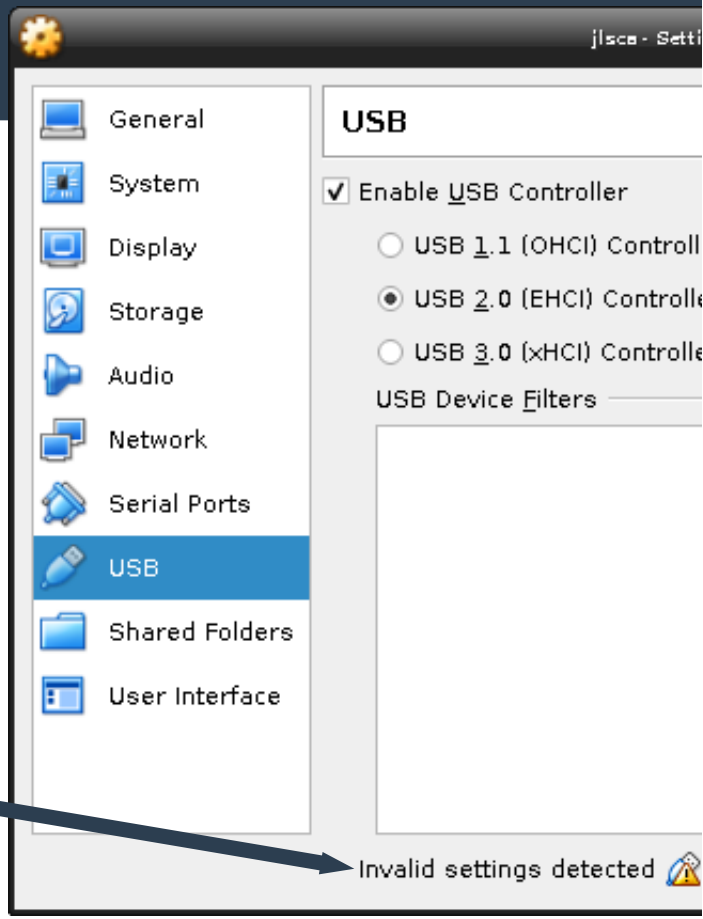
Get the VM to work!

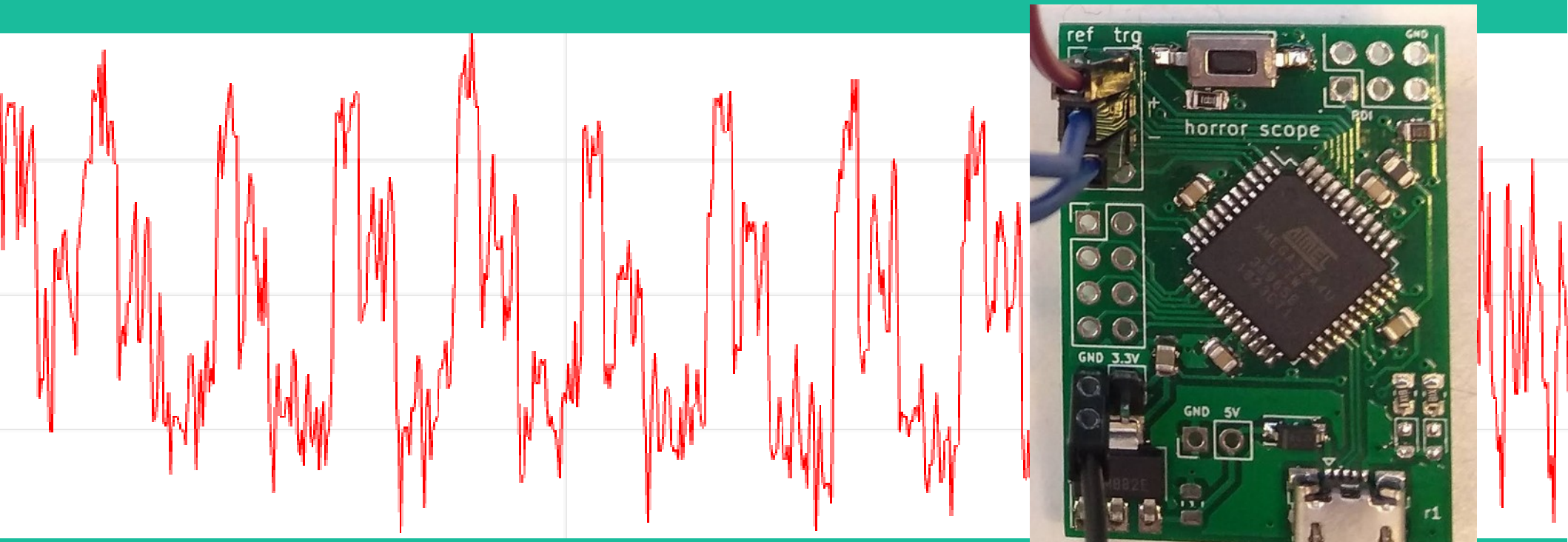
Connect USB device

Start VM → Devices → USB

No devices?

- Machine → settings → USB
- See this error?
  - Add ‘contrib’ and ‘non-free’ repos
  - sudo apt-get update
  - sudo apt-get install virtualbox-ext-pack
- No error?
  - sudo adduser <YOUR USERNAME> vboxusers
  - Logout and login again





The Cheapiest SCA workshop

Albert Spruyt
Alyssa Milburn

Agenda

- **Introduction talk** Courtesy of Ilya Kihzvatov
- **Our “hardware”**

- **Exercise 1a: Connect everything**
- **Exercise 1b: Collect traces**
- **Exercise 2: Preprocess**
- **Exercise 3: Get the key!**

Our problem

So, side channels are great!

- Measure power → Maths magic → get Keys!

But we are cheap and lazy:

- Crypto attacks are tricky to get right
- Analog stuff is complicated and mathsy
- Don’t want to buy an oscilloscope

Previous Work

- **Hardware (oscilloscopes) are expensive**
  - Real oscilloscope: $400+
  - Colin O'Flynn: ChipWhisperer $180–$250
    - https://newae.com/files/ChipWhispererIntroCOSADE2014.pdf
  - Rafa Boix Carpi (with a Hantek) $60
    - https://www.rsaconference.com/writable/presentations/file_upload/mbs-w02-cheapscate-attacking-iot-with-less_-than-60_.pdf
  - ChipWhisperer Nano $50

Previous Work

- **Hardware (oscilloscopes) are expensive**
  - ChipWhisperer $180 – $250
  - Rafa Boix Carpi (Hantek USB) $60
  - ChipWhisperer Nano $50
  - That’s a lot of beers!

Solving our hardware woes

Let’s build an awesome scope?

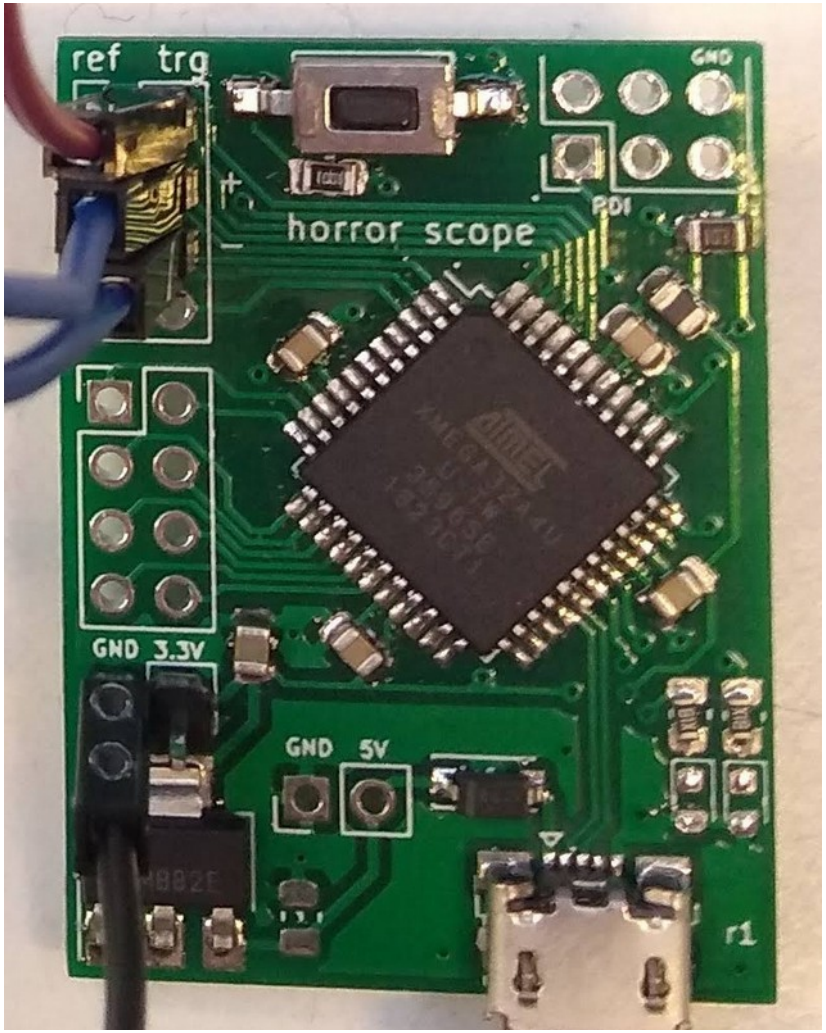
However:

- We have no idea how to do PCB design
- We don’t know how to solder things
- Generally electronics stuff seems hard

Let’s hack something together!

