Google

A hacker guide to deep-learning based AES side channel attacks

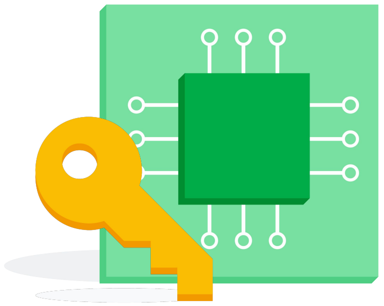

Elie Bursztein
Google, @elie


Jean-Michel Picod
Google, @jmichel_p

with the help of **many** Googlers and external collaborators

Security and Privacy Group

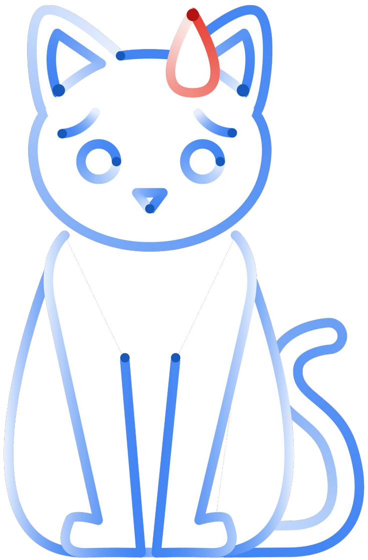
Side channel attacks are one of the most efficient ways to attack secure hardware



A side-channel attack was used to recover the Trezor bitcoin wallet private key

Google

https://jochen-hoenicke.de/crypto/trezor-power-analysis/



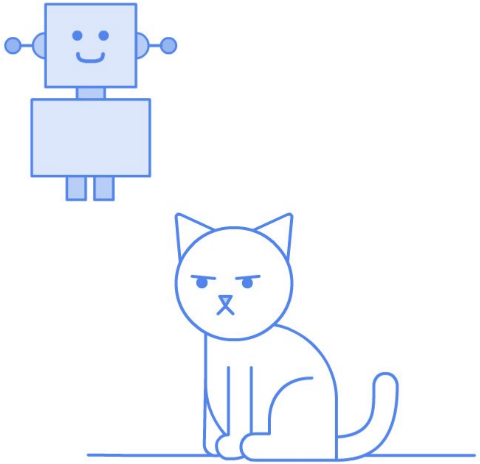
Side-channels attacks **requires a lot of domain expertise** and are **implementation specific**

Is there a better and more generic way to perform side-channels attacks?

Deep-learning is posed to revolutionize hardware side-channel cryptanalysis

Google

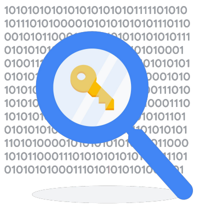
Security and Privacy Group



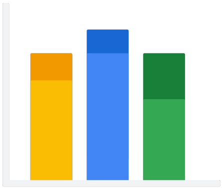
AI? Really?

Template attack on steroids

No trace processing



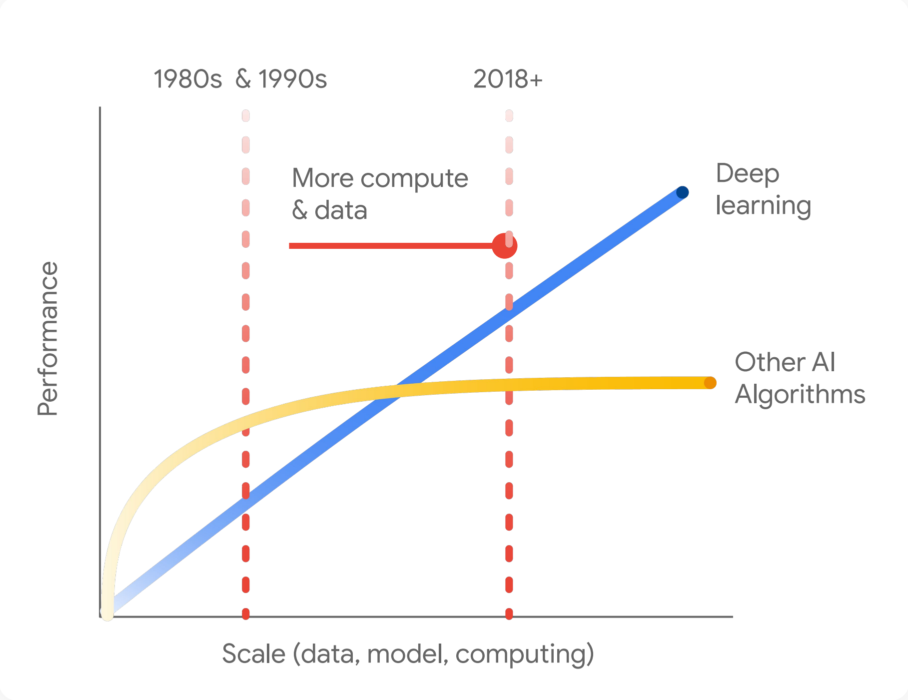
Direct attack point targeting

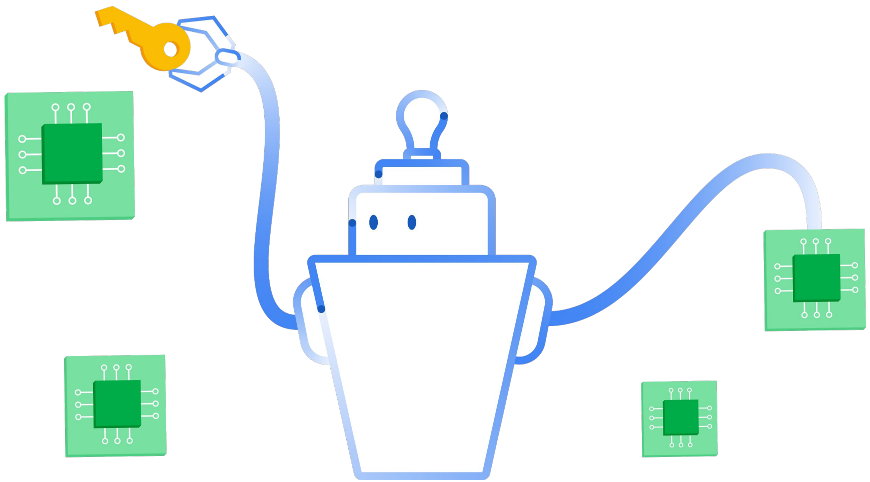


Efficient probabilistic attack

Better and intuitive success metrics

Attacks are going to be better over time as deep learning scales with data and computing

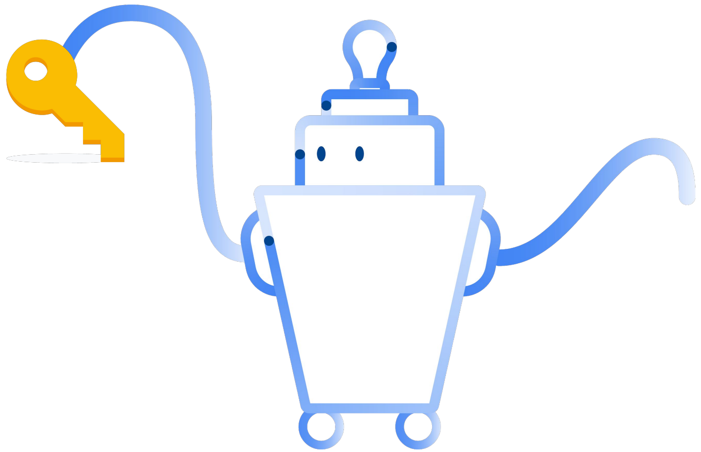




How to use deep-learning to recover AES keys baked in hardware **in practice**

Google

Security and Privacy Group



Side Channel Attacks Assisted with Machine Learning

Google

Security and Privacy Group

Talk is based on some of the results of a joint research project with many collaborators on **hardening hardware cryptography**

Google

Security and Privacy Group

Code and slides

https://elie.net/scaaml

Disclaimer

This talk purposely focuses on showcasing how to a get SCAAML attack working end-to-end rather than discussing state of the art attacks.

Agenda

- What are side-channels?
- What is deep-learning?
- Hacker's guide to AES SCAAML attacks
- What's next

What are side-channels?

Google

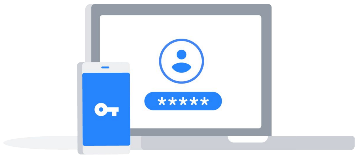
Security and Privacy Group

A side-channel attack is an indirect measurement of a computation result via an auxiliary mechanism

