

# Automatic Wireless Protocol Reverse Engineering

Johannes Pohl and Andreas Noack

University of Applied Sciences Stralsund

August 13, 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

- Manufacturers 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<sup>a</sup>
- Flexibility and extendability with *custom software*

<sup>a</sup>e.g. HackRF: 1 MHz - 6 GHz

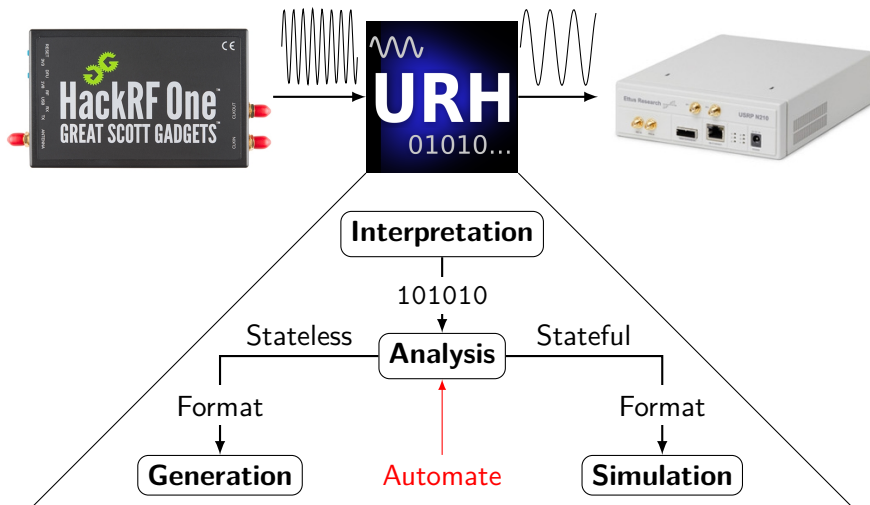


(a) USRP N210



(b) HackRF

# Universal Radio Hacker



# Structure of a Wireless Protocol



## Purpose of Fields

- **Preamble:** Synchronize clocks with fixed preamble pattern, e.g., 101010...
- **Synchronization:** Indicate start of transmission with sync sequence, e.g., 0x9a7d
- **Length:** Contains the size of following data, usually in bytes
- **SRC/DST:** Source / Destination addresses of communicating devices
- **Sequence Number:** Increasing counters used for flow control and freshness
- **Checksum:** Verify integrity of received data (recognize transmission errors)

The **message format** determines the order and type of fields for a message. The **message type** describes which message format to use. A protocol can contain various message types such as DATA and ACK (acknowledgement).

# Example Protocol: Communication between two Smart Home Devices

The screenshot displays the AWRE interface for analyzing a captured communication packet. The left sidebar shows the 'Protocols' tab with a list of protocols: 'not assigned', 'CCU (C)', and 'Socket (S)'. The 'View data as:' dropdown is set to 'Hex', and the 'Decoding:' dropdown is set to 'Non Return To Zero (NRZ)'. The 'Decoding errors:' section shows 'No message selected'. The 'Analyze Protocol' button is highlighted with a red circle. The main packet list shows 12 packets, with the first packet selected. The detailed packet view shows the packet data in a hex dump format, with columns for bit positions (1-40) and rows for data bytes. The packet data is as follows:

Packet	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
1 (C)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	0	b	2	4	a	6	4	0	3	9	2	7	c	c	3	1	0	1	c	c	0	2	1	1
2 (S)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	1	1	2	4	a	0	0	2	3	1	0	1	c	c	3	9	2	7	c	c	0	4	3	f
3 (C)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	1	9	2	4	a	0	0	3	3	9	2	7	c	c	3	1	0	1	c	c	b	0	c	5
4 (S)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	0	e	2	4	8	0	0	2	3	1	0	1	c	c	3	9	2	7	c	c	0	0	8	9
5 (C)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	0	b	2	5	a	6	4	0	3	9	2	7	c	c	3	1	0	1	c	c	0	1	0	9
6 (S)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	1	1	2	5	a	0	0	2	3	1	0	1	c	c	3	9	2	7	c	c	0	4	1	c
7 (C)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	1	9	2	5	a	0	0	3	3	9	2	7	c	c	3	1	0	1	c	c	9	3	2	d
8 (S)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	0	e	2	5	8	0	0	2	3	1	0	1	c	c	3	9	2	7	c	c	0	0	4	f
9 (C)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	0	b	2	6	a	6	4	0	3	9	2	7	c	c	3	1	0	1	c	c	0	1	0	9
10 (S)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	1	1	2	6	a	0	0	2	3	1	0	1	c	c	3	9	2	7	c	c	0	4	1	c
11 (C)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	1	9	2	6	a	0	0	3	3	9	2	7	c	c	3	1	0	1	c	c	9	3	2	d
12 (S)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	0	e	2	6	8	0	0	2	3	1	0	1	c	c	3	9	2	7	c	c	0	0	4	f