We built:

- **The HorrorScope**
  - ~5 euro
  - Xmega32a4u breakout
  - Built in USB2@12Mbit/s
  - 12 bit ADC@2 MSPS
  - You can add capacitors if you really want to
  - Also does glitching :D



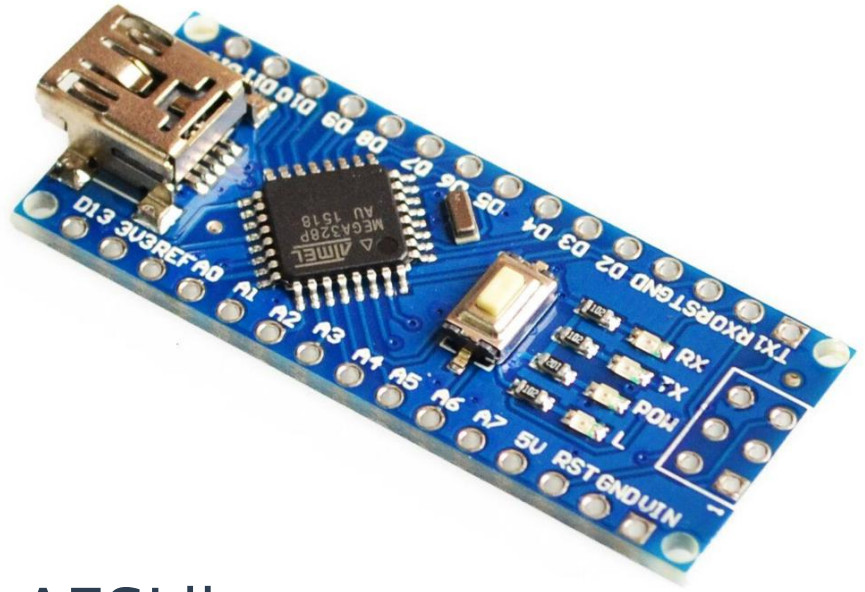
Bill of Materials (BOM)

- **~5 euro ex. VAT**
- **Why so expensive?**
  - Xmega: 2.50eur (or more)
  - PCBs: ~1eur in small quantities
  - Headers, USB connector, LDO, caps .. add up :(
- **This excludes:**
  - Soldering – go to your local hackerspace!

Which target?

Problem:

We need a cheap but familiar target...



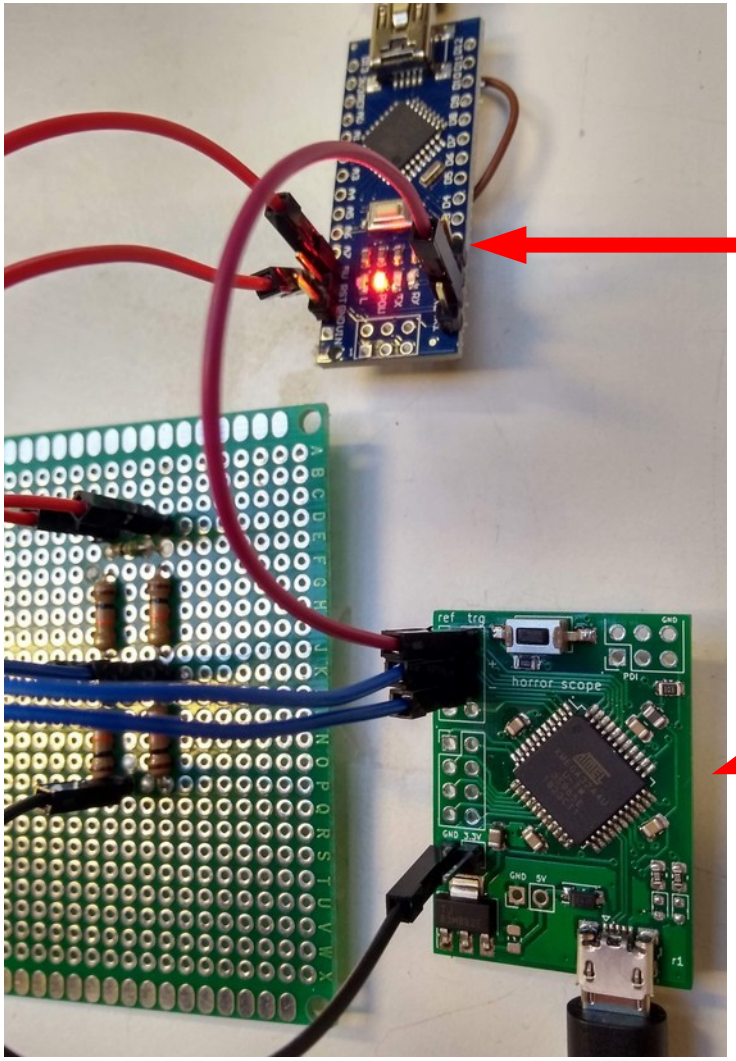
Arduino Nano

- 16 Mhz
- ~3-5 euro

Software:

- Generic AES from github
- If you have time: Arduino AESLib

~~More problems~~
Design considerations



We have:

"Target": Arduino Nano + AES

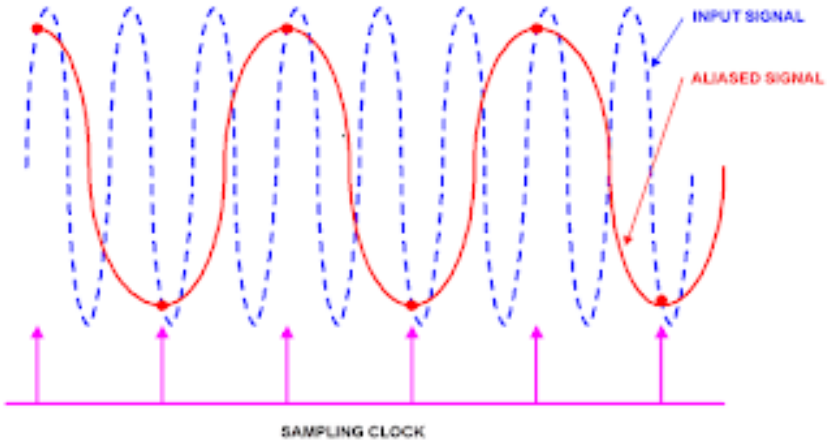
"Oscilloscope": HorrorScope

What's left?

- ~~Problems~~ Challenges

~~More problems~~
Design considerations

Sampling below Nyqist frequency



Source: http://blog.teledynelecroy.com/2013/06/back-to-basics-sampling-rate.html

Solution:

- take more traces... maybe average them?
- maybe ignore the datasheet?

~~More problems~~
Design considerations

No analog front-end (amplifier)

- Use 12-bits ADC? Take more traces?

