Radboud University

# AI and Side-channel analysis: An overview

IACR School on Applied Cryptography
February 2, 2023

Lejla Batina

Institute for Computing and Information Sciences
Radboud University
lejla@cs.ru.nl

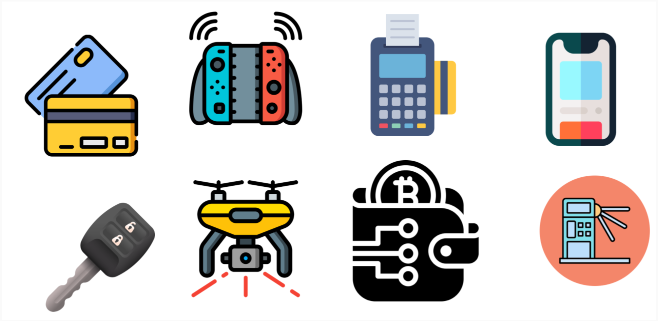
## Outline

## About me

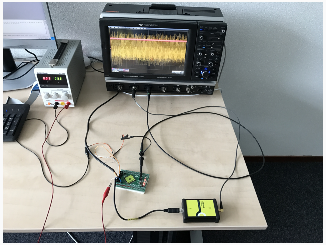
- 2001: PDEng in engineering, Mathematics for Industry
- 3 years with semiconductor industry as a cryptographer
- 2005: PhD from KU Leuven on hardware impl. of Public-key cryptosystems
- 2006-2009: PostDoc at KU Leuven
- 2009- : Digital Security group at Radboud University, full prof. since 2017
- CESCA lab: 10+ PostDocs/PhD students working on:
    - Physical attacks on embedded systems
    - Secure cryptographic implementations
    - FPGA security
    - AI and security
    - Lightweight cryptography
    - Leakage simulators

## Intro to side-channel analysis

## Known challenge: embedded crypto devices



## Implementation attacks



## Relevance

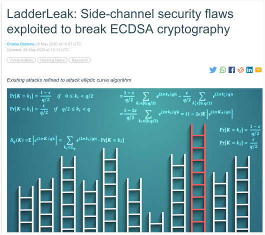
**November 13, 2019**

**October 3, 2019**

Researchers Discover ECDSA Key Recovery Method

October 3, 2019 • Add Comment • by Emma Davis

**May 28, 2020**

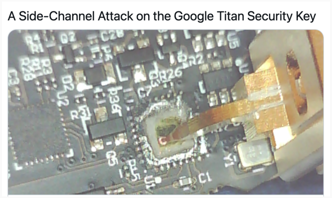

LadderLeak: Side-channel security flaws exploited to break ECDSA cryptography

Charlie Osborne 28 May 2020 at 14:07 UTC
Updated: 28 May 2020 at 15:13 UTC

Vulnerabilities Hacking News Research

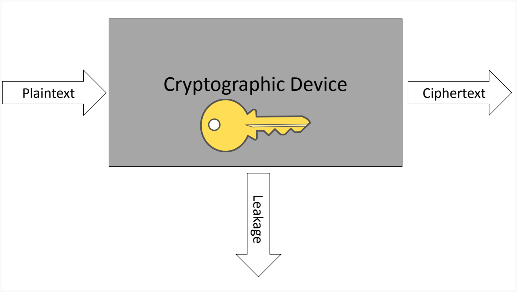
*Existing attacks refined to attack elliptic curve algorithm*

**January 7, 2021**


A Side-Channel Attack on the Google Titan Security Key

## Side-channel Analysis (SCA) Attacks

## Greybox/Whitebox scenario



Greybox = SCA adversary in the wild:

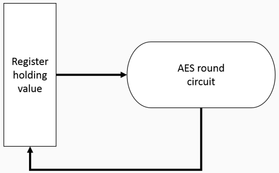
- Crypto is **implemented on a real device** such as a microcontroller, FPGA, ASIC
- Adversary can measure and process *physical quantities* in the device's vicinity
- Adversary's goal: secret key or plaintext recovery by observing plaintext/ciphertext pairs **and a side channel**

Whitebox = Security evaluator:

- Algorithms and implementation details are (partially) known
- Adversary's goal: secret key or plaintext recovery by observing plaintext/ciphertext pairs while trying all known attacks, including profiling

## Power side-channel: Modeling the leakage

- The Hamming distance model counts the number of $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions
- Example 1: Assume a hardware register R storing the result of an AES round. The register initially contains value $v_0$ and gets overwritten with value $v_1$

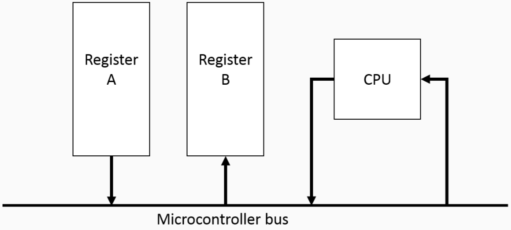


- The power consumption because of the register transition $v_0 \rightarrow v_1$ is related to the number of bit flips that occurred
- Thus it can be modeled as $\text{HammingDistance}(v_0, v_1) = \text{HammingWeight}(v_0 \oplus v_1)$

## Power side-channel: Modeling the leakage

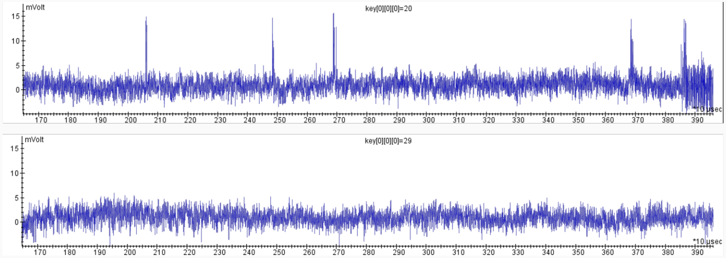
- Example 2: In a microcontroller, assume register A with value $v_0$ and an assembly instruction that moves the contents of register A to register B

```
mov rB, rA
```



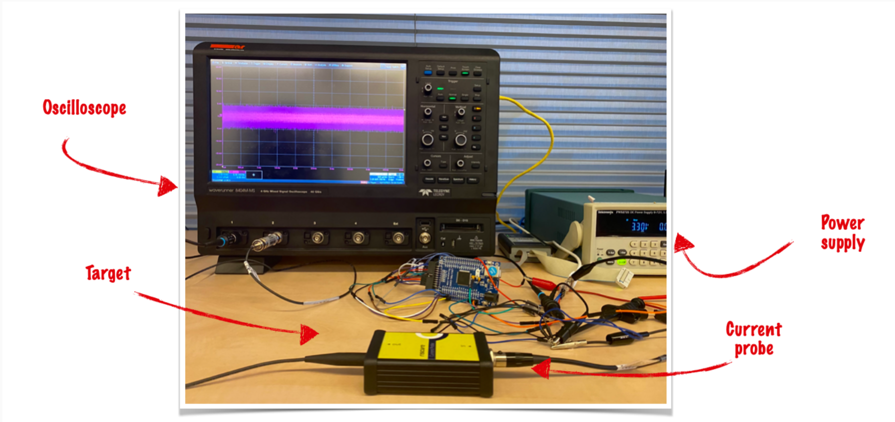
- In general-purpose processors the instruction will transfer value $v_0$ from register A to B via the CPU, using the bus
- Often the bus is a very leaky component and also precharged to all bits to zeros (or all to 1) i.e. busInitialValue
- The power consumption of the assembly instruction can be modeled as $\text{HammingDistance}(\text{busInitialValue}, v_0) = \text{HammingWeight}(v_0 \oplus 0) = \text{HW}(v_0)$