The bottom section of the interface shows the 'Message types' table, which is currently empty. The 'Default' message type is selected. The 'Add new message type' button is visible at the bottom left.

# Example Protocol after hitting the Analyze Protocol Button

Protocols Participants

- ☒ not assigned
- ☒ CCU (C) [3927cc]
- ☒ Socket (S) [3101cc]

View data as: Hex

Decoding: Non Return To Zero (NRZ)

Decoding errors: 0 (0.00%)

- ☐ Mark diffs in protocol
- ☐ Show only diffs in protocol
- ☐ Show only labels in protocol

Analyze Protocol

Enter pattern here Search -∞ dBm Timestamp: 2018-03-19 16:42:23.985552 (+25,03 µs)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
1 (C)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	0	b	2	4	a	6	4	0	3	9	2	7	c	c	3	1	0	1	c	c	0	2	1	1
2 (S)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	1	1	2	4	a	0	0	2	3	1	0	1	c	c	3	9	2	7	c	c	0	4	3	f
3 (C)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	1	9	2	4	a	0	0	3	3	9	2	7	c	c	3	1	0	1	c	c	b	0	c	5
4 (S)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	0	e	2	4	8	0	0	2	3	1	0	1	c	c	3	9	2	7	c	c	0	0	8	9
5 (C)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	0	b	2	5	a	6	4	0	3	9	2	7	c	c	3	1	0	1	c	c	0	1	0	9
6 (S)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	1	1	2	5	a	0	0	2	3	1	0	1	c	c	3	9	2	7	c	c	0	4	1	c
7 (C)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	1	9	2	5	a	0	0	3	3	9	2	7	c	c	3	1	0	1	c	c	9	3	2	d
8 (S)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	0	e	2	5	8	0	0	2	3	1	0	1	c	c	3	9	2	7	c	c	0	0	4	f
9 (C)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	0	b	2	6	a	6	4	0	3	9	2	7	c	c	3	1	0	1	c	c	0	1	0	9
10 (S)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	1	1	2	6	a	0	0	2	3	1	0	1	c	c	3	9	2	7	c	c	0	4	1	c
11 (C)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	1	9	2	6	a	0	0	3	3	9	2	7	c	c	3	1	0	1	c	c	9	3	2	d
12 (S)	a	a	a	a	a	a	a	a	e	9	c	a	e	9	c	a	0	e	2	6	8	0	0	2	3	1	0	1	c	c	3	9	2	7	c	c	0	0	4	f

Bit: 1100 Hex: c Decimal: 12 1 column(s) selected

Message types

Name	Edit	Name	Color	Display format	Order [Bit/Byte]	Value
<input checked="" type="checkbox"/> Default		<input checked="" type="checkbox"/> preamble	Yellow	Bit	MSB/BE	101010101010101010101010101010
<input checked="" type="checkbox"/> Inferred #1		<input checked="" type="checkbox"/> synchronization	Green	Bit	MSB/BE	11101001110010101110100111001010
<input checked="" type="checkbox"/> Inferred #2		<input checked="" type="checkbox"/> length	Red	Decimal	MSB/BE	11
<input checked="" type="checkbox"/> Inferred #3		<input checked="" type="checkbox"/> sequence number	Blue	Decimal	MSB/BE	37
		<input checked="" type="checkbox"/> source address	Brown	Hex	MSB/BE	3927cc
		<input checked="" type="checkbox"/> destination address	Dark Red	Hex	MSB/BE	3101cc
		<input checked="" type="checkbox"/> checksum	Dark Blue	Hex	MSB/BE	5d10 (should be 5d10)