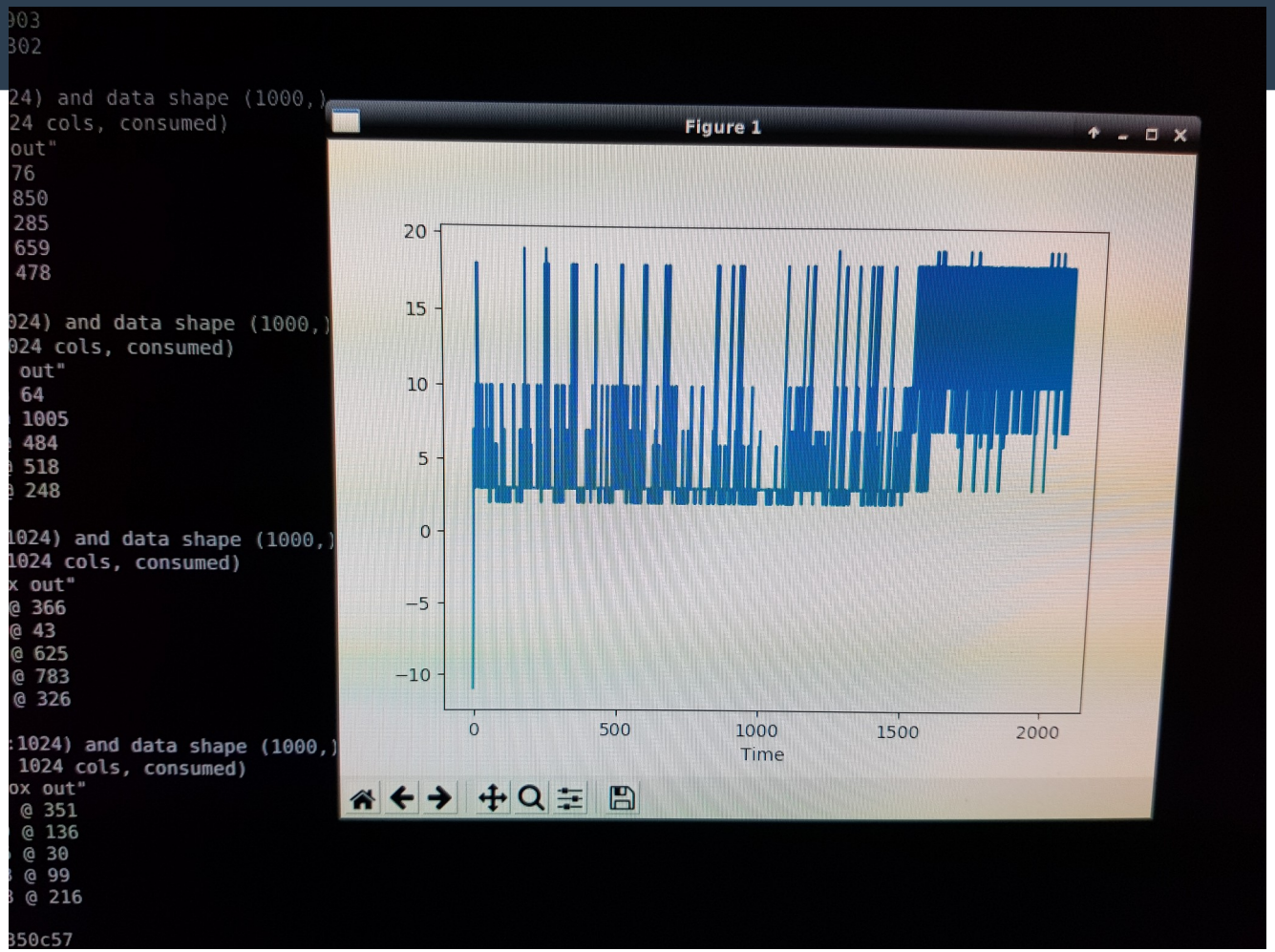
Random noise

- Postprocess in software? Take more traces?

DC offset, resolution, etc

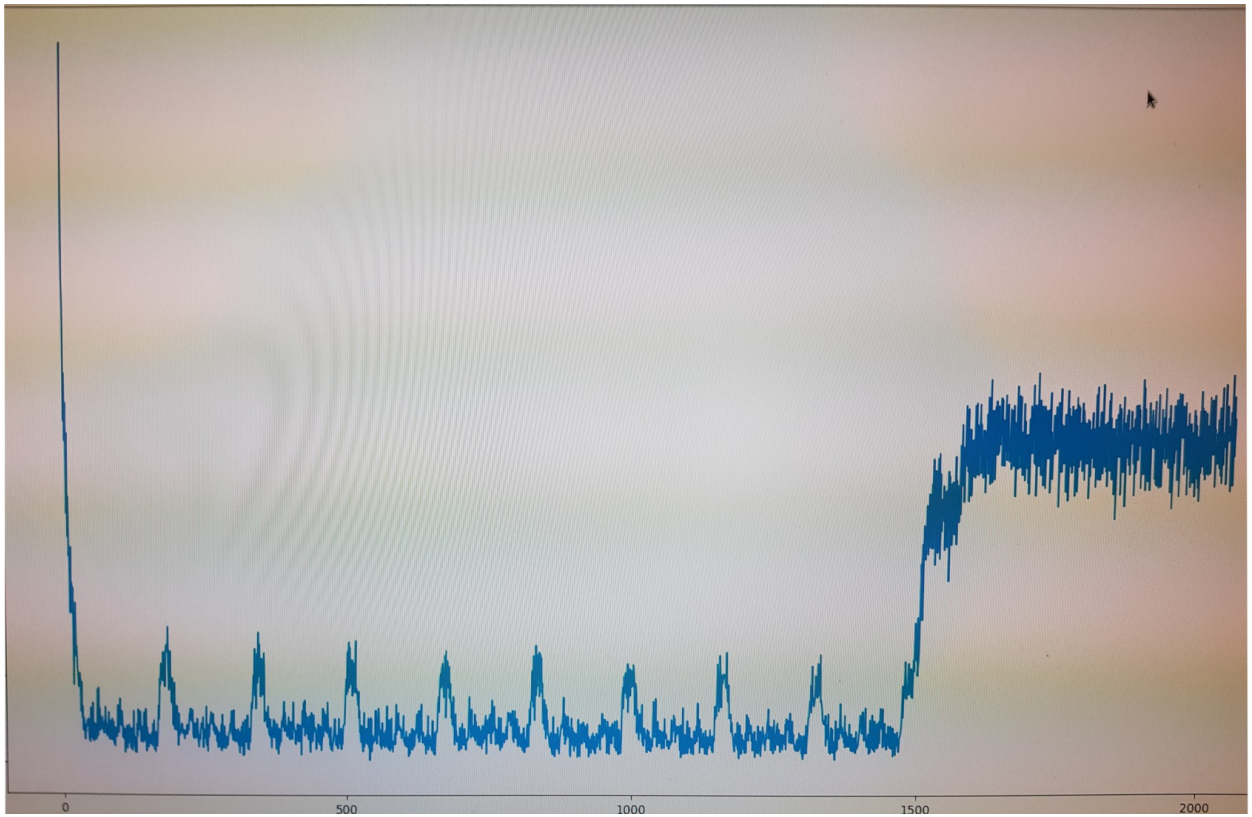
- ~~AC coupling,~~ ~~use AREF~~
- Take more traces!

Funny story time



Datasheet: “Don’t do this”

More traces!



100 traces averaged: A wild AES appears!

How do we get the keys?

Take advantage of other people's generosity (aka open source):

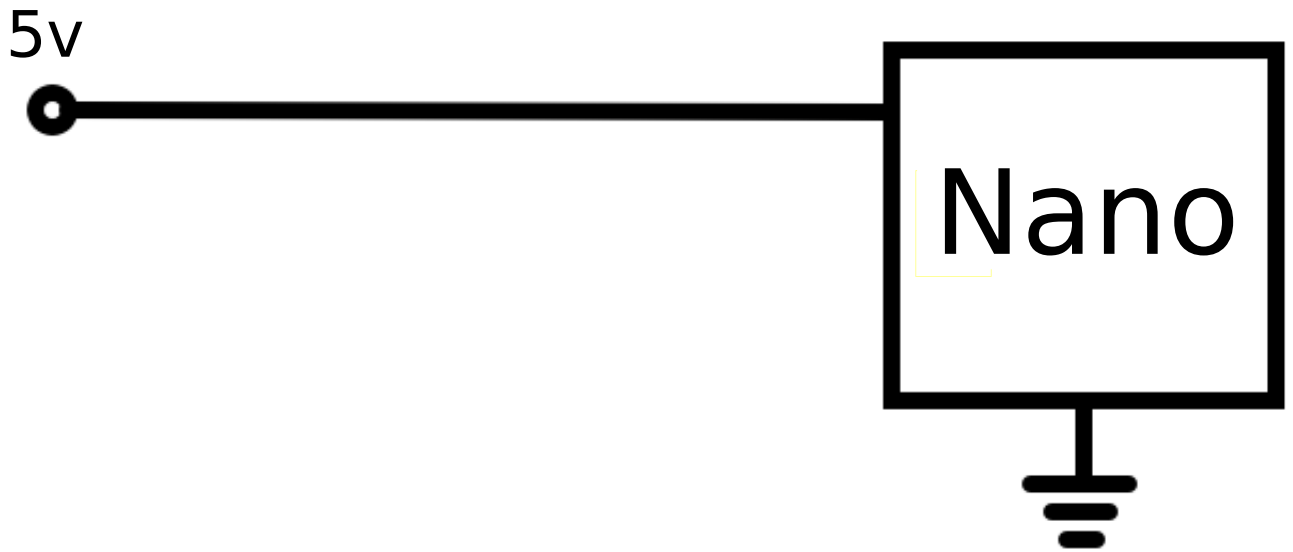
JLsca: Cees (+ Ilya)

For the purposes of this workshop we will refer to this as 'magic'. Run script, keys come out!