## Differential Power Analysis (DPA)



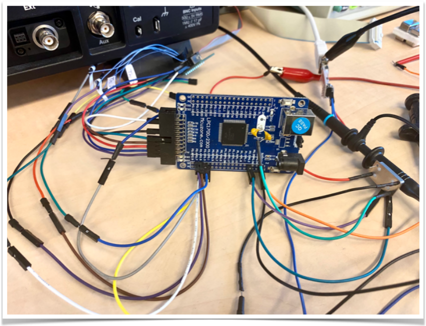
- The most popular side-channel attack
- Aims at recovering the secret key by using a large number of power measurements (traces)
- Nowadays often combined/replaced with a leakage evaluation methodology such as TVLA

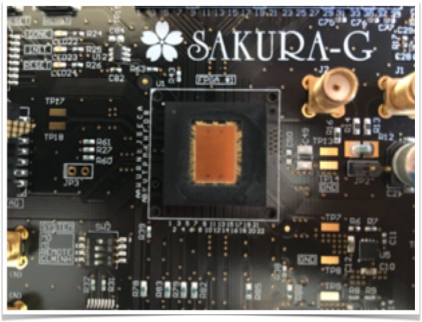
## Typical setup for power side-channel analysis



@CESCA, Radboud University

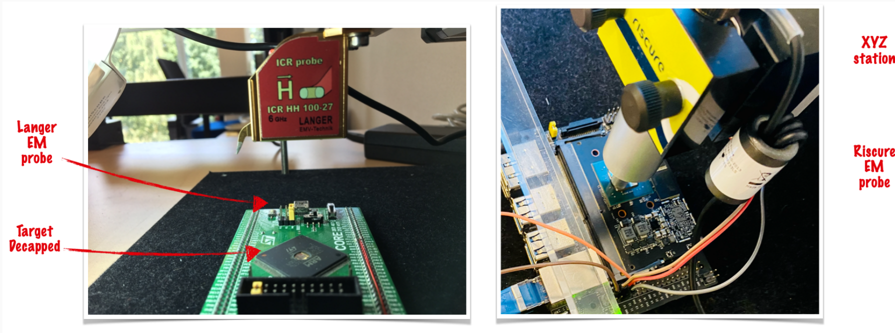
## Typical setup for power side-channel analysis





@CESCA, Radboud University

## Typical setup for electromagnetic analysis



@CESCA, Radboud University

## SCA Countermeasures

## Main idea

Goal: break the link between the actual data and power consumption

- Masking: power consumption remains dependent on the data on which computation is performed but not the actual data
- Hiding: power consumption is independent of the intermediate values and of the operations

## Masking

Boolean masking: a $d$th-order (Boolean) masking scheme splits an internal sensitive value $v$ into $d+1$ shares $(v_0, v_1, ..., v_d)$, as follows:

$$v = v_0 \oplus v_1 \oplus \cdots \oplus v_d$$

*Probing-secure scheme.* We refer to a scheme that uses certain families of shares as $d-$probing-secure iff any set of at most $d$ intermediate variables is independent from the sensitive values.

Consequently, the leakage of up to $d$ values does not disclose any information to the attacker.

## Masking with 2 shares

- $X = X_1 \oplus X_2$
- The leakage $L(X) = HW(X_1, X_2)$ depends on two variables.
- It does not reveal info on the value of $X$ when a DPA is performed, in theory

Masking in practice: unintended interactions between values in the processor cause leakage in 1st order (caused often by transitional effects and glitches).

If a program that processes a secret value $X$ contains two consecutive instructions (the first uses $X_1$ and the second uses $X_2$), then the transitional effect of changing the contents of the bus leaks the Hamming distance between $X_1$ and $X_2$.

## Leakage evaluation

## Independent Leakage Assumption (ILA) [RSVKF11]

Independent computations give rise to independent leakage.

Practically: The underlying assumption of this model is that the adversary can only observe a single intermediate value with every probe used.

As a consequence: All masks (shares) need to be processed independently

Physical side-effects when implementing masking, such as glitches and distance-based leakages, violate ILA in practice.

## Test Vector Leakage Assessment (TVLA)

- Leakage assessment of a device is very important for the semiconductor and the security evaluation industries
- Number of attacks to check the device's resistance against keeps on growing
- Various attackers' models possible but security evaluation often goes for the strongest adversary
- It is using Welch's $t$-test to differentiate between two sets of measurements, one with fixed inputs and the other with random inputs
- Far from perfect, false positive and negatives are possible
- Leakage from combining multiple points is not detected

## What we learned so far

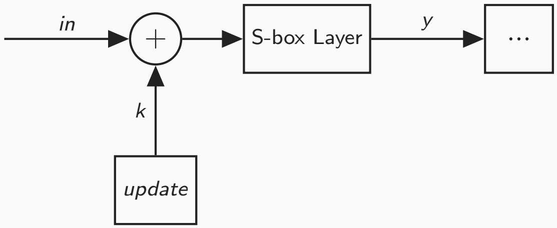
- Side-channel attacks are a threat to all implementations of cryptography
- The adversaries vary in capabilities and assumed knowledge of the target
- The SCA attacker needs to define: sensitive variable, leakage model and SCA distinguisher
- Countermeasures should be implemented considering a realistic adversary and cost-win tradeoffs
- Now we will learn about new type attacks, so-called profiling attacks assuming a very powerful adversary

## Profiling attacks

## Types of attacks/analyses and attacker models

- Simple Power Analysis (SPA): one or a few measurements e.g. when attacking 1-time keys
- Differential power analysis (DPA): multiple measurements available with the same key e.g. of unprotected implementations using “long-term” keys
- Higher order attacks: multiple measurements - for protected implementations
- Profiled attacks: training i.e. profiling with many measurements and just 1 measurement (or several) for key recovery
- Advanced attacks: using AI, theoretical cryptanalysis, combination of attacks etc.
- Fault analysis: using Fault Injection (FI) techniques to force computation errors

## Template motivation example

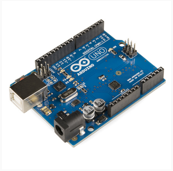


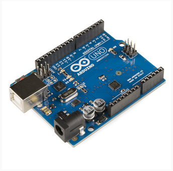
- Key $k$ is refreshed before every encryption
- Does classical DPA work?
- No! It requires a constant key
- We typically rely on several assumptions (for attacks):
  1. We assume the leakage to be related to the Hamming weight (or distance) of the manipulated data
  2. We look for leakage in the S-box output only
  3. We assume the leakage to be univariate
- Ideally we would like to extract more leakage with minimal assumptions

## Template Attack

- Attacking a well-protected device directly is hard
- We often do not get many traces with the same secret (key)
- So, we use an unprotected device of the same model

**Figure:** protected device (left), unprotected device (right)





- We profile, i.e. template the unprotected device
- We use the profile to break the protected device

## Template Attack Procedure

(1) Choose a **model** that describes the power consumption

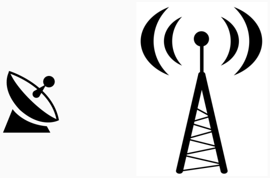
(2) Profile the **unprotected device** to create the template (Template Building)

(3) Use the template to break the **protected** device (Template Matching)

The same steps are always performed.

The **model** can be different.

