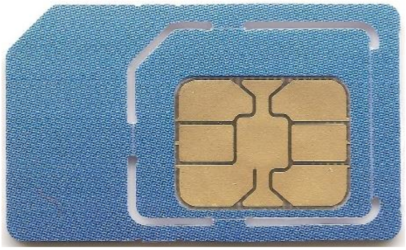
How Deep Learning Helps Compromising USIM

Martin Brisfors, Sebastian Forsmark, Elena Dubrova
School of Electrical Engineering and Computer Science
Royal Institute of Technology (KTH)

Overview

- Motivation
- Background
  - AKA, MILENAGE, AES
- Measurment setup
  - Locating the attack point
- Training & key extraction using CNN
- Demo of a USIM attack
- Summary and open problems

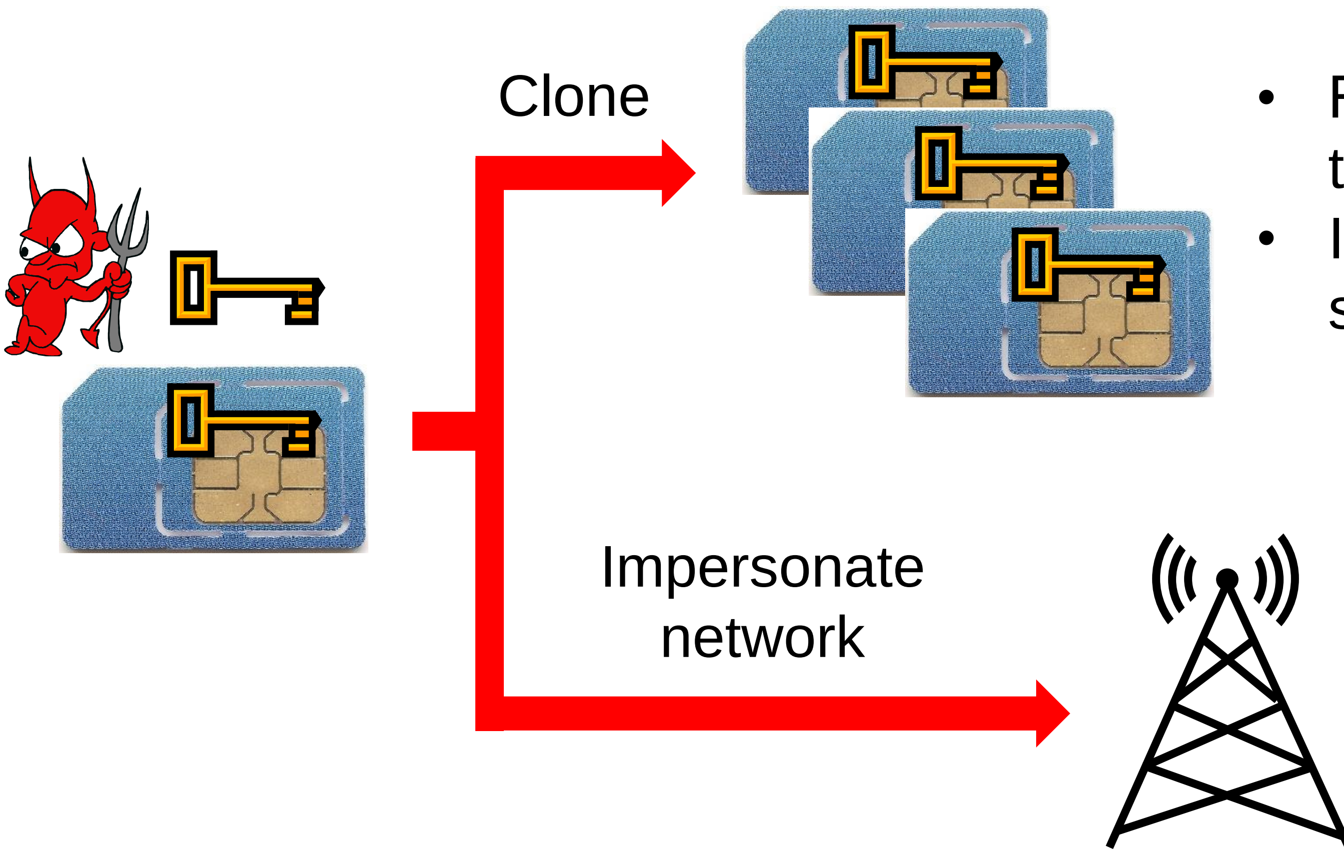
Universal Subscriber Identity Module (USIM)



Source:Telefónica O_2 Europe

- USIM is a type of smart card
- Contains:
  - Secret key K pre-shared with home subscriber server
  - International Mobile Subscriber Identity (IMSI)
  - Operator Variant Algorithm Configuration Field (OP)
  - …
- All cryptographic operations involving K are carried out within the USIM

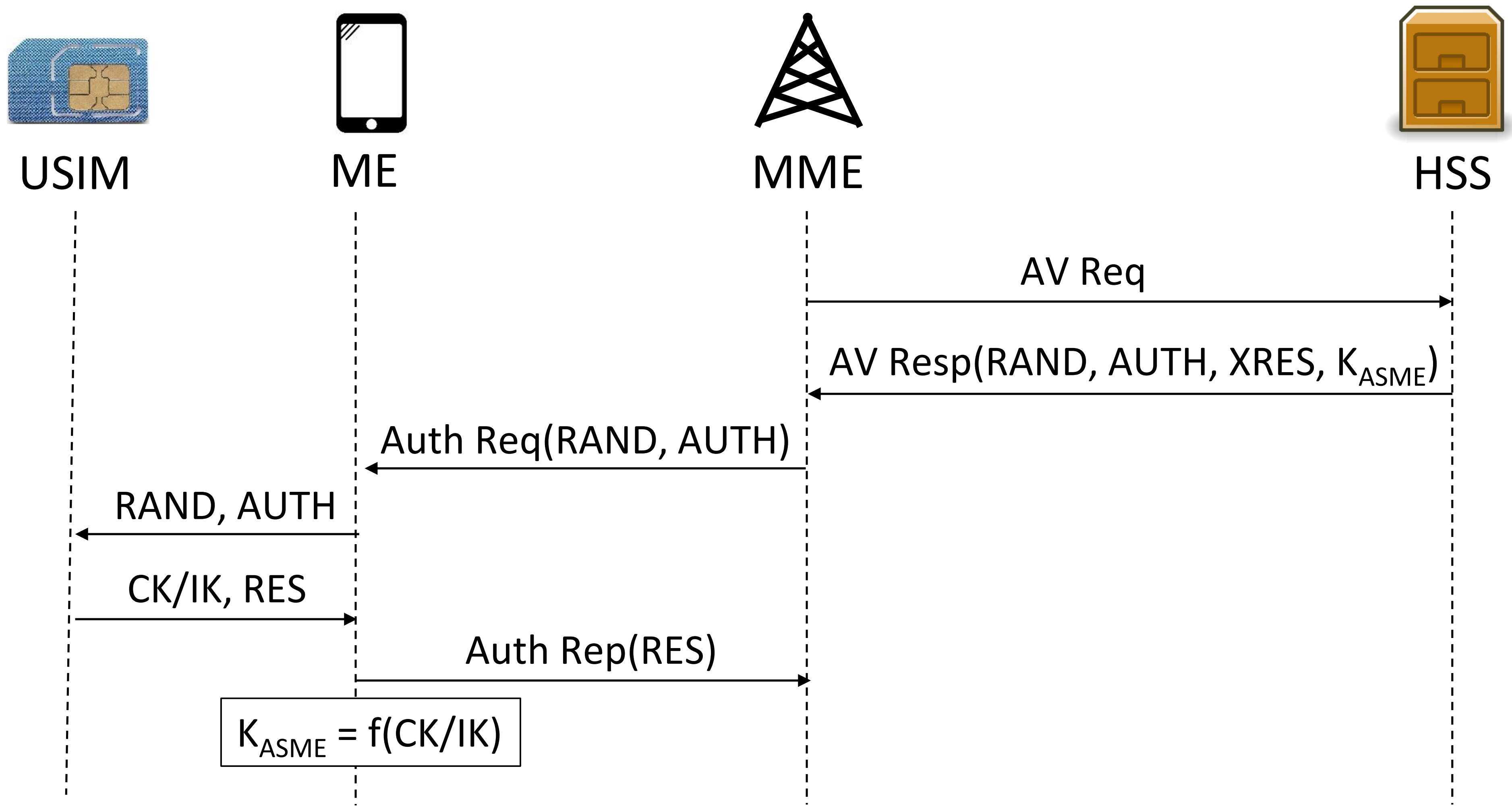
3G/4G/5G security relies on the USIM's key



