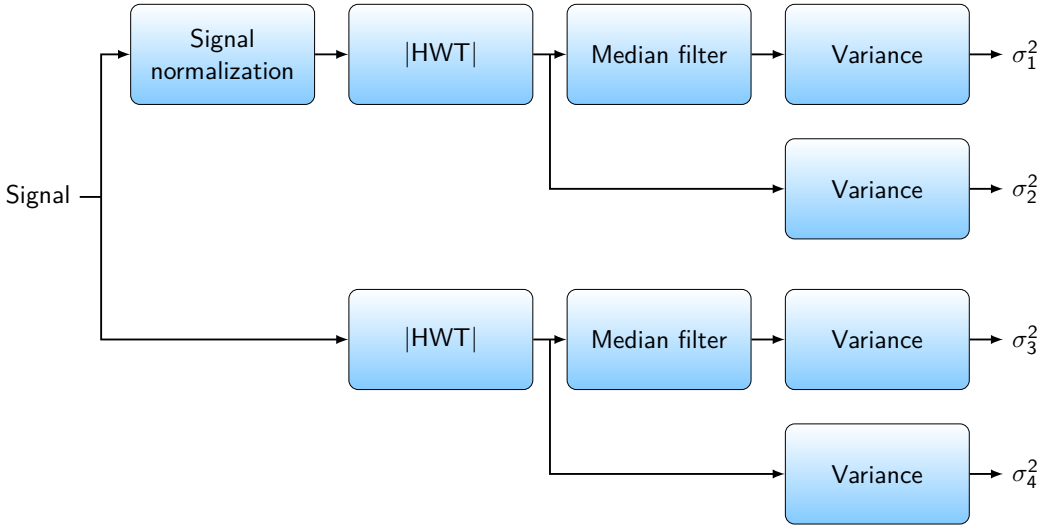
(e) Normalized 2-ASK



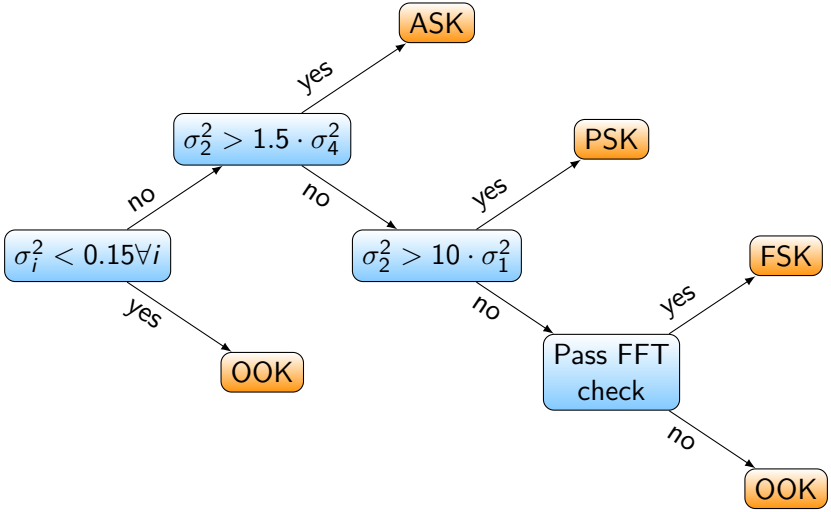
(f) Normalized 2-PSK

Figure: Wavelet transforms for FSK/ASK/PSK signals and their amplitude normalized versions

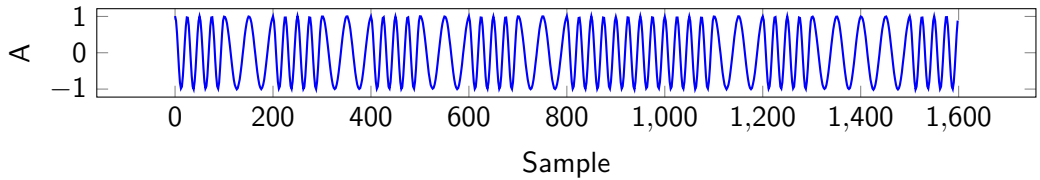
Modulation Detection: Feature Extraction



