LoRa Reverse Engineering and AES EM Side-Channel Attacks using SDR

Pieter Robyns

UHASSELT EDM
fwo Research Foundation Flanders
Opening new horizons

About me

- PhD student at Hasselt University since 2014
  - Since 2016 on FWO SBO research grant
- Researching wireless security
  - Protocol security, location tracking, fingerprinting
  - Machine learning and side channel analysis
  - Wi-Fi, GSM, LoRa, proprietary protocols
- Website: https://robyns.me

  Email: pieter.robyns@uhasselt.be

Motivation for researching LoRa

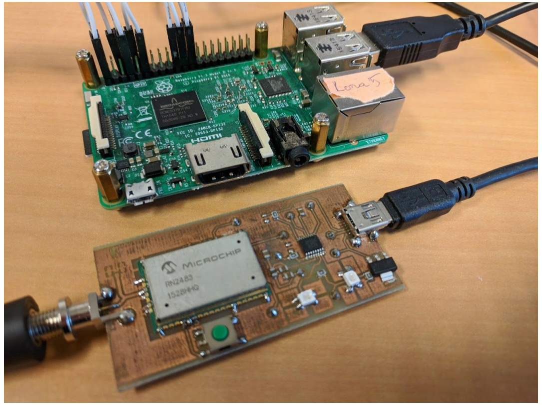
- Project started in April 2016 → LoRa was relatively new
  - Introduced to LoRa by co-advisor
- A lot of opportunities to learn new things
  - No working software-based decoders available, only simulations
    - → Building a GNU Radio OOT module from scratch
  - Limited description of the PHY layer: patents and blog posts
    - → Reverse engineering low-level aspects of a protocol
  - Fingerprinting and tracking devices over long ranges
    - → Machine learning applied to fingerprinting instead of expert feature selection
  - Side-channel attacks
    - → IoT devices are inherently more vulnerable

Part 1

Unlocking the LoRa PHY

Unlocking the LoRa PHY

- Hardware LoRa radios can only be interfaced with over a serial connection



Microchip RN2483 + custom board made by my co-advisor

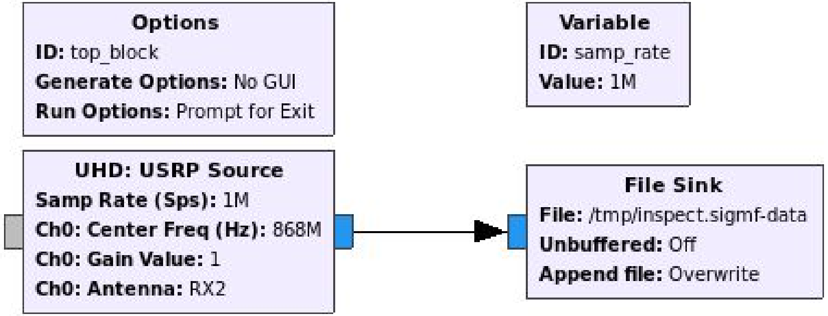
- We need access to the raw PHY signal for fingerprinting

  ⇒ Where do we start?

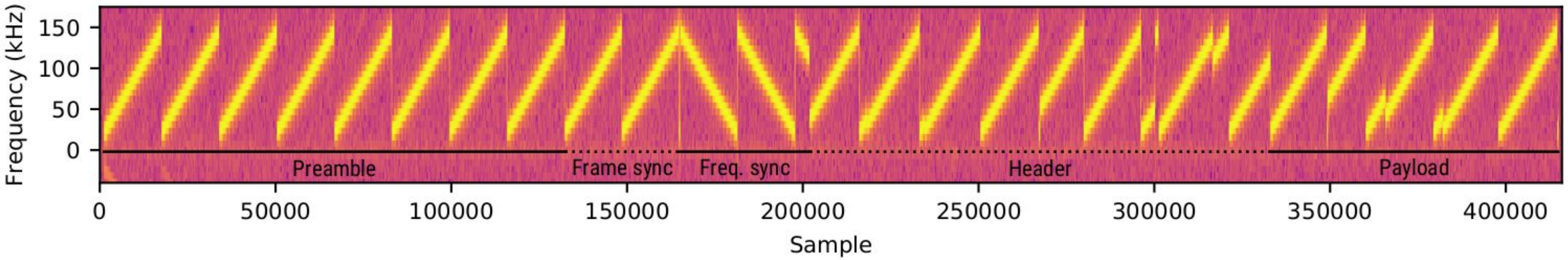
Unlocking the LoRa PHY

- GNU Radio to the rescue! Let's inspect a transmission using a simple flowgraph

GNURadio
THE FREE & OPEN SOFTWARE RADIO ECOSYSTEM



Unlocking the LoRa PHY



- Frame structure can be easily derived from patent
  - See [Patent EP2763321 A1]
  - Also contains information on:
    - → Modulation
    - → Interleaving
  - Some other info located in datasheets:
    - → Whitening and coding
- Let's build a receiver!

Patents

Find prior art | Discuss this application | Vie

Low power long range transmitter

EP 2763321 A1

ABSTRACT

A transmitter device arranged to encode a set of digital input data into a succession of modulated chirps, whereby said digital input data are encoded according to a Gray code into codewords (320, 321, 322) having a plurality of bits, and having an interleaver that distributes the bits (C_{00},... C_{nn}) of each codeword into a series of digital modulation values (S_0, ... S_7), at different bit positions, and to synthesize a series of modulated chirps whose cyclical shifts are determined by the modulation values. A special frame structure is defined in order to ensure high robustness, and variable bit-rate flexibility.

Publication number	EP2763321 A1
Publication type	Application
Application number	EP20130154071
Publication date	Aug 6, 2014
Filing date	Feb 5, 2013
Priority date	Feb 5, 2013
Also published as	CN103973626A, US9252834, US20140219329
Inventors	Olivier Bernard André Seller, Nicolas Sornin
Applicant	Semtech Corporation
Export Citation	BiBTeX, EndNote, RefMan

Patent Citations (7), Referenced by (8), Classifications (9), Legal Events (5)

External Links: Espacenet, EP Register

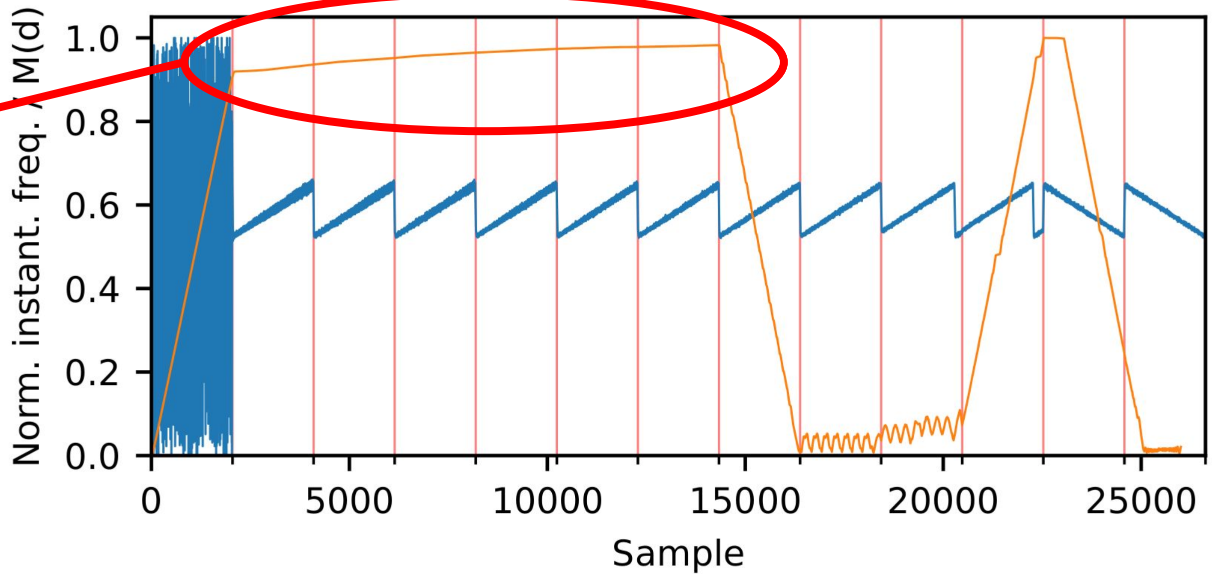
IMAGES (5)

How do we detect the signal?

- Detecting: pretty standard problem in signal processing
- Multiple solutions possible; I chose Schmidl-Cox algorithm
  - Autocorrelation exploiting the repeating property of the preamble

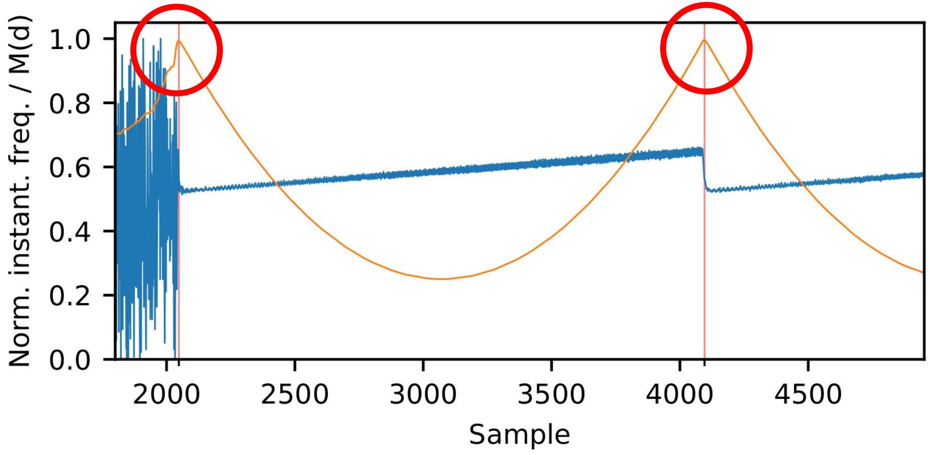
Preamble is here, but where does it start?

Thresholding = bad!



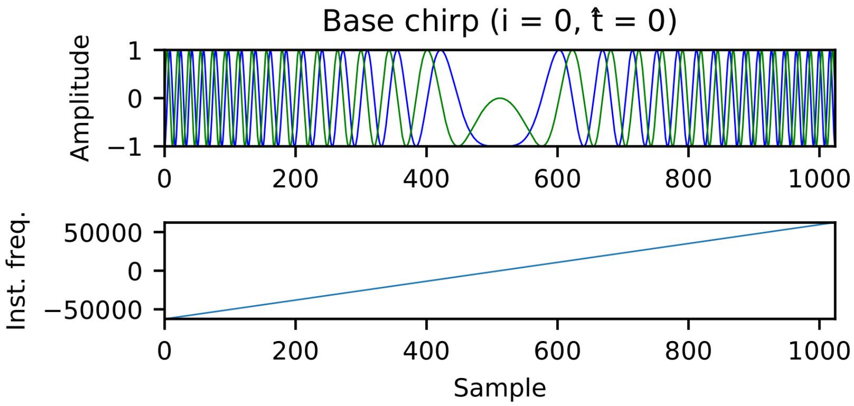
How do we synchronize to the signal?

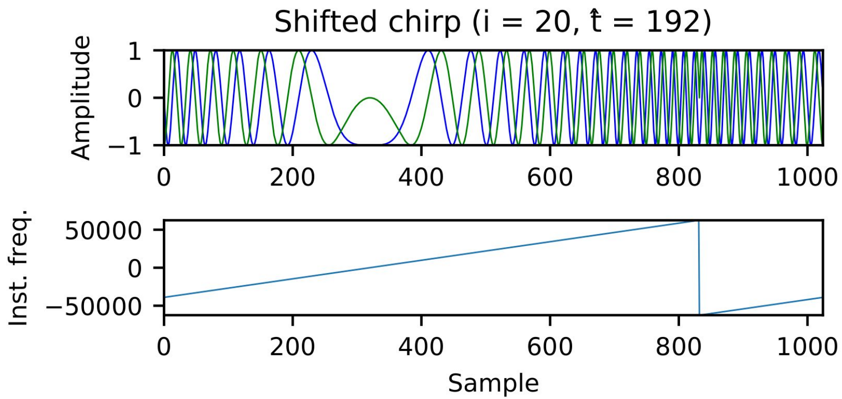
- Again multiple possibilities:
  - Demodulate preamble symbol → supposed to be 0
    - → Offset from 0 indicates a time shift (basic principle of LoRa modulation as we will see)
    - → However: ambiguity because a frequency shift also causes an offset from 0!
  - Cross-correlate instantaneous frequency with locally generated preamble
    - → Higher sensitivity to noise, but no ambiguity



How do we demodulate a single symbol?

- Modulation of LoRa is based on Chirp Spread Spectrum
- Chirp = signal that linearly increases in frequency
- To modulate a value “i” onto chirp: cyclically time shift it!