## Noisy Communication Example



- Assume a transmitter sends a bit, 0 or 1, to a receiver
- Bit 0 is encoded as 0 Volts, bit 1 as +5 Volts
- The receiver gets a noisy signal
- The receiver must decide if the bit is 0 or 1

## Noisy Communication Example

- The receiver measures $+5.1$ Volts
- Bit is probably 1
- The receiver measures -0.3 Volts
- Bit is probably 0
- The receiver measures $+3.15$ Volts
- Bit value 1 is more likely than bit value 0

**Detection and estimation theory studies such phenomena and is able to quantify the uncertainty in our problems.**

## Reduced Templates - Template Building

(1) Force the transmitter to send bit 0, e.g. 1000 times

(2) Measure the voltage in the receiver: ${}^{0}v_1,\ {}^{0}v_2, \ldots,\ {}^{0}v_{1000}$

(3) Compute the mean: ${}^{0}\bar{v} = (1/1000) * \sum_{i=1}^{1000}\ {}^{0}v_i$

(1) Similarly, force the transmitter to send bit 1, e.g. 1000 times

(2) Measure the voltage in the receiver: ${}^{1}v_1, {}^{1}v_2, \ldots, {}^{1}v_{1000}$

(3) Compute the mean: ${}^{1}\bar{v} = (1/1000) * \sum_{i=1}^{1000}\ {}^{1}v_i$

We have computed the templates $T_0$ and $T_1$.

## Reduced Templates - Template Matching

(1) Observe a measurement $v$ in the receiver

(2) Match $v$ to template $T_0$, i.e. compute $score_0 = \mid v -^0 \bar{v} \mid$

(3) Match $v$ to template $T_1$, i.e. compute $score_1 = \mid v -^1 \bar{v} \mid$

(4) *if* $score_0 \geq score_1$ *then* bit=0 *else* bit=1

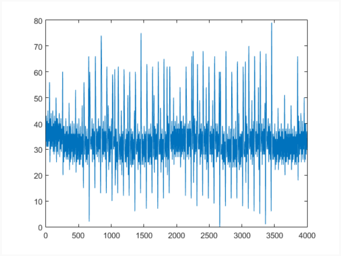
## Reduced Templates - Formal Definition

(1) Model the voltage measurement in the receiver using a random variable $V$ and the bit using random variable $B$

(2) Create a template for the random variable $T_0 = (V|B = 0)$
Create a template for the random variable $T_1 = (V|B = 1)$

(3) Match a measurement $v$ to $T_0$ or $T_1$ using the *score* function

## Reduced Templates - Side Channels



- Side-channel analysis focuses on modeling real measurement traces
- We model a measurement as a random variable $V$
- A *realization* (or instance) of $V$ is a measurement $v$, e.g. $v = 5.1\,\text{Volts}$
- A trace consists of several (or many) measured samples e.g. values in certain time points
- We model a trace as a vector of random variables, also called a random vector $\mathrm{L}$, where $\mathrm{L} = [L^1, L^2, \ldots, L^{no\_samples}]$
- A *realization* of the random vector $\mathrm{L}$ is a trace $\mathrm{l}$ e.g. $\mathrm{l} = [0.41, 0.10, 0.12, 0.17, 0.36]$

## Reduced Templates - Template Building

- Template for $key_0$
  1. Force the cryptographic device to encrypt $n$ times with key $K = key_0$
  2. Measure $n$ traces $\mathsf{l}_i$ with $K = key_0$
  3. The template for $\mathsf{T}_{\mathsf{key}_0} = (\mathsf{L}|K = key_0)$ is the mean vector $^{key_0}\bar{\mathsf{l}} = 1/n * \sum_{i=1}^{n} {}^{key_0}\mathsf{l}_i$
- Template for $key_1$
  1. Force the cryptographic device to encrypt $n$ times with $K = key_1$
  2. Measure $n$ traces with $K = key_1$
  3. The template for $\mathsf{T}_{\mathsf{key}_1} = (\mathsf{L}|K = key_1)$ is the mean vector $^{key_1}\bar{\mathsf{l}} = 1/n * \sum_{i=1}^{n} {}^{key_1}\mathsf{l}_i$

## Reduced Templates - Template Matching

(1) Observe a trace t

(2) Match the trace t to templates $T_{key_0}$ and $T_{key_1}$

(3) Use the matching score to decide the key used by the trace **t**

(4) $score_0 = (\mathrm{t} - ^{key_0} \bar{\mathrm{l}}) * (\mathrm{t} - ^{key_0} \bar{\mathrm{l}})^T$
$score_1 = (\mathrm{t} - ^{key_1} \bar{\mathrm{l}}) * (\mathrm{t} - ^{key_1} \bar{\mathrm{l}})^T$

(5) Decide $key_0$ or $key_1$

We examined template attacks using the **reduced model**

Next up is the **univariate Gaussian model**

## Estimation Theory Basics

- So far we did not discuss models in detail, so how can we model data?
- **Probability functions have the ability to model experimental data**
- **The Templates are random variables that have certain probability density functions (p.d.f.)**
- Do you know some examples?

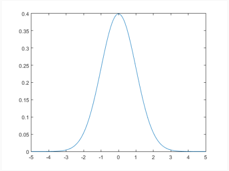
- The Poisson p.d.f. (probability density function) is often used to model Internet traffic
- Gaussian p.d.f. can describe e.g. the weight of an adult population, demonstrating the mean and variance observable
- Gaussian p.d.f. can also be used to model the power consumption measurements used in side-channel analysis attacks

## Gaussian distribution

- **Every distribution includes one or more parameters**
- Example: the weight of a population is “drawn from” the Gaussian distribution with mean $\mu = 81$ and variance $\sigma^2 = 4.5$
  Formally: the weight is described by the random variable $X$ and $X \sim \mathcal{N}(\mu, \sigma^2)$
- **Estimation theory describes how to determine the parameters of our model, using our experimental data**

## Estimation Theory Basics – Dota 2

- Assume that the Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ is a good model for our experiment



- Start from the experimental data:

| Match | No of Deaths |
|---|---|
| 1 | 6 |
| 2 | 9 |
| 3 | 2 |
| 4 | 13 |

- Estimate the parameters $\mu$ and $\sigma^2$ of the Gaussian distribution that models our data.

## Maximum Likelihood Estimation

- Maximum Likelihood Estimation (MLE) describes a method to estimate the parameter(s) of a distribution.
- Assume that $X \sim f_X(x;\theta)$, where $\theta$ is the unknown parameter
- In our example, $X$ is the random variable describing the No. of deaths and $\theta = (\mu, \sigma)$
- We have captured a random sample of size $n: x_1, x_2, \ldots, x_n$
- In our example, $\{x_1, x_2, x_3, x_4\} = \{6, 9, 2, 13\}$

## Maximum Likelihood Estimation

- We **fit** our dataset into the model by estimating the distribution parameters
- $\mu_d = \frac{\sum_{i=1}^{n} x_i}{n}$
- $\sigma_d^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \mu_d)^2$
- The est. no. of deaths is $\mu_d = 7.5$, $\sigma_d = 4.03$

We demonstrated **template building** for the **univariate Gaussian model**
Next, we focus on **template matching** for the univariate Gaussian model

## Hypothesis Testing and Decision Theory

- **In Binary Hypothesis testing we know that either event $H_0$ or $H_1$ is true**
- Ex.1: radar detected “no airplane” or “an airplane” (military)
- Ex. 2: recognition system of an amusement park decides that the client has paid the ticket or not (entertainment)
- Ex. 3: transmitting source sent $bit = 0$ or $bit = 1$ (telecom)
- Ex. 4: cryptographic implementation used $key_0$ or $key_1$ (crypto)