+ Add new message type

# Overview of Proposed Algorithm

Raw data with messages and RSSI

0.31 dBm	AA D9 04 13 37 01 28
0.75 dBm	AA D9 02 13 EB
0.80 dBm	AA D9 04 37 13 02 7B
0.27 dBm	AA AA D9 02 37 DD
0.34 dBm	AA D9 04 13 37 03 26
0.79 dBm	AA D9 02 13 EB

Preprocessor

AA D9 04 13 37 01 28
AA D9 02 13 EB
AA D9 04 37 13 02 7B
AA AA D9 02 37 DD
AA D9 04 13 37 03 26
AA D9 02 13 EB

Checksum Engine	SEQ Nr Engine	Address Engine	Length Engine
04 13 37 01 28	04 13 37 01 28	04 13 37 01 28	04 13 37 01 28
02 13 EB	02 13 EB	02 13 EB	02 13 EB
04 37 13 02 7B	04 37 13 02 7B	04 37 13 02 7B	04 37 13 02 7B
02 37 DD	02 37 DD	02 37 DD	02 37 DD
04 13 37 03 26	04 13 37 03 26	04 13 37 03 26	04 13 37 03 26
02 13 EB	02 13 EB	02 13 EB	02 13 EB

Result: Aggregated protocol fields

Message Type 1 (DATA)

Preamble	SYNC	LEN	DST	SRC	SEQ	CRC
----------	------	-----	-----	-----	-----	-----

Message Type 2 (ACK)

Preamble	SYNC	LEN	DST	CRC
----------	------	-----	-----	-----

## Design Goals

- Work on limited number of messages
- Tolerant against transmission errors
- Bootstrap unknown protocols but also consider **prior knowledge**
- Work solely on captured messages, i.e., program binary is not accessible



# Preprocessing: Align messages on (unknown) sync words

## Preprocessor

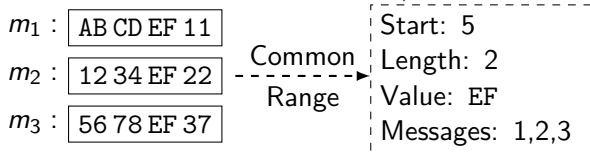
AA D9	04 13 37 01 28
AA D9	02 13 EB
AA D9	04 37 13 02 7B
AA AA D9	02 37 DD
AA D9	04 13 37 03 26
AA D9	02 13 EB

## Purpose of Preprocessing

- Identify **preamble**:  $(a^n b^m)^k$  with  $a, b \in \{0, 1\}$ ,  $a \neq b$  and  $n, m, k \in \mathbb{N}^+$ , e.g., 10101010 with  $n = m = 1$  and  $k = 4$ .
- Identify **sync word(s)**
- **Align** messages on sync word(s): Pass only data behind sync to subsequent engines

# Overview Field Type Inference

- 1 Assign messages to engine-specific **clusters**. For example, the length engine clusters messages based on their physical length in bytes.
- 2 Find **common ranges** inside and/or between clusters.



- 1 Score common ranges with an engine-specific *scoring function*.
  - 2 Return common ranges with highest score if they surpass a minimum score  $s_{\min}$ .
  - 3 If possible, merge the resulting ranges.
- 3 Add found labels to the current *message type* or create a new one, if necessary.