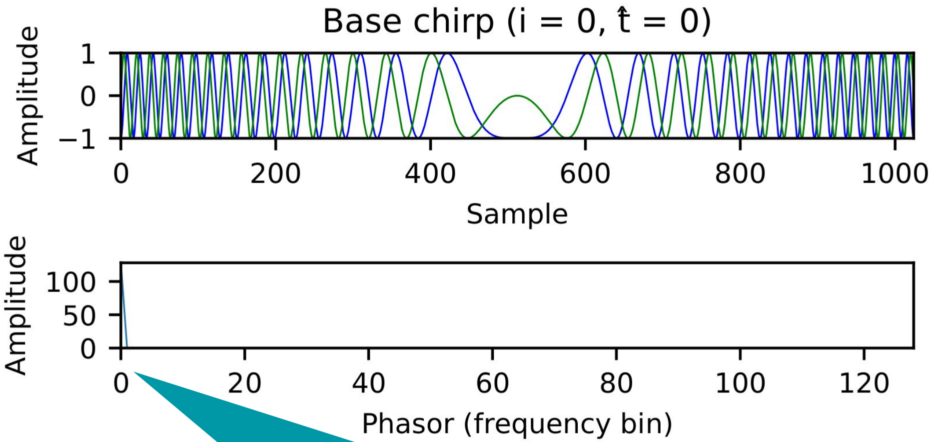


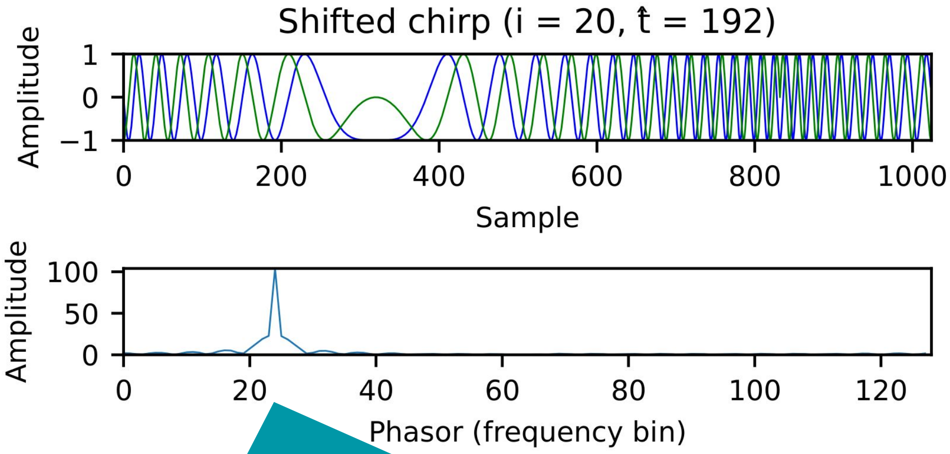
Value: 0 (unmodulated)

Value: 20 (spoiler: indexing ;))

How do we demodulate a single symbol?

- Cyclic shift results in a peak in the frequency domain when multiplied by a conjugate base chirp (+ resampling at chirp rate) ⇒ details not important for now
- Index is “gray” decoded. Encode to demodulate!





gray(0) == 0 == i

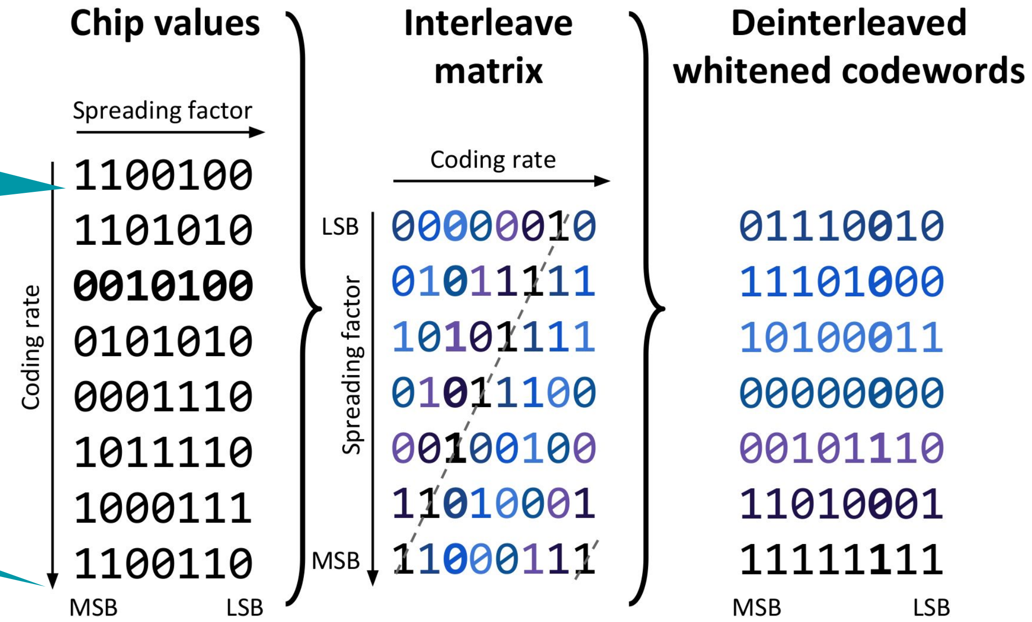
gray(24) == 20 == i

Demodulation continued: interleaving

- Interleaving is trivial: algorithm provided in patent
  - Spreading factor determines bits per symbol value (here: 7)
  - Coding rate determines symbol values per interleave matrix (here: 8)

Binary value of FFT peak index

Only pitfall: the bit order → interleave direction



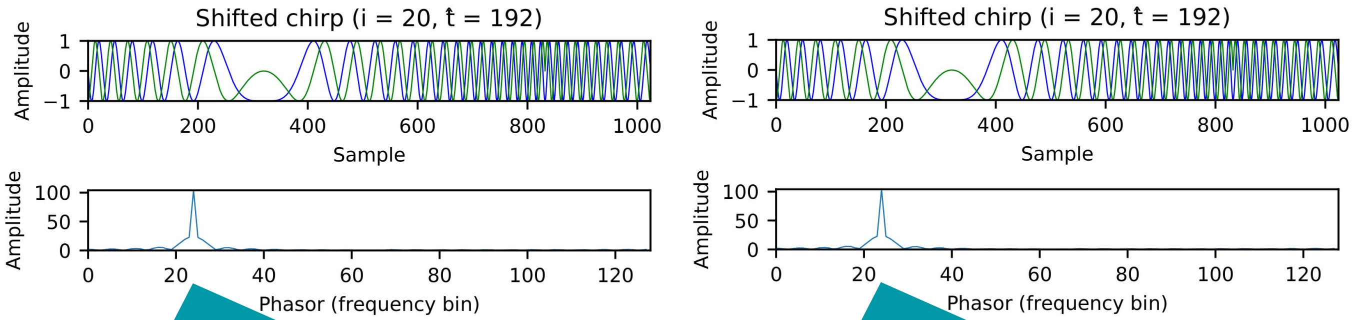
Unlocking the LoRa PHY: unknown aspects

- What's left to be done?
  - ~~How do we detect the signal?~~
  - ~~How do we synchronize to the signal?~~
  - ~~How does the modulation and interleaving work?~~
  - What is the relation between a raw symbol and its integer value?
  - In which stage of the decoding is whitening performed and how?

- Not discussed in this presentation:
  - Header structure
  - Clock drift correction
  - Swapping of nibbles + CRCs
  - See my paper for more info!

Relation between symbol and integer value?

- Patent states “gray coding” is used
  - Total of 4 possible mappings to symbol values:



gray(24) or degray(24)?

gray(103) or degray(103)?

} Inverted x-axis

- To check correctness: implement decoder up to interleaving and look for patterns
  - Header is unwhitened ⇒ use header to check previous stages

c. Relation between symbol and integer value?

- Example: sending packets with increasing payload sizes (SF 7)

Bin data → ; Hex len →

Right to left (FFT bin) 127 → 0

Hex len	Gray encoding					Gray decoding				
01:	10001100	00001000	10000011	01000010	00101000	00001100	01001010	10000011	01000010	00000000
02:	10001100	00001000	01100100	01000010	00101001	10001000	01001010	01000101	01000010	00100001
03:	10001100	000010[illegible]	00000111	01000010	00100000	10001000	01001010	00000111	01100010	00001000
10:	10001100	10000010	0[illegible]100011	01000001	00100001	10001100	11000010	01000010	01000001	00100001
11:	10001100	10000010	00[illegible]0000	01000001	00101000	10001100	11000010	00100000	01000001	00101000
12:	10001100	10000010	111001[illegible]	01000001	00101001	00001000	11[illegible]0010	11000110	01000001	00101001
20:	10001100	10000010	10100110	00000000	00100001	00001000	1000[illegible]0	10000111	00000000	00100001
21:	10001100	10000010	11000101	00000000	00101000	00001000	10000010	11100101	00000000	00101000
22:	10001100	10000010	00100010	00000000	00101001	10001100	10000010	[illegible]0011	00000000	00101001

Left to right (FFT bin) 0 → 127

Hex len	Gray encoding					Gray decoding				
01:	00000000	10001011	10011100	00000000	10001011	10011000	10001011	100110[illegible]	00010000	00011110
02:	00000000	01001110	10011100	00000000	00101101	00011100	01001110	01111100	00010000	11100000
03:	00000000	11000110	10011100	00000000	01001110	00011100	11000101	00011110	000[illegible]0000	10001010
10:	10001011	00000000	10011100	10001011	11111111	10010011	10001000	01111011	10011000	11110111
11:	10001011	10001011	10011100	10001011	10011100	10010011	00000011	00011001	10011000	10011101
12:	10001011	01001110	10011100	10001011	01100011	00010111	11000110	11111111	10011000	01100011
20:	01001110	00000000	10011100	10001011	00111010	11010010	11001000	10111110	11011001	00110010
21:	01001110	10001011	10011100	10001011	01011001	11010010	01000011	11011100	11011001	01011000
22:	01001110	01001110	10011100	10001011	10100110	01010110	10000110	00111010	11011001	10100110

Whitened?

Inconsistent

How do we decode the obtained codewords?

```
01: 00000000 10001011 10011100 00000000 10001011
02: 00000000 01001110 10011100 00000000 00101101
03: 00000000 11000110 10011100 00000000 01001110
10: 10001011 00000000 10011100 10001011 11111111
11: 10001011 10001011 10011100 10001011 10011100
12: 10001011 01001110 10011100 10001011 01100011
20: 01001110 00000000 10011100 10001011 00111010
21: 01001110 10001011 10011100 10001011 01011001
22: 01001110 01001110 10011100 10001011 10100110
```

- Coding: 4/5 - 4/8 as options imply Hamming coding
- Payload whitening: XOR with random LFSR
  - Mentioned but specified algorithm doesn't work in practice :(.
  - In what stage is the data whitened?
  - Only payload is whitened → very useful!

How do we decode the obtained codewords?

- Fastest solution: brute force
- Whitening: send payload with all zeros

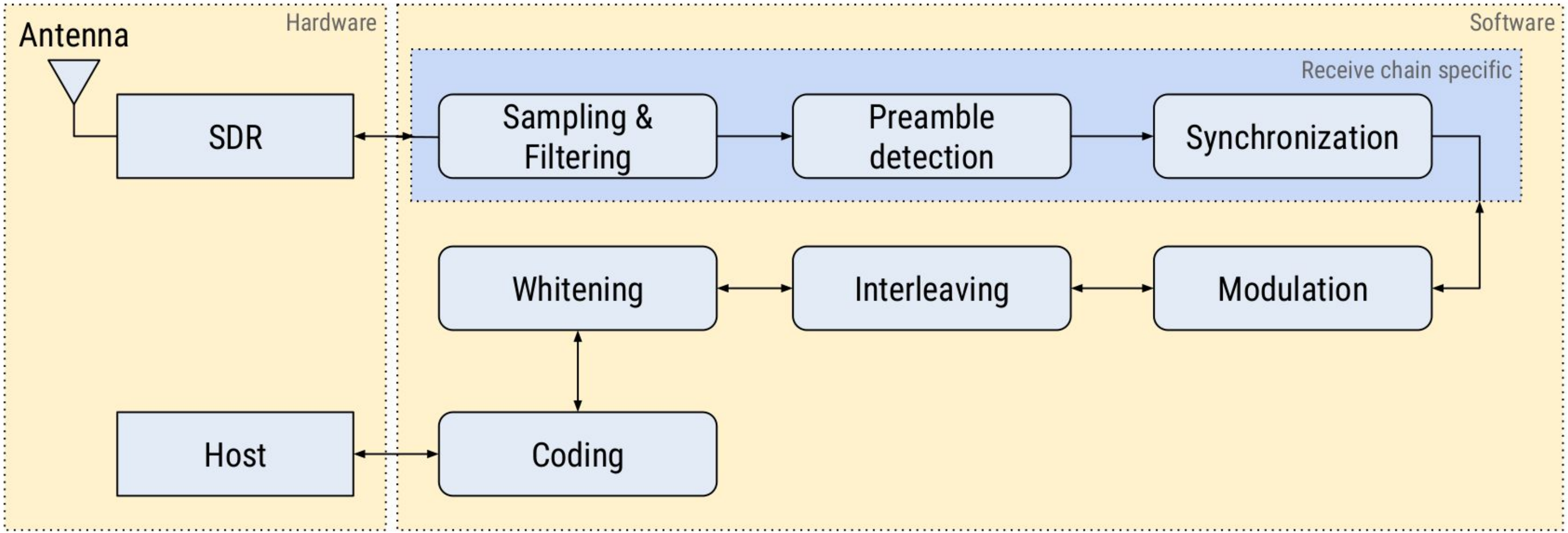
`00100010 XOR 00000000`

  - Hamming code of 0000 is 00000000, which is convenient
  - Ideas for determining LFSR algebraically welcome!
- Hamming codes
  - Try all possible bit permutations for a header byte. Choose the one without decode errors
  - Verify with multiple (all possible) header byte values
  - $\frac{8!}{(8-4)!} = 1680$