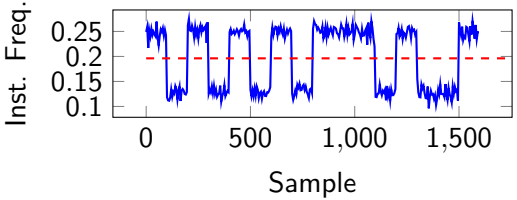
Modulation Detection: Decision Tree



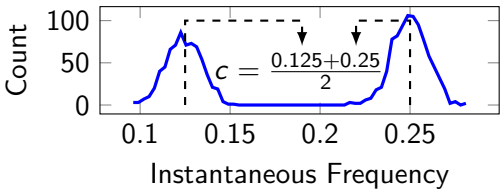
Center Detection: Take mean of histogram peaks



(a) 2-FSK modulated message



(b) Rectangular signal $R(n)$ after Quad Demod



(c) Histogram of $R(n)$ with two peaks

Bit-Length and Tolerance Detection

How to determine the Bit-Length?

- Count subsequent samples above/below found *center* $\Rightarrow$ **plateau lengths vector**
- In theory, vector only contains multiples of bit-length; but: interrupted by outliers
- Set **tolerance** to maximum of values smaller than 5% of maximum plateau length
- **Merge** plateaus based on found tolerance like this:

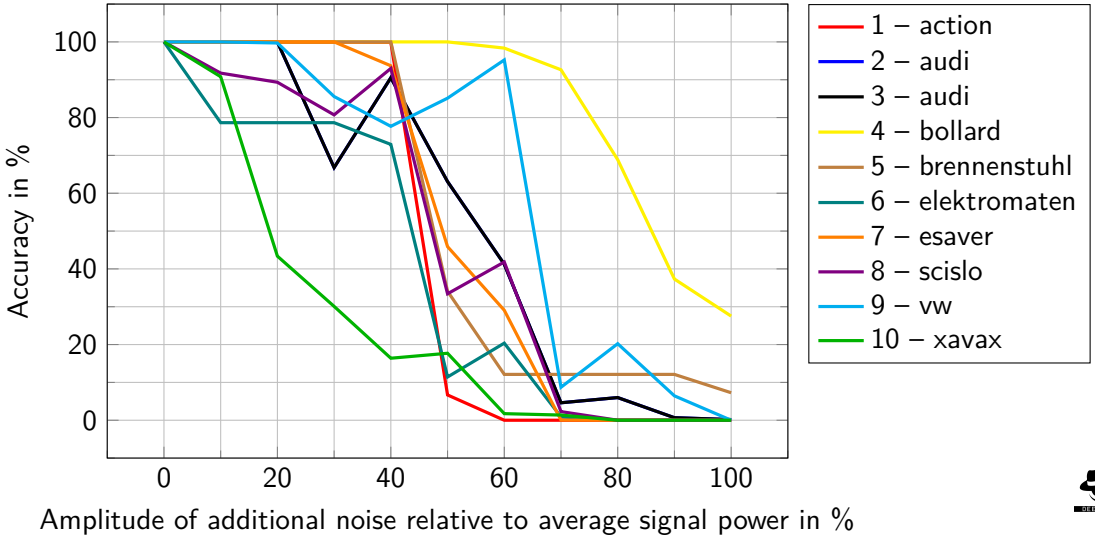
$$(\underbrace{200}_{\text{Hi}}, \underbrace{\mathbf{53}}_{\text{Lo}}, \underbrace{\mathbf{3}}_{\textit{Hi}}, \underbrace{\mathbf{44}}_{\text{Lo}}, \underbrace{100}_{\text{Hi}}) \rightarrow (\underbrace{200}_{\text{Hi}}, \underbrace{\mathbf{100}}_{\text{Lo}}, \underbrace{200}_{\text{Hi}})$$