# Included Engines

## Length Engine

- Cluster messages by physical length
- Give higher score to ranges those decimal value matches physical length

## Address Engine

- Assign a participant to every message
- Infer **participant address candidates**
- Find fields with address candidates

## Sequence Number Engine

- Calculate a matrix  $E$  of decimal differences between adjacent messages
- Evaluate columns of  $E$  with only constants or zeros (when SeqNr is constant between some messages)

## Checksum Engine

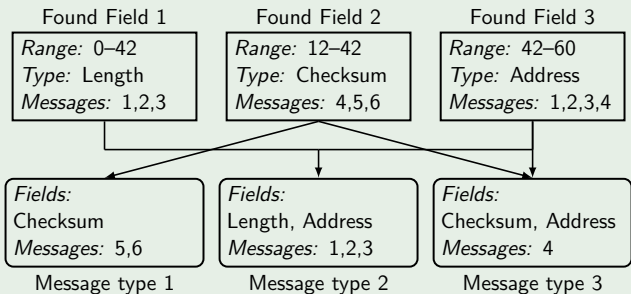
- Find checksums such as CRC16-CCITT by testing common CRC parameters and other checksums
- CRC testing algorithm uses a CRC cache for increased performance

# Message Type Creation and Assignment

## Assignment of Message Types

- Engines return a set of labels
- Group these labels into message types based on their **message indices**
- Create message type for non-overlapping fields with matching message indices
- In conflict case: Choose range(s) that maximize the total score of message type

## Example for three found fields



# Considering Prior Knowledge

## Rules for prior knowledge

- ① Labels must not be changed.
- ② Labels must not be removed from a message type.
- ③ Messages must keep their assigned message type.

## Dealing with prior knowledge

- Skip engines of already present fields
- Engines ignore all ranges of a message that are already labeled
- If new message type needs to be created: **split** original one (=copy over all labels)

## Run this for each message type to consider prior knowledge

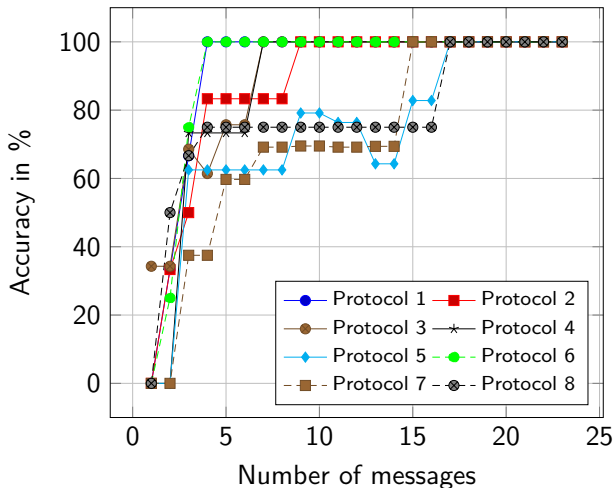
```
1 while new_fields_found and max_iteration_not_exceeded:
2     for mt in existing_message_types:
3         new_fields = []
4         for engine in engines:
5             if field_of_engine not in mt:
6                 new_fields.extend(engine.run(mt))
7         add_to_message_type(mt, new_fields) # Split message type if necessary
```

# Overview of Generated Protocols

**Table:** Properties of tested protocols whereby  $\times$  means field is not present and  $N_P$  is the number of participants.

#	Comment	$N_P$	Message Type	Even/odd message data	Size of field in bit (BE=Big Endian, LE=Little Endian)						
					Preamble	Sync	Length	SRC	DST	SEQ Nr	CRC
1	common protocol	2	data	8/64 byte	8	16	8	16	16	8	$\times$
2	unusual field sizes	2	data	8/64 byte	<b>72</b>	16	8	<b>24</b>	<b>24</b>	16 (BE)	$\times$
3	contains ack and CRC8 CCITT	2	data	10/10 byte	16	16	8	16	16	8	<b>8</b>
			ack	$\times$	16	16	8	$\times$	16	$\times$	<b>8</b>
4	contains ack and CRC16 CCITT	2	data	8/64 byte	16	16	8	16	16	$\times$	<b>16</b>
			ack	$\times$	16	16	8	$\times$	16	$\times$	<b>16</b>
5	three participants with ack frame	3	data	8/64 byte	16	16	8	16	16	8	$\times$
			ack	$\times$	16	16	8	$\times$	16	$\times$	$\times$
6	short address	2	data	8/64 byte	$\times$	16	8	<b>8</b>	$\times$	8	$\times$
7	four participants, varying preamble size, varying sync words	4	data	8/8 byte	<b>16</b>	<b>16</b>	8	<b>24</b>	<b>24</b>	$\times$	16
			ack	$\times$	<b>8</b>	<b>16</b>	$\times$	$\times$	<b>24</b>	$\times$	16
			kex	64/64 byte	<b>24</b>	<b>16</b>	$\times$	<b>24</b>	<b>24</b>	$\times$	16
8	nibble fields + LE	1	data	542/260 byte	<b>4</b>	<b>4</b>	16 (LE)	$\times$	$\times$	16 (LE)	$\times$