10001011

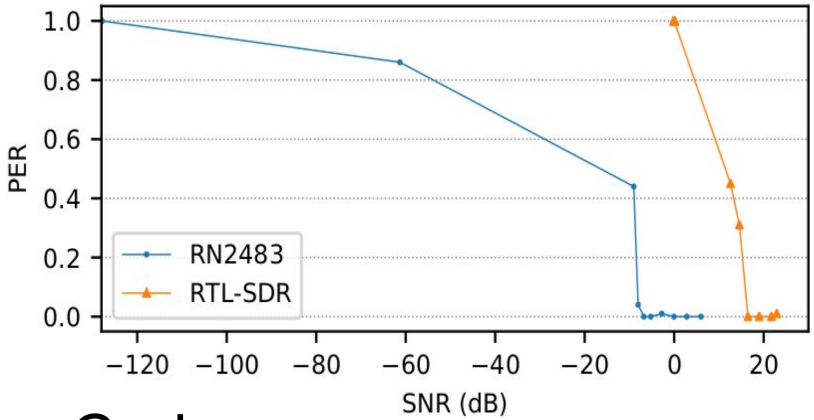
Results

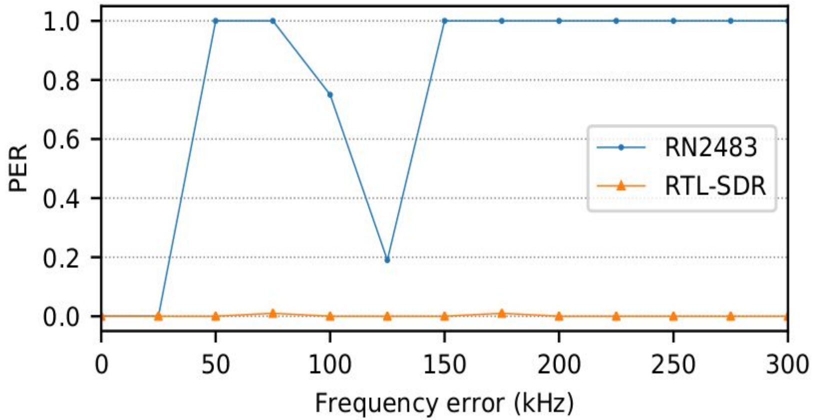
- Overview of all components linked together:



Results

- Comparison with real hardware:





- Code: https://github.com/rpp0/gr-lora
  - Special thanks to my student William for implementing some optimizations
- Other decoders / related work
  - LoRa-SDR: https://github.com/myriadrf/LoRa-SDR
  - BastilleResearch's gr-lora: https://github.com/BastilleResearch/gr-lora

Application

Fingerprinting LoRa devices using neural networks

Why fingerprint devices?

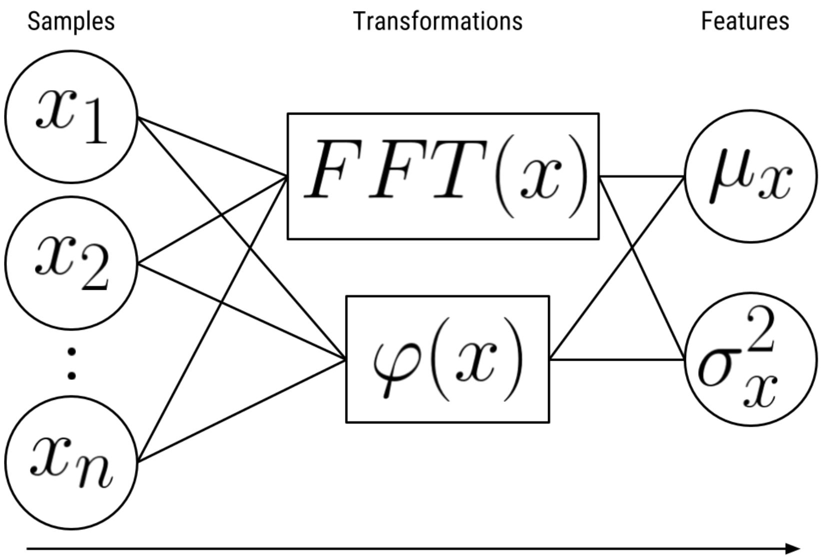
- Defensive
  - Extra layer of defense in critical infrastructure → detect unknown devices
  - Possibly counter relay attacks
  - Measure degree of privacy provided by device
- Offensive
  - Linking anonymous transmissions (e.g. defeat MAC randomization)
  - Tracking the location of sensors (e.g. to take them down)
  - Mimic radio signature of a device to defeat IDSs
- Caveat: cat-and-mouse game between attacker and defender!

PHY-layer fingerprinting theory

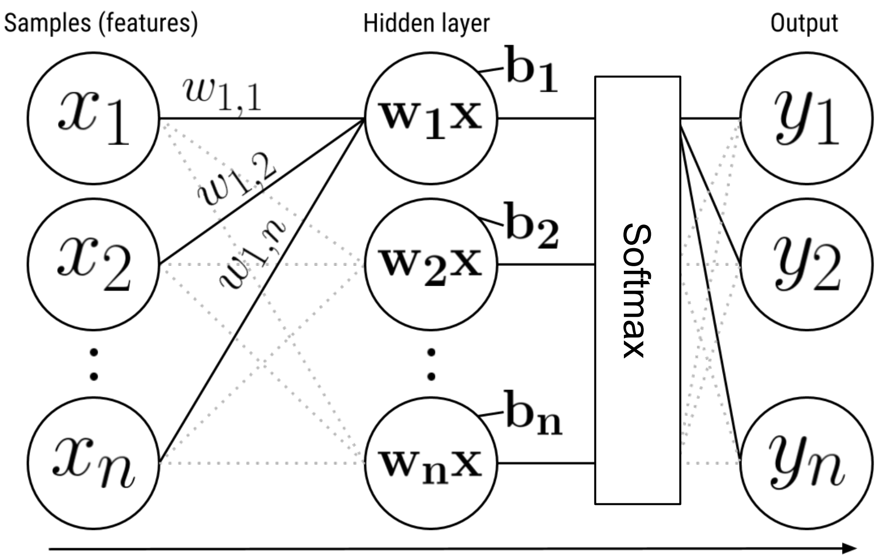
- Hypothesis: no two radios can be perfectly identical
  - Manufacturing differences in circuits, crystal oscillators, components, …
    - → Manifest as per-device transmission errors (e.g. frequency offset)
    - → Error tolerance typically defined within data sheets (e.g. ± 12 KHz)
    - → *Larger tolerance implies more entropy*

- Challenge: distinguish noise from errors caused by the radio hardware
  - Traditional approach: use statistical measures on “expert features”
    - → Carrier Frequency Offset, Sampling Frequency Offset, Preamble Transient,...
  - My approach: apply machine learning to the **raw radio signal**
    - → Similar techniques applied in face recognition, image classification, etc.

Simplified comparison

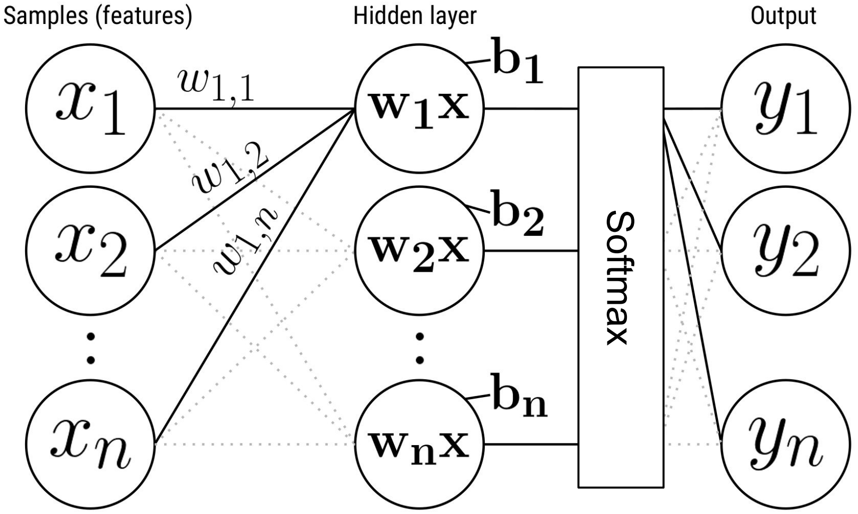


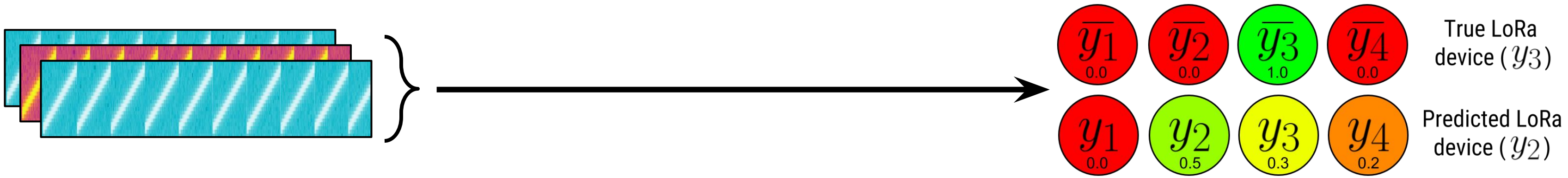
- “Human” filtering at feature level
- Resulting features can be learned with ML or statistical distance measures



- Unimportant features are filtered through weight values
- Consider raw samples as features

Training the neural network



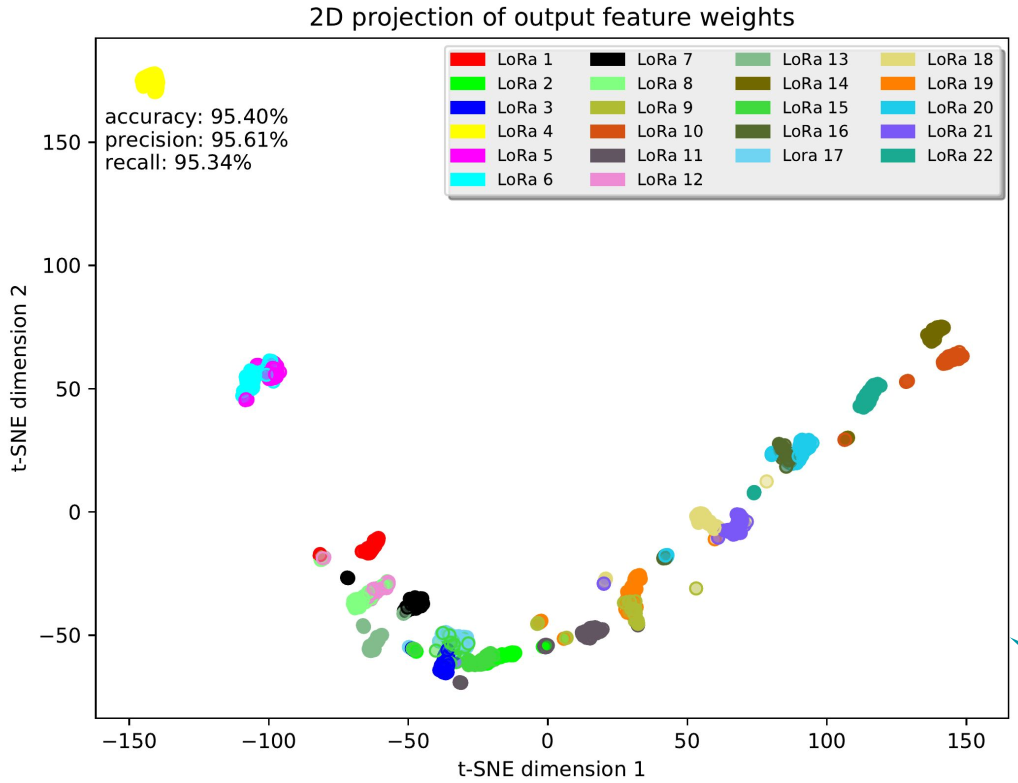


1. Label transmission with LoRa device.
2. Feed data through neurons and check resulting outputs.
3. Evaluate the result in terms of a “loss” function, and update the neuron weights accordingly. Repeat step 2.