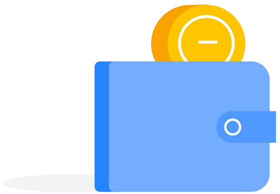
SCA real-world applications

Recover Encryption keys

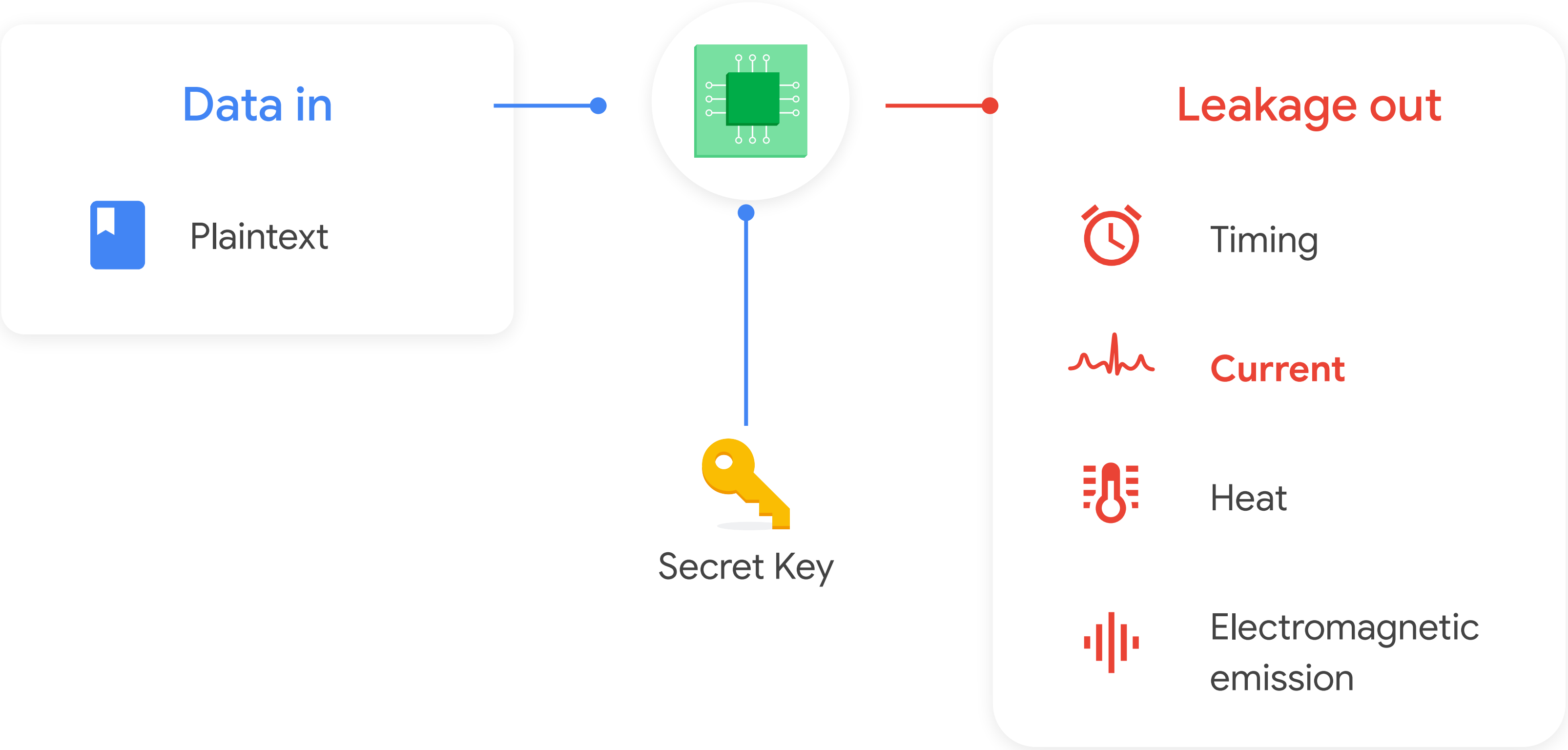
Perform blind SQL injections



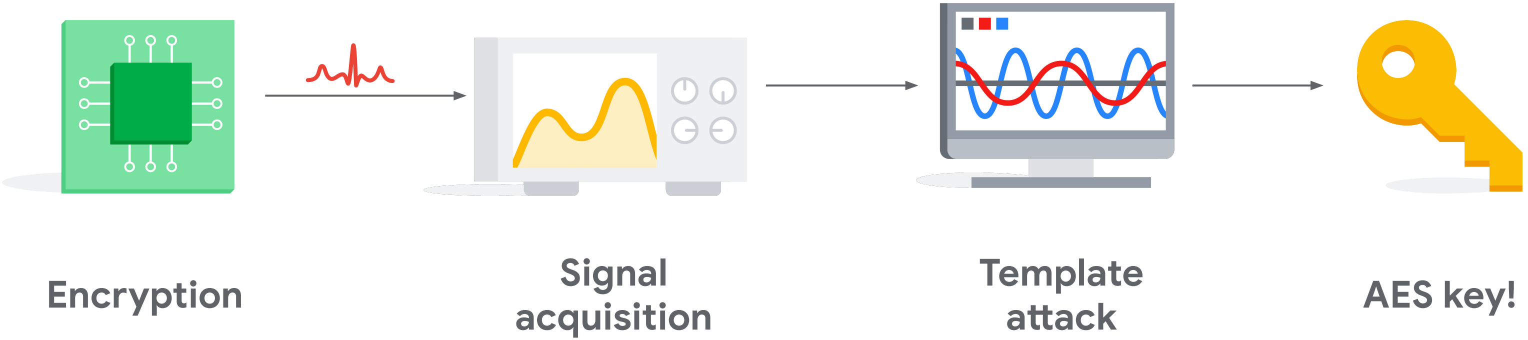
Steal passwords & pins



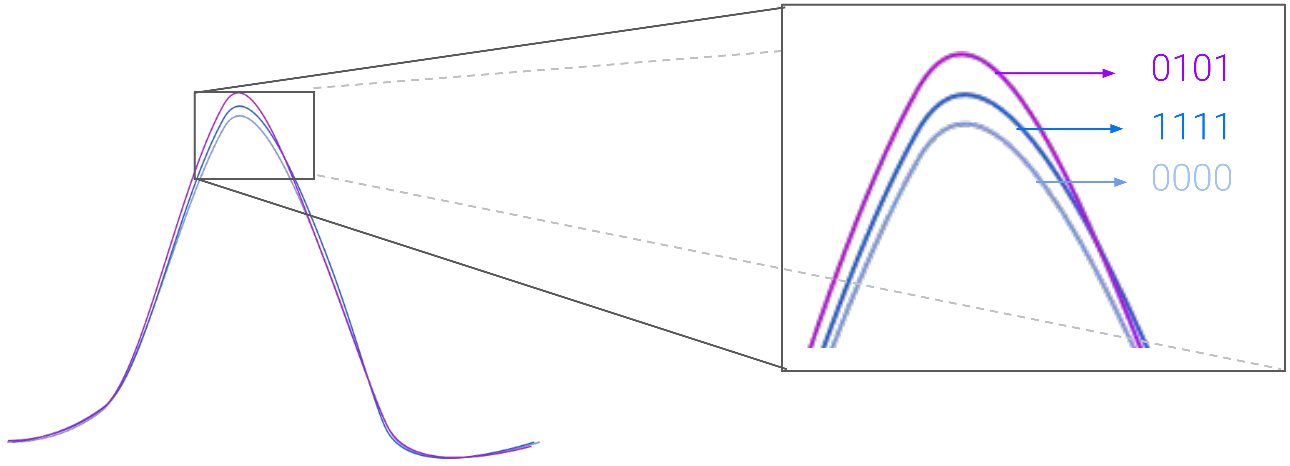
Extract cryptowallet private keys


Data in
Plaintext
Secret Key
Leakage out
Timing
Current
Heat
Electromagnetic emission

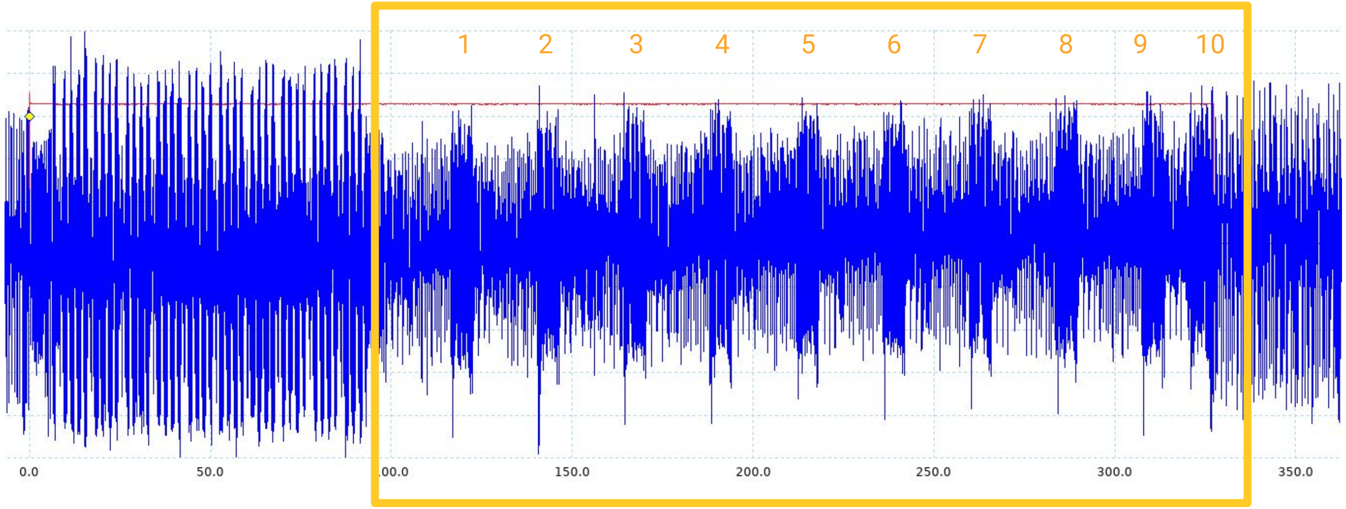
SCA in a nutshell



Crypto computation side-effects are measurable



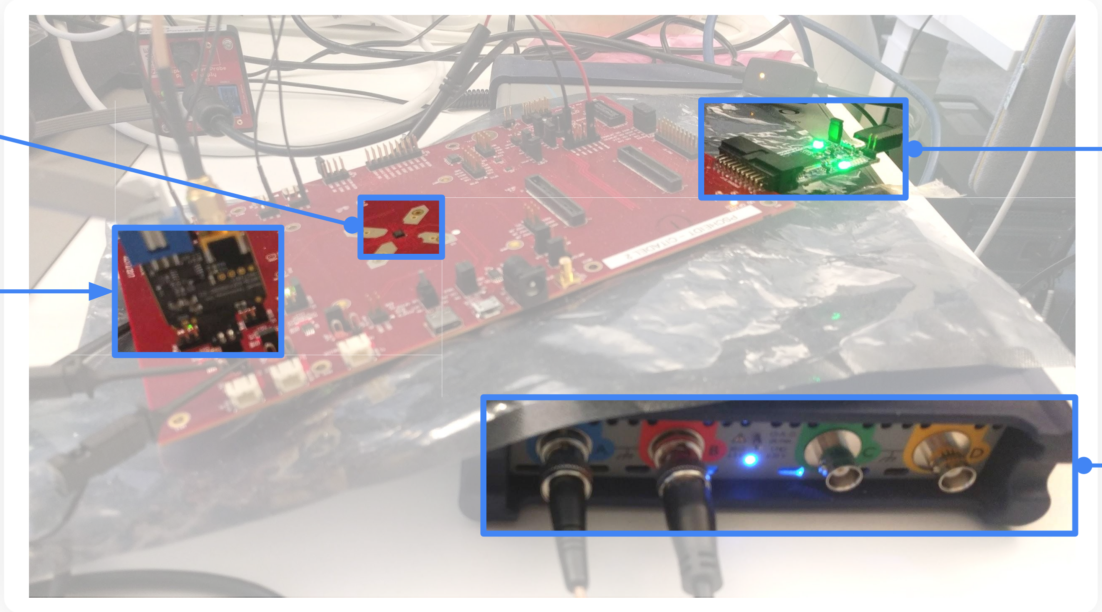
Lightly protected AES power trace



Google

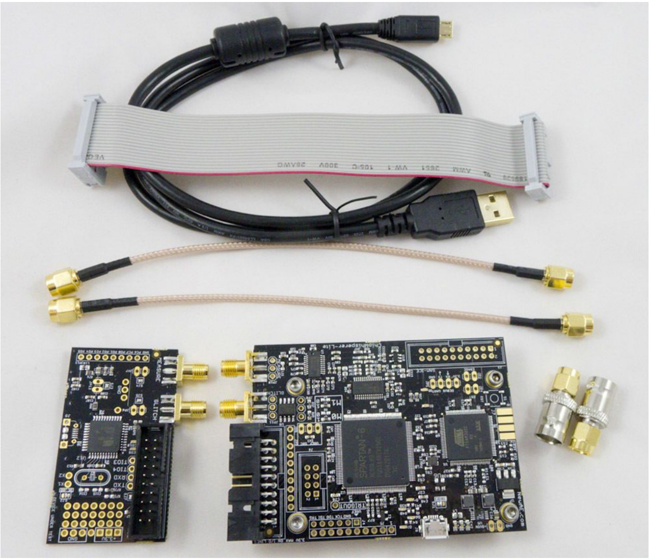
Security and Privacy Group

DIY hardware setup from early days

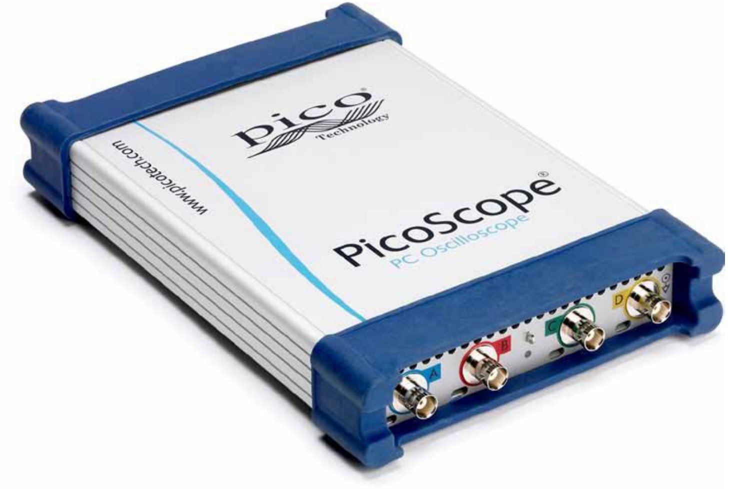


NewAE Chipwhisperer is an easy and cheap all-in-one entry to side-channel attacks