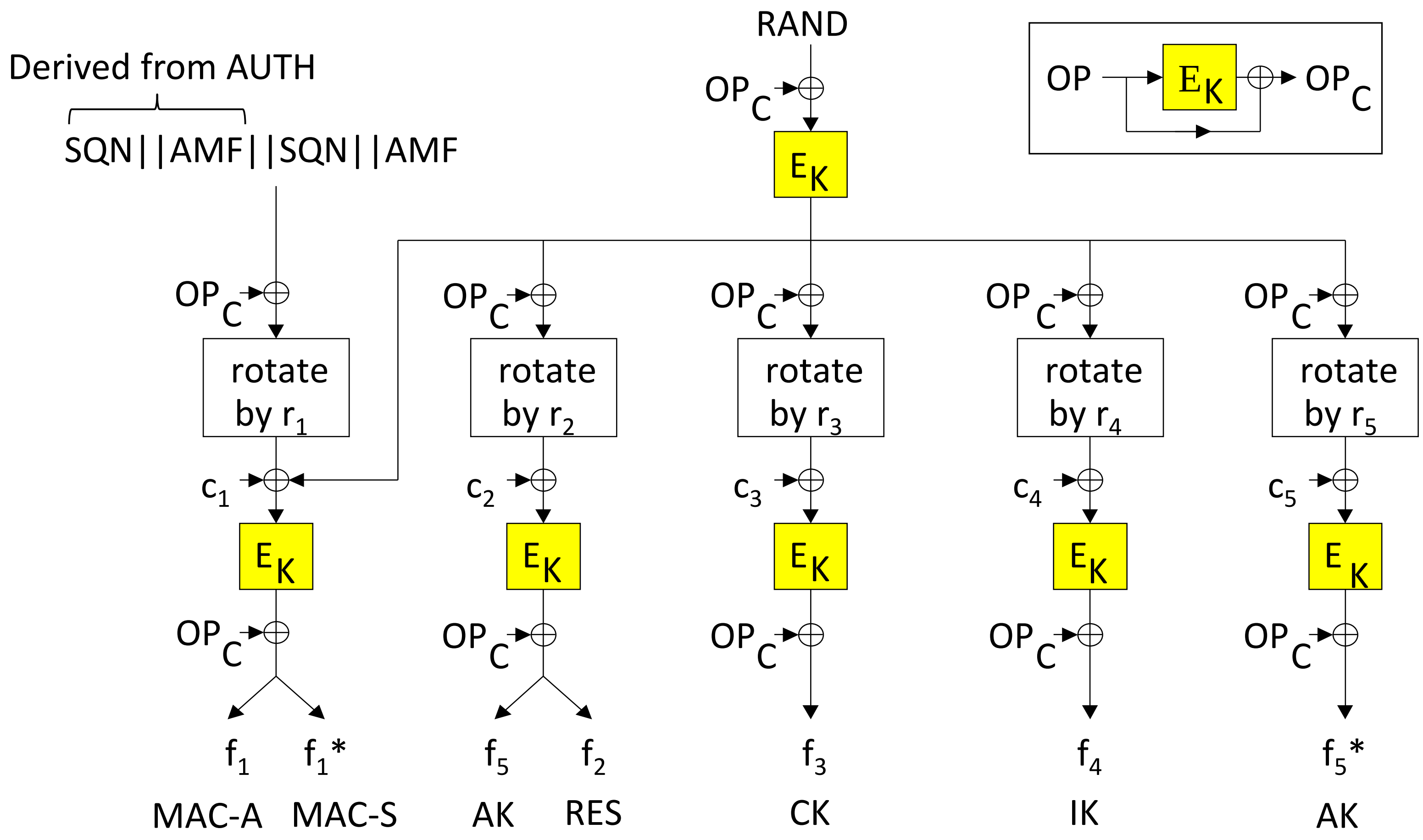
- Receive subscriber's texts and calls
- Impersonate subscriber

- Extract information from subscriber

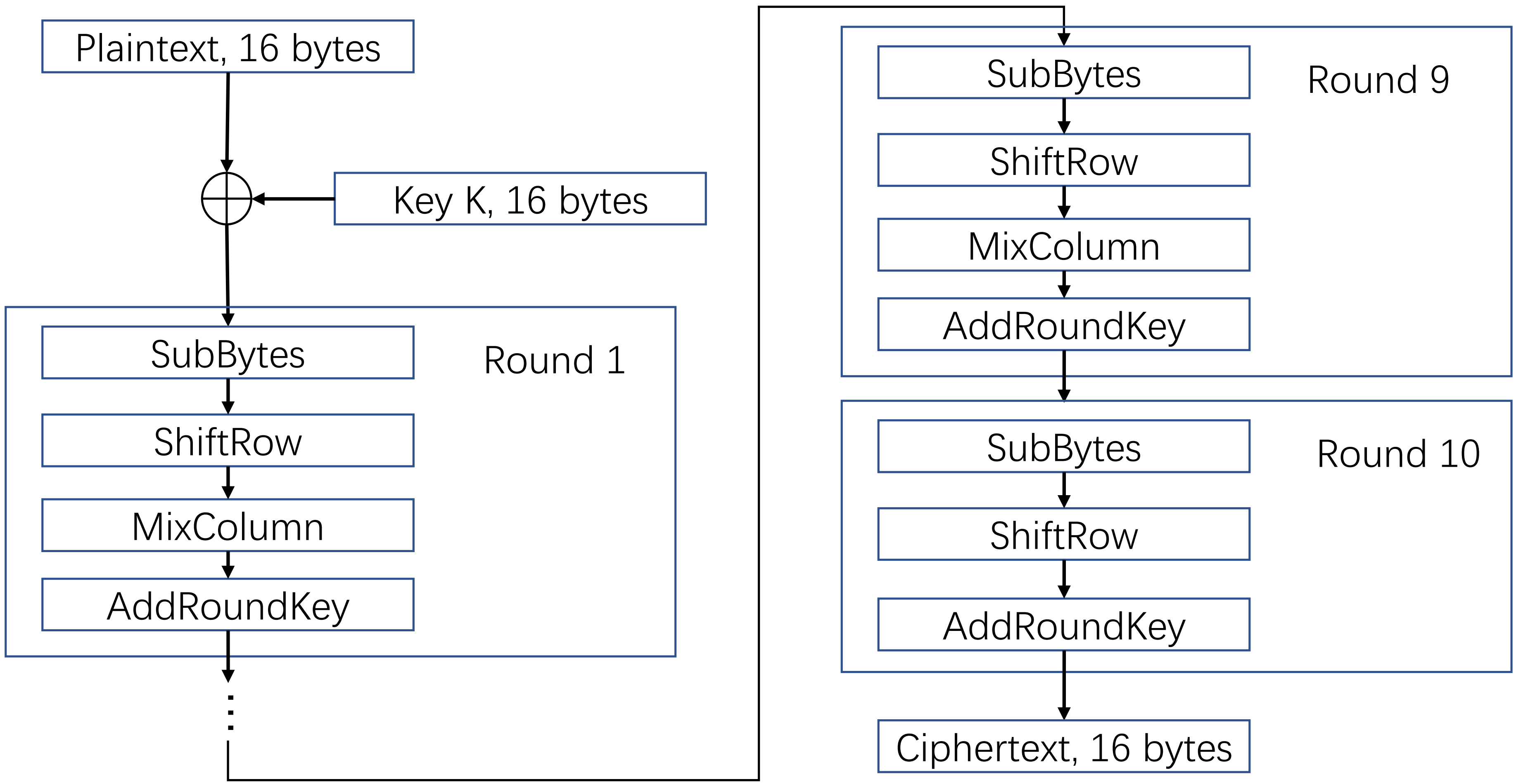
Authentication and Key Agreement (AKA) in 4G