## Hypothesis Testing and Decision Theory

**For every experiment with $H_0$ and $H_1$ what are the 4 possible courses of action?**

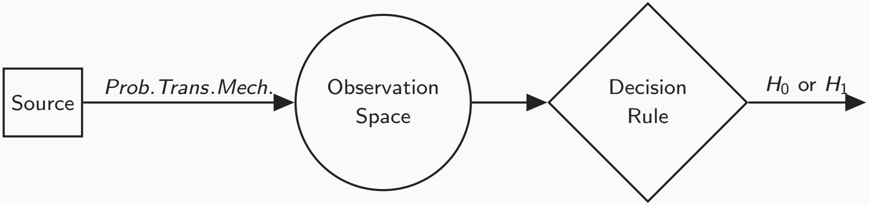
- Decide $H_0$; $H_0$ true (correct choice)
- Decide $H_0$; $H_1$ true (false negative, type II error)
- Decide $H_1$; $H_0$ true (false positive, type I error)
- Decide $H_1$; $H_1$ true (correct choice)

**What are the implications of type I and type II errors for military radar, entertainment park, telecom, crypto?**

- Military: What does a false negative/positive mean for a radar?
  False negative is equivalent to airspace violation, false positive results in more work and expenses.
- Entertainment park: What does a false negative/positive mean for ticket verification?
  False negative could trigger customer anger and false positive could result in monetary loss.
- Telecom & crypto:
  False negative is actually equivalent to false positive scenario. In both cases the bit/key is detected incorrectly.

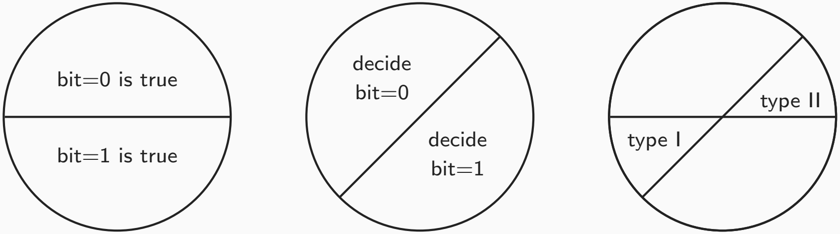
## Basic Definitions for Detection

In order to formalize the error notions, we introduce the concepts of the Source, Probabilistic Transition Mechanism, Observation Space and Decision Rule



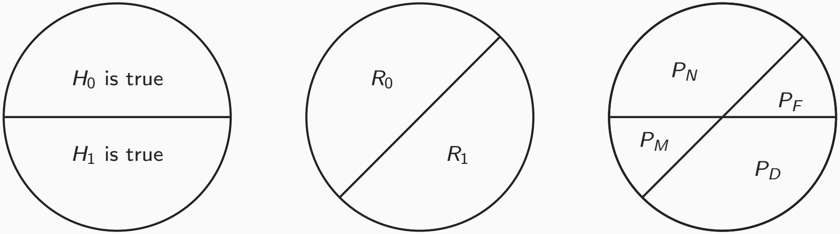
- "A transmission antenna sends bit 0 or 1 over the air. A receiver antenna recovers the emitted noisy signal and tries to deduce if the transmitted bit was 0 or 1"
- **Hypotheses:** $H_0$ equals transmitted bit 0, $H_1$ equals transmitted bit 1
- **Source:** transmission antenna, knows if $H_0$ or $H_1$ is true
- **Probabilistic Transition Mechanism:** noise addition to the signal
- **Observation Space:** noisy received signal
- **Decision Rule:** the process that decides if $bit = 0$ or $bit = 1$

## Bayes Criterion



- Assume the circle is the 2D observation space R
- The reality partitions the space in two (left circle)
- The source knows this partitioning but the receiving antenna does not
- The receiving antenna creates its own decision rule partitioning the observation space $R$ in two (mid circle)
- The region *decide bit* $= 0$ results in decision $H_0$ and the region *decide bit* $= 1$ results in decision $H_1$ (mid circle)
- Imperfect partition results in type I and type II errors (right circle)

## Bayes Criterion



- Trace r, Probability density functions $f(\mathrm{r}|H_0)$ and $f(\mathrm{r}|H_1)$
- $P$(decide $H_1|H_0$ true)= probability of false alarm $P_F = \int_{\mathrm{r}\in\mathrm{R}_1} f(\mathrm{r}|H_0)\,d\mathrm{r}$
- $P$(decide $H_0|H_1$ true)= probability of miss $P_M = \int_{\mathrm{r}\in\mathrm{R}_0} f(\mathrm{r}|H_1)\,d\mathrm{r}$
- $P$(decide $H_0|H_0$ true)= probability $P_N = \int_{\mathrm{r}\in\mathrm{R}_0} f(\mathrm{r}|H_0)\,d\mathrm{r} = 1 - P_F$
- $P$(decide $H_1|H_1$ true)= probability of detection $P_D = \int_{\mathrm{r}\in\mathrm{R}_1} f(\mathrm{r}|H_1)\,d\mathrm{r} = 1 - P_M$

## Bayes Criterion

- The result/score is the *Likelyhood Ratio Test*, denoted as:

- $\Lambda(\mathrm{r}) \underset{H_0}{\overset{H_1}{\gtrless}} 1$
- $\Lambda(\mathrm{r}) = P(\mathrm{r}|H_1)/P(\mathrm{r}|H_0) = \frac{f_{R|H_1}(\mathrm{r})}{f_{R|H_0}(\mathrm{r})}$, i.e. the ratio of the distributions of $H_1$ and $H_0$, for the observation r

## Template Matching - Univariate Gaussian Model

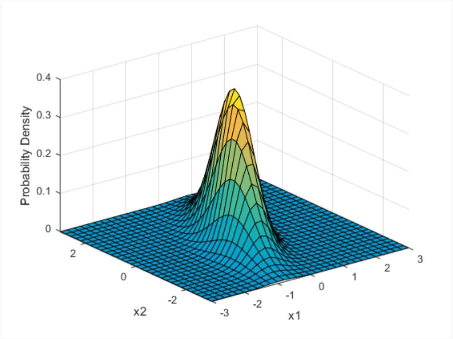
- A common choice for the side-channel attack p.d.f. is the univariate Gaussian distribution:
- $P(\mathrm{r}|H_0) \sim \mathcal{N}(\mu_{key_0}, \sigma_{key_0})$
- $P(\mathrm{r}|H_1) \sim \mathcal{N}(\mu_{key_1}, \sigma_{key_1})$
- The parameters $\mu_{key_0}, \sigma_{key_0}, \mu_{key_1}, \sigma_{key_1}$ have already been estimated using the estimation formulas discussed before
- **We will check if an observation** $\mathrm{r}_{\mathrm{test}}$ **matches** $\mathcal{N}(\mu_{key_0}, \sigma_{key_0})$ **or** $\mathcal{N}(\mu_{key_1}, \sigma_{key_1})$
- $\Lambda(\mathrm{r}) = \mathcal{N}(\mathrm{r}_{\mathrm{test}}, \mu_{key_0}, \sigma_{key_0}) / \mathcal{N}(\mathrm{r}_{\mathrm{test}}, \mu_{key_1}, \sigma_{key_1})$
- if $\Lambda(\mathrm{r}) > 1$ we decide that $key_0$ matches
- if $\Lambda(\mathrm{r}) < 1$ we decide that $key_1$ matches

## Recap

- We used p.d.f.s for templates.
- We build the templates by estimating the parameters of the p.d.f.s
- We demonstrated **template matching** for the **univariate Gaussian model**

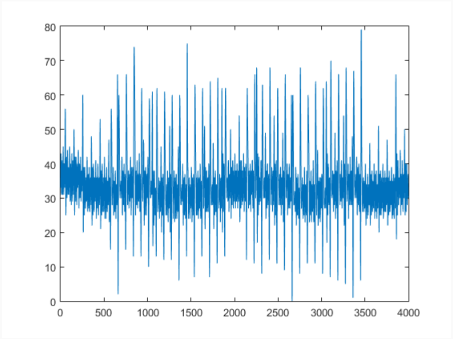
Next step would be to extend **template building and matching** for the **multivariate Gaussian model**.