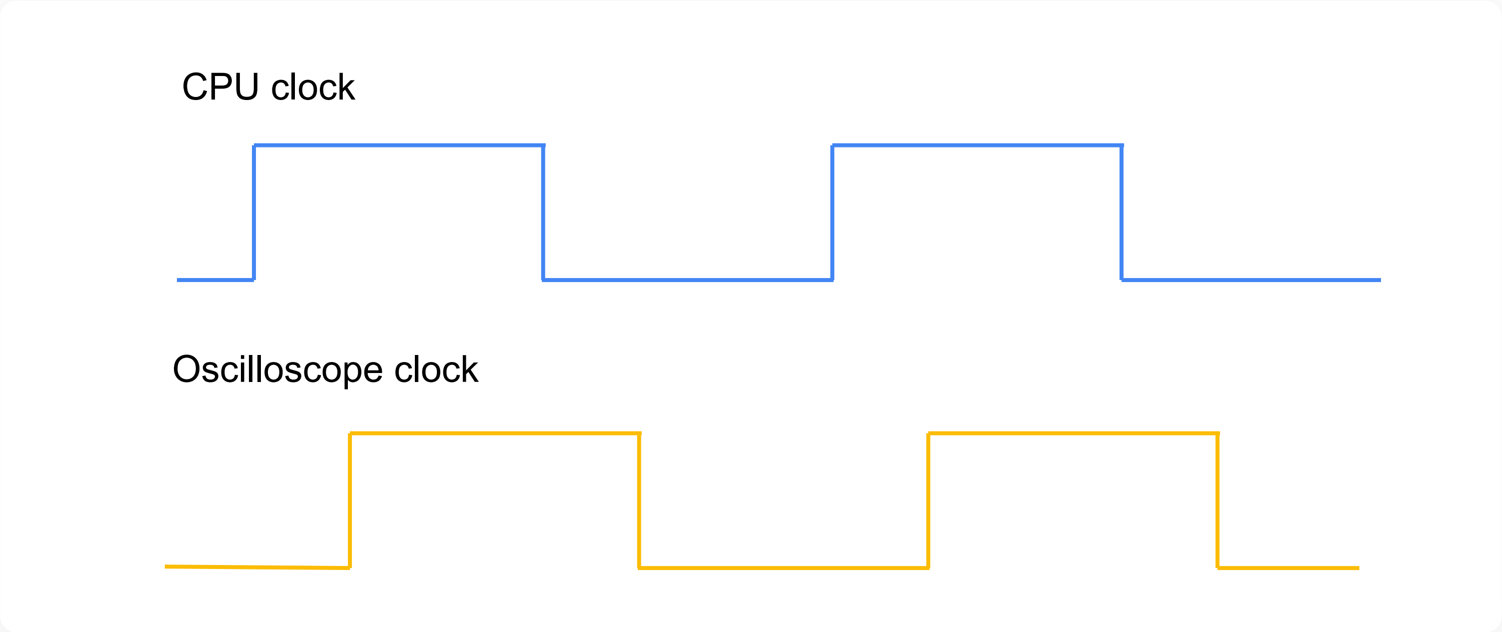
https://newae.com/tools/chipwhisperer/



Recommendation based on usage



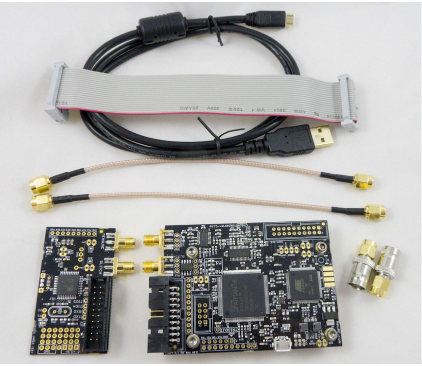
For many chips a higher sampling rate is needed due to their clock speed so you need a faster oscilloscope

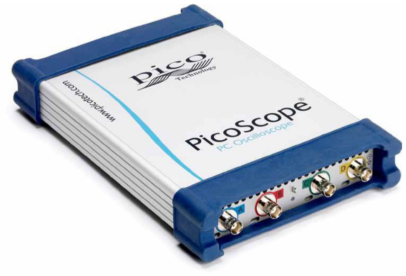


Asynchronous capture used for blackbox attacks like SCAAML needs **at least 4x the CPU clock speed**

NewAE Chipwhisperer Pro + Picoscope 6000 is what we use for our SCA research

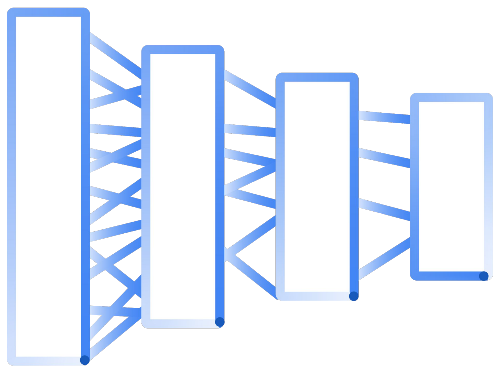
This is not an ad :) it is a recommendation
based on what we use

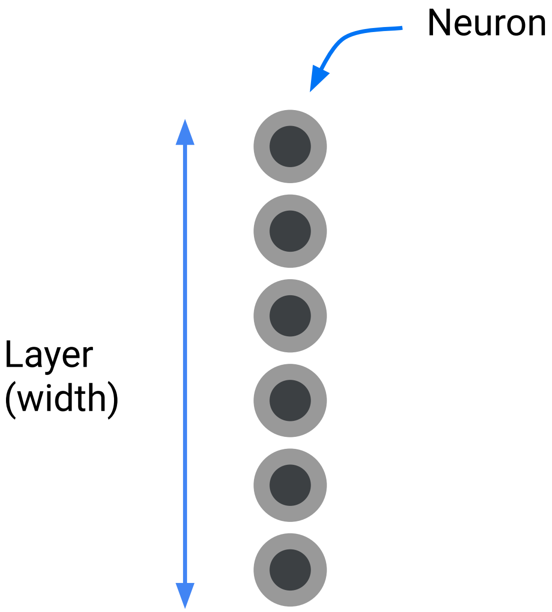


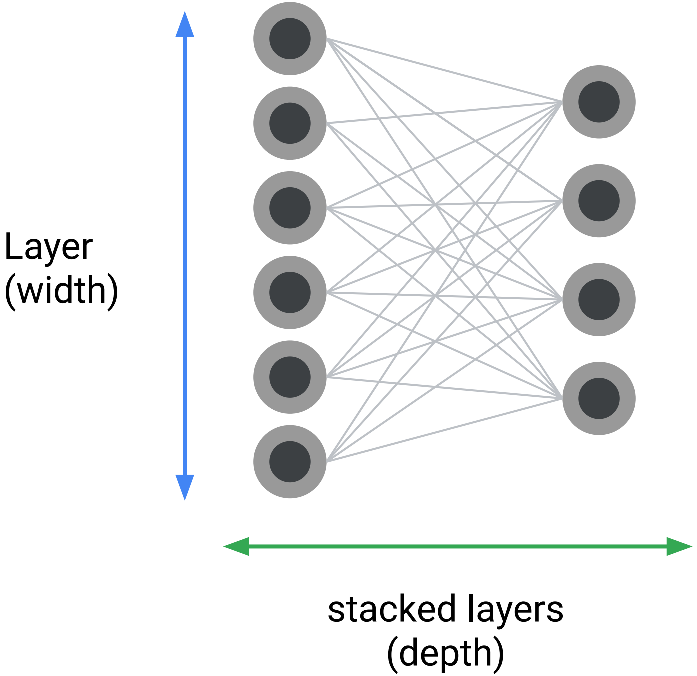


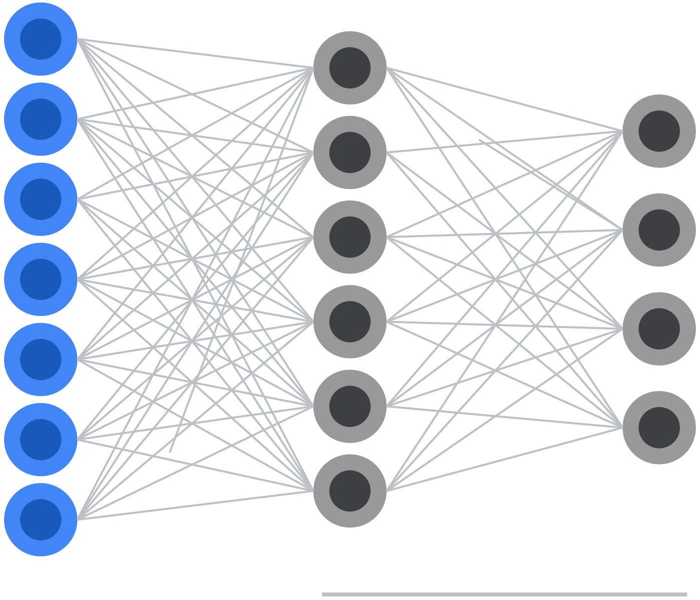
What is deep-learning?

At its core deep-learning is basically a neural network with many layers



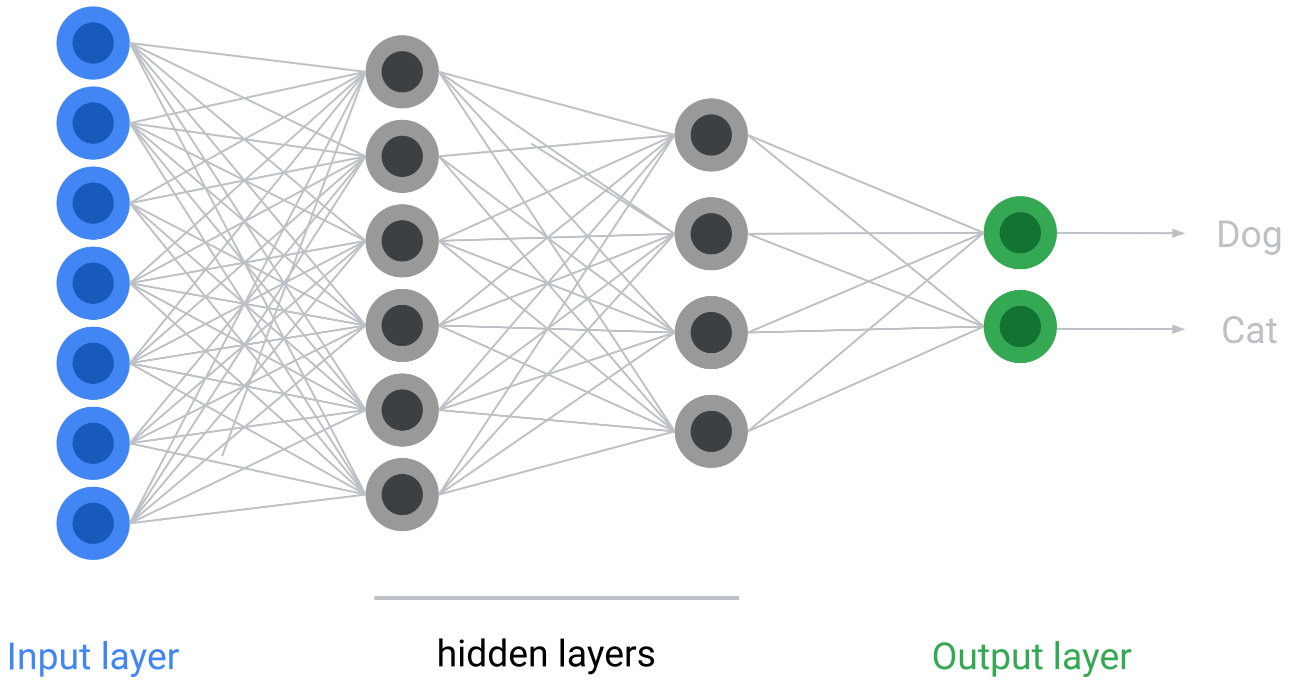

Neuron
Layer
(width)


Layer
(width)
stacked layers
(depth)