- **Count** how often each plateau length *nearly* divides other lengths, e.g., for $(40, 40, 40, 40, 40, 30, 50, 30, 90, 40, 40, 80, 160, 30, 50, 30)$ the counts are $N_{\text{near}} = \{30 : 10, 40 : 35, 50 : 3, 80 : 2\}$ so bit-length is 40 (most frequent)

Evaluation with real-world signals

#	Manufacturer	Description	Mod.	Samplerate	SNR	Bitlen	#Msgs	∅ Length
1	Action	remote (four buttons) for a LED light	OOK	2 MS/s	10.8 dB	500	19	11.95 Byte
2	Audi	car open command	OOK	5 MS/s	25.8 dB	2400	1	106 Byte
3	Unknown	command to sink a bus bollard	OOK	1 MS/s	18.9 dB	300	17	5 Byte
4	Brennenstuhl	wireless socket remote (four buttons)	OOK	1 MS/s	11.7 dB	300	64	13 Byte
5	Elektromaten	open command for parking gate	OOK	2 MS/s	16.2 dB	600	11	17 Byte
6	ESaver	remote (four buttons) for a wireless socket	2-FSK	1 MS/s	28.3 dB	100	12	42 Byte
7	RWE	pairing command of a wireless socket	2-FSK	1 MS/s	12.7 dB	100	18	27.17 Byte
8	Scislo	garage door open command	2-FSK	500 kS/s	14.6 dB	200	8	64.75 Byte
9	Volkswagen	car open command	OOK	1 MS/s	32.3 dB	2500	1	53 Byte
10	Xavax	radiator valve temperature command	2-FSK	1 MS/s	21.8 dB	100	6	231.5 Byte

Results when additional noise is added



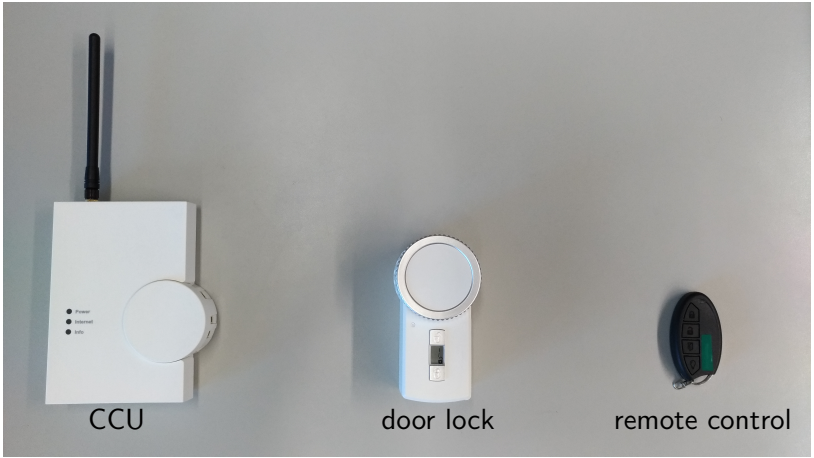
Why does the accuracy of Xavax (Signal #10) drop so early?