## Multivariate Gaussian Distribution



## Multivariate Gaussian Model

- Build templates with a multivariate Gaussian distribution
- Perform estimation (templates building) and detection (template matching) in this context
- The joint distribution of several samples in the trace will make us distinguish $key_0$ and $key_1$ more effectively



## Multivariate Gaussian Model - POIs

- Not all samples contain sensitive information (CPA assignments)
- We refer to the key-dependent sample points as Points of Interest (POIs)
- Usually heuristics are used to find them

- **Difference of Means heuristic:**
  - Assuming 2 keys ($key_0, key_1$) use the traces with $key_0$ to compute $\mu_{key_0}$ and the traces with $key_1$ to compute $\mu_{key_1}$ ($\forall$ sample)
  - We get 2 vectors: $[\mu_{key_0}^{sample_0} \dots \mu_{key_0}^{sample_t}]$ for $key_0$ and vector $[\mu_{key_1}^{sample_0} \dots \mu_{key_1}^{sample_t}]$ for $key_1$ where $t + 1 = \#$ samples
  - Find the $m$ samples with the highest $\mu_{key_0} - \mu_{key_1}$, where $m$ is the choice of the evaluator
- **Sum of Square Differences (SOSD):** Find $m$ samples with the highest $(\mu_{key_0} - \mu_{key_1})^2$
- **Standard Deviation**: Find the $m$ samples with the highest $\sigma$
- **Correlation Power Analysis:** Best points according to Pearson corr. coeff.
- **Principal Component Analysis (PCA)**

## Multivariate Gaussian Model - Estimation

**Multivariate Gaussian p.d.f. for $m$ POIs:**

$$f(l_1, l_2, \ldots, l_m) = \frac{1}{\sqrt{(2\pi)^m * det(\Sigma)}} exp[-\frac{1}{2}((\mathrm{l} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathrm{l} - \boldsymbol{\mu}))]$$

- The leakage vector of random variables $\mathrm{L} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$
- We need to estimate two parameters: the mean vector $\boldsymbol{\mu}$ and the covariance matrix $\Sigma$

## Template Building

**Multivariate Gaussian p.d.f. for $m$ POIs:**

$$f(l_1, l_2, \dots, l_m) = \frac{1}{\sqrt{(2\pi)^m * det(\Sigma)}} exp[-\frac{1}{2}((\mathrm{l} - \boldsymbol{\mu})^T * \Sigma^{-1}(\mathrm{l} - \boldsymbol{\mu}))]$$

- For all $n$ traces $\mathrm{l}_i$ encrypting with $key_0$:

  Mean vector: $\boldsymbol{\mu}_{key_0} = 1/n * \sum_{i=1}^{n} {}^{key_0}\mathrm{l}_i$

  Covariance matrix: $\Sigma_{key_0} = cov(\mathrm{T}), \mathrm{T} = (\mathrm{l}_1, \dots, \mathrm{l}_n)^T$

- Do the same for all key candidates $key_1, key_2, \dots$

- The template for $key_i$ is $(\boldsymbol{\mu}_{key_i}, \Sigma_{key_i})$, for $i = 0, \dots, no\_keys - 1$

## Template Matching

**Multivariate Gaussian p.d.f. for *m* POIs:**

$$f(l_1, l_2, \ldots, l_m) = \frac{1}{\sqrt{(2\pi)^m * det(\Sigma)}} exp[-\frac{1}{2}((\mathrm{l}_{\mathrm{attack}} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathrm{l}_{\mathrm{attack}} - \boldsymbol{\mu}))]$$

- Observe a trace $\mathrm{l}_{\mathrm{attack}}$
- Match the trace $\mathrm{l}_{\mathrm{attack}}$ to the templates for $key_0$ and $key_1$
- Multivariate Likelyhood Ratio Test:
- $\Lambda(\mathrm{l}_{\mathrm{attack}}) = \frac{1}{2} * (\mathrm{l}_{\mathrm{attack}} - \boldsymbol{\mu}_{key_0})^T * inv(\Sigma_{key_0}) * (\mathrm{l}_{\mathrm{attack}} - \boldsymbol{\mu}_{key_0}) - \frac{1}{2} * (\mathrm{l}_{\mathrm{attack}} - \boldsymbol{\mu}_{key_1})^T * inv(\Sigma_{key_1}) * (\mathrm{l}_{\mathrm{attack}} - \boldsymbol{\mu}_{key_1})$
- $\gamma = \frac{1}{2} * log(det(\Sigma_{key_1})) - \frac{1}{2} * log(det(\Sigma_{key_0}))$
- $\Lambda(\mathrm{t}) \underset{H_0}{\overset{H_1}{\gtrless}} \gamma$

## Final recap on Profiling attacks

- We saw three models for template attacks: Reduced model, Univariate Gaussian, Multivariate Gaussian
- The two phases in profiling are:
  - Template Building
  - Template Matching
- Profiling attacks are actually a sort of supervised learning
- As a consequence, machine and deep learning are commonly used in side-channel attacks (more in the next lecture)

## Literature

- Harry van Trees: Detection, Estimation and Modulation Theory.
- Steven M. Kay: Fundamentals of Statistical Signal Processing, Volume I and II.
- R.Larsen, M.Marx: Introduction to Mathematical Statistics.
- Suresh Chari, Josyula R. Rao, Pankaj Rohatgi: Template Attacks. CHES 2002: pp. 13-28.
- Cedric Archambeau, Eric Peeters, Franois-Xavier Standaert, Jean-Jacques Quisquater: Template Attacks in Principal Subspaces. CHES 2006: pp. 1–14 [Easy-to-use formulas for template construction]
- Choudary and Kuhn: Efficient, Portable Template Attacks. IEEE TIFS 2018.
- L. Batina, M. Djukanovic, A. Heuser and S. Picek: It Started with Templates: The Future of Profiling in Side-Channel Analysis.

## SCA and AI

## AI for Side-channel Analysis (SCA): Why

- Machine learning for SCA was a natural direction:
  - PCA to assist profiling/template attacks (dimensionality reduction)
  - PCA for pre-processing measurement traces
  - Machine Learning (ML)-based SCA distingushers
- Deep learning in SCA:
  - neural nets for profiling attacks
  - defeating countermeasures (e.g. skip the alignment phase)
  - leakage assessment/simulators
  - TEMPEST-like techniques e.g. screen gleaning
  - location-based SCA

## AI for SCA: When

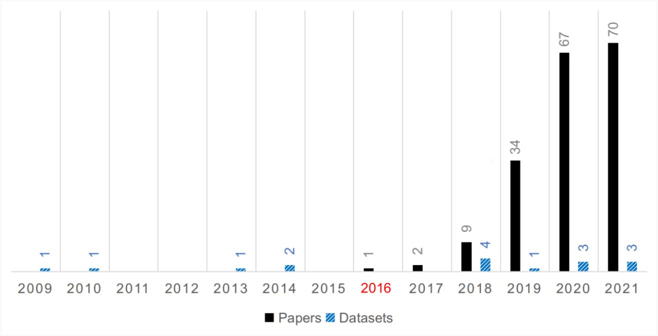


**Figure:** Deep learning papers and datasets.

S. Picek, G. Perin, L. Mariot, L. Wu and L. Batina, SoK: Deep Learning-based Physical Side-channel Analysis, `https://eprint.iacr.org/2021/1092`, accepted at ACM Computing Surveys, 2022.