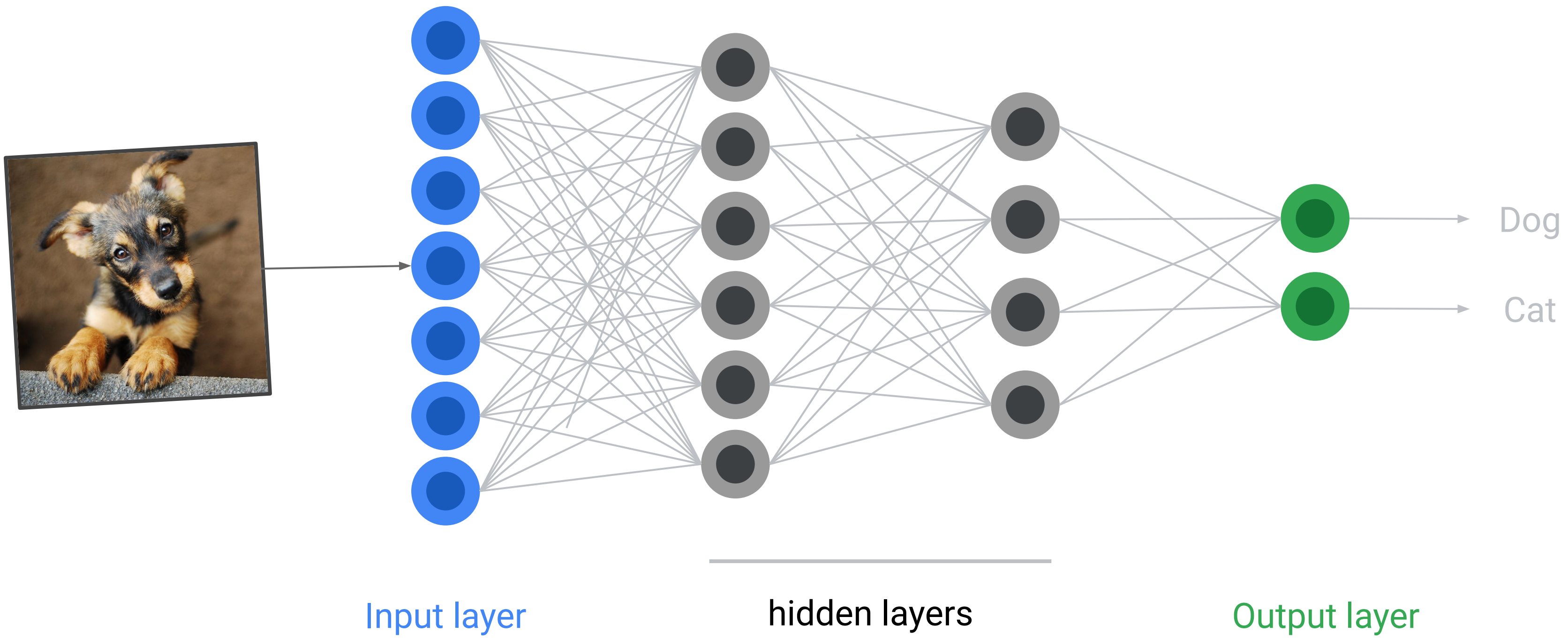


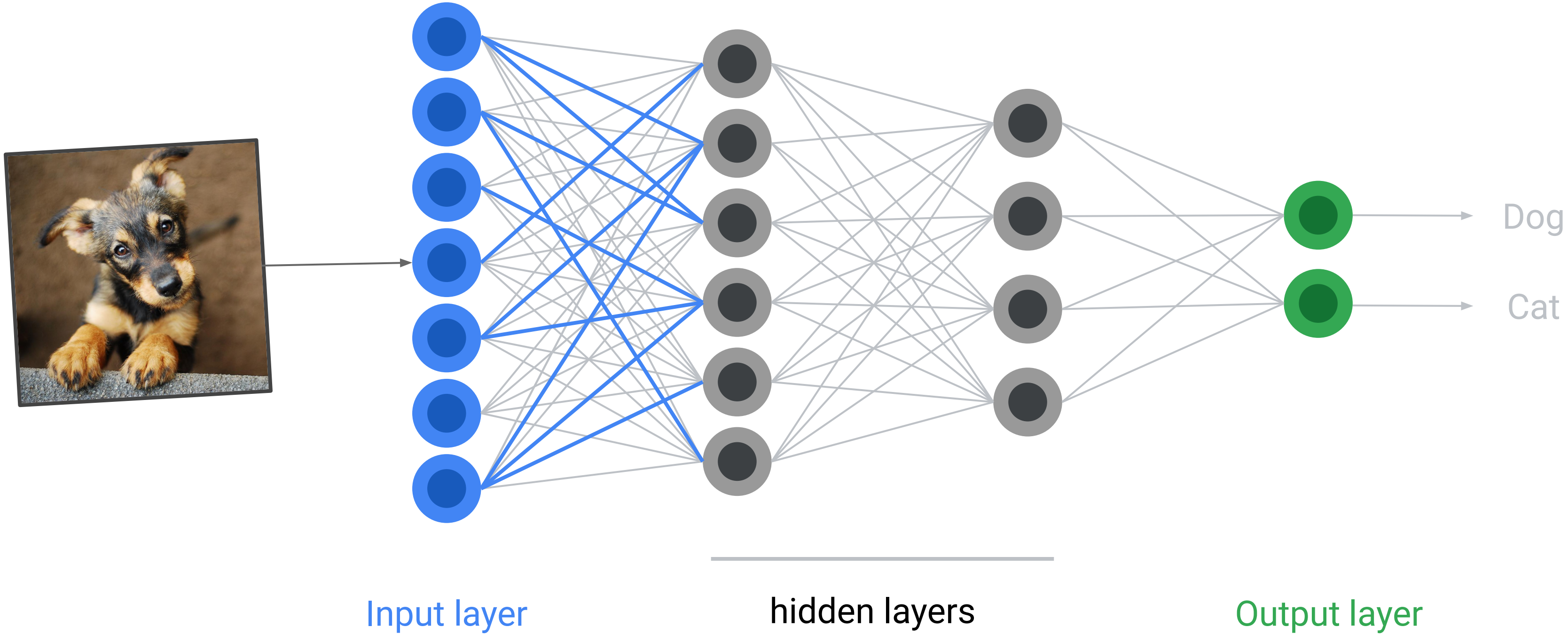
Input layer

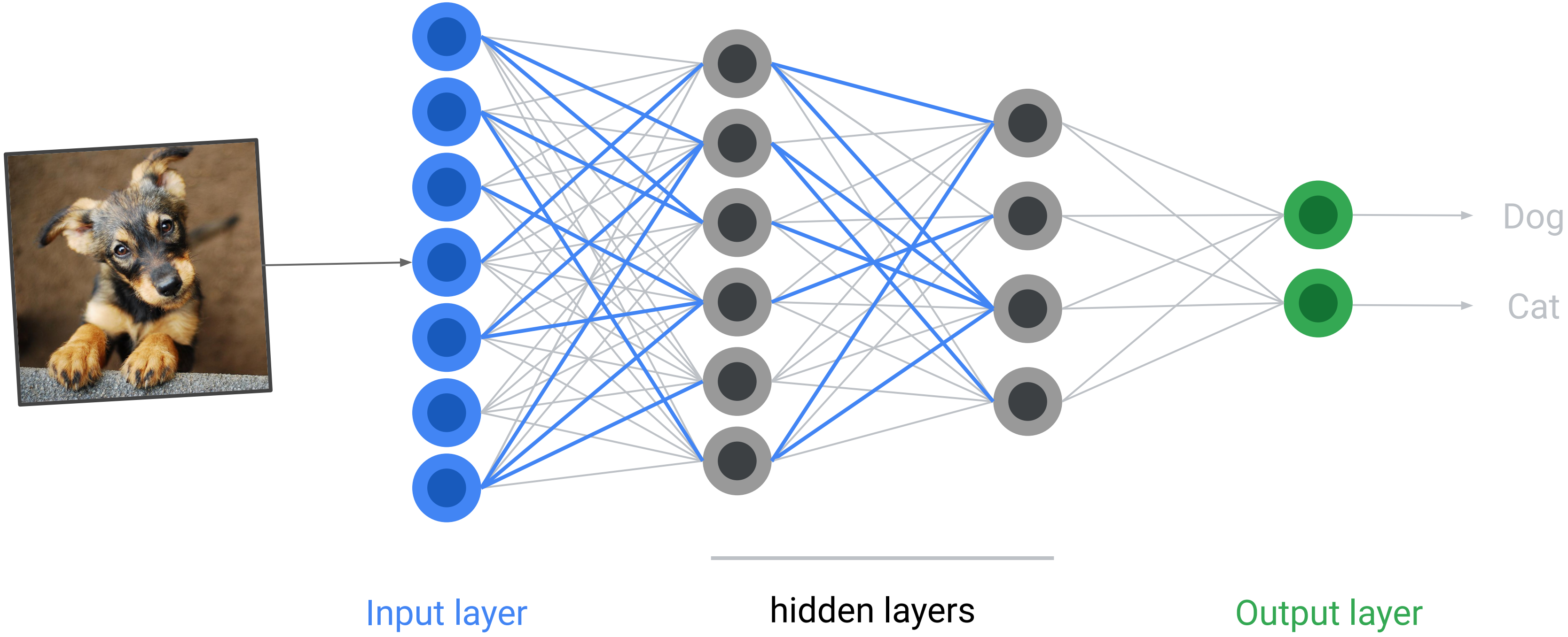
hidden layers

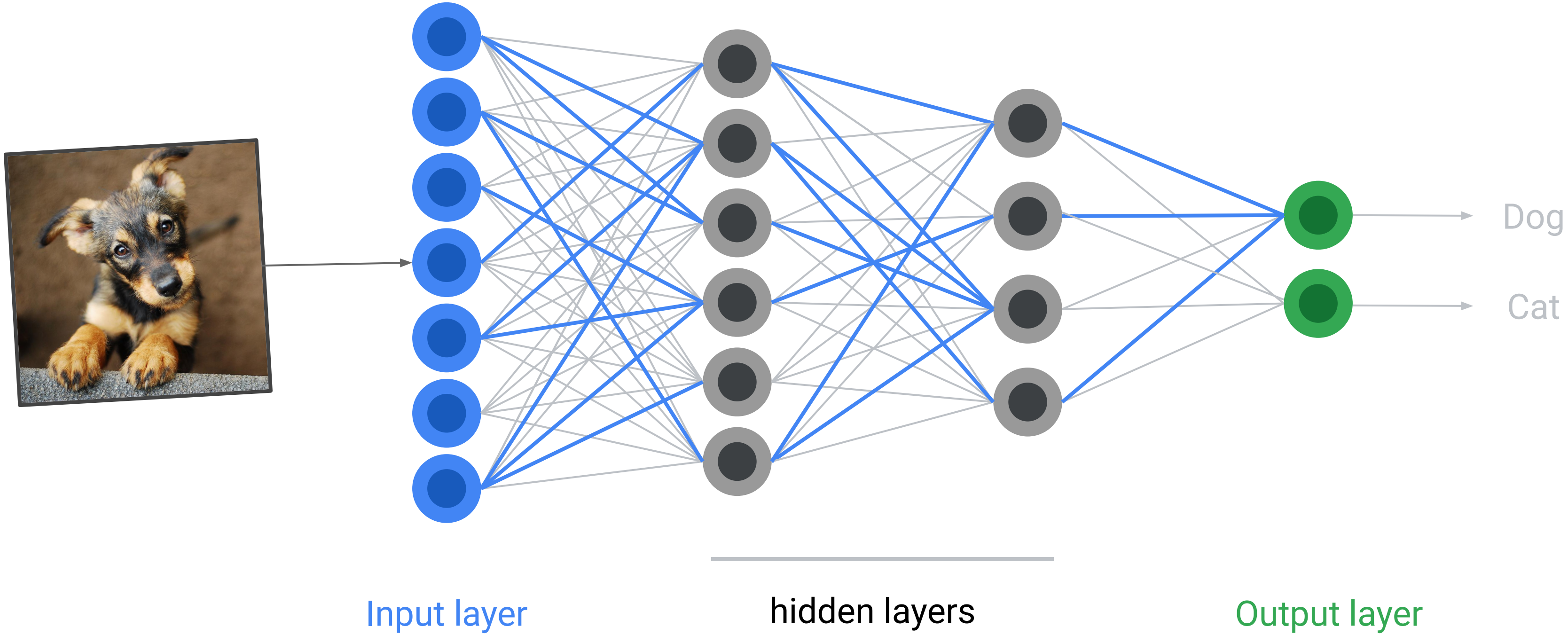


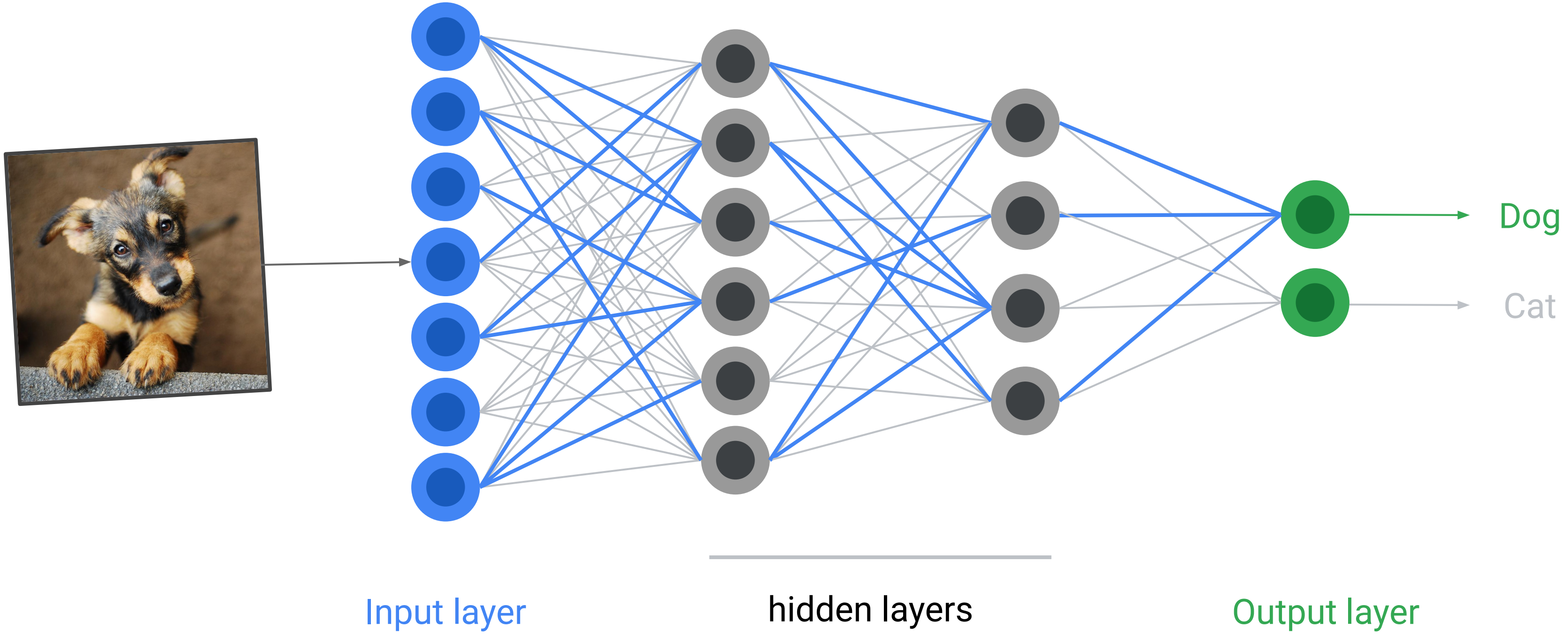
Dog
Cat
Input layer
hidden layers
Output layer


Dog
Cat
Input layer
hidden layers
Output layer

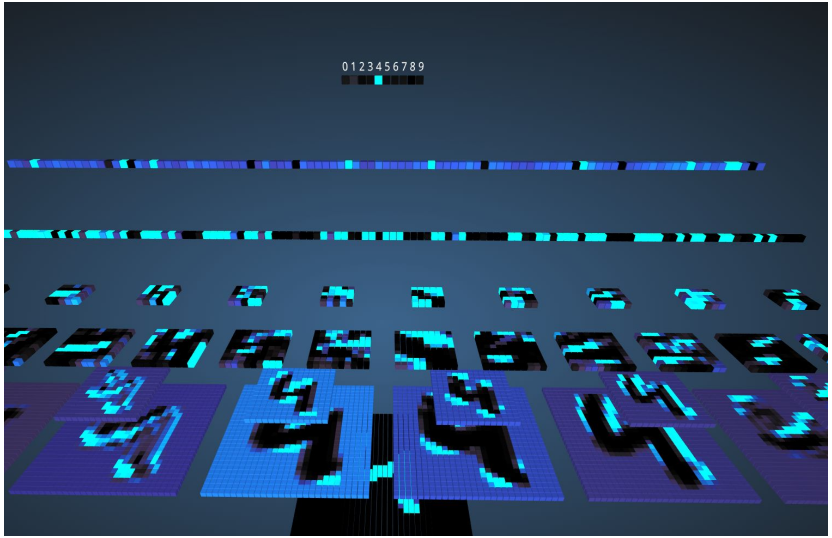

Dog
Cat
Input layer
hidden layers
Output layer


Dog
Cat
Input layer
hidden layers
Output layer


Dog
Cat
Input layer
hidden layers
Output layer


Dog
Cat
Input layer
hidden layers
Output layer

Different use-cases use different types of layers and network architectures



Handwritten digit recognition visualization from
http://scs.ryerson.ca/~aharley/vis/conv/

What do I need to train deep learning models?

Tensorflow to write and train your model

https://www.tensorflow.org/

TPU v2

You need a **hardware accelerator (GPU or TPU)** as training on CPU is impossibly slow