MILENAGE algorithm



AES-128 algorithm



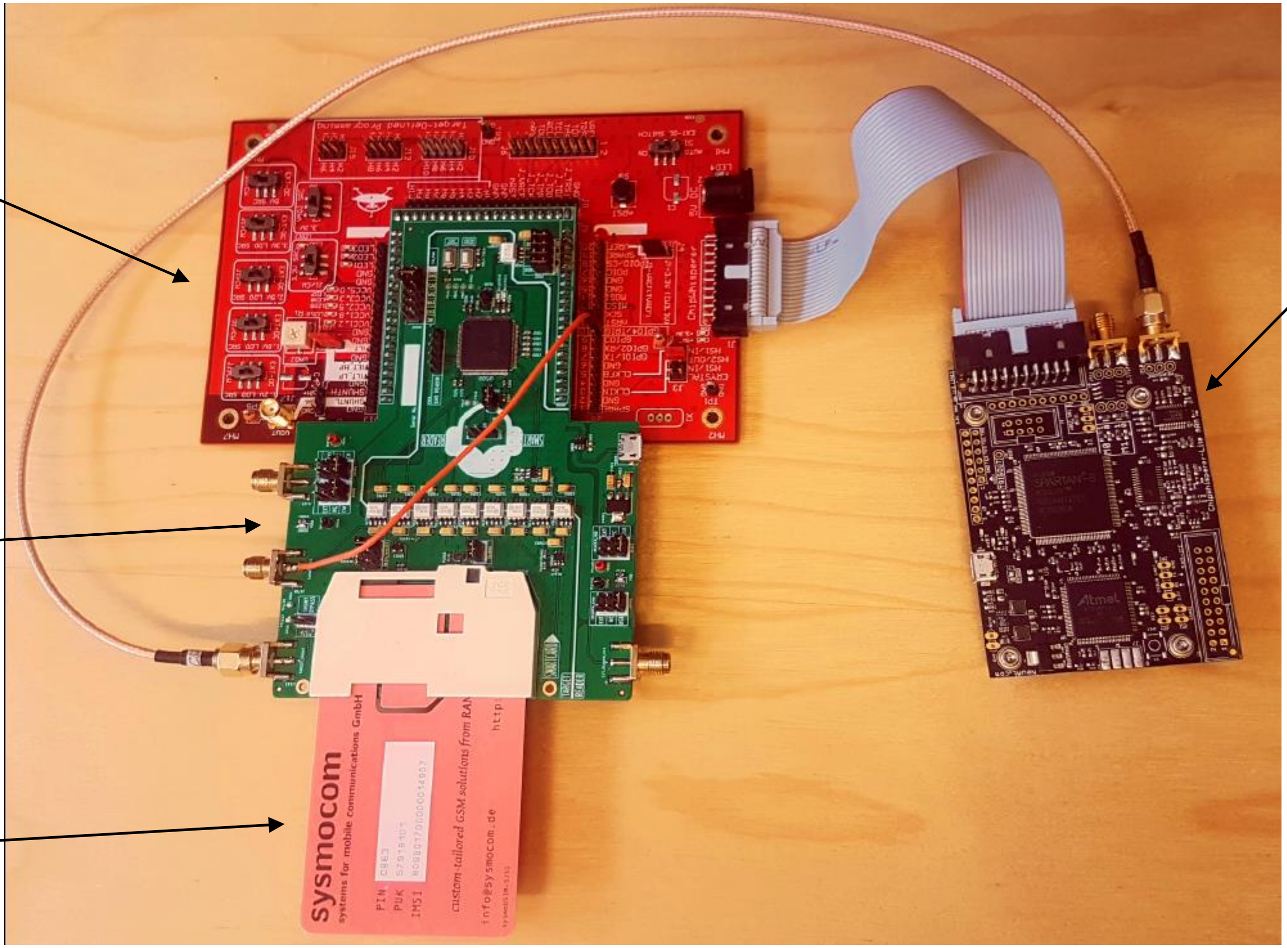
picture credit: Ruize Wang

Measurment setup

CW308 UFO

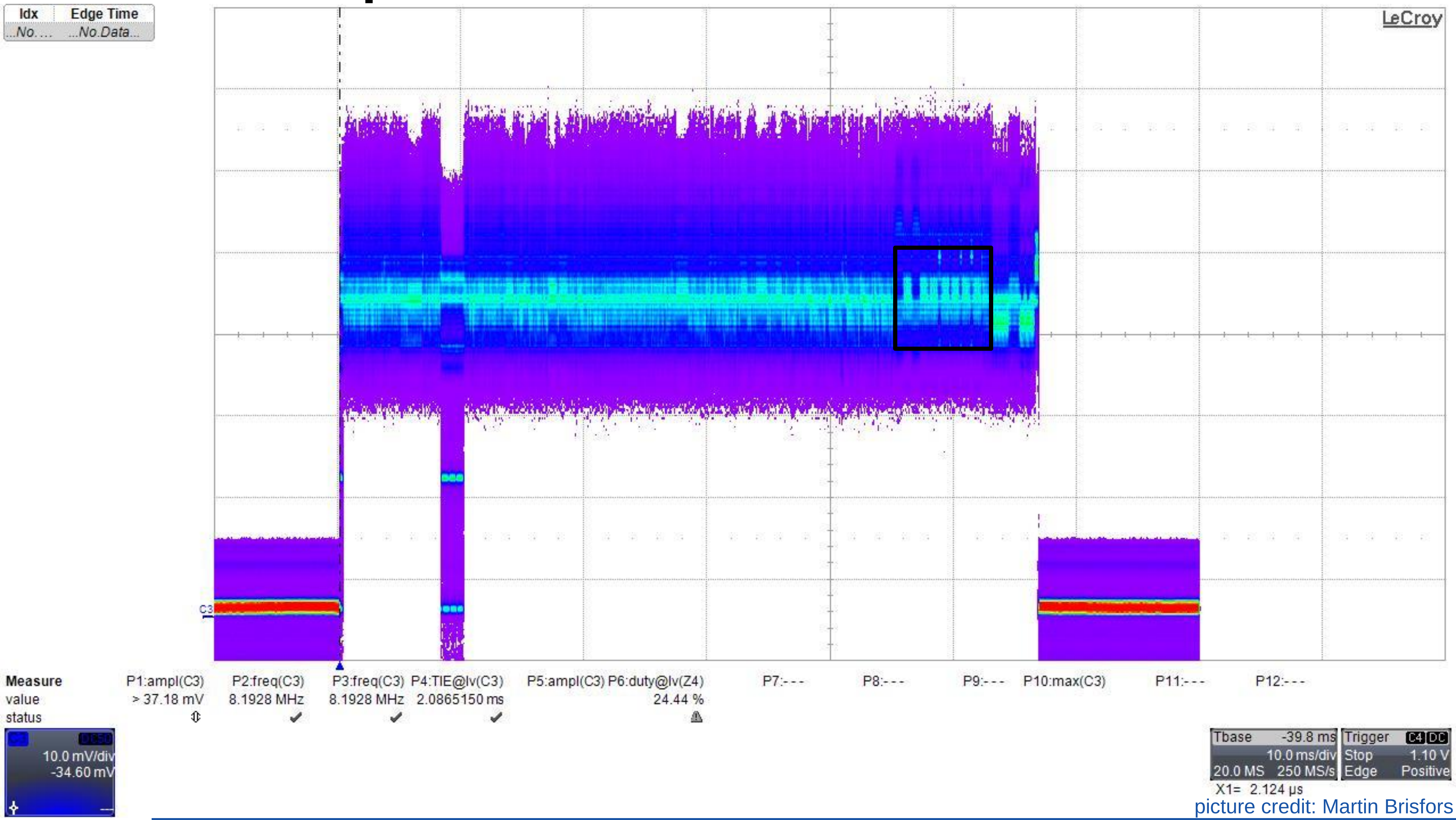
LEIA

USIM



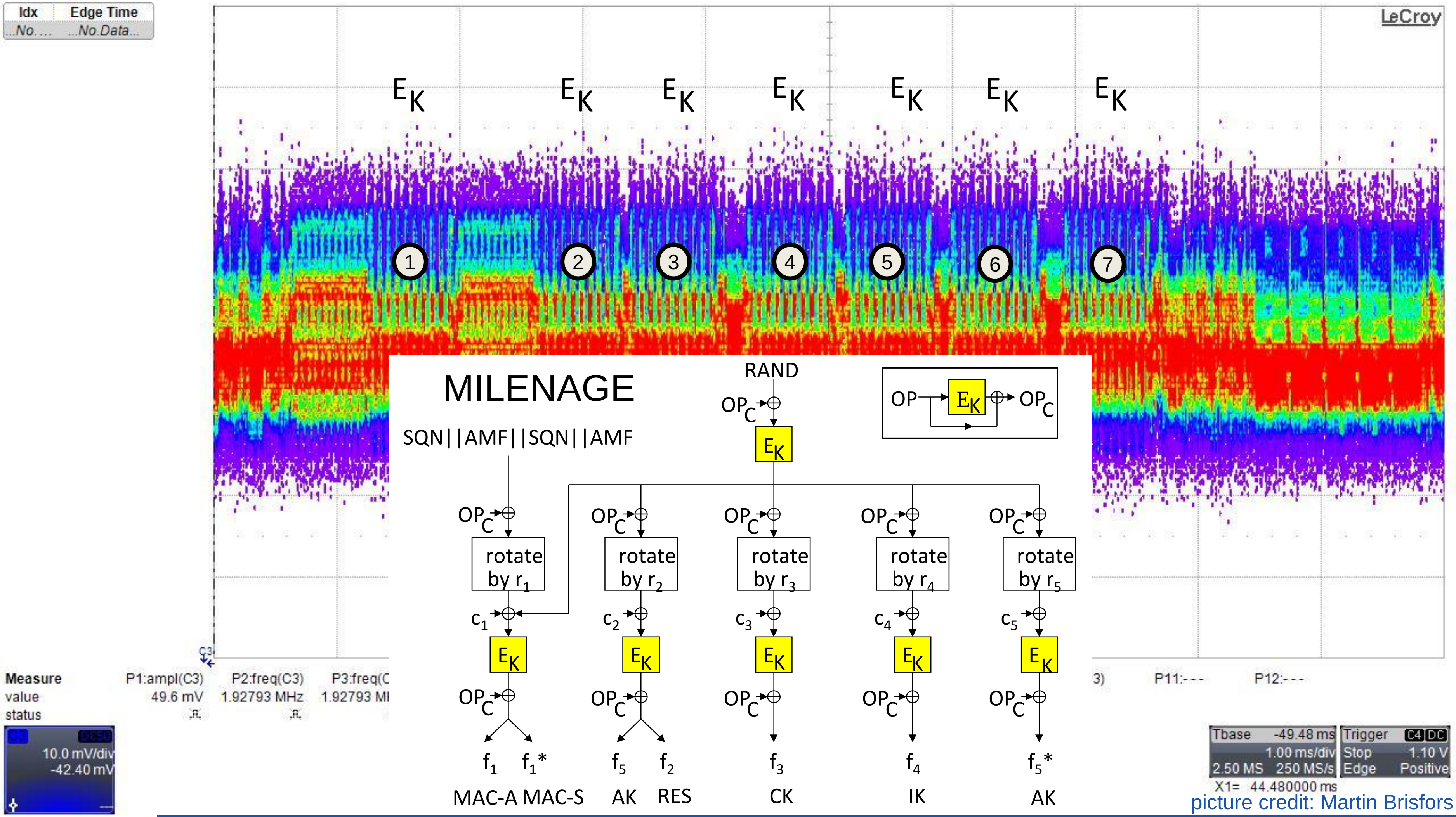
ChipWisperer