LoRa fingerprinting experiment

- Experiment: can we uniquely identify 22 LoRa devices?
  - 3 different vendors
    - → 1 SX1272
    - → 2 RF96
    - → 19 RN2483
  - Model: simple MLP from previous slides
  - Training data: ~100,000 symbols
  - Test data: ~1,000 symbols
- 95% accuracy
  - However: tradeoff between sensitivity to noise and being able to detect fine-grained differences between devices → noise is a problem

Results



Outline: predicted device

Fill: true device

Correct

Incorrect

Each point is one symbol! (>16 symbols per frame)

Part 2

EM side-channel attacks on AES

What is a side channel attack?

- Implementation leaks information through “side channel”
- Attacker gains *advantage* based on this information
- Numerous types of side channels:
  - Timing
  - Acoustic
  - Power consumption
  - Temperature
  - Cache
  - **Electromagnetic**

Correlated?

Motivation

- EM side-channel attacks (on AES) are interesting
  - Used by LoRa, Wi-Fi, TLS, IPsec, apps, ...
- Attack techniques have been around for quite some time, but expensive equipment often required
- Can we do these TEMPEST-style attacks with cheap SDRs?
  - We will discuss a simple Correlation Power Attack (more complicated attacks exist)

Examples of EM side channel attacks

1. (Attacker sends data to encrypt)

2. Victim inadvertently leaks info through electromagnetic radiation

3. Attacker captures info and predicts key based on a **model**

EM models

- Behavior of system can be approximated with a model
- Accuracy of model is crucial for successful attack
- Some observations:
  - Amplitude of electromagnetic radiation is proportional to power
  - Power is required to change state of a circuit

  ⇒ State changes cause variations in the amplitude of EM radiation, proportional to their power consumption
- What happens if we would AM demodulate AES encryptions?

Case: AES on ATmega 328p

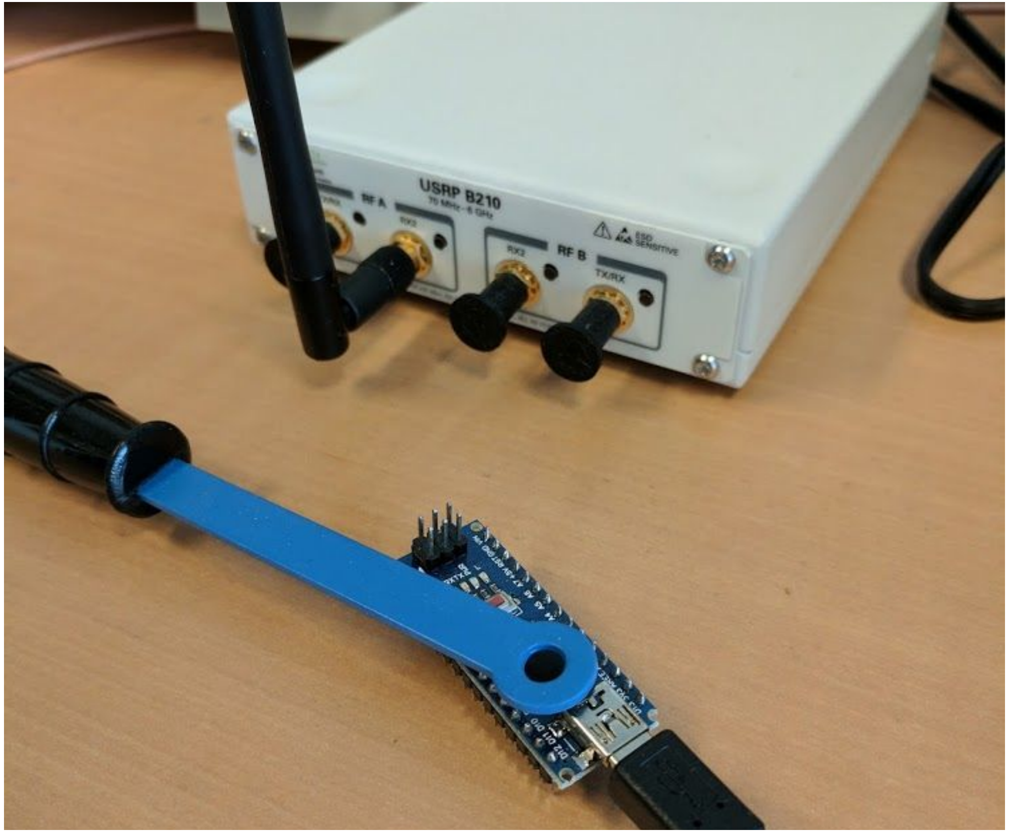
- Case study: AM demodulated AES encryptions performed by an ATmega 328p (Riscure competition)
  - Key size and key unknown; black box

- What we can learn from related works:
  - Lower frequencies must be favored[1]
  - Harmonics of CPU clock frequency contain useful information[2]

- Equipment: USRP B210 + amplifier + EM probe
  - ~18,000 traces. More = better

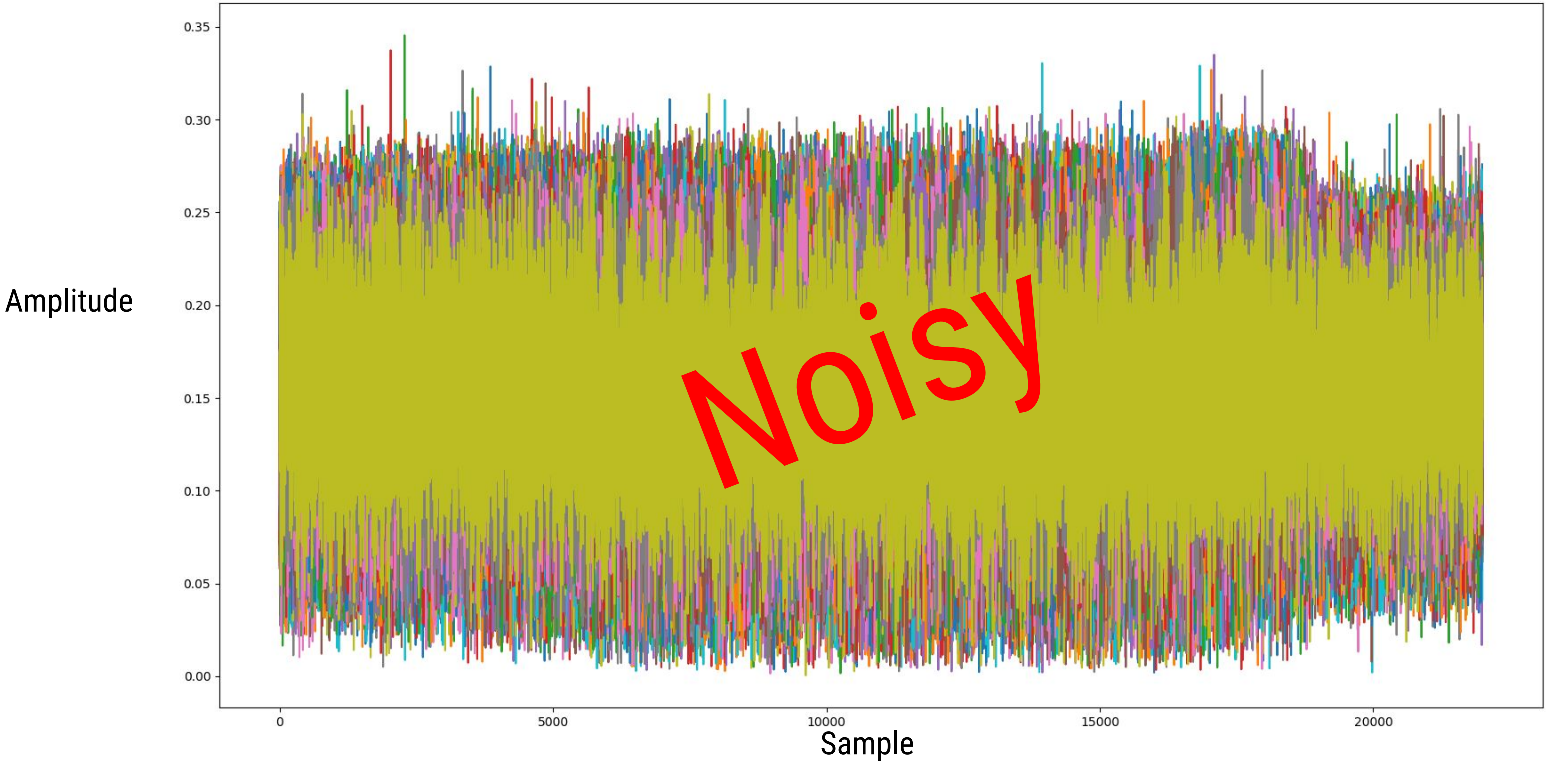
[1] A Frequency Leakage Model and its application to CPA and DPA, Sébastien Tiran et al., IACR Cryptology ePrint Archive, 2013
[2] The EM Side–Channel(s):Attacks and Assessment Methodologies, Dakshi Agrawal et al., CHES 2002.

Case: AES on ATmega 328p



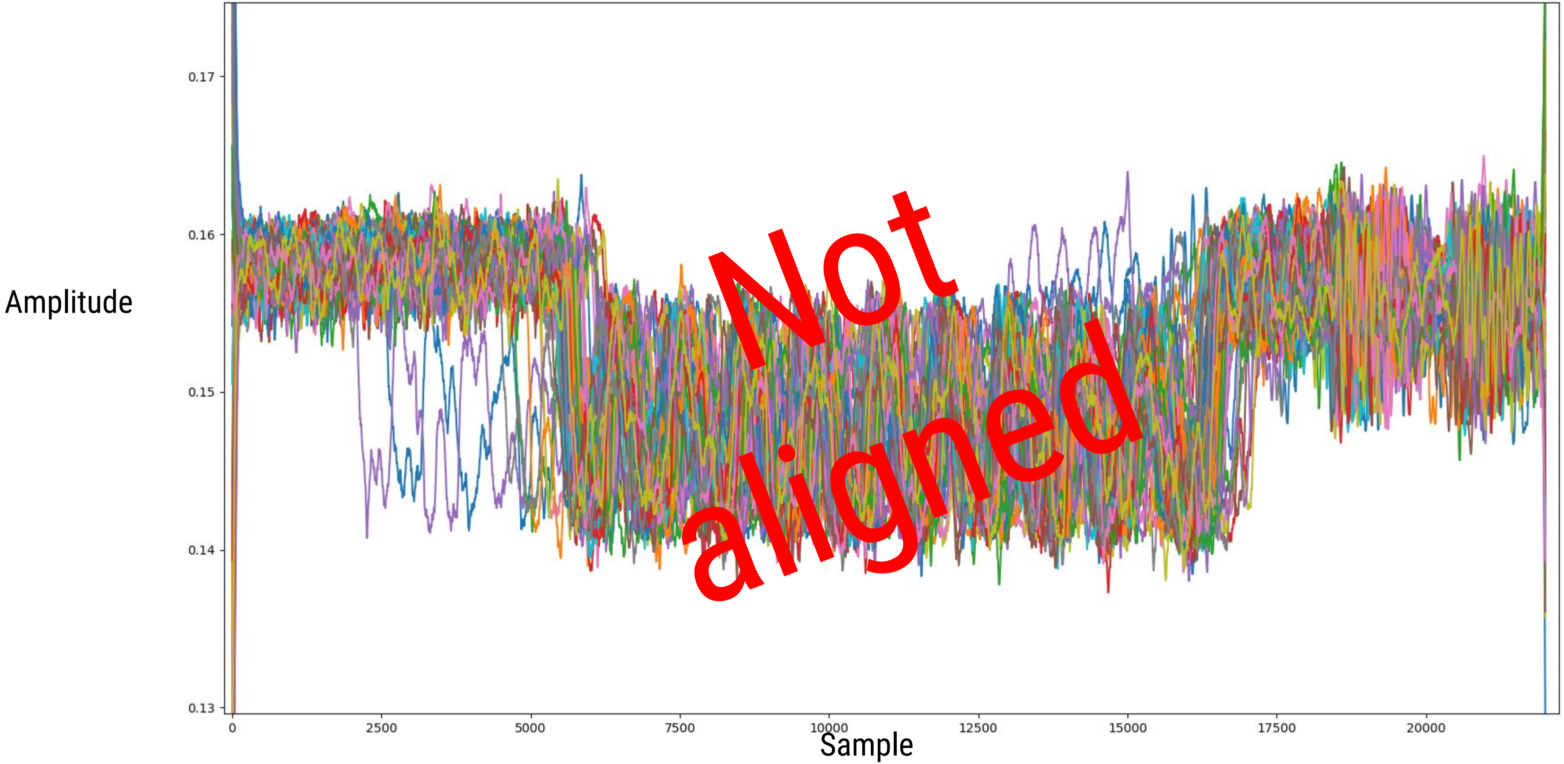
Case: AES on ATmega 328p

- AM demodulation of raw capture:



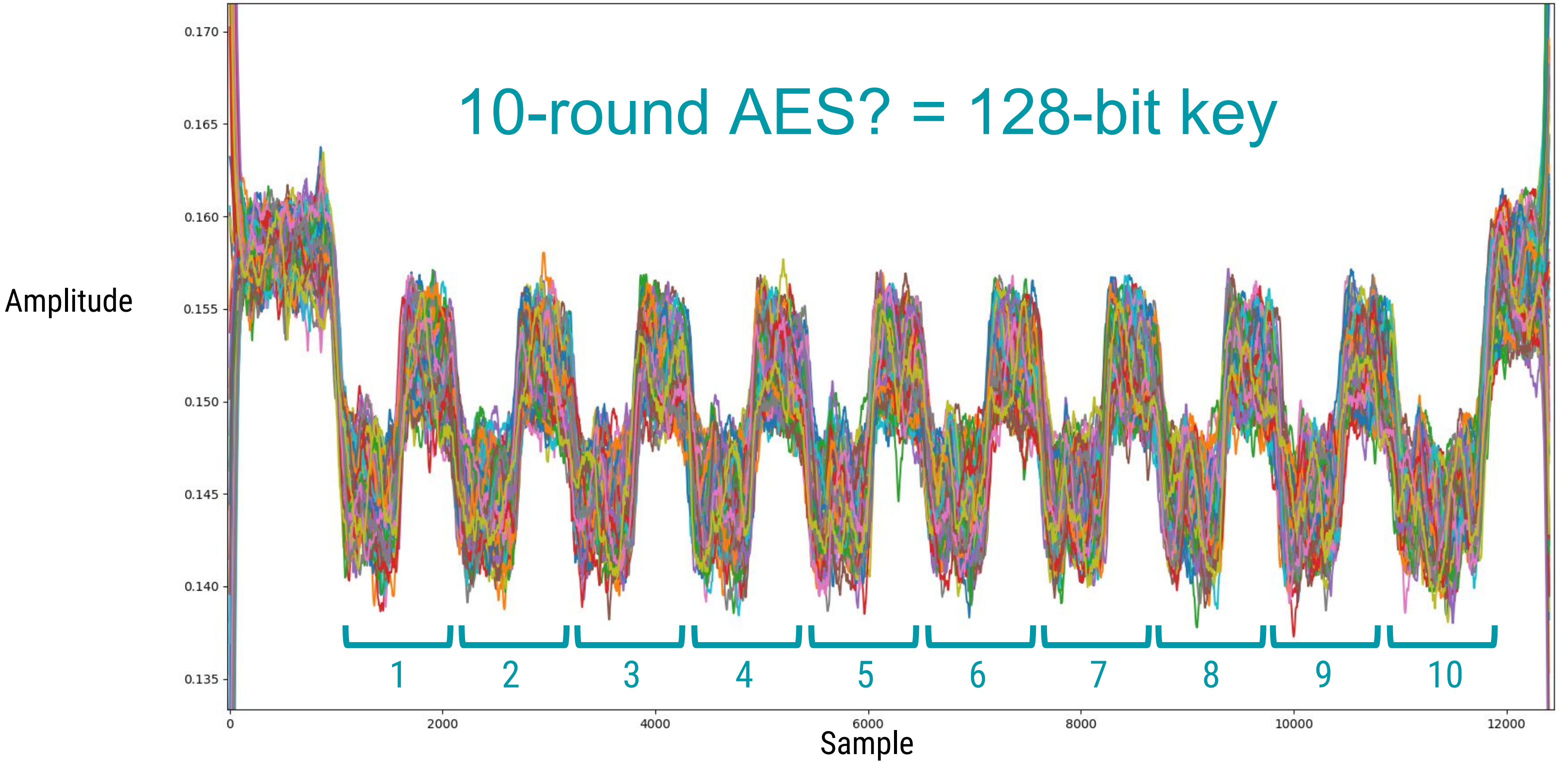
Case: AES on ATmega 328p

- After low pass filter



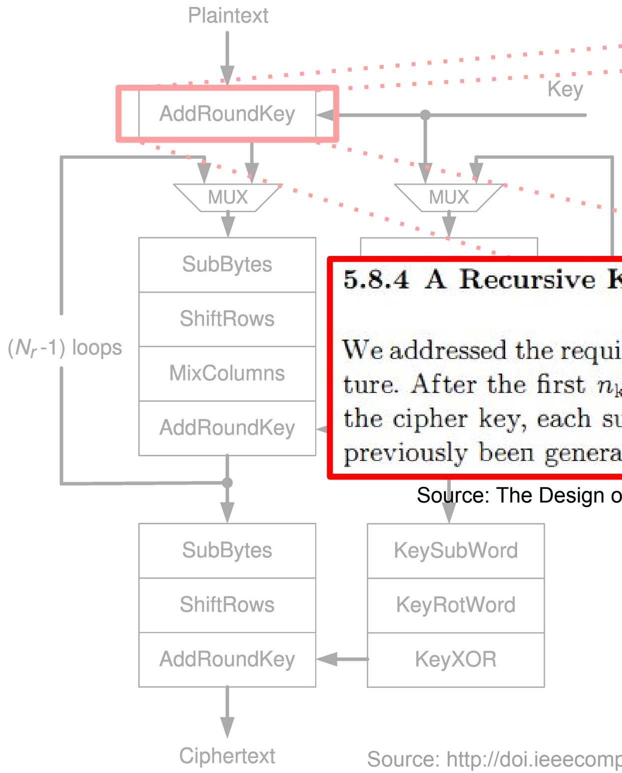
Case: AES on ATmega 328p

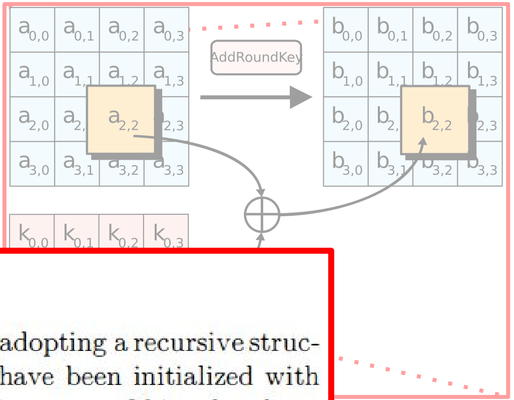
- After cross-correlation with reference signal



Extending our model to attack AES

- Where is the secret key in AES used?





5.8.4 A Recursive Key Expansion

We addressed the requirements discussed above by adopting a recursive structure. After the first n_k bits of the expanded key have been initialized with the cipher key, each subsequent bit is computed in terms of bits that have previously been generated.

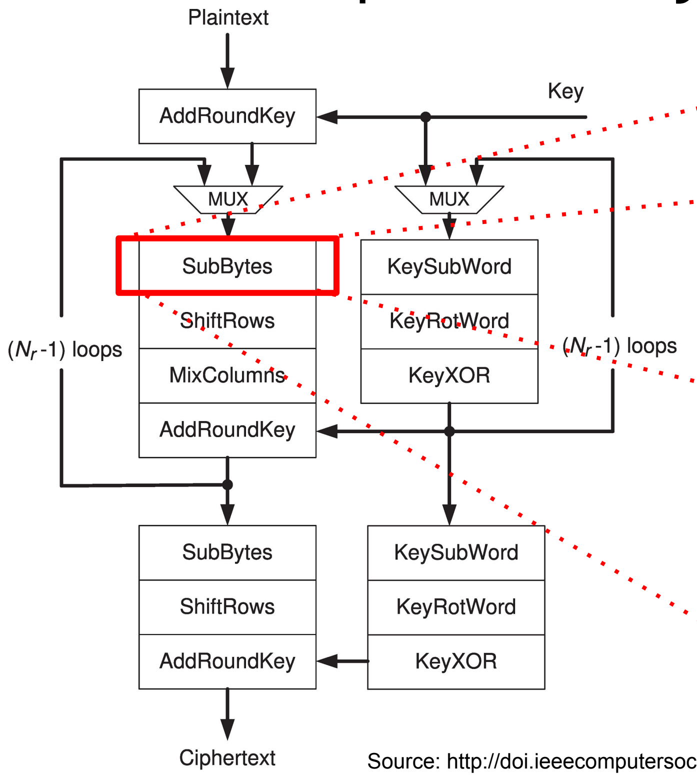
Source: The Design of Rijndael, Joan Daemen and Vincent Rijmen, Springer, 2002.

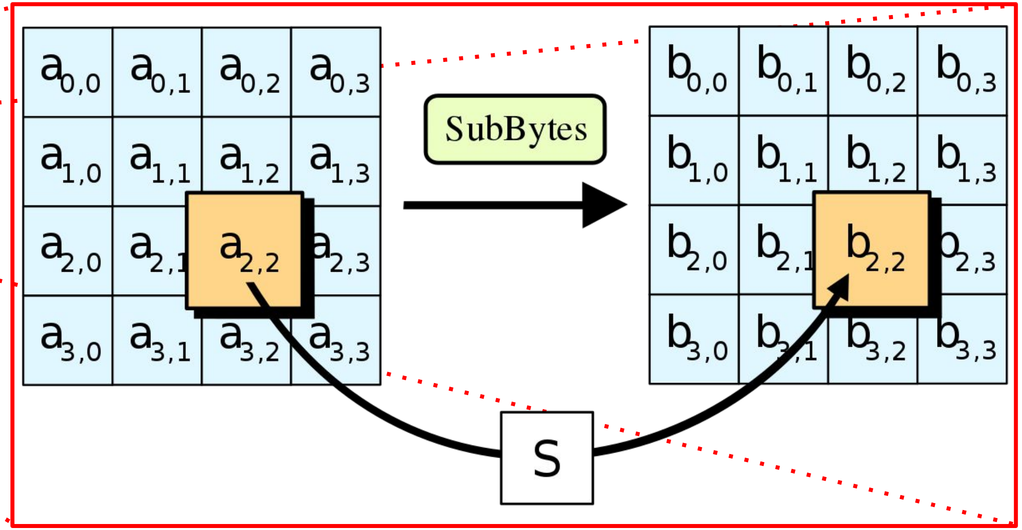
dRoundKey.svg/2000px-AES-AddRoundKey.svg.png

Source: http://doi.ieeecomputersociety.org/cms/Computer.org/dl/trans/tc/2013/03/figures/ttc20130305361.gif

Extending our model to attack AES

- Assume output of `SubBytes` is vulnerable for now





https://upload.wikimedia.org/wikipedia/commons/thumb/a/a4/AES-SubBytes.svg/1200px-AES-SubBytes.svg.png

Source: http://doi.ieeecomputersociety.org/cms/Computer.org/dl/trans/tc/2013/03/figures/ttc20130305361.gif

Extending our model to attack AES

- What happens inside the chip?
  - Initial state is unknown reference state R
  - After `AddRoundKey` and `SubBytes`, the state is $D = sbox[p_d \oplus k_d]$

- Current consumed ~ state changes on clock edge
  - Therefore, it's given by Hamming distance between R and D

R 00100110
D 10101000

Hamming Distance = 4

- Hamming weight also works in practice if R = 0

Case: AES on ATmega 328p

$$h_d = HW(sbox[p_d \oplus k_d])$$

$$h_d = HW(sbox[p_d \oplus \texttt{0x00}])$$

$$h_d = HW(sbox[p_d \oplus \texttt{0x01}])$$

$$\vdots$$

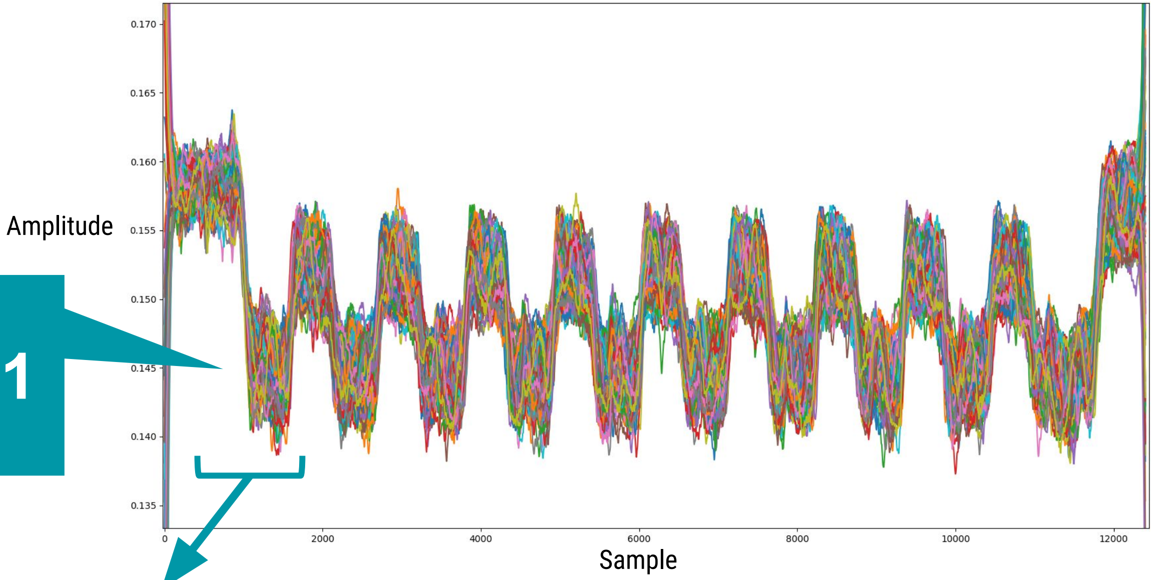
$$h_d = HW(sbox[p_d \oplus \texttt{0xff}])$$