Demo code is available on Colab: a hosted Python notebook with Tensorflow and free GPU/TPU time

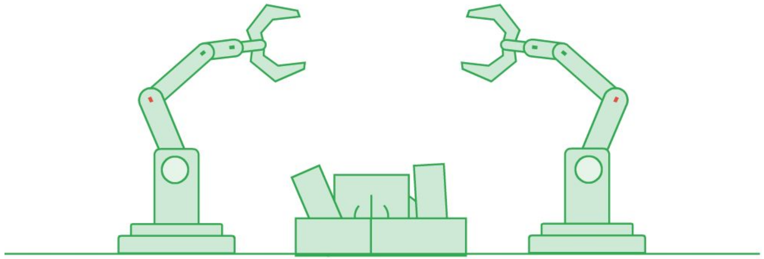
https://colab.research.google.com/

Hacker guide to SCAAML attacks

Google

Security and Privacy Group

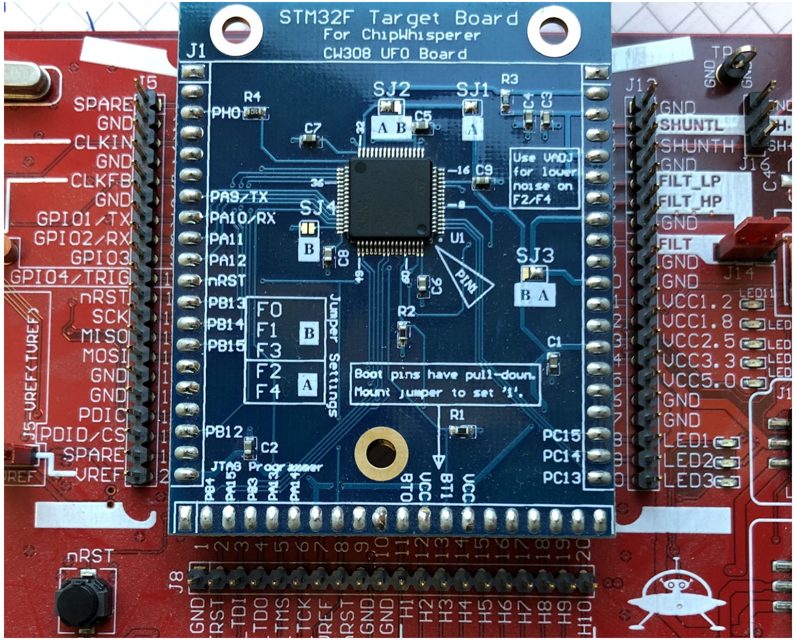
Performing a SCAAML attack **step by step**



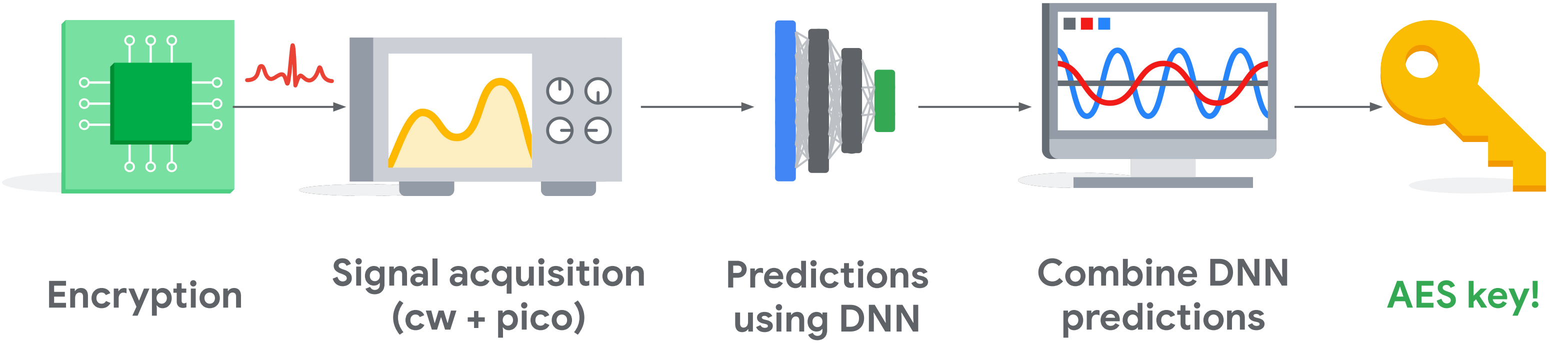
Google

Security and Privacy Group

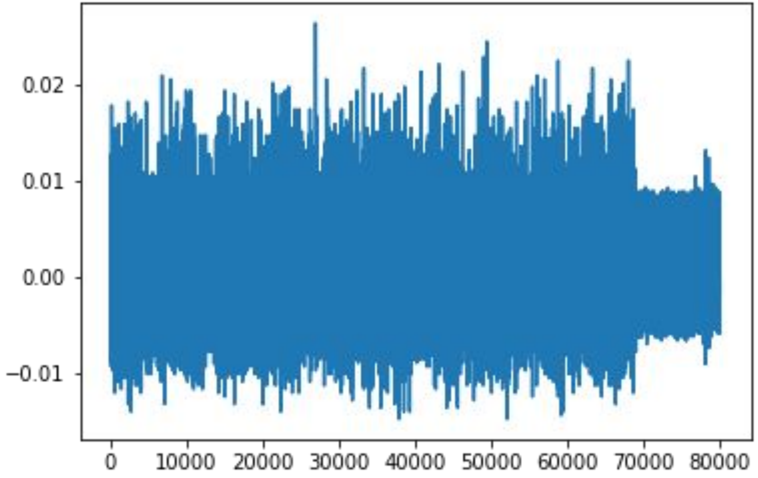
Goal: train a model that can recover the AES keys from the STM32F415 TinyAES implementation using as few power traces as possible



SCAAML game plan



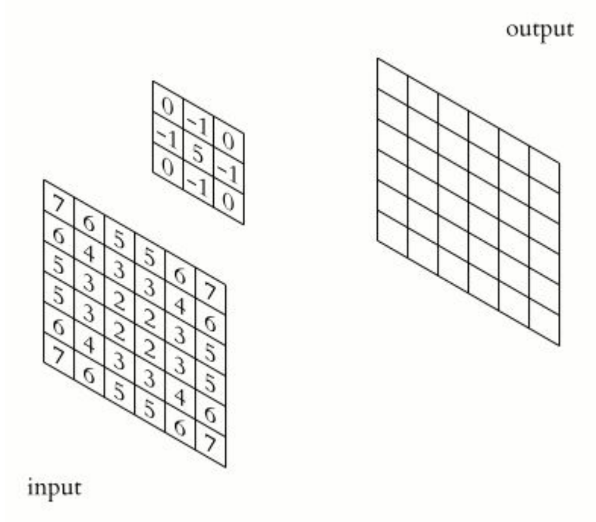
Dataset is composed of **50000 raw power traces with 80000 points per trace**, without any processing or cutting, that were connected asynchronously



Sample trace from the TinyAES dataset used in this talk

What model architecture to use?