## Deep-learning in SCA

## Universal approximation theorem

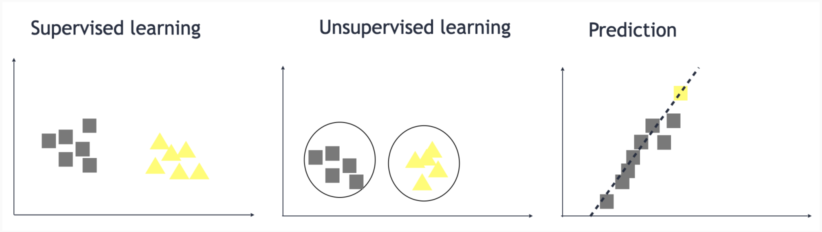
Any neural networks aims to find a function $f(x) = y$, which can maps input $x$ to output $y$;

Universal Approximation Theorem:[1]a well-guided and engineered deep neural network can approximate any arbitrary complex and continuous $f$ among the input variable $x$ and the output variable $y$.

- "Prodigy" of machine learning algorithms, powered by the availability of cheap large scale computational resources;
- Good performance in reducing the feature engineering step in SCA (signal processing, feature reduction, etc.)
- From 2017, certification bodies mandate the application of deep learning in side channel analysis;

[1]Kurt Hornik, Maxwell Tinchcombe, Halbert White *Multilayer Feedforward Networks are Universal Approximators* Neural Networks. Vol. 2. Pergamon Press. pp. 353–366,(1989).

## Applications for machine learning



- Supervised learning: the machine learns with a supervisor, for every example we tell the machine what the correct answer is;
- Unsupervised learning: the machine discovers hidden patterns in the data; training is done on unlabelled data;
- Prediction: the machine predicts the future, based on past events;
- Supervised learning is the most common application of machine learning for side channel analysis.

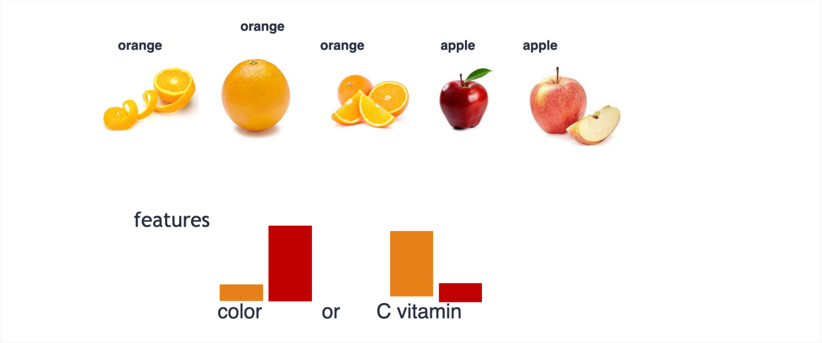
## A simple classification machine

Problem: we want to create a machine which can distinguish between apples and oranges.

- We need to describe to the machine the concept of oranges and apples.
- Features are measurable properties which describe the objects we want to classify.
- Choosing the correct features is crucial for the performance of any machine learning algorithm.

## A simple classification machine

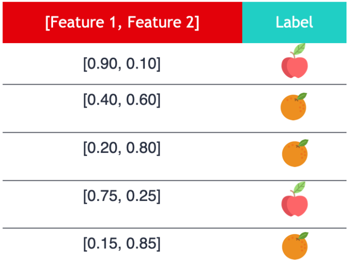
Problem: we want to create a machine which can distinguish between apples and oranges.



We will describe every orange and apple in terms of the level of colour (e.g. ratio of red vs. orange) and C-vitamin which we measured in each fruit.

## A simple classification machine

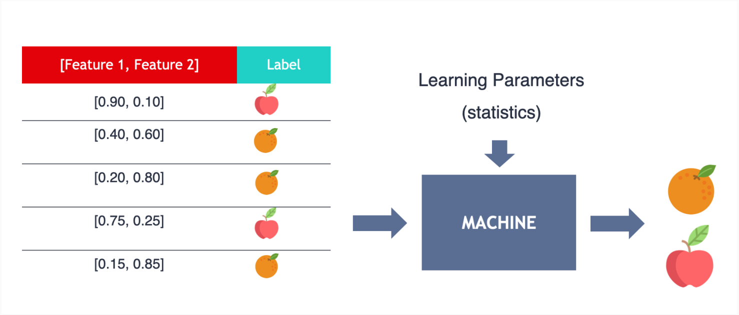
The labelled dataset, $\mathcal{D}$, will look like this (simplified example):



| [Feature 1, Feature 2] | Label |
|---|---|
| [0.90, 0.10] | 🍎 |
| [0.40, 0.60] | 🍊 |
| [0.20, 0.80] | 🍊 |
| [0.75, 0.25] | 🍎 |
| [0.15, 0.85] | 🍊 |

- The effort of creating a labelled dataset should not be underestimated
- Before creating your own dataset, check for a publicly available dataset

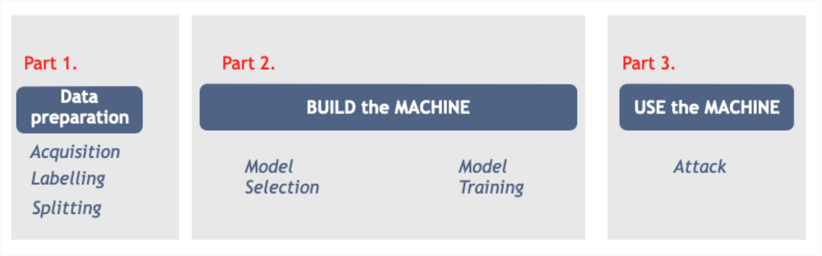
## Workflow deep learning



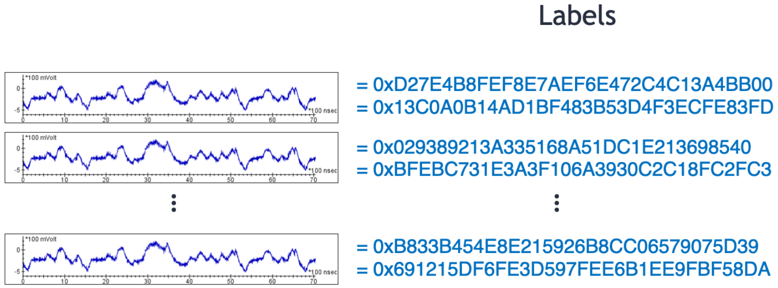
Machine is a function $f(x, \theta) = y$, where $x$ is the input, $y$ are the labels and $\theta$ are parameters [2] of the machine;

[2] the coefficients (weights and bias) of the model, optimized during training

## Workflow machine learning



## Steps in preparing data for SCA evalution


Labels
= 0xD27E4B8FEF8E7AEF6E472C4C13A4BB00
= 0x13C0A0B14AD1BF483B53D4F3ECFE83FD
= 0x029389213A335168A51DC1E213698540
= 0xBFEBC731E3A3F106A3930C2C18FC2FC3
= 0xB833B454E8E215926B8CC06579075D39
= 0x691215DF6FE3D597FEE6B1EE9FBF58DA
*100 mVolt
*100 nsec