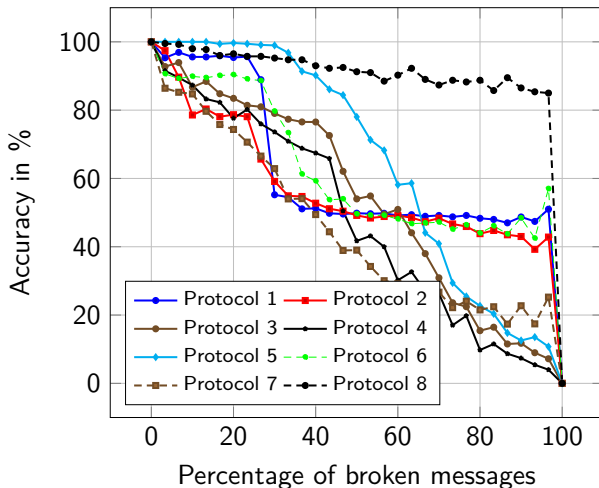
# Test against number of messages



## Results

- 100% accuracy for all protocols when more than 17 messages are available
- 100% accuracy for five out of eighth protocols when at least 7 messages are available
- Protocol 5 and 7 have more participants involved so algorithm needs more messages to infer address fields correctly

# Test against errors

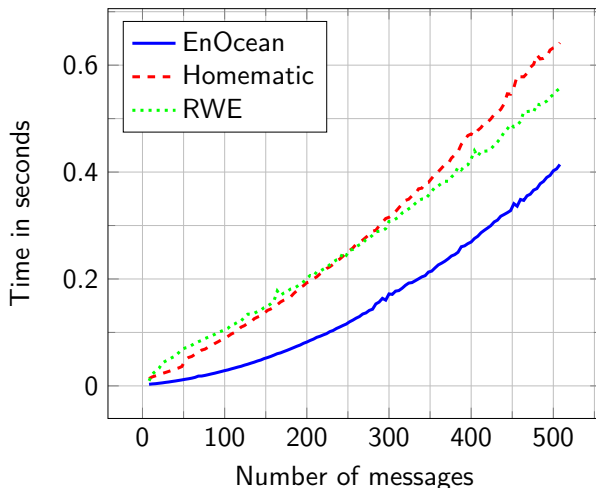


## Setup and Results

- Break messages by setting bits to random values beginning at a random position (30 messages total)
- Worst case for the algorithm because some data remains valid in broken messages
- Majority of protocols are labeled with more than 80% accuracy when 20% of messages are broken



# Performance measurement with real-world smart home protocols



## Measurement Setup

- Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz
- 16 GB RAM
- Message length between 8 and 61 bytes
- For every number of messages perform 100 performance measurements and take the mean performance

# Conclusion and Future Work

## Conclusion

- Framework for automatic reverse engineering of proprietary wireless protocols
- Dedicated engines to find Preamble, Synchronization, Length, Sequence Number, Address and Checksum fields
- **Bootstrap** unknown protocols but also able to consider **prior knowledge**
- Verified with simulated and real-world protocols

## Future Work

- Suggestion of attacks based on the found fields
- Detection of cryptography in message payload
- Ultimate goal: automated security score based on found cryptography and protocol complexity for initial security assessment right from captured messages



 <https://github.com/jopohl/urh>   

## Contact

- E-Mail: [Johannes.Pohl90@gmail.com](mailto:Johannes.Pohl90@gmail.com)  
E-Mail: [Andreas.Noack@hochschule-stralsund.de](mailto:Andreas.Noack@hochschule-stralsund.de)
- Slack: <https://bit.ly/2LGpsra>
- GitHub: <https://github.com/jopohl>