We are going to use a **convolutional network architecture**



https://de.wikipedia.org/wiki/Convolutional_Neural_Network

Constants

```
dropout_rate = 0.3
filters = 32
kernel_size = 5
num_convolutions = 5
pool_size = 4
```

Input

```
inputs = layers.Input(shape=(trace_size, 1))
x = inputs
```

Pooling

```
x = layers.MaxPooling1D(pool_size)(x)
```

Convolutions

```
for _ in range(num_convolutions):
    x = layers.SeparableConv1D(filters, kernel_size)(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    filters *= 2
```

Pooling

```
x = layers.GlobalMaxPool1D()(x)
```

Denses

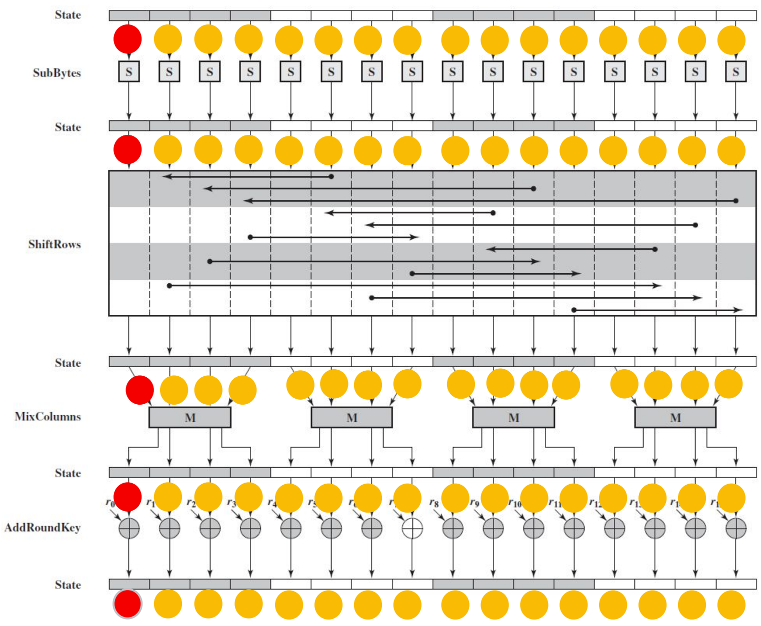
```
x = layers.Dropout(dropout_rate)(x) # better with it
x = layers.Dense(256, activation='relu')(x)
x = layers.BatchNormalization()(x)  # helps
x = layers.Dropout(dropout_rate)(x)
```

softmax

```
outputs = layers.Dense(256, activation='softmax')(x)
```

What the model should predict?

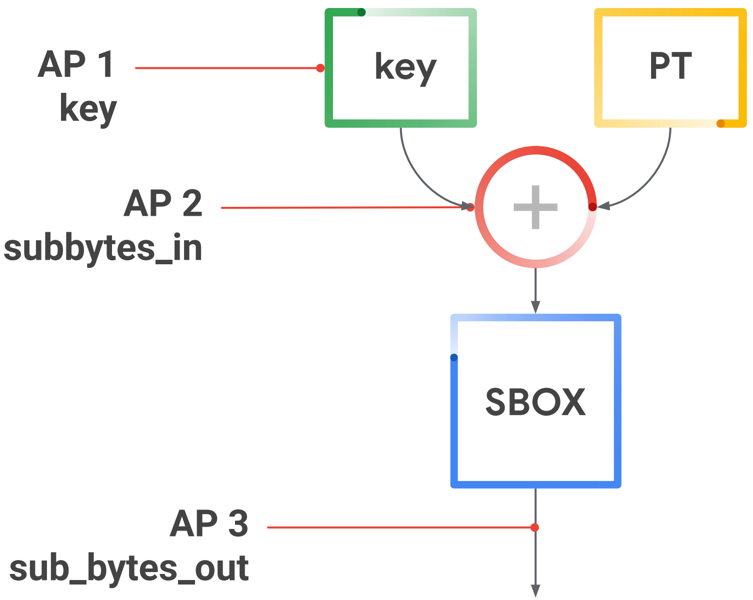
AES attack points

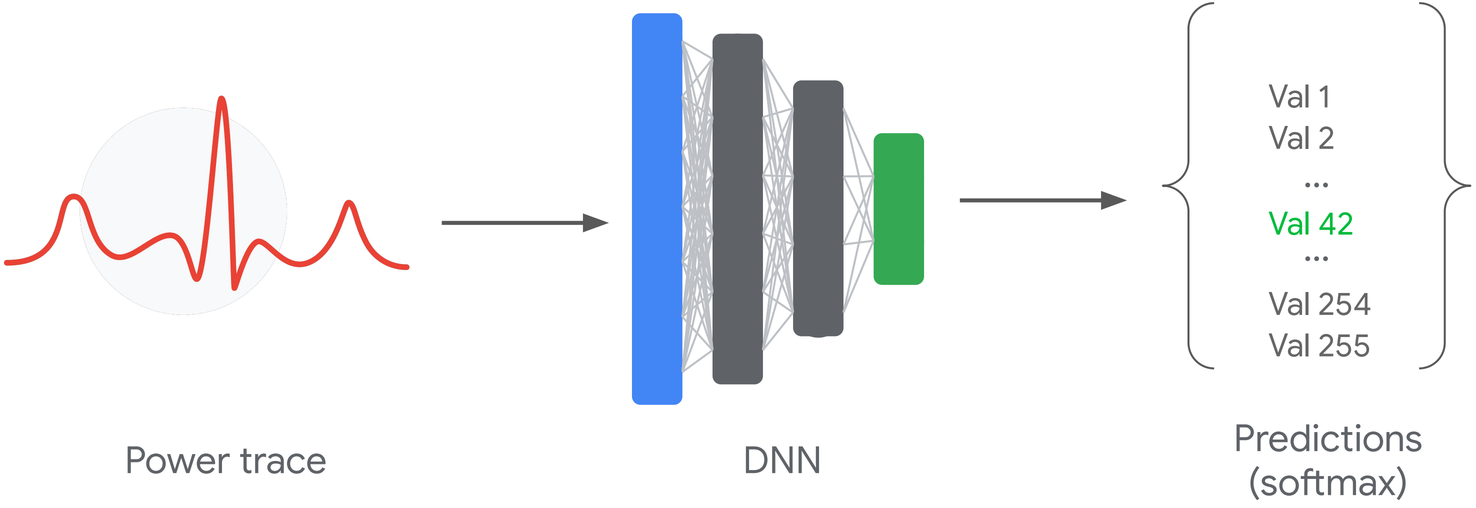


Google

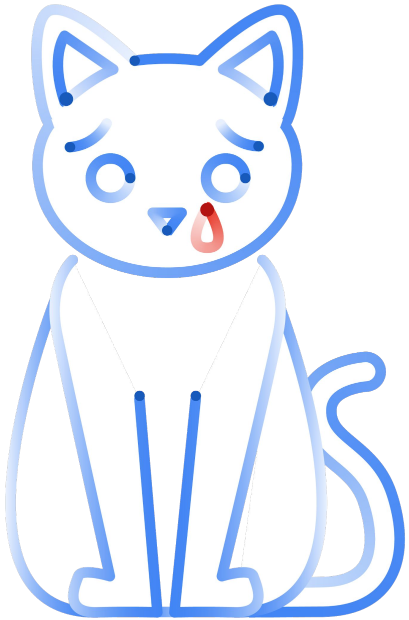
Security and Privacy Group

Target any of the initial three AES attacks points as they are easily invertible





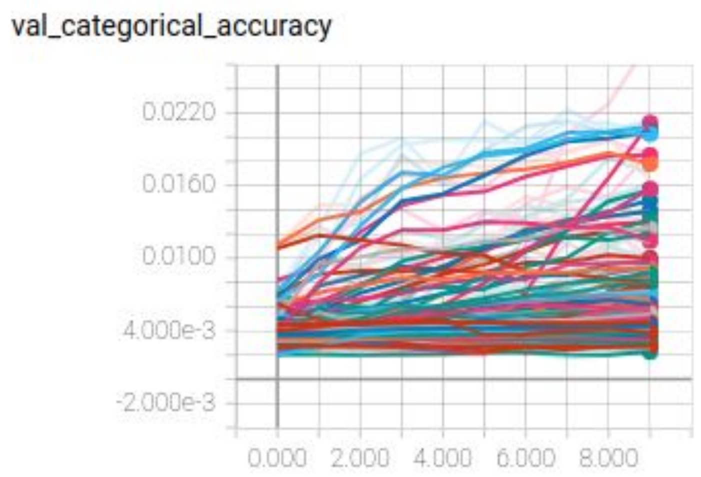
predict a single byte at the time
256 predictions per model: one for each attack point potential value



Google

Security and Privacy Group

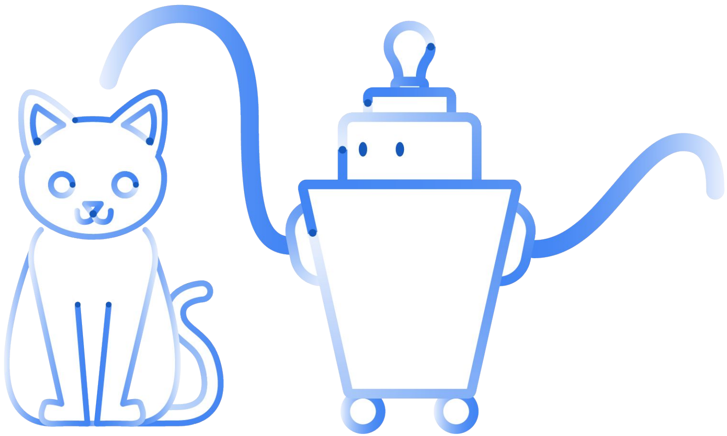
Learning crypto is hard … most models won’t converge



How do I find a model that work?

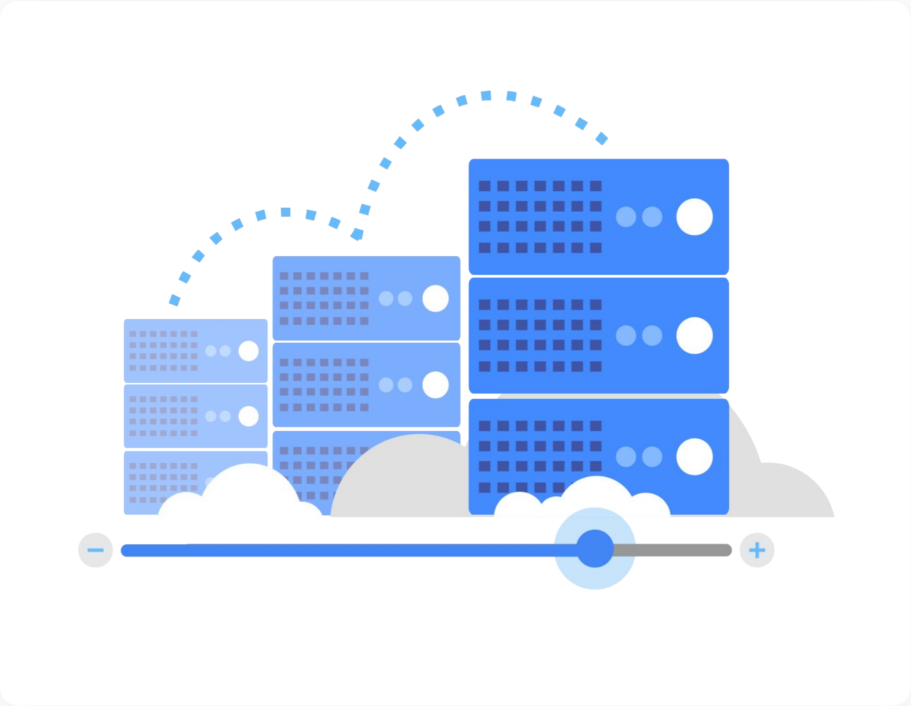
Google

Security and Privacy Group



SCAAML models **are hard to find by hand** so instead it is best to **use hyper-tuning to find the right model automatically**