USIM power trace for one MILENAGE call



picture credit: Martin Brisfors

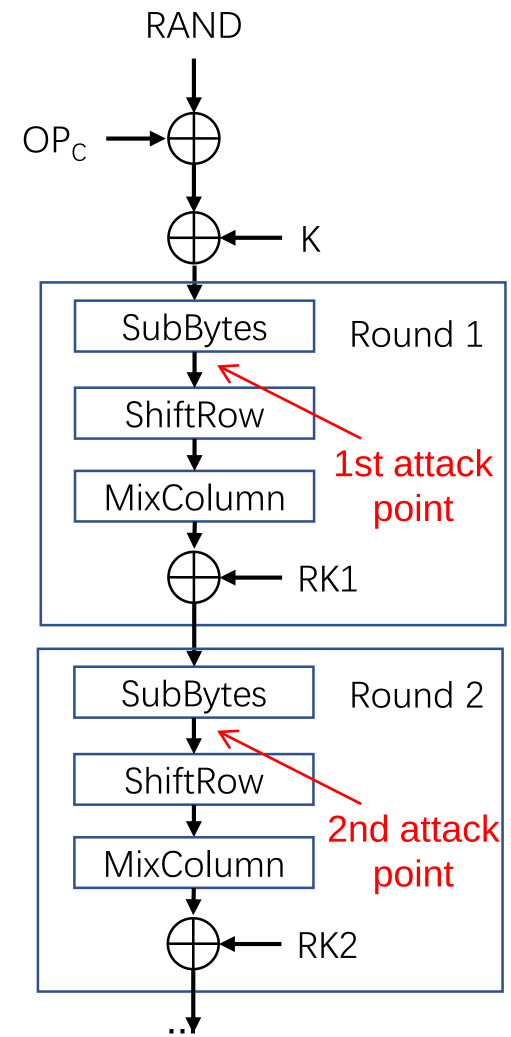
Zoomed interval of MILENAGE execution



picture credit: Martin Brisfors

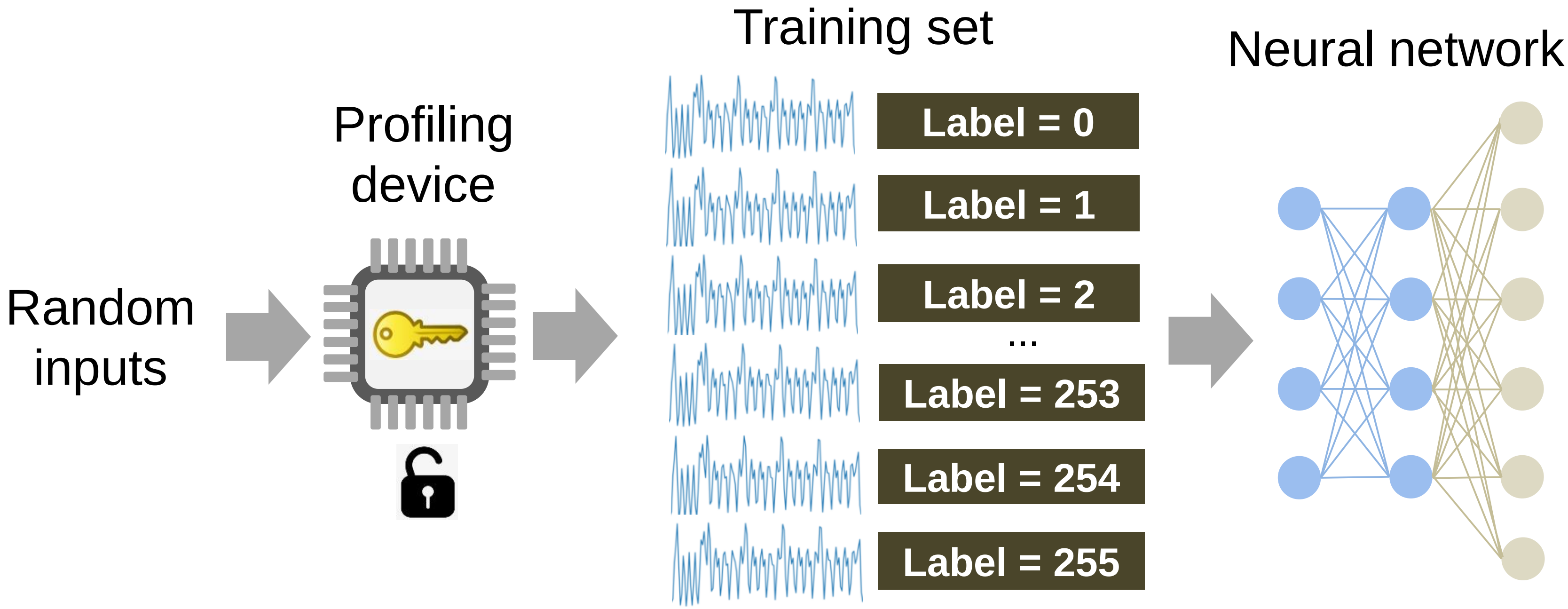
Correlation Power Analysis (CPA)