(a) Original signal, no additional noise added

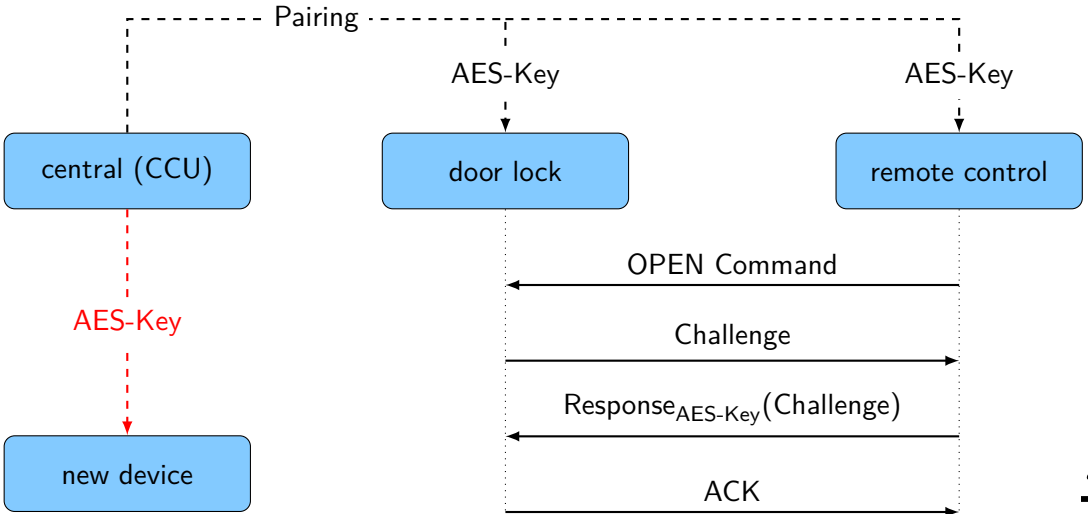
(b) Noise with 20% amplitude of mean signal power added

Figure: Reason for accuracy drop of signal #10: The two weaker messages get marked as noise when noise with 20% amplitude of mean signal power added is added.

Setup



Use-case: Attacking a Wireless Door Lock



Adapting parameters live during a recording

Motivation

- Parameters like center and noise level can change between recordings (varying power levels of devices, changed distances, different antennas)
- Attacking stateful protocols: Messages need to be demodulated live
- Avoid annoying record-analyze-adjust cycles

We have to update noise level and center based on continuously received chunks C_R.

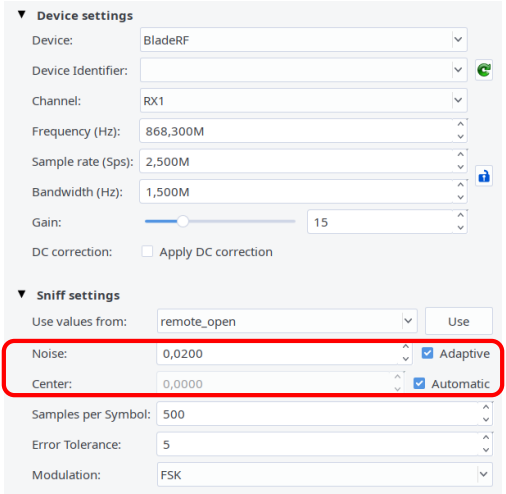
Adaptive Noise Level for received chunk C_R

$$T_{\text{noise}} = \begin{cases} 0.9 \cdot T_{\text{noise}} + 0.1 \cdot \max|C_R| & \text{if } \overline{|C_R|} < T_{\text{noise}} \\ T_{\text{noise}} & \text{else} \end{cases}$$

Automatic Center

Once full message in receive buffer: perform Center Detection from slide 15.