Build models for each possible key byte

Chosen by attacker and varied each trace

Case: AES on ATmega 328p

- Measure reality



Round 1

m_d

One-point amplitude measurement for byte d of key

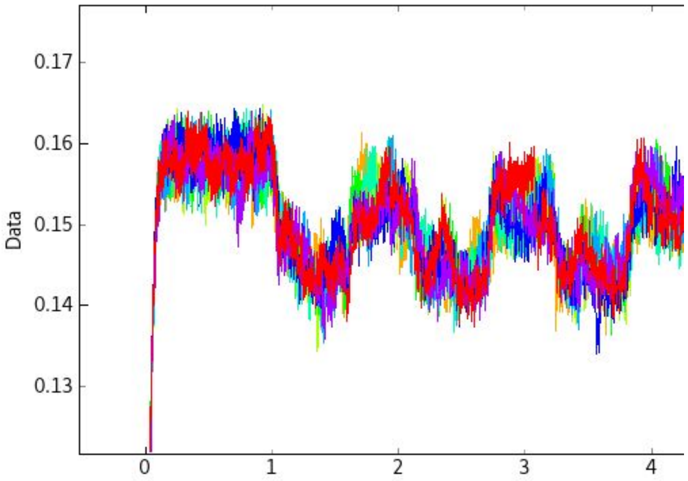
Case: AES on ATmega 328p

- Final step: correlate reality with model for each trace
- Highest correlation hypothesis is most likely key byte
- Absolute value of Pearson correlation
  - Note: only linear correlation!
- "Correlation Power Attack"

$$|\rho_{m_d,h_d}| = \left|\frac{cov(m_d, h_d)}{\sigma_{m_d}\sigma_{h_d}}\right|$$

Case: AES on ATmega 328p

- Using ChipWhisperer to perform CPA attack:



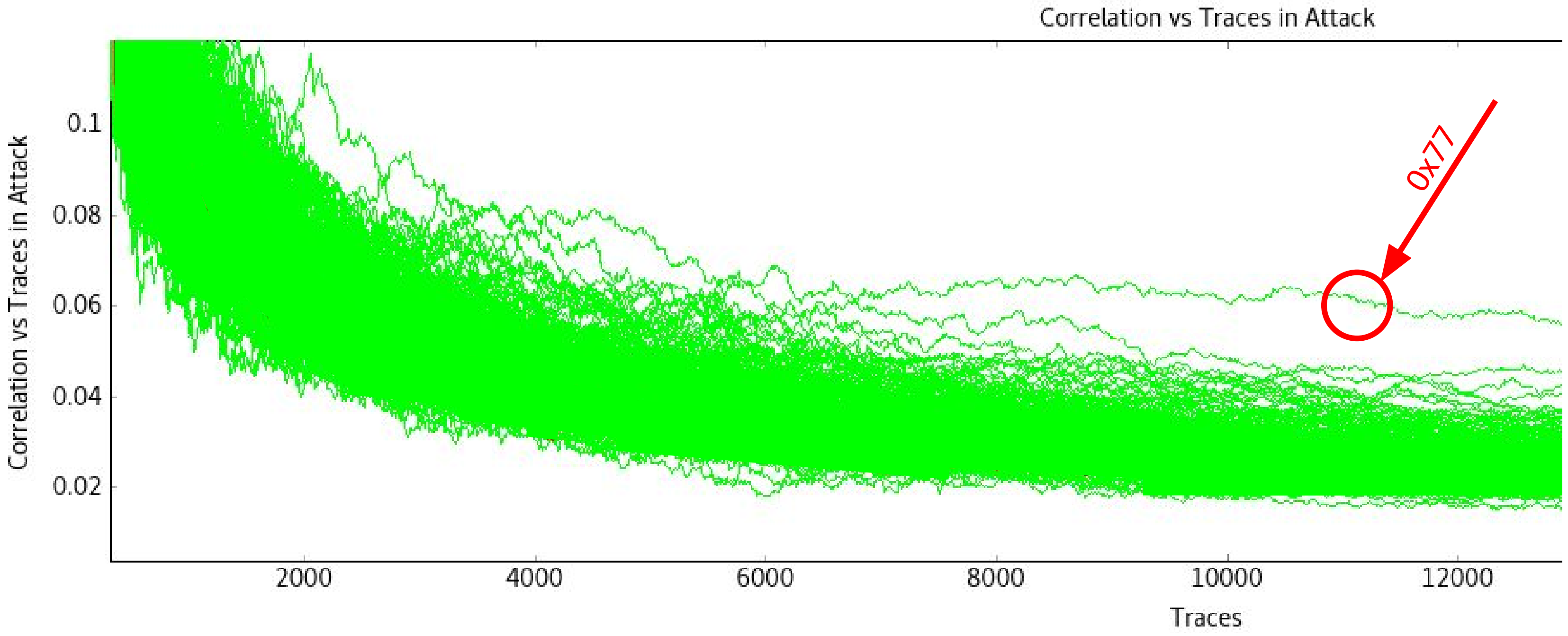
	0	1	2	3	4	5	6	7
PGE	185	85	29	149	193	9	242	2
0	0E 0.0430	EB 0.0498	A7 0.0499	43 0.0563	00 0.0497	9D 0.0508	67 0.0599	D 0.04
1	C9 0.0315	15 0.0324	00 0.0449	A4 0.0372	9C 0.0322	03 0.0341	3B 0.0343	D 0.03
2	45 0.0306	3D 0.0303	69 0.0331	17 0.0325	37 0.0315	68 0.0333	40 0.0326	4 0.03
3	A3 0.0304	72 0.0292	28 0.0312	E9 0.0298	0E 0.0308	A8 0.0331	65 0.0325	D 0.03
4	19 0.0289	4B 0.0290	F3 0.0309	26 0.0298	CF 0.0293	A2 0.0329	DA 0.0323	6 0.03
5	BA 0.0279	DE 0.0288	63 0.0303	68 0.0287	3A 0.0290	92 0.0327	D7 0.0315	2 0.03
6	14 0.0278	AC 0.0287	1D 0.0301	53 0.0286	7D 0.0274	3C 0.0321	AD 0.0310	2 0.02
7	77 0.0276	87 0.0283	37 0.0299	9B 0.0283	27 0.0273	DC 0.0314	69 0.0310	4 0.02
8	BB 0.0274	89 0.0280	35 0.0299	28 0.0273	53 0.0270	6A 0.0298	47 0.0308	2 0.02

Extra: SDR plugin for NewAE ChipWhisperer

Available at: http://research.edm.uhasselt.be/probyns/cw_hacky_usrp_plugin.zip

Case: AES on ATmega 328p

- Using ChipWhisperer to perform CPA attack:



Case: AES on ATmega 328p

- Using EMMA (soon-to-be open source)
  - Uses multiple cores per node and can run on multiple machines

```
Num entries: 19825
Subkey 15: elapsed: 56
Num entries: 19825

    0           1           2           3           4           5           6           7           8           9           10          11          12          13          14          15
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 0.04 (0e) | 0.05 (eb) | 0.05 (a7) | 0.05 (43) | 0.04 (00) | 0.04 (9d) | 0.06 (67) | 0.04 (d2) | 0.05 (e5) | 0.05 (63) | 0.05 (cf) | 0.04 (4c) | 0.04 (5c) | 0.05 (b0) | 0.05 (77) | 0.05 (cb) |
 0.02 (45) | 0.02 (15) | 0.04 (00) | 0.03 (a4) | 0.03 (37) | 0.03 (03) | 0.03 (da) | 0.03 (03) | 0.03 (32) | 0.03 (7c) | 0.03 (53) | 0.04 (30) | 0.03 (56) | 0.03 (94) | 0.03 (cc) | 0.04 (f4) |
 0.02 (c9) | 0.02 (69) | 0.03 (69) | 0.03 (26) | 0.03 (ca) | 0.03 (a2) | 0.03 (3d) | 0.03 (27) | 0.02 (7d) | 0.03 (71) | 0.03 (4b) | 0.02 (d4) | 0.02 (62) | 0.03 (d7) | 0.03 (79) | 0.03 (c1) |
 0.02 (77) | 0.02 (3d) | 0.03 (98) | 0.03 (9b) | 0.03 (0e) | 0.03 (3b) | 0.03 (5f) | 0.03 (d7) | 0.02 (42) | 0.03 (76) | 0.02 (3d) | 0.02 (3d) | 0.02 (1c) | 0.02 (16) | 0.03 (c3) | 0.03 (d4) |
 0.02 (a0) | 0.02 (2b) | 0.03 (35) | 0.03 (53) | 0.03 (31) | 0.03 (57) | 0.03 (b9) | 0.02 (d3) | 0.02 (94) | 0.03 (dd) | 0.02 (5e) | 0.02 (a4) | 0.02 (06) | 0.02 (98) | 0.03 (eb) | 0.03 (07) |
 0.02 (11) | 0.02 (01) | 0.03 (0b) | 0.02 (68) | 0.02 (80) | 0.02 (3c) | 0.02 (47) | 0.02 (4e) | 0.02 (99) | 0.03 (65) | 0.02 (7c) | 0.02 (0c) | 0.02 (7f) | 0.02 (ff) | 0.03 (4a) | 0.03 (29) |
 0.02 (bb) | 0.02 (a5) | 0.03 (20) | 0.02 (a0) | 0.02 (8c) | 0.02 (6a) | 0.02 (6d) | 0.02 (2c) | 0.02 (10) | 0.03 (54) | 0.02 (b9) | 0.02 (df) | 0.02 (3f) | 0.02 (51) | 0.03 (f4) | 0.03 (8b) |
 0.02 (2d) | 0.02 (08) | 0.03 (1d) | 0.02 (38) | 0.02 (c5) | 0.02 (50) | 0.02 (d7) | 0.02 (f7) | 0.02 (da) | 0.02 (24) | 0.02 (f9) | 0.02 (13) | 0.02 (9c) | 0.02 (9b) | 0.03 (de) | 0.03 (0f) |
 0.02 (e2) | 0.02 (6f) | 0.03 (fb) | 0.02 (17) | 0.02 (c8) | 0.02 (ca) | 0.02 (3b) | 0.02 (48) | 0.02 (7b) | 0.02 (42) | 0.02 (70) | 0.02 (54) | 0.02 (f3) | 0.02 (22) | 0.03 (72) | 0.03 (bb) |
 0.02 (41) | 0.02 (5a) | 0.02 (e4) | 0.02 (8e) | 0.02 (a3) | 0.02 (68) | 0.02 (8a) | 0.02 (7c) | 0.02 (39) | 0.02 (16) | 0.02 (25) | 0.02 (b1) | 0.02 (b5) | 0.02 (02) | 0.02 (82) | 0.03 (1d) |
 0.02 (31) | 0.02 (de) | 0.02 (d0) | 0.02 (24) | 0.02 (a0) | 0.02 (92) | 0.02 (de) | 0.02 (4b) | 0.02 (c9) | 0.02 (ce) | 0.02 (be) | 0.02 (97) | 0.02 (5f) | 0.02 (6f) | 0.02 (18) | 0.03 (9c) |
 0.02 (74) | 0.02 (89) | 0.02 (59) | 0.02 (dc) | 0.02 (c7) | 0.02 (20) | 0.02 (71) | 0.02 (9c) | 0.02 (a2) | 0.02 (97) | 0.02 (57) | 0.02 (4a) | 0.02 (b4) | 0.02 (74) | 0.02 (0f) | 0.03 (3b) |
 0.02 (d0) | 0.02 (34) | 0.02 (e2) | 0.02 (60) | 0.02 (9c) | 0.02 (1d) | 0.02 (96) | 0.02 (3e) | 0.02 (a5) | 0.02 (de) | 0.02 (e6) | 0.02 (36) | 0.02 (45) | 0.02 (4b) | 0.02 (7d) | 0.03 (7c) |
 0.02 (56) | 0.02 (19) | 0.02 (37) | 0.02 (62) | 0.02 (9f) | 0.02 (05) | 0.02 (79) | 0.02 (8e) | 0.02 (b0) | 0.02 (5c) | 0.02 (d6) | 0.02 (b2) | 0.02 (07) | 0.02 (dc) | 0.02 (20) | 0.02 (35) |
 0.02 (72) | 0.02 (af) | 0.02 (0a) | 0.02 (fe) | 0.02 (11) | 0.02 (76) | 0.02 (ad) | 0.02 (b0) | 0.02 (0e) | 0.02 (ee) | 0.02 (b5) | 0.02 (b6) | 0.02 (8d) | 0.02 (41) | 0.02 (b2) | 0.02 (09) |
 0.02 (ce) | 0.02 (75) | 0.02 (a3) | 0.02 (02) | 0.02 (d7) | 0.02 (78) | 0.02 (32) | 0.02 (75) | 0.02 (eb) | 0.02 (08) | 0.02 (17) | 0.02 (6f) | 0.02 (c8) | 0.02 (ab) | 0.02 (21) | 0.02 (5c) |
 0.02 (e3) | 0.02 (f3) | 0.02 (ba) | 0.02 (10) | 0.02 (1e) | 0.02 (cc) | 0.02 (e9) | 0.02 (9d) | 0.02 (5d) | 0.02 (19) | 0.02 (b2) | 0.02 (82) | 0.02 (a4) | 0.02 (d3) | 0.02 (31) | 0.02 (3c) |
 0.02 (da) | 0.02 (05) | 0.02 (3f) | 0.02 (4a) | 0.02 (17) | 0.02 (ce) | 0.02 (f3) | 0.02 (f6) | 0.02 (40) | 0.02 (01) | 0.02 (5f) | 0.02 (e1) | 0.02 (29) | 0.02 (fb) | 0.02 (d2) | 0.02 (69) |
 0.02 (3b) | 0.02 (e5) | 0.02 (af) | 0.02 (bb) | 0.02 (20) | 0.02 (c7) | 0.02 (35) | 0.02 (72) | 0.02 (d9) | 0.02 (34) | 0.02 (fe) | 0.02 (0e) | 0.02 (f7) | 0.02 (e3) | 0.02 (60) | 0.02 (0a) |
 0.02 (03) | 0.02 (ff) | 0.02 (c0) | 0.02 (77) | 0.02 (b1) | 0.02 (38) | 0.02 (85) | 0.02 (36) | 0.02 (79) | 0.02 (84) | 0.02 (20) | 0.02 (27) | 0.02 (96) | 0.02 (c8) | 0.02 (05) | 0.02 (d1) |
0e eb a7 43 00 9d 67 d2 e5 63 cf 4c 5c b0 77 cb
Cleaning up
[probyns@compute-4 emma]$
```