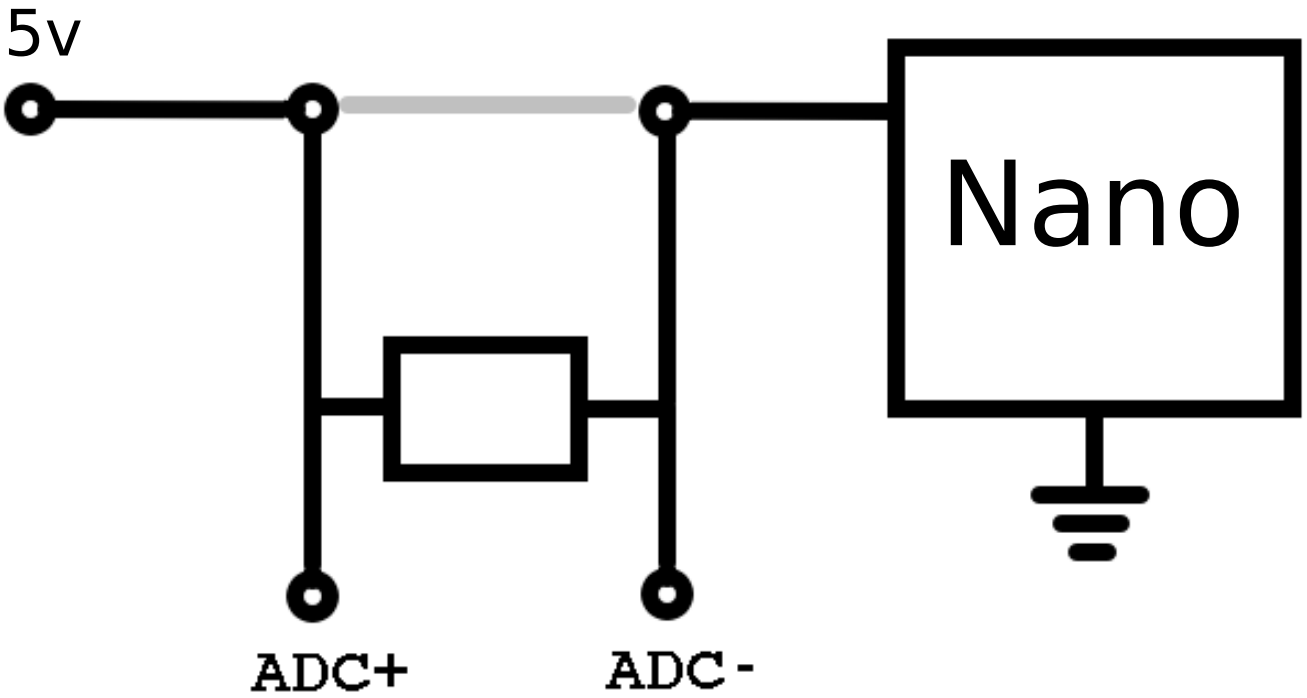
How do we measure

- **Measure voltage drop over shunt resistor in series with IC**
  - Voltage drop depends on IC power usage

Making the cut



Making the cut

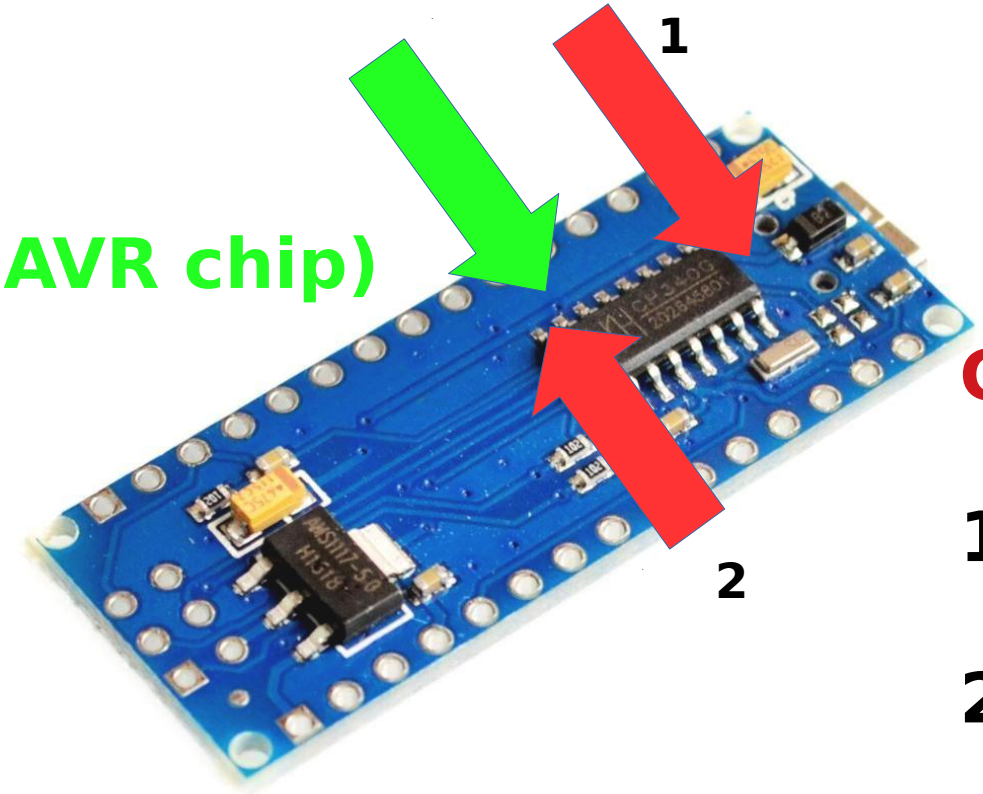


Horror Scope

VOLTAGE CUT

To power the Arduino:

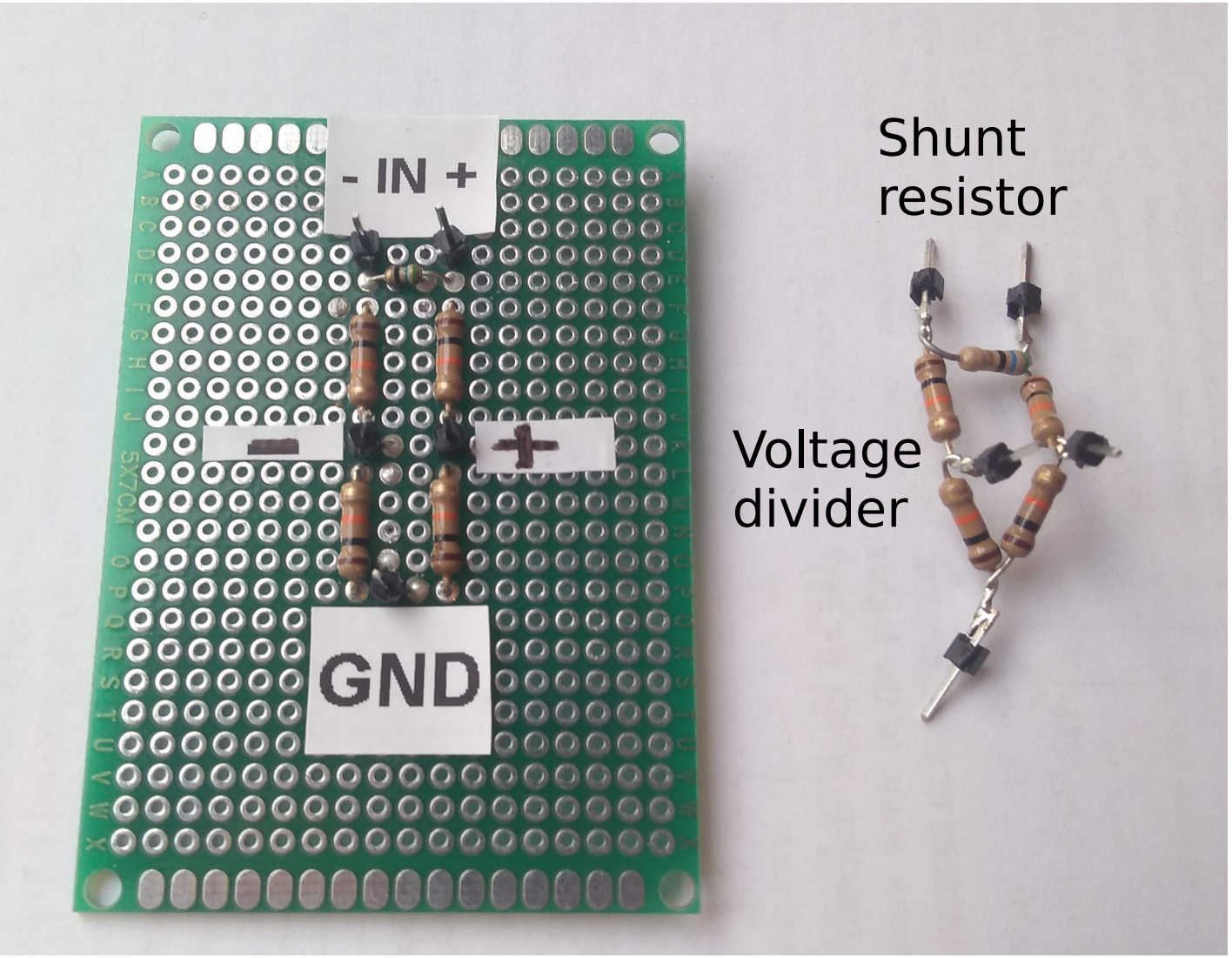
– Connect 5v to Extra pin (connect 1 and 2)



CUTs **1 & 2**

1: connect diode to 5v
2: connect chipside to pin

Resistor clusters!