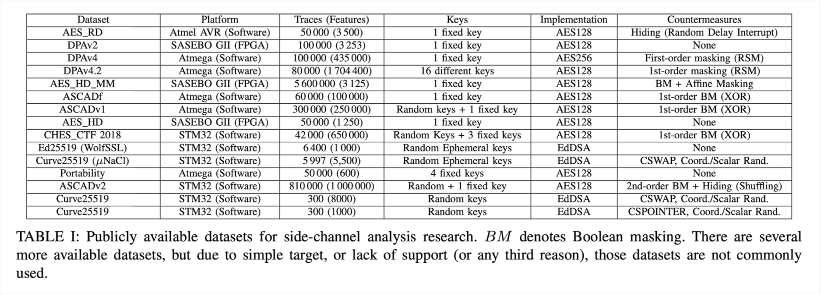
(1) Acquisition, collecting traces for profiling a model from a given target;

(2) Labelling, the process of annotating the side-channel traces with a label. The label is determined by the choice of the target intermediate and leakage model.

(3) Splitting, the data is split into training and testing sets.

## Public datasets

Available datasets for side-channel analysis. Saves the acquisition of the traces, labelling is performed for each attack.



| Dataset | Platform | Traces (Features) | Keys | Implementation | Countermeasures |
|---|---|---|---|---|---|
| AES_RD | Atmel AVR (Software) | 50 000 (3 500) | 1 fixed key | AES128 | Hiding (Random Delay Interrupt) |
| DPAv2 | SASEBO GII (FPGA) | 100 000 (3 253) | 1 fixed key | AES128 | None |
| DPAv4 | Atmega (Software) | 100 000 (435 000) | 1 fixed key | AES256 | First-order masking (RSM) |
| DPAv4.2 | Atmega (Software) | 80 000 (1 704 400) | 16 different keys | AES128 | 1st-order masking (RSM) |
| AES_HD_MM | SASEBO GII (FPGA) | 5 600 000 (3 125) | 1 fixed key | AES128 | BM + Affine Masking |
| ASCADf | Atmega (Software) | 60 000 (100 000) | 1 fixed key | AES128 | 1st-order BM (XOR) |
| ASCADv1 | Atmega (Software) | 300 000 (250 000) | Random keys + 1 fixed key | AES128 | 1st-order BM (XOR) |
| AES_HD | SASEBO GII (FPGA) | 50 000 (1 250) | 1 fixed key | AES128 | None |
| CHES_CTF 2018 | STM32 (Software) | 42 000 (650 000) | Random Keys + 3 fixed keys | AES128 | 1st-order BM (XOR) |
| Ed25519 (WolfSSL) | STM32 (Software) | 6 400 (1 000) | Random Ephemeral keys | EdDSA | None |
| Curve25519 ($\mu$NaCl) | STM32 (Software) | 5 997 (5,500) | Random Ephemeral keys | EdDSA | CSWAP, Coord./Scalar Rand. |
| Portability | Atmega (Software) | 50 000 (600) | 4 fixed keys | AES128 | None |
| ASCADv2 | STM32 (Software) | 810 000 (1 000 000) | Random + 1 fixed key | AES128 | 2nd-order BM + Hiding (Shuffling) |
| Curve25519 | STM32 (Software) | 300 (8000) | Random keys | EdDSA | CSWAP, Coord./Scalar Rand. |
| Curve25519 | STM32 (Software) | 300 (1000) | Random keys | EdDSA | CSPOINTER, Coord./Scalar Rand. |

TABLE I: Publicly available datasets for side-channel analysis research. $BM$ denotes Boolean masking. There are several more available datasets, but due to simple target, or lack of support (or any third reason), those datasets are not commonly used.

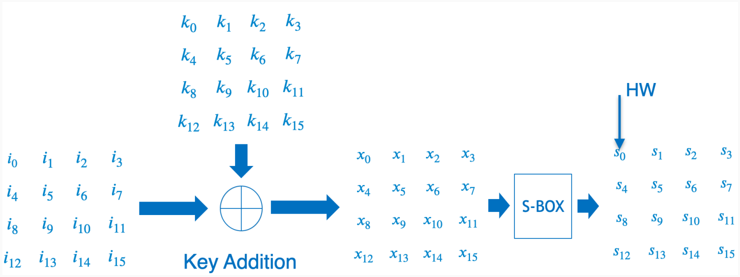
- Few datasets for hardware implementations, existing datasets built with fixed keys;

Source for the table: Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu and Lejla Batina, *SoK: Deep Learning-based Physical Side-channel Analysis*, 2021;

## Data preparation for SCA

Example (labelling):

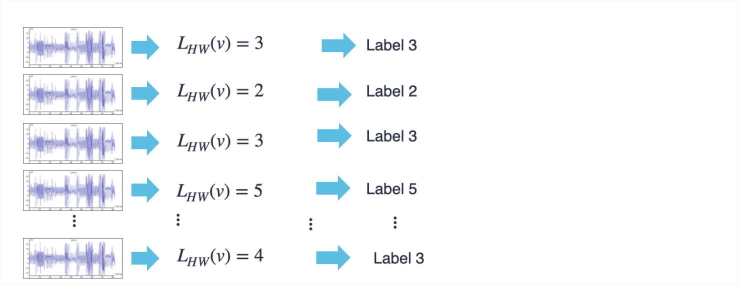
- The target algorithm is an AES-128 encryption function;
- The target intermediate ($v$) is `byte 0` of the `S-box out` operation in `round 1`;
- The leakage model function ($L$) of choice is the Hamming Weight;



$k_0$ $k_1$ $k_2$ $k_3$
$k_4$ $k_5$ $k_6$ $k_7$
$k_8$ $k_9$ $k_{10}$ $k_{11}$
$k_{12}$ $k_{13}$ $k_{14}$ $k_{15}$
$i_0$ $i_1$ $i_2$ $i_3$
$i_4$ $i_5$ $i_6$ $i_7$
$i_8$ $i_9$ $i_{10}$ $i_{11}$
$i_{12}$ $i_{13}$ $i_{14}$ $i_{15}$
Key Addition
$x_0$ $x_1$ $x_2$ $x_3$
$x_4$ $x_5$ $x_6$ $x_7$
$x_8$ $x_9$ $x_{10}$ $x_{11}$
$x_{12}$ $x_{13}$ $x_{14}$ $x_{15}$
S-BOX
HW
$s_0$ $s_1$ $s_2$ $s_3$
$s_4$ $s_5$ $s_6$ $s_7$
$s_8$ $s_9$ $s_{10}$ $s_{11}$
$s_{12}$ $s_{13}$ $s_{14}$ $s_{15}$

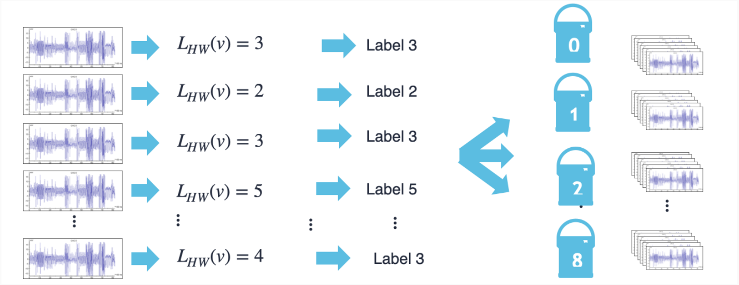
## Data preparation for SCA

For each trace in $\mathcal{D}$ we calculate the associated label, according to the chosen target intermediate $v$ and desired leakage model $L$.