Configuring it in the Universal Radio Hacker



Automatic Parameter Estimation

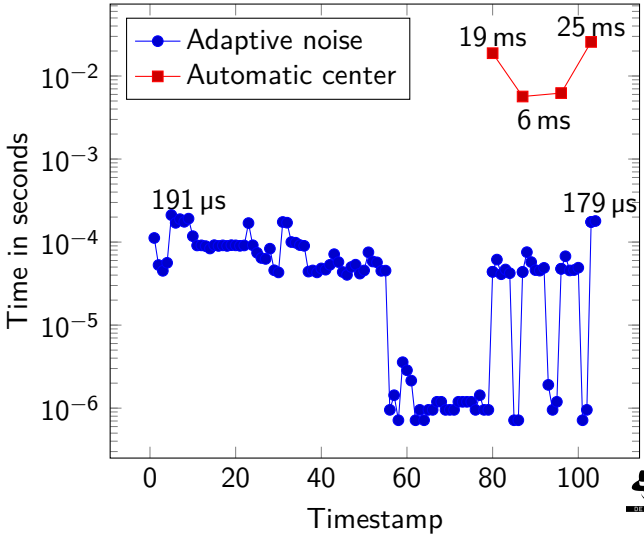
- Noise and Center will be adapted live during simulation time
- Both parameters do not need to be manually changed when using a different SDR or antenna
- Experimental validation proved that setting these parameters automatically is as successful as setting them manually to the correct value

Performance measurement

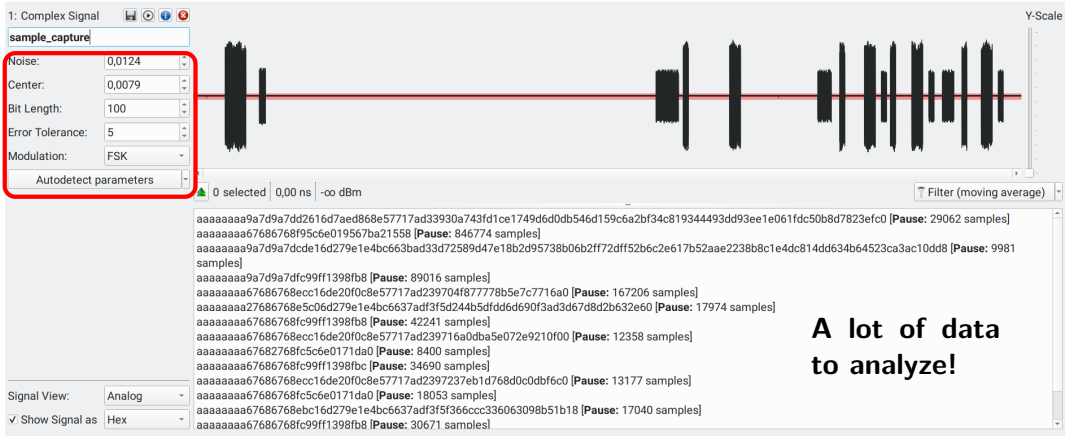
Why performance matters?

- Devices have time windows in which they expect a response
- Time window here: 200 ms
- In this time window, we need to demodulate `Challenge` and, additionally, calculate and modulate correct `Response`

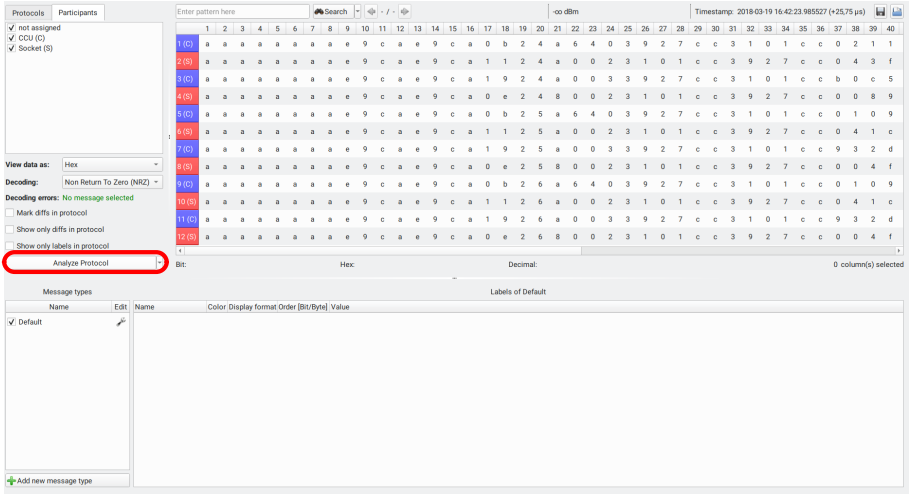
Tested on PC with i7-6700K CPU@4.00GHz and 16GB RAM