Exercise 1a

Connect everything!

- Shunt resistor to Nano power cuts
  - Pin sticking out (5V from USB, power cut 1)
  - 5V on Nano (5V to chip, power cut 2)
- Trigger: Xmega portA2 → Nano RX (serial)
- Measurement: Xmega Port A3 & A5
- **Get the OK from us BEFORE you..**
- .. connect it all to USB
  - Ideally there is no magic smoke

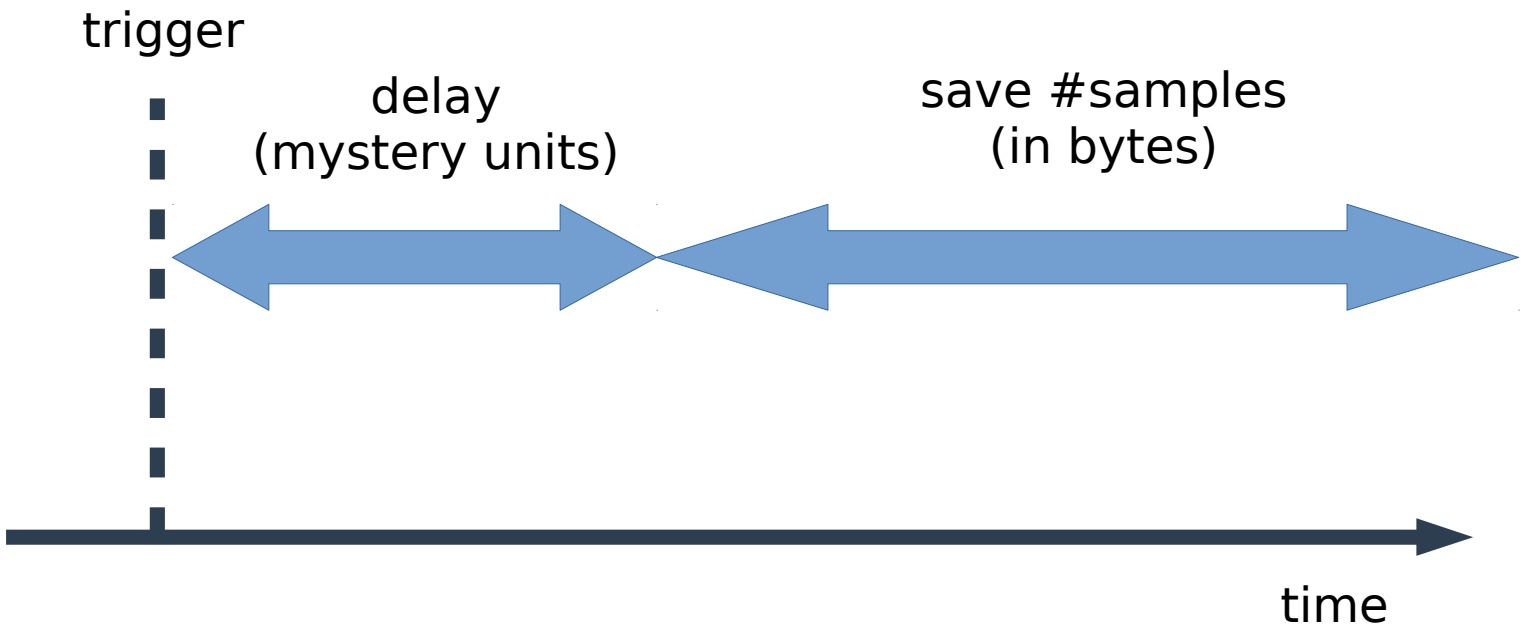
Exercise 1b – Start encrypting

We want to run AES on target (Nano):

- ./runTarget.py
  - **Sends:** random strings
  - **Receives:** encrypted random strings

- **Measure:** power trace
- **Exercise:** What is the key?

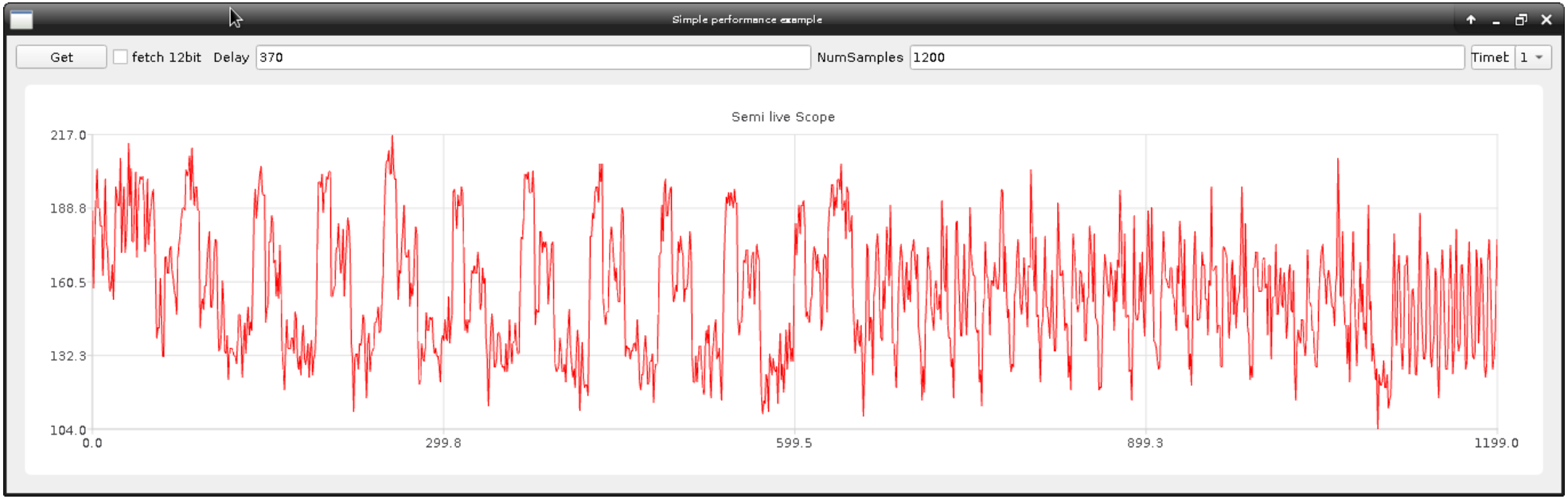
Parameters



Exercise 1c – Take Traces

Play around in our fancy GUI

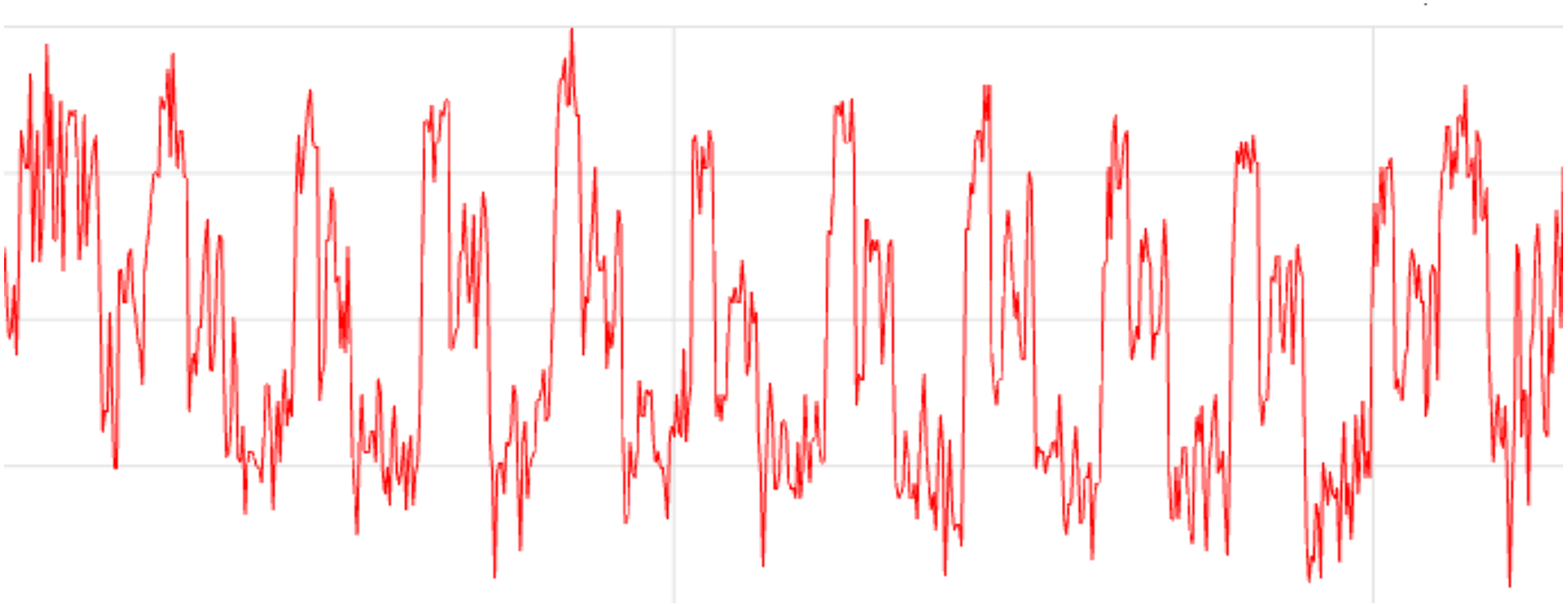
- Keep running AES on target: ./runTarget.py
- “Semi-live” scope: ./gui.py



Unlabeled axes? Where we’re going, we don’t need units

What are we looking for?