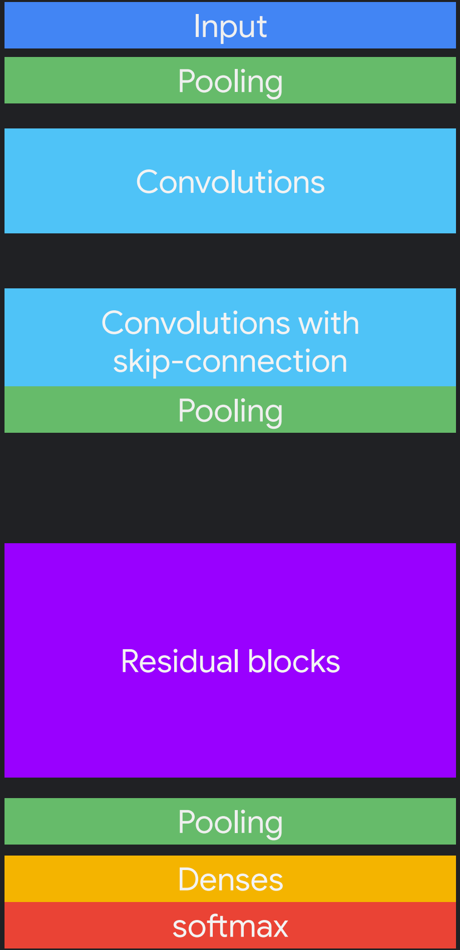
Trained 1000+ to find the right one using Keras Tuner and Kubernetes on Google Cloud



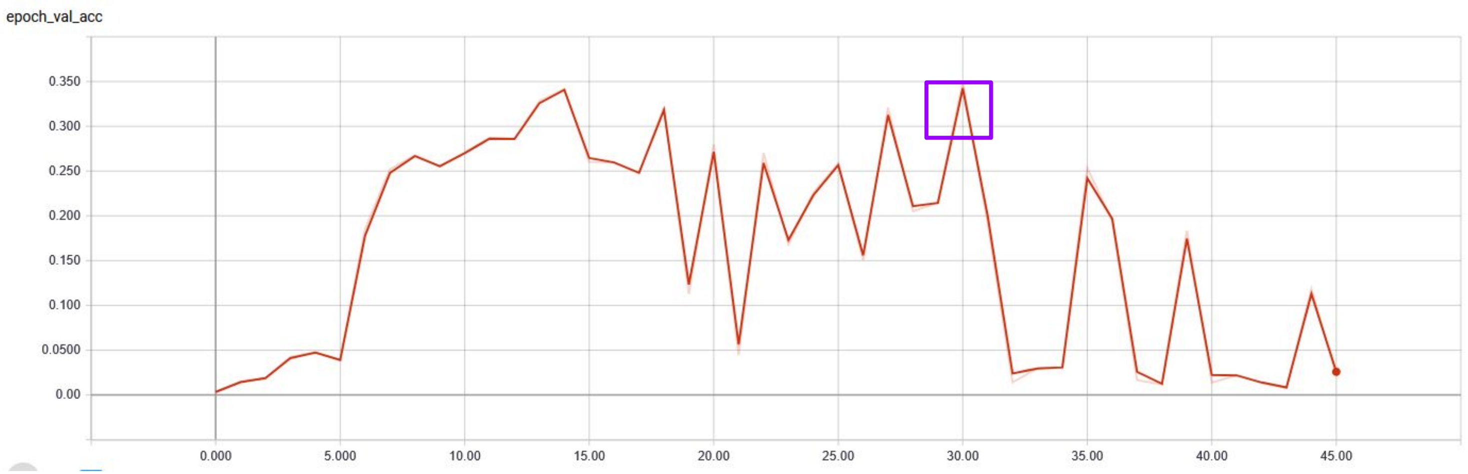
Google

Security and Privacy Group

Hypertuning found **very effective models** however **none of them are simple**

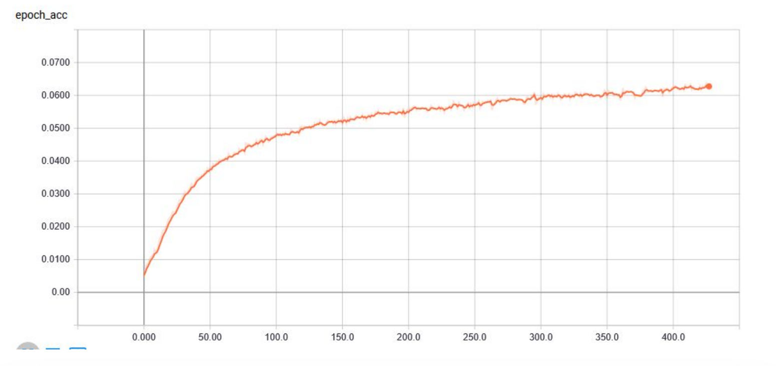


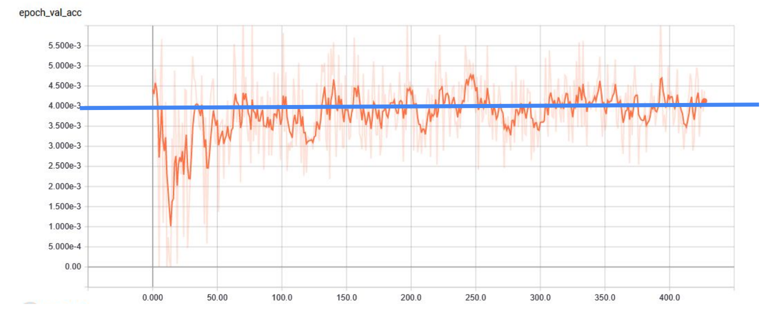

```python
x = inputs

x = layers.MaxPooling1D(pool_size)(x)  # helps

x = layers.Conv1D(16, kernel_size, strides=strides, padding='same',
activation='relu')(x)
x = layers.BatchNormalization()(x)

x = layers.Conv1D(filters, kernel_size, strides=strides, padding='same',
activation='relu')(x)
x = layers.BatchNormalization()(x)

for idx in range(num_convolutions):
    filters *= 2
    residual = layers.Conv1D(filters, 1, strides=strides, padding='same')(x)
    x = layers.SeparableConv1D(filters, kernel_size, padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.Conv1D(filters, kernel_size, padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.MaxPooling1D(kernel_size, strides=strides, padding='same')(x)
    x = layers.add([x, residual], name='sortcut_%s' % (idx))

for idx in range(nun_residuals):
    residual = x
    x = layers.Conv1D(filters, kernel_size, padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.Conv1D(filters, kernel_size, padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.Conv1D(filters, kernel_size, padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.add([x, residual], name='residual_%s' % (idx))

x = layers.GlobalMaxPool1D()(x)

x = layers.Dense(256, activation='relu')(x)
x = layers.BatchNormalization()(x)  # helps
outputs = layers.Dense(256, activation='softmax')(x)
```

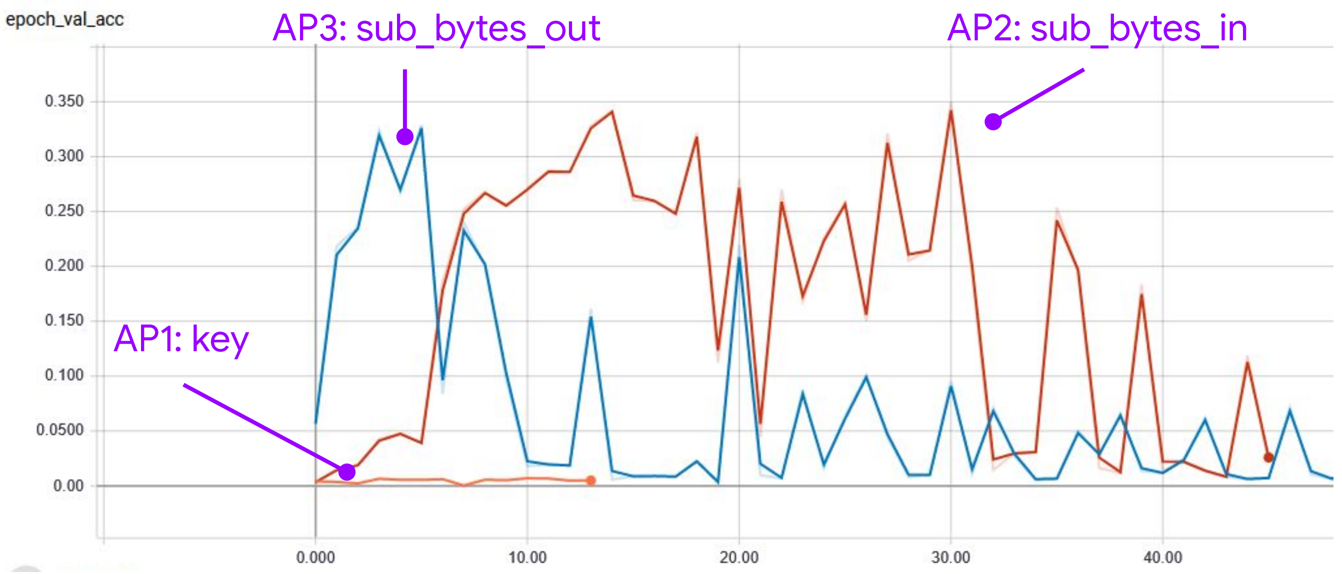


our model reached 34.94% validation accuracy before collapsing

Data augmentation can help but if badly configured it prevents the model from converging



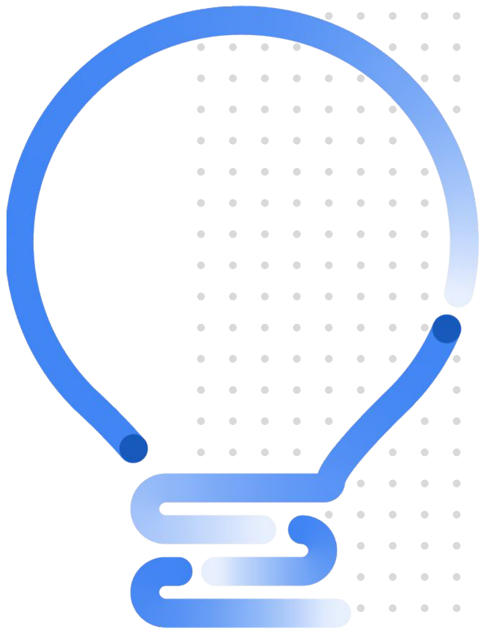




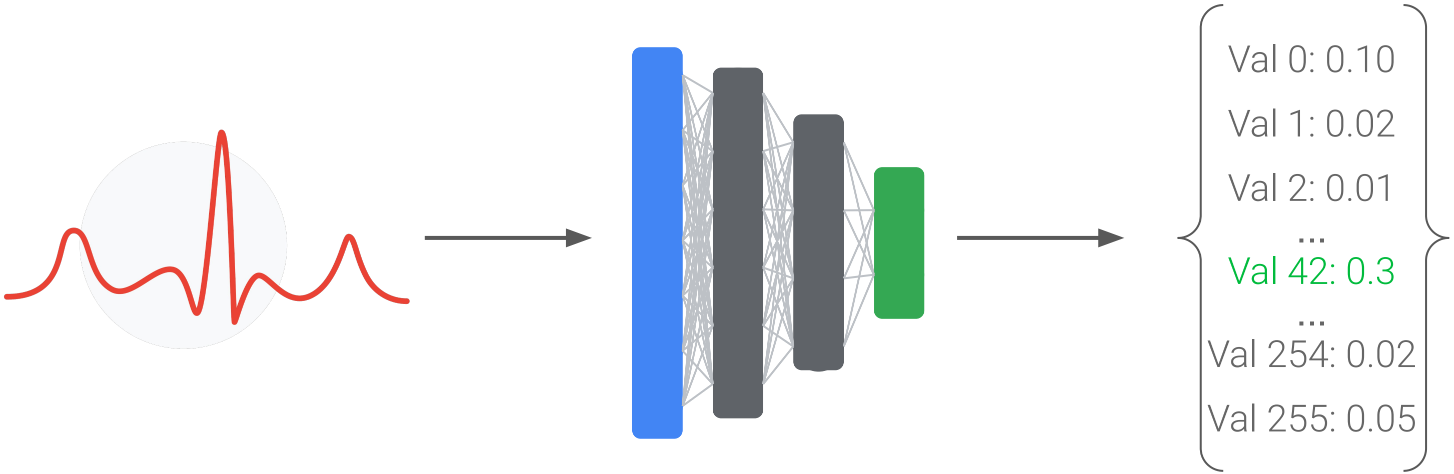
Choosing the right attack point matters to get the best performance. The **best attack point varies from architecture to architecture**

How do I recover the key?