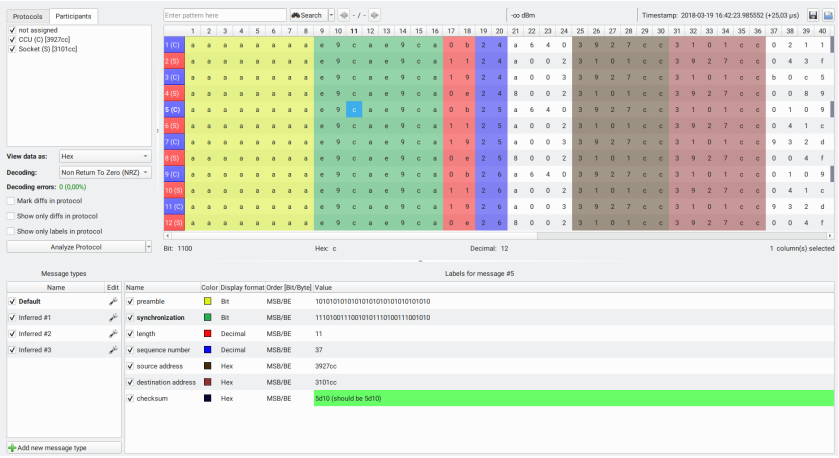
Result of Interpretation for a typical signal



Example Protocol: Communication between two Smart Home Devices



Example Protocol after hitting the Analyze Protocol Button



Published at USENIX WOOT 2019 [4]

Conclusion

- Contribute a multipart system that detects modulation parameters (**modulation type**, **noise level**, **center**, **bit-length** and **tolerance**) of a wireless signal
- Each parameter is returned so it can be fine-tuned afterwards, if needed
- Speed up security investigations and lower hurdle for wireless hacking beginners
- Aimed at *proprietary* protocols with *unknown* modulation parameters operating on frequencies such as 433.92 MHz or 868.3 MHz usually using binary modulations
- Basis for future automations such as automatic protocol field inference
- Future work is support for higher order modulations

https://github.com/jopohl/urh/releases

Contact

- E-Mail: Johannes.Pohl90@gmail.com
  E-Mail: Andreas.Noack@hochschule-stralsund.de
- Slack: https://bit.ly/2LGpsra
- GitHub: https://github.com/jopohl

Publications I

[1] Johannes Pohl. "Attacking Internet of Things with Software Defined Radio (Workshop)". In: *DeepSec* (2018).

[2] Johannes Pohl. "Universal Radio Hacker: Investigate wireless protocols like a boss". In: *Blackhat Arsenal USA* (2017).

[3] Johannes Pohl. "Universal Radio Hacker v2: Simulate Wireless Devices with Software Defined Radio". In: *Blackhat Arsenal Europe* (2018).

[4] Johannes Pohl and Andreas Noack. "Automatic Wireless Protocol Reverse Engineering". In: *13th USENIX Workshop on Offensive Technologies (WOOT 19)*. Santa Clara, CA: USENIX Association, Aug. 2019. URL: https://www.usenix.org/conference/woot19/presentation/pohl.

[5] Johannes Pohl and Andreas Noack. "Universal Radio Hacker: A Suite for Analyzing and Attacking Stateful Wireless Protocols". In: *12th USENIX Workshop on Offensive Technologies (WOOT 18)*. Baltimore, MD: USENIX Association, 2018. URL: https://www.usenix.org/conference/woot18/presentation/pohl.

[6] Johannes Pohl and Andreas Noack. "Universal Radio Hacker: A Suite for Wireless Protocol Analysis". In: *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy (CCS)*. Dallas, Texas, USA: ACM, 2017, pp. 59–60. DOI: 10.1145/3139937.3139951.