- In MILENAGE, RAND ⊕ OP_C is first computed and then the result is encrypted
- If E_k is AES-128, the key K can be recovered in two steps:
  1. Recover OP_C ⊕ K by a CPA with S-box output in the first round as the attack point
  2. Recover the 1st round key, RK1, by a CPA with the S-box output in the second round as the attack point
  3. Compute K from RK1
  4. $OP_C = (OP_C \oplus K) \oplus K$



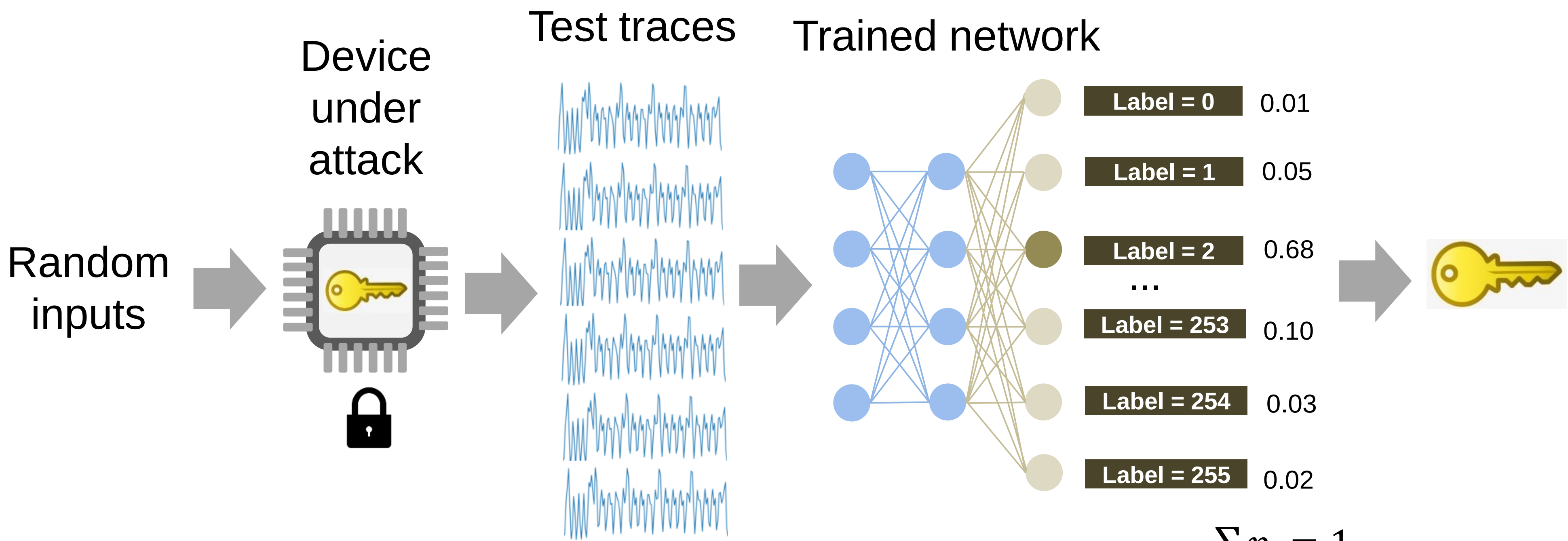
How deep learning is used in power analysis

Profiling stage: Train a neural network using traces from profiling devices



How deep learning is used in power analysis

Attack stage: Use the trained network to classify traces from the device under attack



$$\sum p_i = 1$$

Training setup

- We used Tensorflow with Keras in Python 3.6 for our training code.
- Due to the file size of our training set we trained our network on a high-performance computing system.
- Training using our method could be done on most modern workstations with enough memory.

Development of our neural net design pt. 1

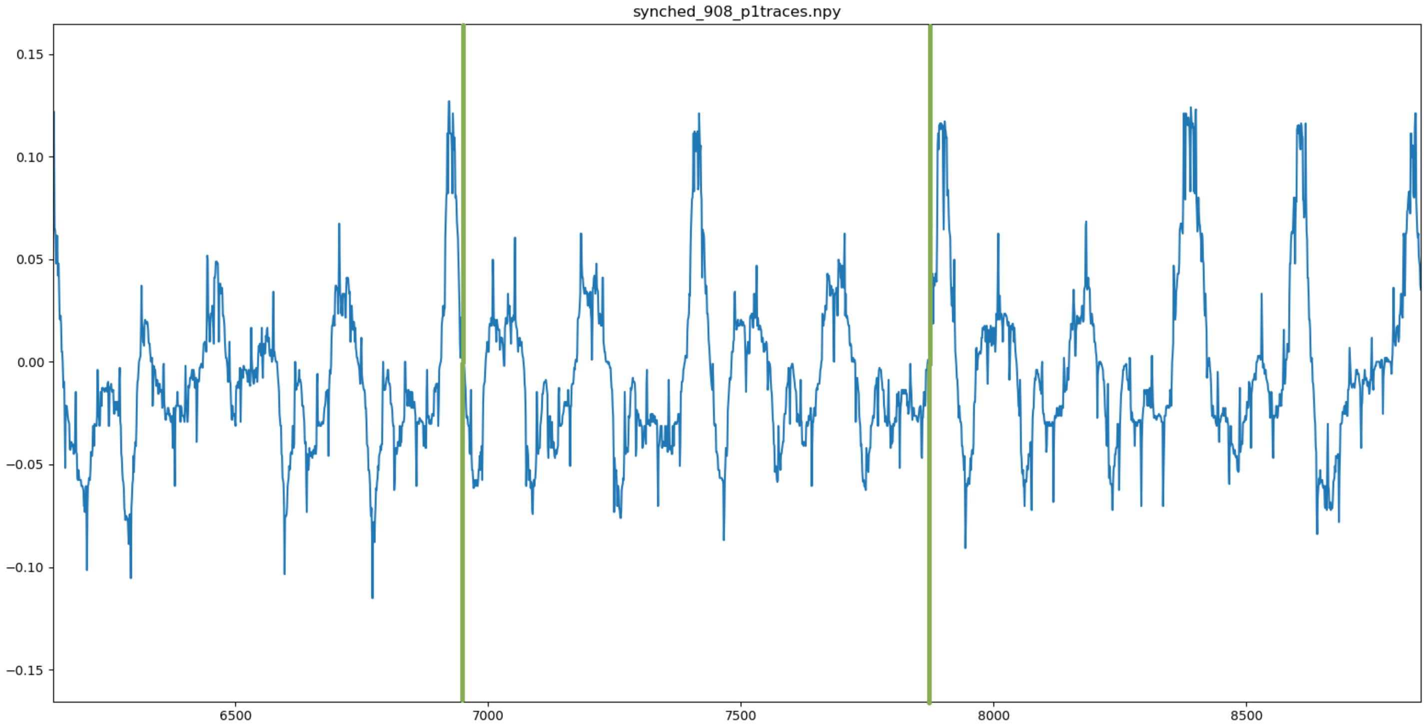
- Our first models were all pure MLP models.
  - Centered on a single block of 4 subkeys.
  - Many were successful in learning to recover subkeys.
  - The improvement compared to CPA was small.
- We trained CNN models centered on 4 subkeys.
  - Performed marginally better than pure MLP models.
  - Early versions lacked specific rationale for the design.
- CNN trained on the entire trace showed a lot of promise.
- We then changed the convolutional layer to be designed with the shape of the traces in mind.