AES has a pre-round, 9 “full” rounds and a final round



Exercise 1d – Collect traces..?

Spotted the AES rounds?

- Use a high timebase
- Count 9 (fences) or 10 (fenceposts) rounds

Pick good parameters + remember them (write them down!)

- Make sure the first round is in view
- Select a good speed (timebase) + delay
  - Make sure there’s enough margin before/after the round

Triggering

- **We have parameters!**
- **Sometimes we don’t see the encryption?!?**
  - What the heck are we seeing?
  - How are we actually trigggering?
- **Real problem: the encryption and measurement are not in sync**
  - which input belongs to which measurement?
- **Solution:** put measurement and encryption in 1 script

runSetup

Program flow

1) Send *most* of the command to the Arduino
2) Ask the scope to get ready (“arm”)
3) Send the last byte to the Arduino → USB
4) Scope triggers on ‘seeing’ the last byte

    (Remember, trigger is on the Arduino’s RX)

5) Save: input & output & (power) measurement
6) GOTO 1

Exercise 1e – Collect traces

Finally: Take traces!!!

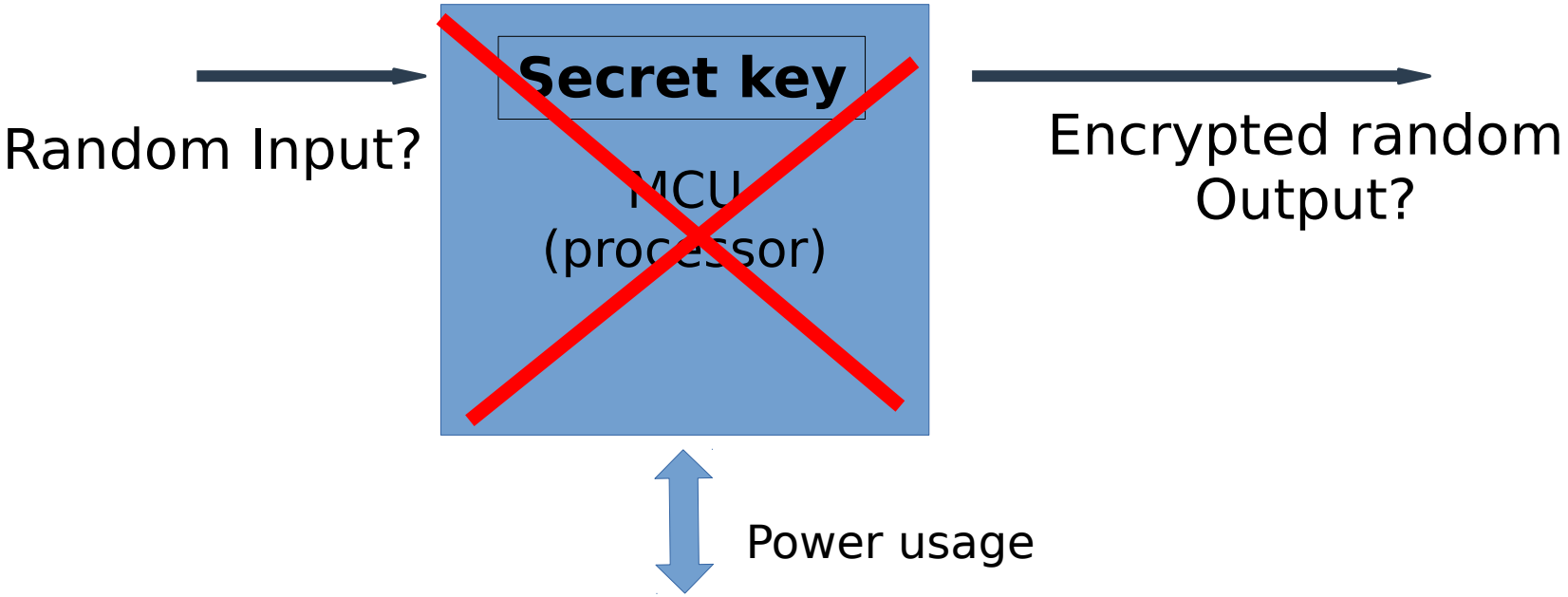
- Remember those parameters
  - Make sure the first round (or last round) is in view
  - Select a good speed, with enough margin
- Copy the settings into runSetup.py
  - There are kind of two different places, sorry :(
  - Timebase turns into ‘sampleSpeed’
- Run: ./runSetup.py
- Collect 10k traces (in the background)

Exercise 2: processing

- **We'll need to preprocess traces**
  - Align the traces
  - Remove any DC offset

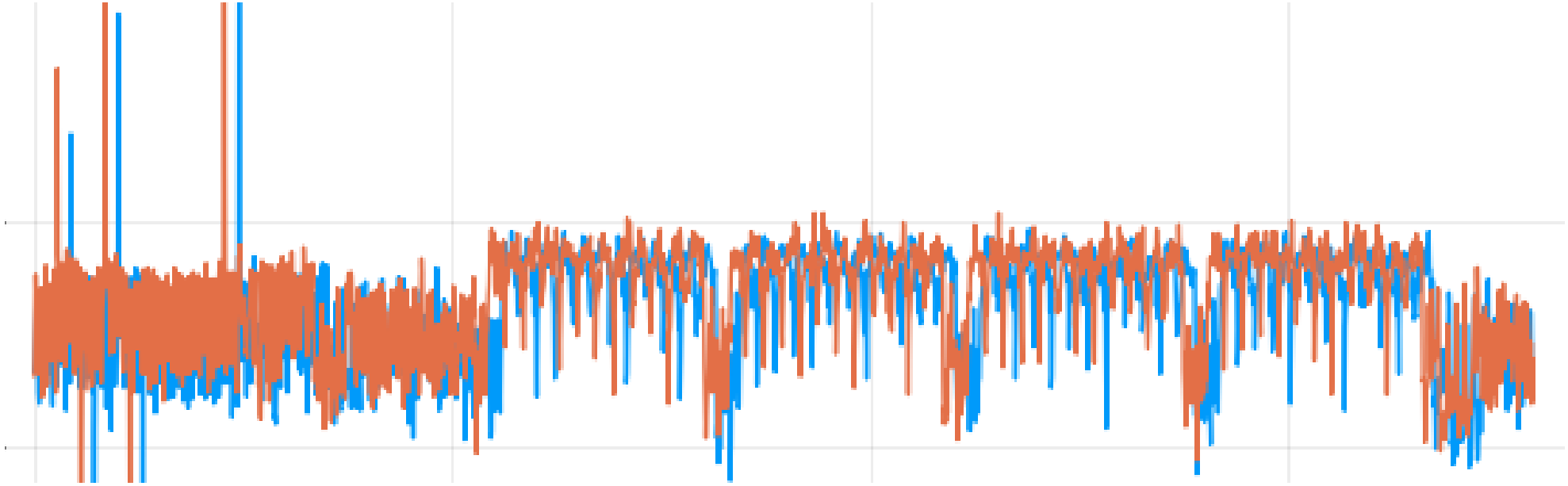
  How do we know the result is good?
  - Look for I/O correlation!
  - Let's check correlation between the input bytes and the power measurement.

What do we know?