Leverage all model predictions on many traces to carry out probabilistic attacks



Probabilistic attack: single trace



Probabilistic attack: summing traces*

Val 0: 0.10
Val 1: 0.02
Val 2: 0.01
...
Val 42: 0.3
...
Val 254: 0.02
Val 255: 0.05

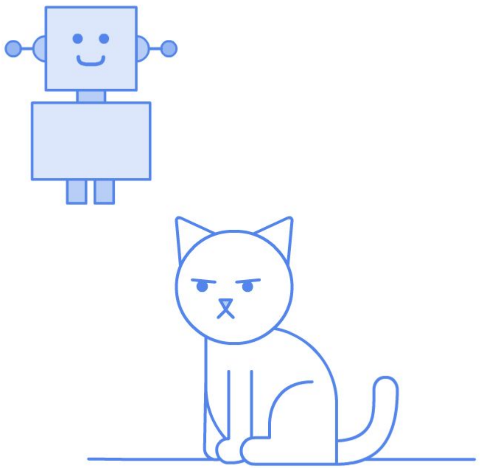
+ . . . +

Val 0: 0.08
Val 1: 0.04
Val 2: 0.05
...
Val 42: 0.12
...
Val 254: 0.03
Val 255: 0.10

→

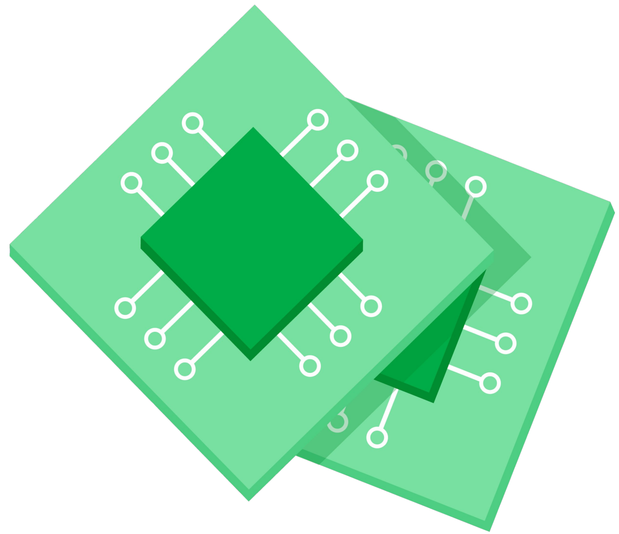
Val 0: 4.4
Val 1: 5.3
Val 2: 3.2
...
Val 42: 21.4
...
Val 254: 2.9
Val 255: 4.2

*sum uses log10 + ε



Does it work across chips?

Use a different chip to create the holdout dataset used to evaluate attack effectiveness

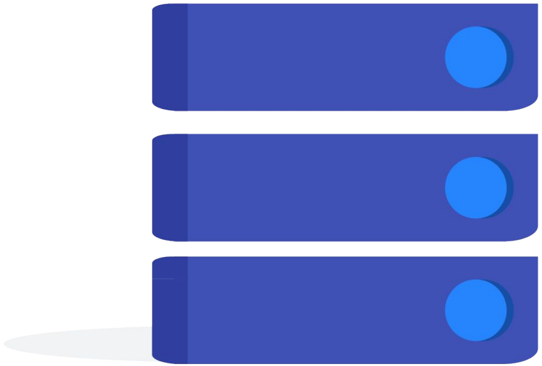


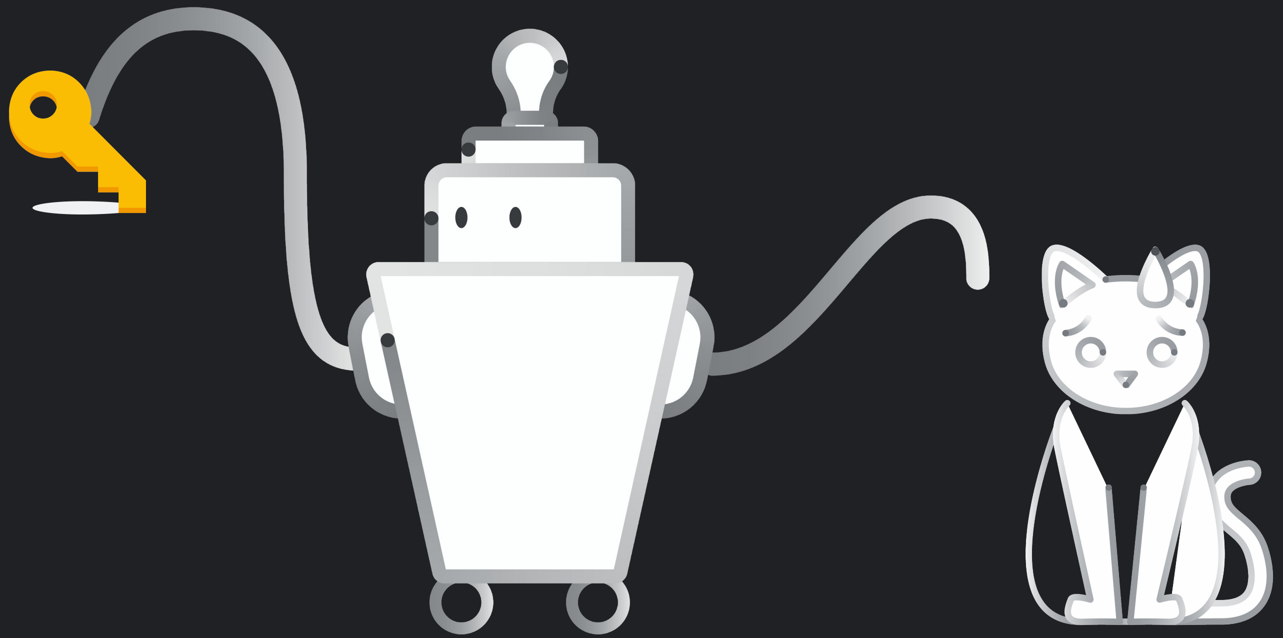
How to evaluate attack effectiveness?

Success metrics

Metric	Description	Baseline
Top 1	Number of bytes correctly predicted	0.004% (1/256)
Top 5	Number of times correct byte is in top5	0.02% (5/256)
Mean rank	Average rank of the correct byte	128
Max rank	Maximum rank of the correct byte	256

Holdout dataset is composed of **100 keys** with **300 power traces** for each key that use a different plaintext

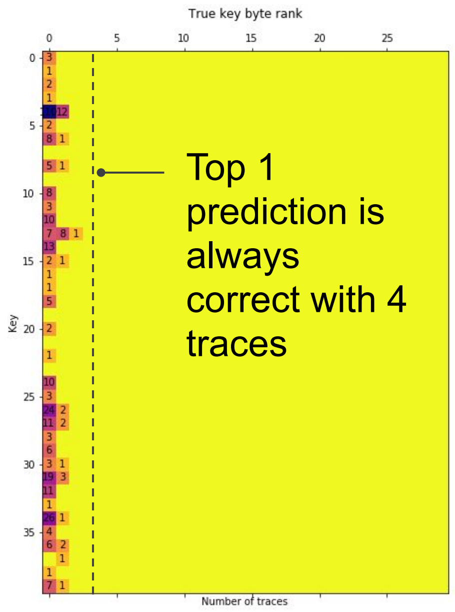




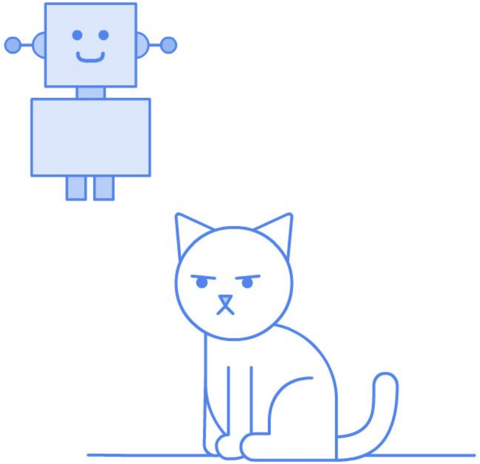
Success! We recovered 100% of the keys!

Results: **perfect score!**

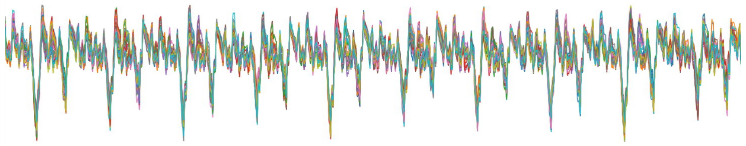
Metric	Baseline	Results
Top 1	0.004% (1/256)	**100%**
Top 5	0.02% (5/256)	**100%**
Mean rank	128	**0**
Max rank	256	**0**



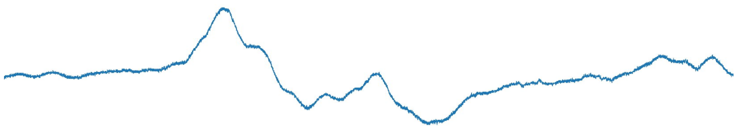
Despite having “only a 30% accuracy” our model allows to **recover automatically 100% of the bytes with at most 4 traces (81% with a single trace!) on a different chip**



How about protected implementations?



Unprotected power trace



Protected power trace

Hardened implementation needs significantly more advanced techniques, computation and data

What's next?

Testbed key numbers

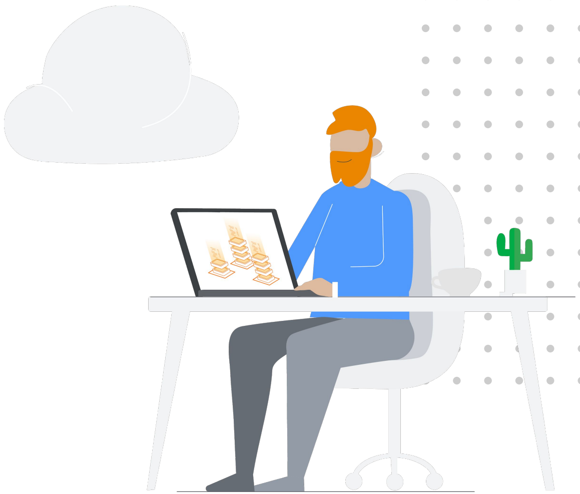
6 AES implementation

9M+ power traces

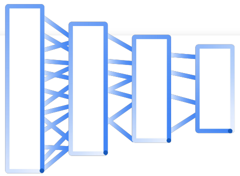
330GB storage

5000+ models trained

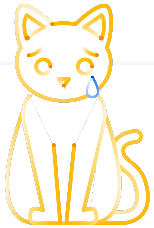
Hope the initial draft of our paper will be public in a few weeks with our results



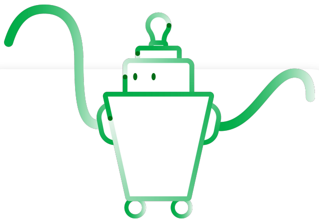
Takeaways



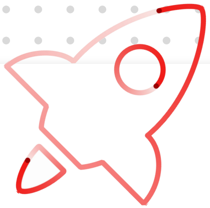
Deep-learning is the future of hardware SCA



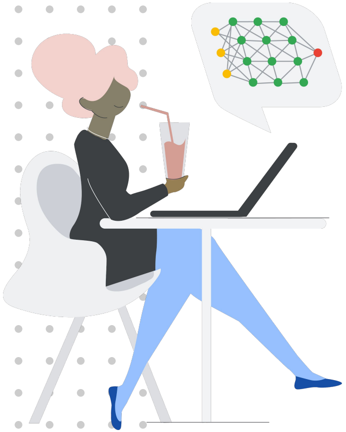
Training model for SCA is hard



Automation is key to success



It’s just the beginning



SCAAML allow to focus on crypto algorithms design and analysis by automatically leveraging computing and AI improvements to assess their security

Keep up with our progress on deep-learning side-channel attacks: https://elie.net/scaaml