Closing statements

- All my finished research is open source

  Decoder: https://github.com/rpp0/gr-lora

  Fingerprinting: https://github.com/rpp0/lora-phy-fingerprinting

  ChipWhisperer plugin: http://research.edm.uhasselt.be/probyns/cw_hacky_usrp_plugin.zip

- Some of my current research directions
  - Relation to machine learning → loss function and features vs. correlation
    - → Can we improve the state of the art in this way?
  - Increasing the range of EM attacks
    - → Analyzing below the noise floor, custom antenna designs, etc.
  - Open to collaborations!

Further reading

- Here are some related papers which I found interesting

Fingerprinting

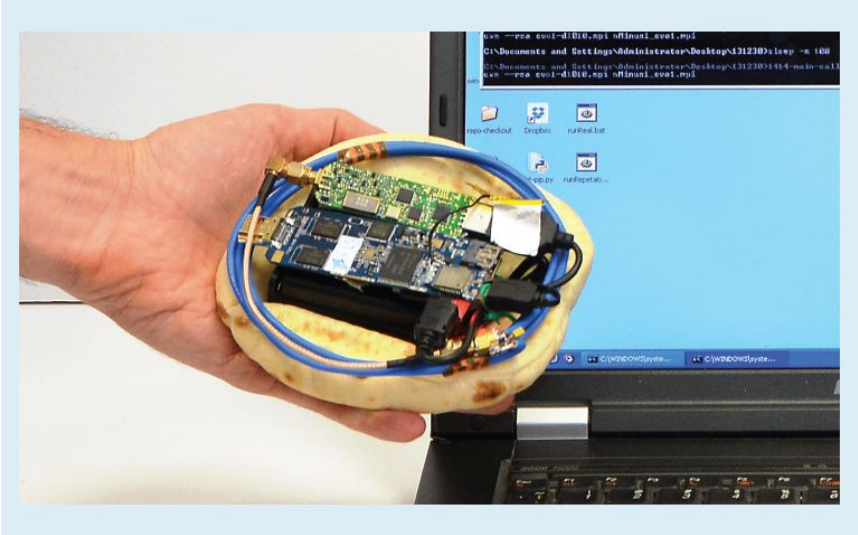
  - *Why MAC address randomization is not enough...* (Mathy Vanhoef et al.)
  - *Challenges to PHY anonymity for Wi-Fi* (Peter Iannucci)
  - *Convolutional Radio Modulation Recognition*... (Timothy O'Shea et al.)
  - *Unsupervised Learning on Neural Network Outputs* (Yao Lu et al.)
  - *Device Fingerprinting in Wireless Networks…* (Qiang Xu et al.)

EM side-channel attacks

  - *Correlation Power Analysis with a Leakage Model* (Eric Brier et al.)
  - *Enhancing Electromagnetic Side-Channel Analysis in...* (David P. Montminy.)
  - *NewAE Wiki page* (https://wiki.newae.com/Main_Page)
  - *Power Analysis Attacks against IEEE 802.15.4 Nodes* (Colin O'Flynn et al.)

Other nice examples of EM side channel attacks

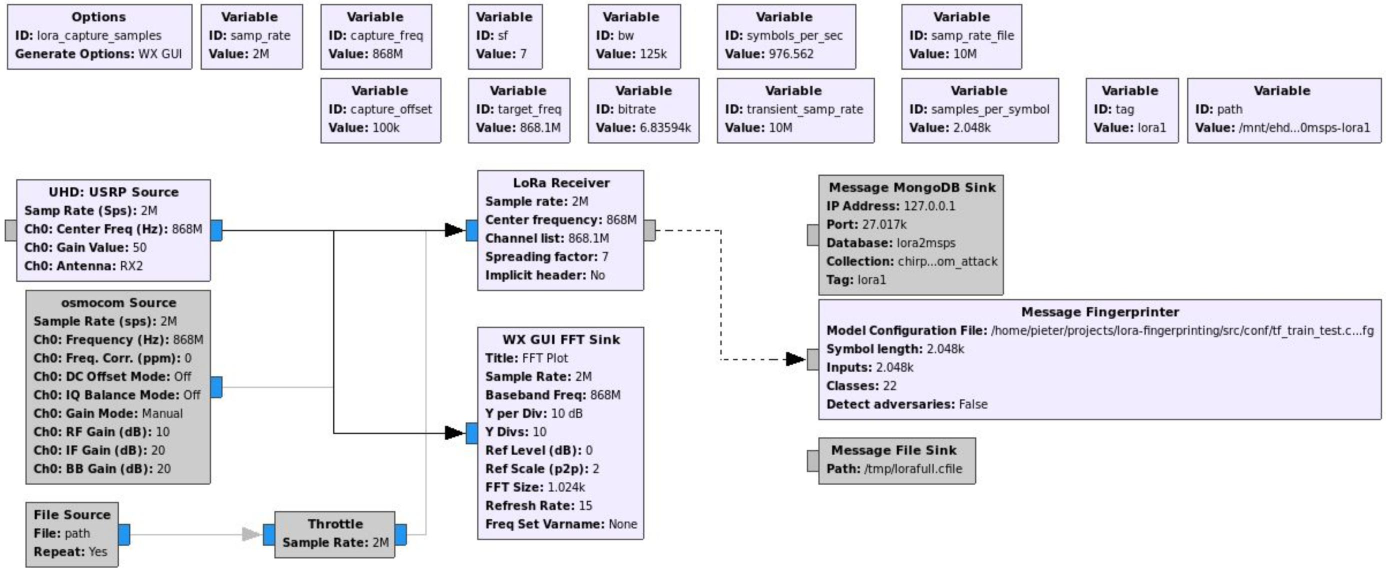
Fully extract decryption keys, by measuring the laptop's chassis potential during decryption of a chosen ciphertext.



Full extraction of ECDSA secret signing keys from OpenSSL and CoreBitcoin running on iOS devices.

Source: https://www.tau.ac.il/~tromer/handsoff/

Demo



Options
ID: lora_capture_samples
Generate Options: WX GUI
Variable
ID: samp_rate
Value: 2M
Variable
ID: capture_freq
Value: 868M
Variable
ID: sf
Value: 7
Variable
ID: bw
Value: 125k
Variable
ID: symbols_per_sec
Value: 976.562
Variable
ID: samp_rate_file
Value: 10M
Variable
ID: capture_offset
Value: 100k
Variable
ID: target_freq
Value: 868.1M
Variable
ID: bitrate
Value: 6.83594k
Variable
ID: transient_samp_rate
Value: 10M
Variable
ID: samples_per_symbol
Value: 2.048k
Variable
ID: tag
Value: lora1
Variable
ID: path
Value: /mnt/ehd...0msps-lora1
UHD: USRP Source
Samp Rate (Sps): 2M
Ch0: Center Freq (Hz): 868M
Ch0: Gain Value: 50
Ch0: Antenna: RX2
osmocom Source
Sample Rate (sps): 2M
Ch0: Frequency (Hz): 868M
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 10
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20
File Source
File: path
Repeat: Yes
Throttle
Sample Rate: 2M
LoRa Receiver
Sample rate: 2M
Center frequency: 868M
Channel list: 868.1M
Spreading factor: 7
Implicit header: No
WX GUI FFT Sink
Title: FFT Plot
Sample Rate: 2M
Baseband Freq: 868M
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
Message MongoDB Sink
IP Address: 127.0.0.1
Port: 27.017k
Database: lora2msps
Collection: chirp...om_attack
Tag: lora1
Message Fingerprinter
Model Configuration File: /home/pieter/projects/lora-fingerprinting/src/conf/tf_train_test.c...fg
Symbol length: 2.048k
Inputs: 2.048k
Classes: 22
Detect adversaries: False
Message File Sink
Path: /tmp/lorafull.cfile

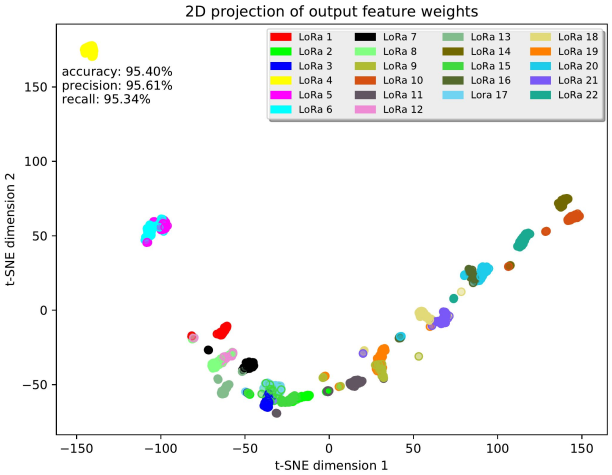
Questions?

pieter.robyns@uhasselt.be

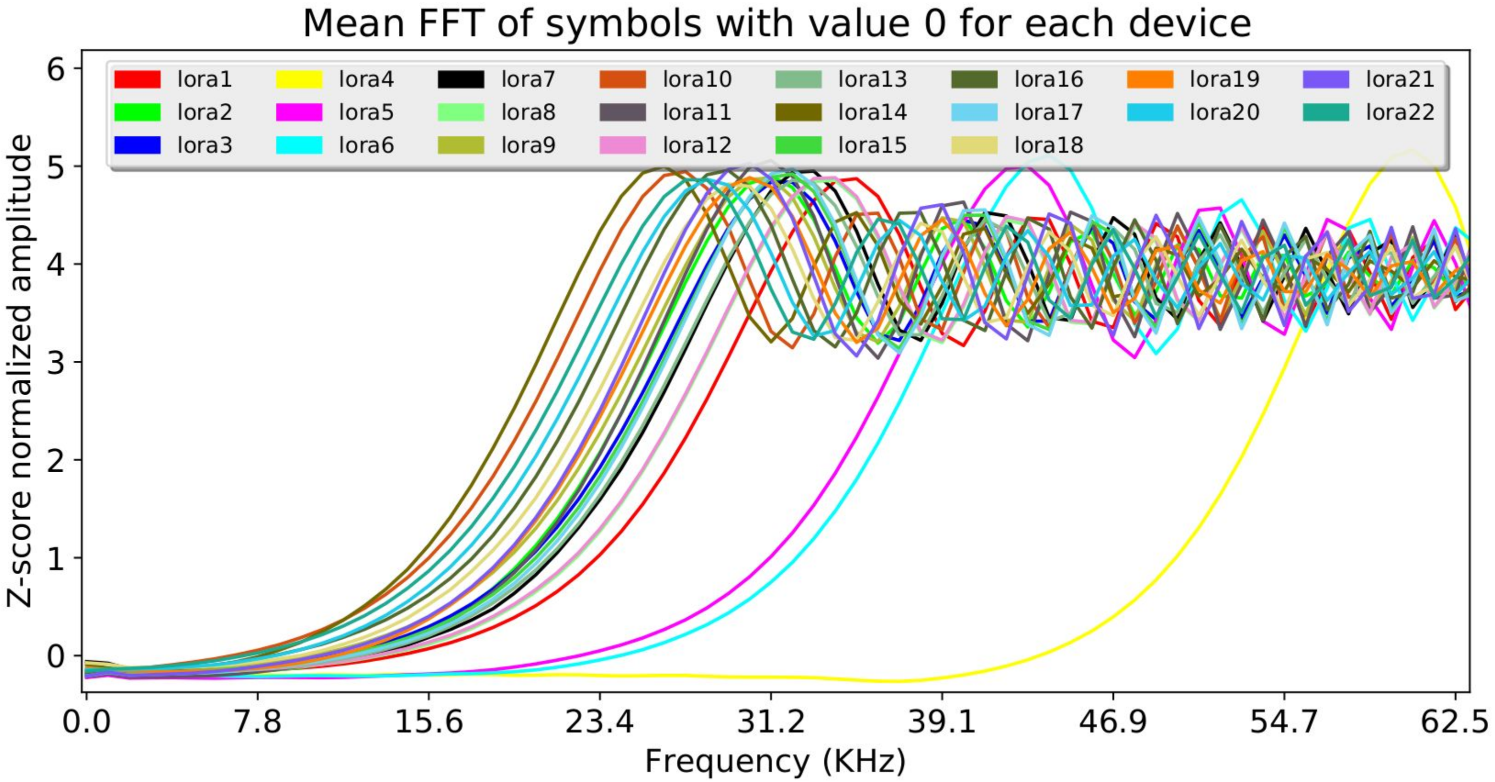
Extra slides

But wait, what about devices that we can't train?