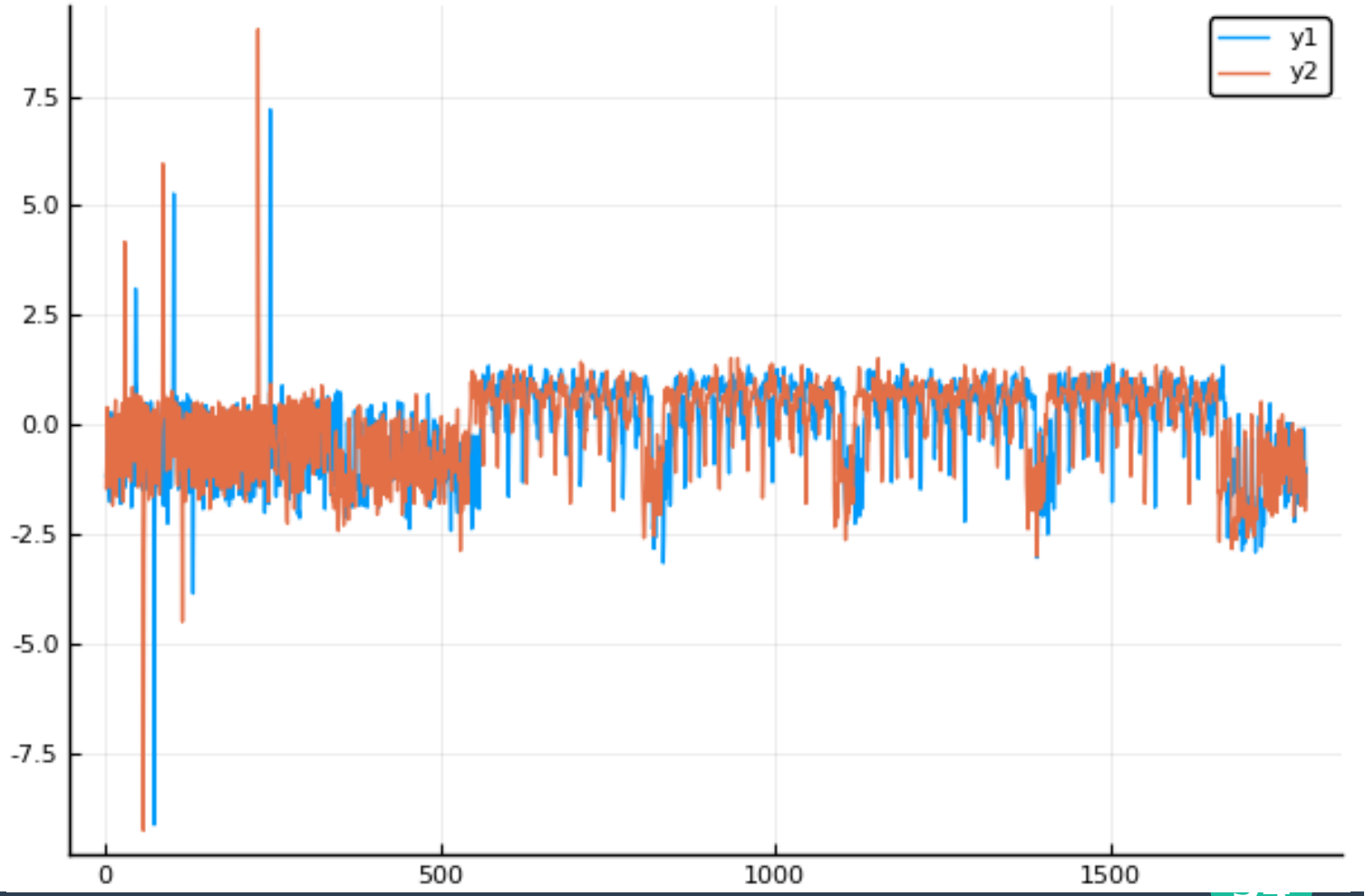
What’s wrong?

- **Misalignment**
- **DC offset**

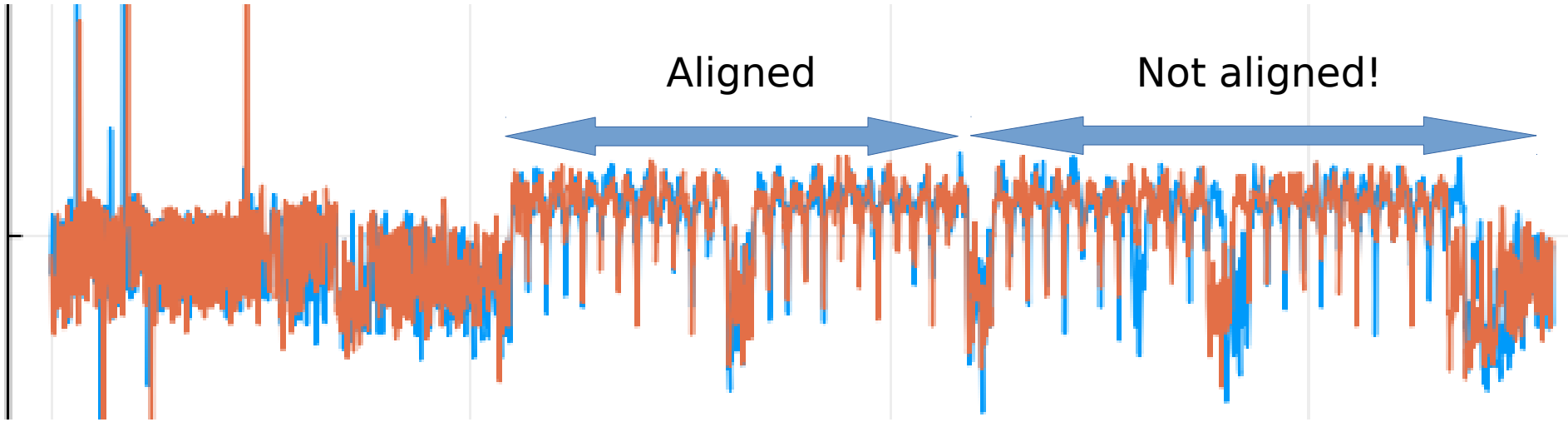


DC Offset fixed!

addSamplePass(trs, x -> (x .- mean(x)) ./ std(x))