Why train on full round?

- Early testing limited data to center on S-box calculations.
  - Makes the process require more expertise.
- Idea: Let the neural net solve the issue. Use all data in trace.
- Training and testing needs no offset.
- We could potentially train a model requiring no synchronization of traces.
  - Would make the process of attacking a victim card even easier. Could potentially attack the victim real time.
  - Future work.

Development of our neural net design pt. 2

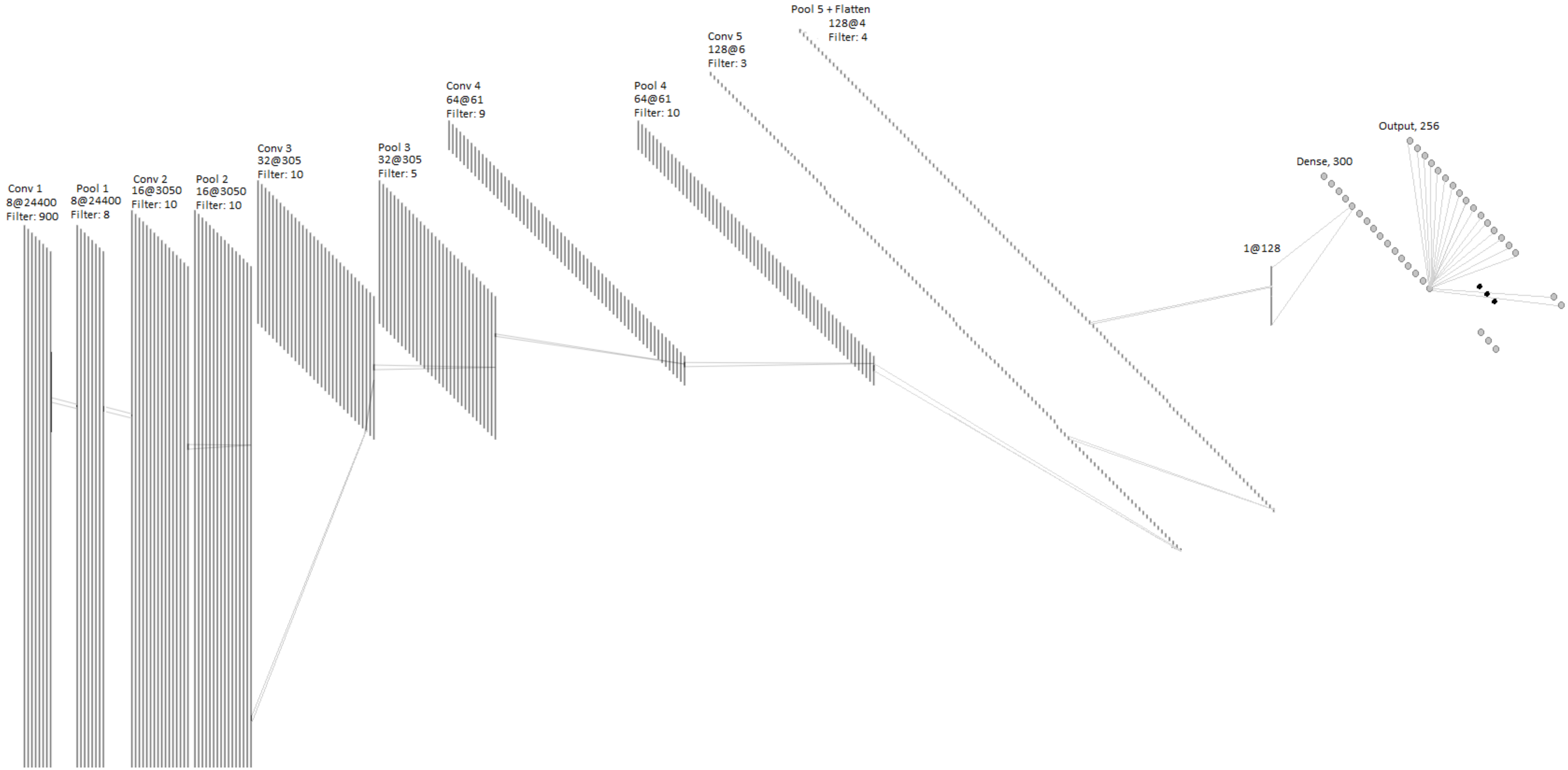
- First convolutional layer uses kernels large enough to contain an entire S-box calculation. We use 900 to be sure.



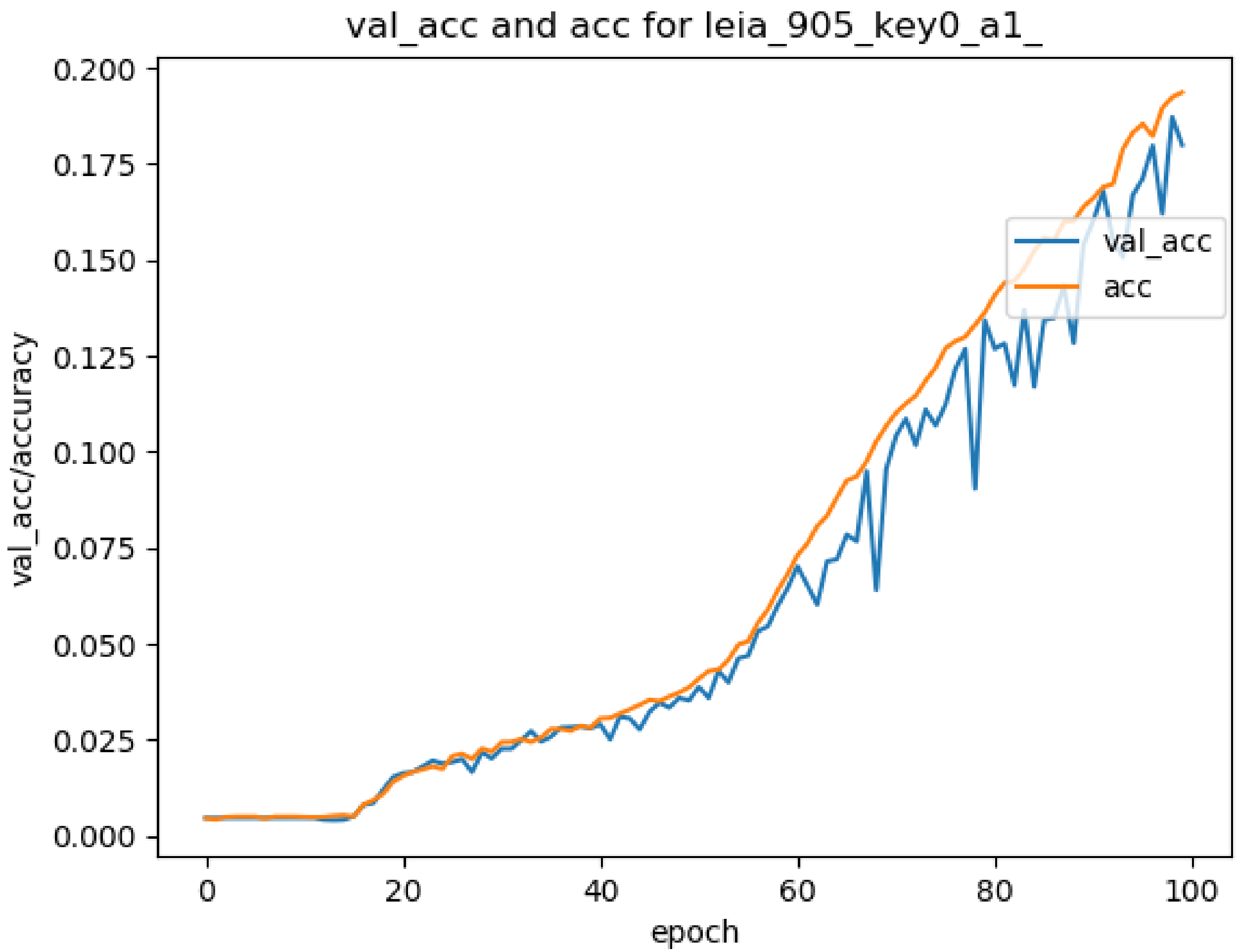
Development of our neural net design pt. 3