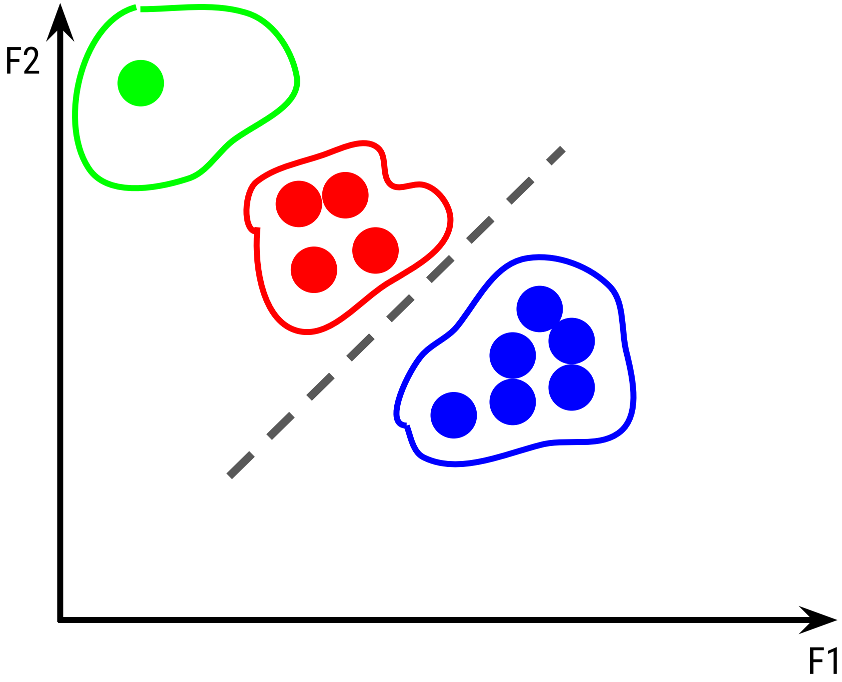
- Technique called zero shot classification
  - Learn "attributes" during training
  - Describe unseen devices using learned attributes
  - Example: cluster on neural network outputs that was trained with a number known LoRa devices



But wait, what about devices that we can't train?

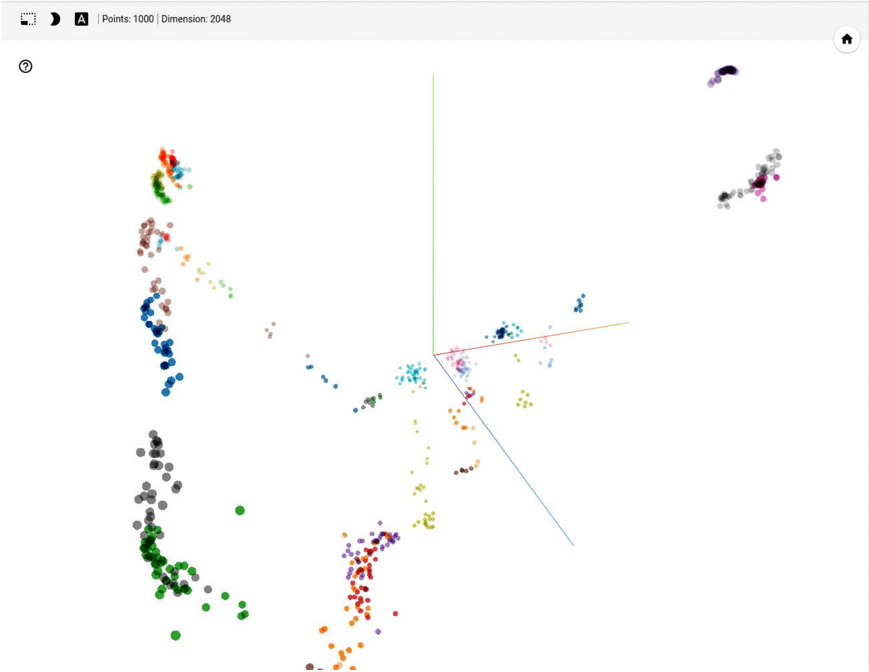


But wait, what about devices that we can't train?

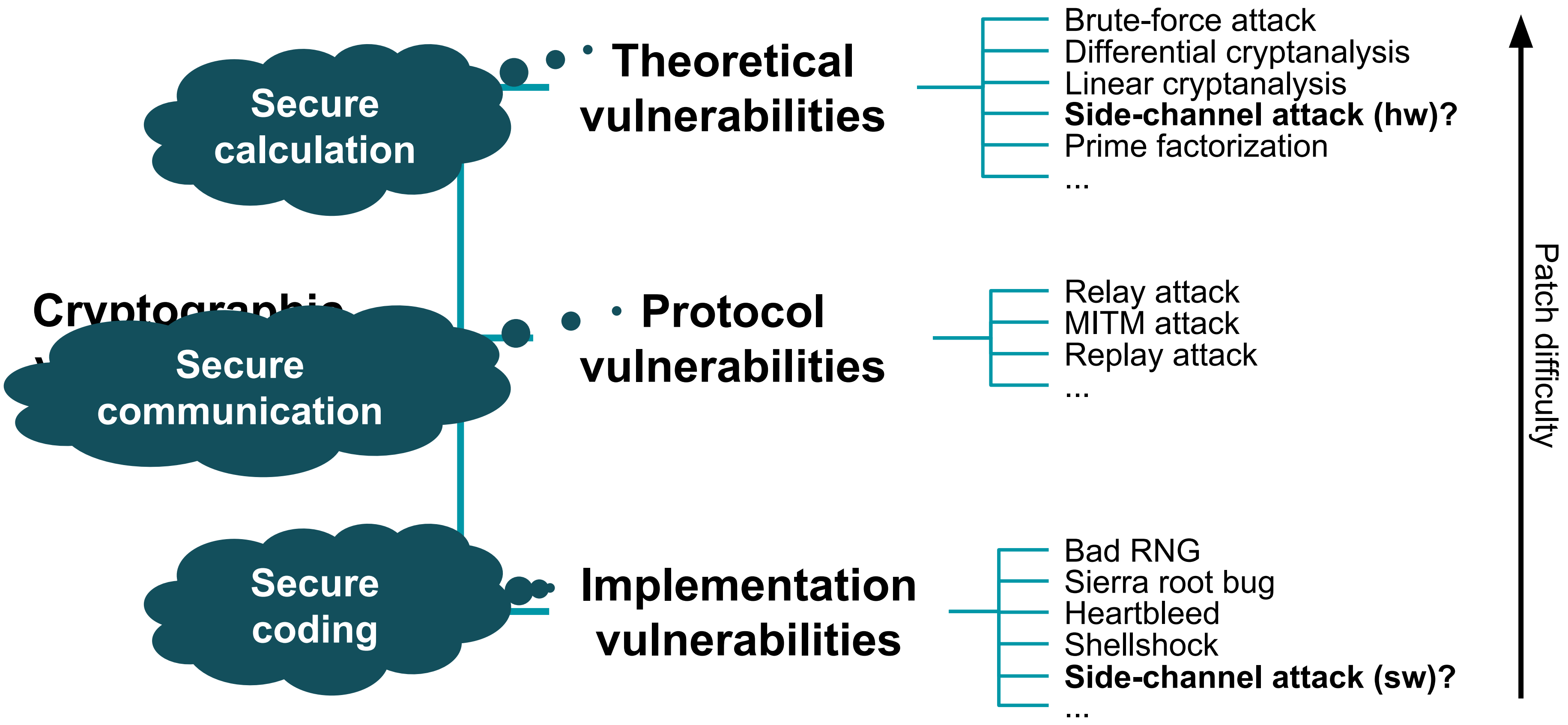


Visualizing the raw data

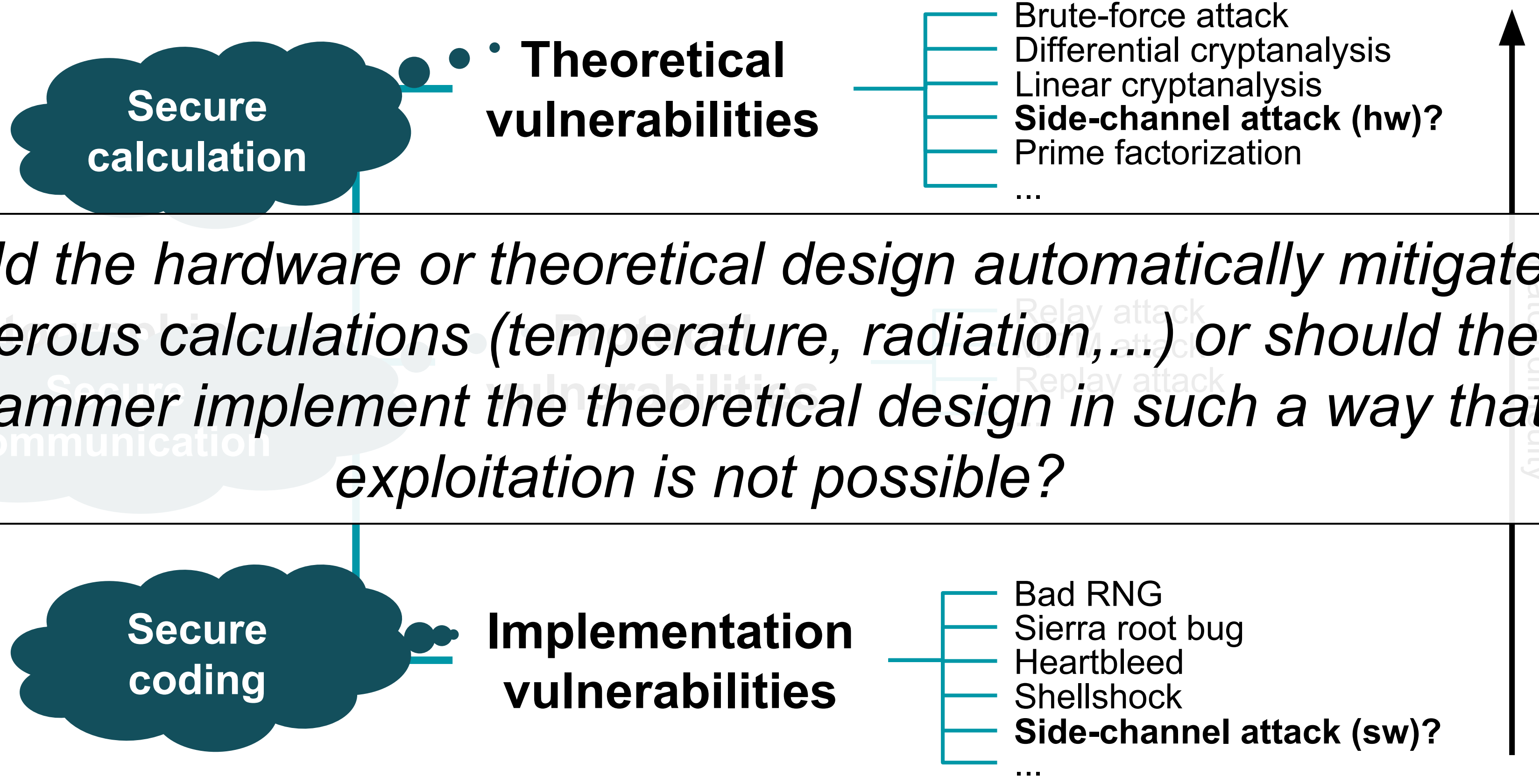
- Visualizing the signal using Principal Component Analysis (PCA):



SCAs within the vulnerability landscape



SCAs within the vulnerability landscape



Should the hardware or theoretical design automatically mitigate dangerous calculations (temperature, radiation,...) or should the programmer implement the theoretical design in such a way that exploitation is not possible?