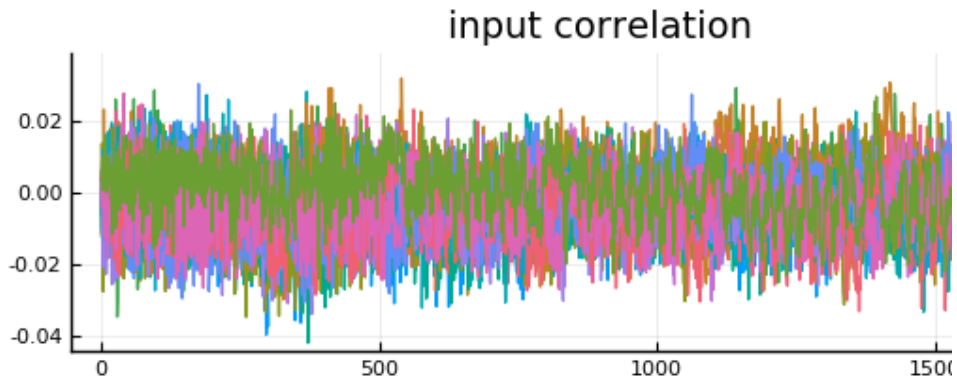


Aligned

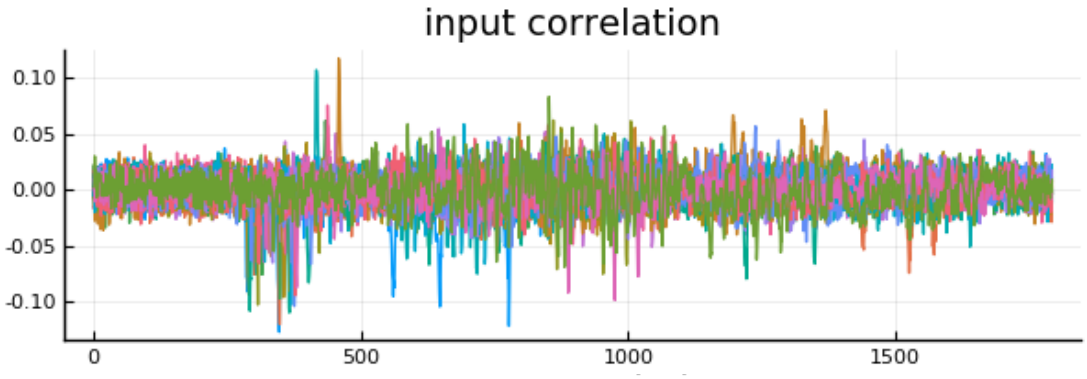


IO correlation

Before

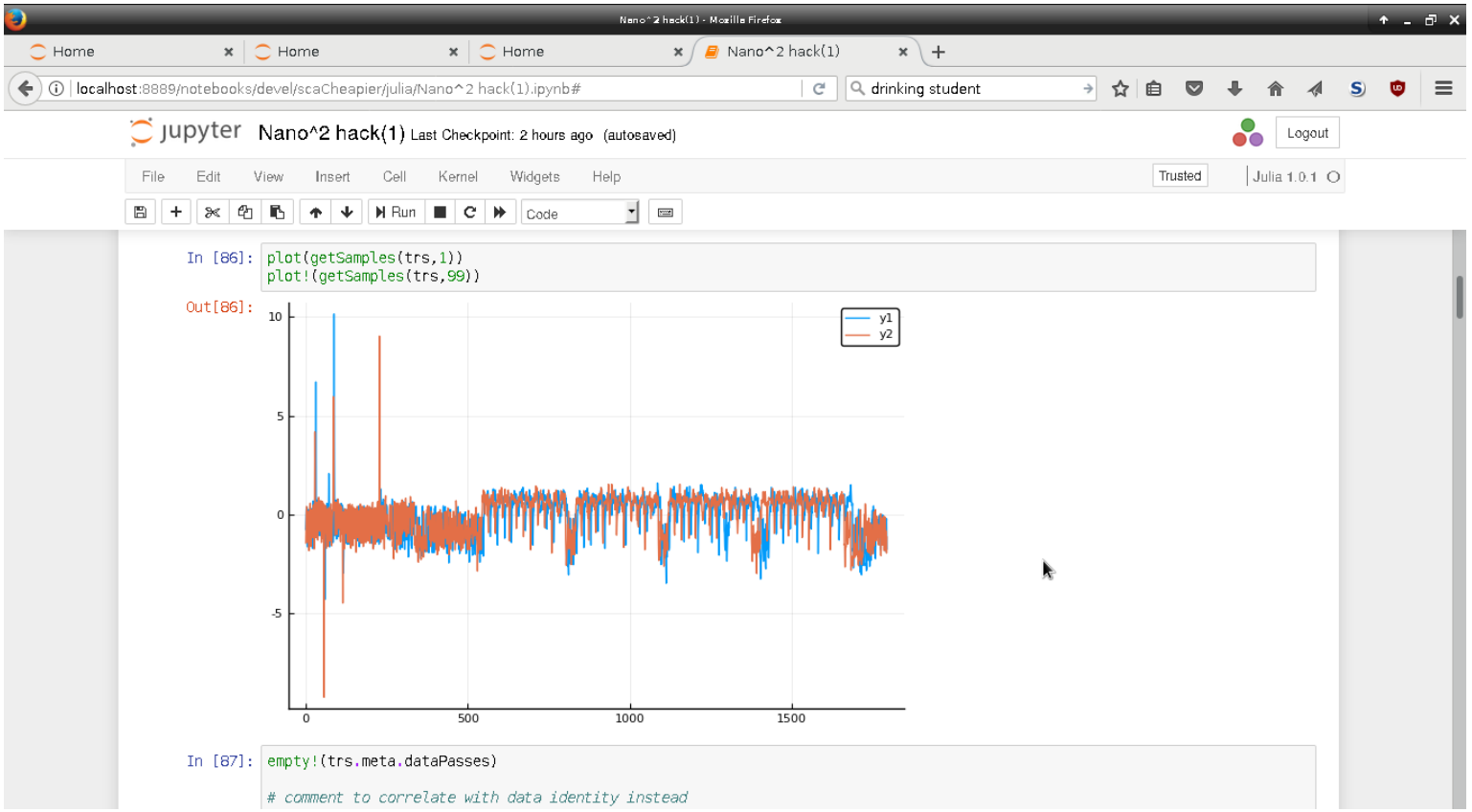


after



Open the “Nano.ipnyb” notebook

- **Open a NEW terminal**
  - #: julia



Get good I/O correlation!

Change the reference parameters

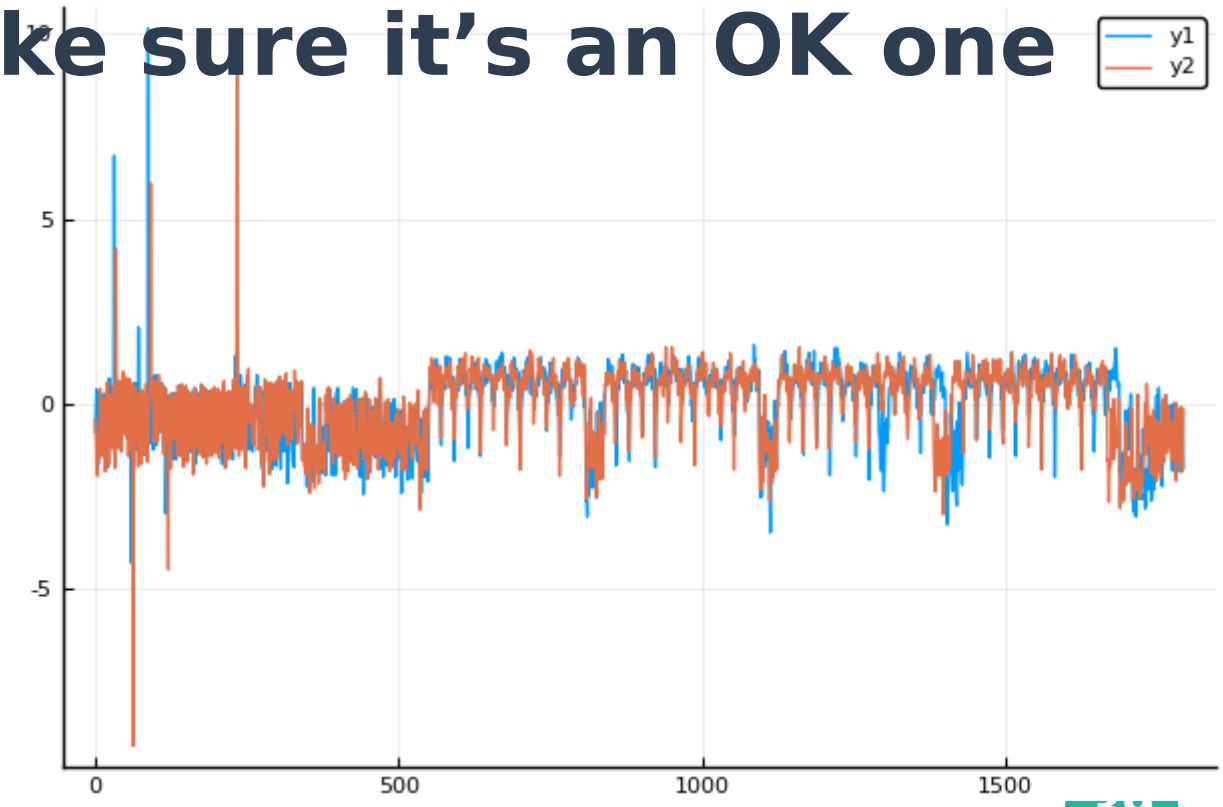
We want to match a PATTERN, so:

1) Trace (!) – make sure it’s an OK one

2) Offset & size

3) Shift

4) Threshold



Exercise 3: Get the key

- I/O correlation
  - Correlate between input byte *a* and power trace
- Guess all values after key addition
  - “Intermediate value” = Sbox(*a* xor K)
- So, for all values of K (we try all of them):
  - Calculate correlation between Sbox(*a* xor K) and power trace
  - Pick the K with the highest correlation score

Homework

- Topics to explore
- Targets: things to break

Next steps

- **Glitch the target and do DFA!**
- **HW hacking: HorrorScope-style**
  - Use AC coupling, use the Aref, add an amplifier
  - Other MCUs: PIC32MK (16msps), LPC (80msps)
  - Write better firmware (e.g., Negative Delays)
- **Use a ChipWhisperer..**

  or a real Oscilloscope
- **Talk to us!**

Target: Assembly AES

Arduino AVR library

- https://github.com/DavyLandman/AESLib
- Based on AVR-CRYPTO-LIB
- A lot faster & leaks less
- You need more traces and processing
- Traces are available in the VM/Tarball

Extra exercises

Fault Attacks!

- The HorrorScope can sort of power the Nano
- Perform Fault Injection and Differential Fault Analysis
- Requires
  - 3.3v FTDI cable
  - 1uF capacitor?
- Talk to us! (and look in the FI folder)

Special Thanks

- **Cees-Bart ‘ceesb’ Breunesse**
  - https://github.com/Riscure/Jlsca
- **Rafa Boix Carpi**
  - https://www.instagram.com/rafacapri/
- **And others ….**

AES

https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

KeyExpansion—round keys are derived from the cipher key using Rijndael's key schedule. AES requires a separate 128-bit round key block for each round plus one more.

Initial round key addition

- **AddRoundKey**—each byte of the state is combined with a block of the round key using bitwise xor.
- **9, 11 or 13 rounds:**
  - **SubBytes**—a non-linear substitution step where each byte is replaced with another according to a lookup table.
  - **ShiftRows**—a transposition step where the last three rows of the state are shifted cyclically a certain number of steps.
  - **MixColumns**—a linear mixing operation which operates on the columns of the state, combining the four bytes in each column.
  - **AddRoundKey**
- **Final round (making 10, 12 or 14 rounds in total):**
  - **SubBytes**
  - **ShiftRows**
  - **AddRoundKey**