- We tried both using no padding and using causal padding.
  - Causal padding seemed to work better.
  - We wanted to try to keep all early convolutions causal to not create an offset between the input and output of convolutions
- We use max pooling and convolutions in 5 layers to convert all information from temporal space to feature space.
- The network presented here uses only a single very wide perceptron layer after the convolutions.
  - Other networks were trained with more depth in the MLP part. Some performed marginally better, but a single hidden layer was more consistent in training successfully.

Development of our neural net design pt. 4



Training pt. 1



Training pt. 2

- To successfully recover the whole key we need to train 32 models.
  - This is because we need 16 models to recover K XOR OPc and then we need 1 model per subkey.
- The total performance will be limited by the worst model.
  - This is why we present models using parameter which most consistently got good results in our testing.
- The network does not always learn. This has to do with the initialization. Fortunately, it is immediately apparent, so training can be reinitialized.
  - There is also an option to retrain a successful CNN model.
  -

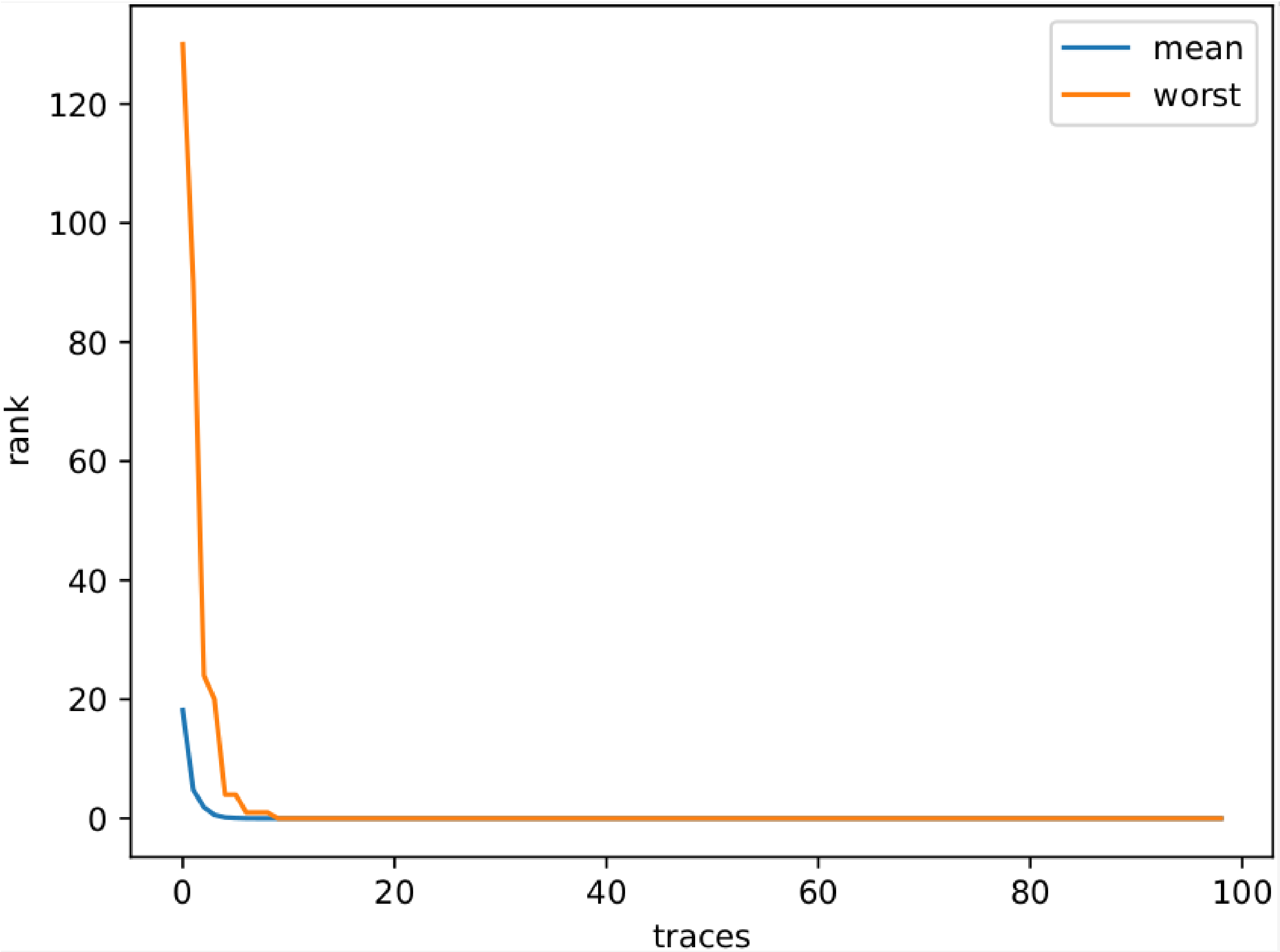
Training pt. 3

- Since we already know the key and OPc for our USIM cards we decided to limit our scope to training 3 models.
- We have 2 models recovering different round 1 bytes, and 1 model recovering a round 2 byte.
  - Specifically, byte 0 and byte 5 of round 1, byte 0 of round 2.
- This is enough to serve as a proof of concept. Previous research on the topic indicates there likely isn’t a significant difference in recovering different bytes.
  - We intentionally chose 2 bytes from different computations for round 1.
  - If bytes of round 1 are correctly predicted, the bytes of round 2 are separate problems.

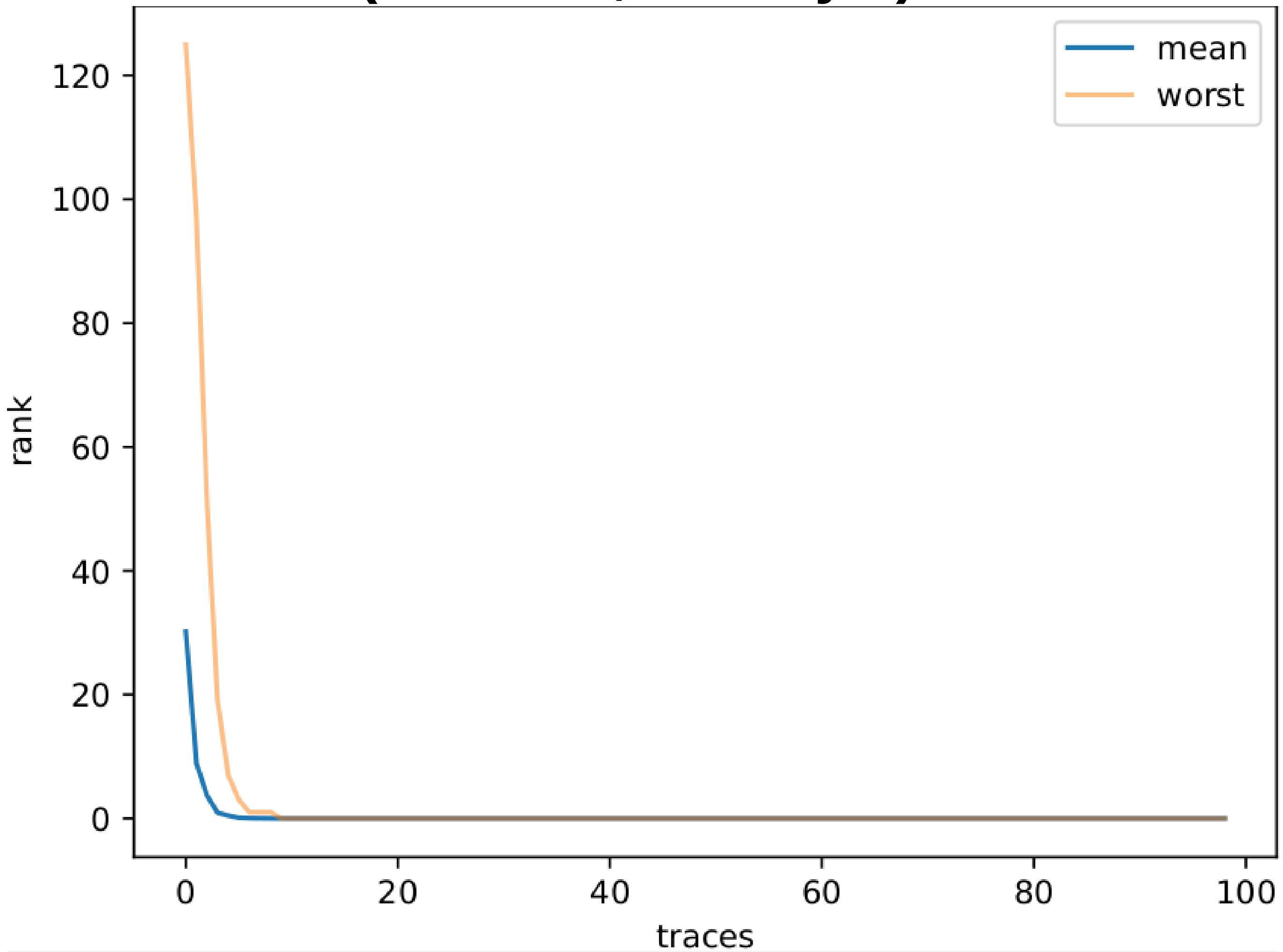
Testing

- Testing was done using the open-source software tool for deep learning side channel analysis called DLSCA.
- The average rank test was used to evaluate model performance. It calculates the average number of guesses which are more likely than the correct one.
- Once the average rank is 0, every iteration of the test has successfully recovered the subkey.
  - The offset should be set to 0 in the test code.
- The tool stores the raw data of the rank progression as well. We used this to calculate the expected number of traces needed to recover the key.

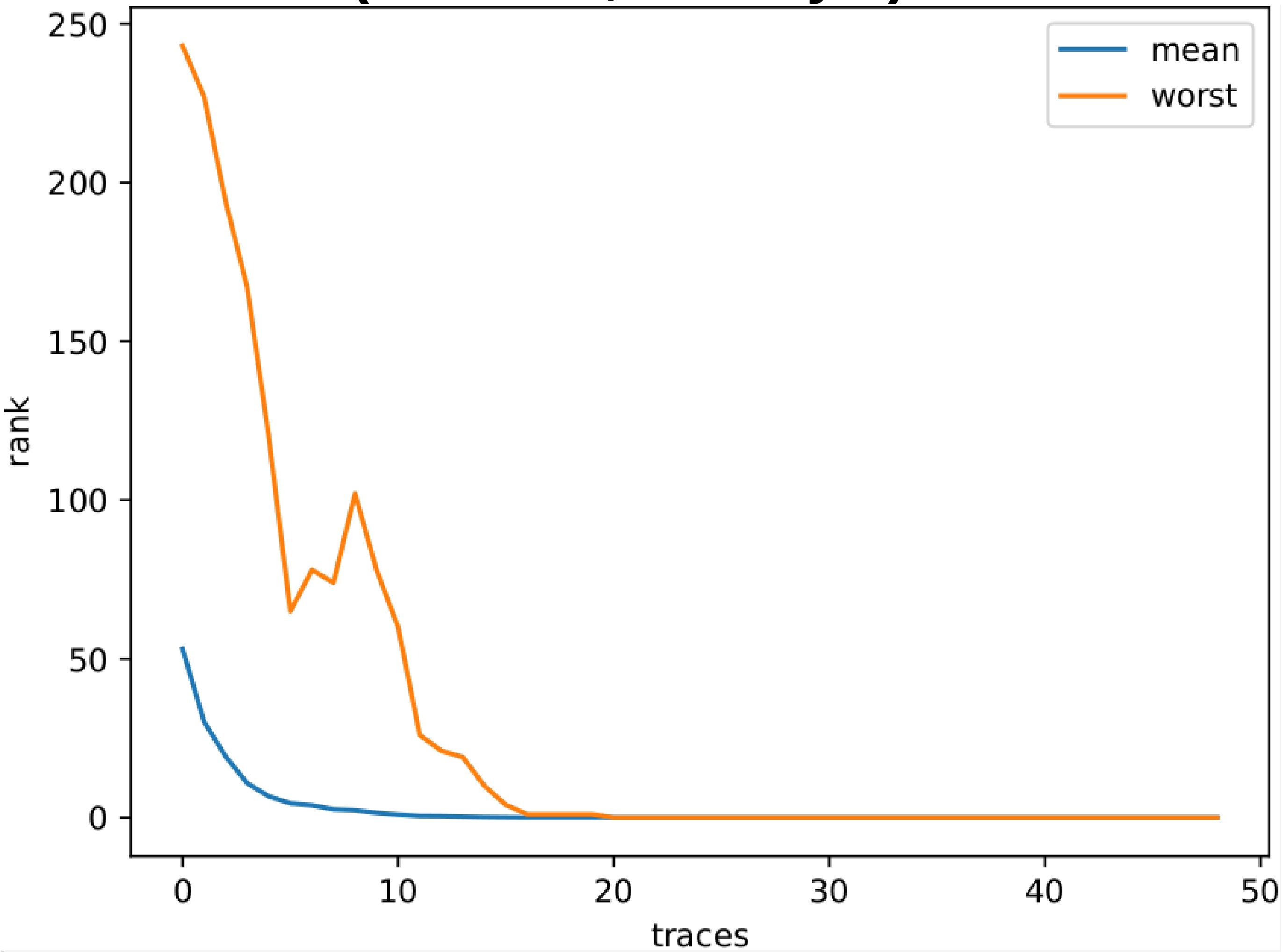
Results (Round 1, subkey 0)



Results (Round 1, subkey 5)



Results (Round 2, subkey 0)



Conclusions pt. 1

- The difficulties associated with performing a SCA on USIM are two-fold:
  - Properly measuring the side channel.
  - Analyzing the measured data to extract hidden information.
- Tools like ChipWhisperer and the LEIA board have made the issue of measurements easier.
  - For capturing traces from other USIM of the same brand it is sometimes as easy as running a capture script.
- Our work partially addresses the second issue. As the demo will show, if someone gives you a pre-trained model for the same brand of USIM, the attack is trivially easy.

Conclusions pt. 2

- Our proposed method also requires an order of magnitude fewer traces to be captured for the attack step compared to previously published research.
  - This makes non-synthetic attack scenarios more likely.
  - If future work can improve the result by another order of magnitude, a single measurement attack may be possible.
- There is likely still room for significant improvements.

Conclusions regarding viability and price

- Equipment cost to perform attack is very low.
  - ChipWhisperer Lite: 250 USD
  - ChipWhisperer UFO board: 240 USD
  - LEIA manufactured in China: 3000 RMB
  - Total <1000 USD
- A third-party malicious actor could train a model and sell. With better generalization it might not be limited to one brand.
- If trace capture was made easier, such as with pattern recognition, an attacker would no longer need any specialized knowledge or skills.
- ML based SCA poses a real threat.

Demo

- Video demo showing steps needed to capture traces, confirm attack point, followed by a test analogous to how an attack would be performed.
- The test only uses the known key to measure performance. This helps us understand how many traces an attacker would need to reach convergence.