## Data preparation for SCA

Next, we place each labelled trace in the corresponding bucket.



## Splitting the data

Once the dataset $\mathcal{D}$ has been labelled, we need to split it into three subsets:

- Training set, $\mathcal{D}_{train}$, is the set of examples used to fit $\theta$.
- Validation set, $\mathcal{D}_{val}$, provides an unbiased evaluation of the model (this data has not been seen) and it is used to tune the hyper-parameters of the machine[3];
- Testing set, $\mathcal{D}_{test}$, is a set of examples used to assess the machine performance;
- Split the training data into $\mathcal{D}_{train}$ and $\mathcal{D}_{val}$, the ratio is typically 80% training and 20% validation, but can vary;

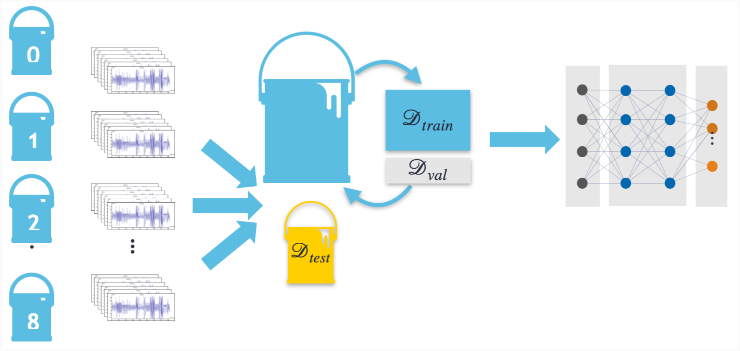
We know that the following holds for the three subsets:

- $\mathcal{D}_{train} \cup \mathcal{D}_{val} \cup \mathcal{D}_{test} = \mathcal{D}$
- $\mathcal{D}_{train} \cap \mathcal{D}_{val} \cap \mathcal{D}_{test} = \emptyset$

[3] hyper-parameters are set manually and not updated during training

## Data preparation for SCA

A visual overview of how data is split:



- Over-fitting an undesired phenomena, when the performance of the model on the training data is very good, while the performance on testing data is poor;
- Cross-validation the process of splitting the training data repeatedly into training and validation sets for more reliable results, which avoid over-fitting;
- Under-fitting when the model does not produce accurate results on the training data;

## The Portability Problem

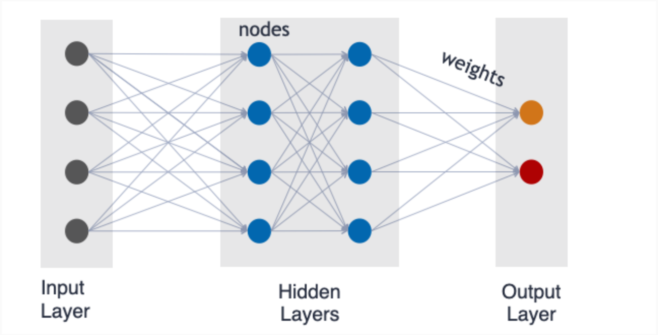
The Portability Problem: the ability to transfer an attack performed on one device to a different device.

- Single-device model, the training and the testing set are collected from the same device:
  - (+) Very popular in academia;
  - (+) Represents the best case for an attacker (least amount of noise);
  - (+) Easy to setup - requires only one device;
  - (-) It is not very realistic;
- Cross-device model, attacker has access to a device with unknown variations to the one he is attacking. Several flavours:
  - identical devices, different physical instances;
  - homogeneous devices, same chip different configurations;
  - heterogeneous devices, same chip, different manufacturers[4];

[4]different implementation for the Instruction Set Architecture (ISA)

## A simple MLP architecture

A simple multilayer perceptron (MLP) neural network architecture, which contains a series of layers formed of connected neurons. The strength of the connection between two neurons is determined by the associated weight.

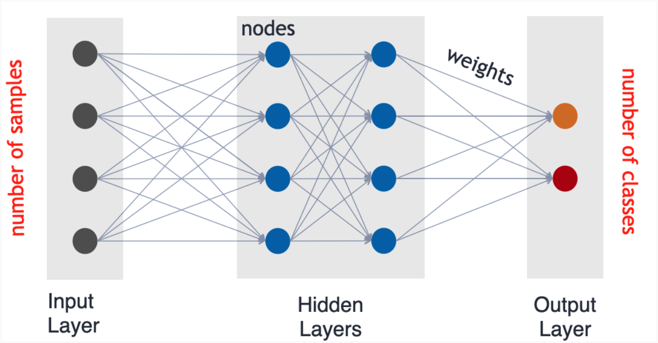


- During training the value of the weights and biases are adjusted;
- In SCA, we use relatively small networks and simple arch.: MLP and CNN[5];

[5] Convolutional Neural Network

## A simple MLP architecture

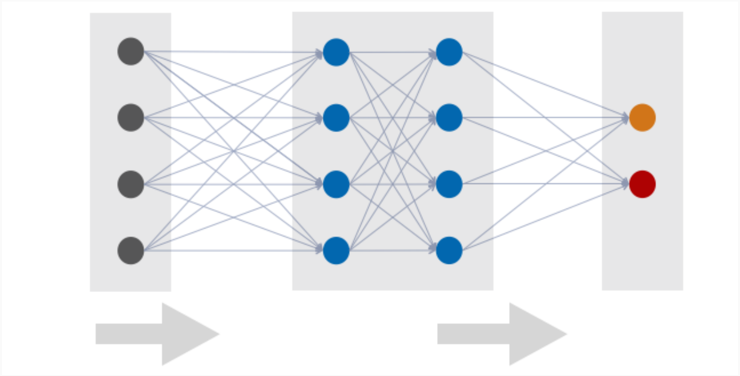
A simple multilayer perceptron (MLP) neural network architecture, which contains a series of layers formed of connected neurons. The strength of the connection between two neurons is determined by the associated weight.



**SCA-context:**

- # nodes in the input layer = # of samples in a trace;
- # nodes in the output layer = # the number of labels (a.k.a. classes);
- We only need to make decisions about the hidden layer;

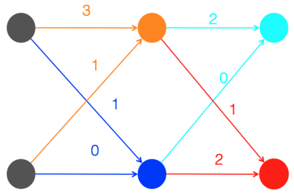
## Mechanics of training



- Forward propagation an algorithm which transforms the input data into output data using the model 's current state;
- Backward propagation an algorithm which adjusts the parameters of the network (weights and bias) to reduce error rates and improve prediction.
- An epoch is one full pass through the training data.

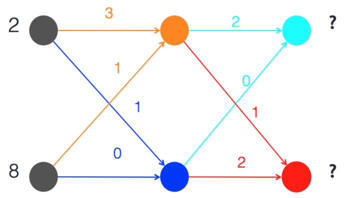
## Forward propagation example

Example: a simple MLP with three layers. Each layers has two nodes. The network has been initialized with random weights.



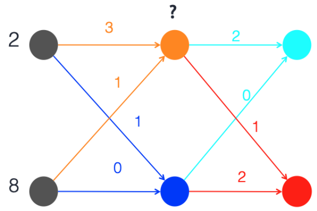
## Forward propagation example

Example: a simple MLP with three layers. Each layers has two nodes. The network has been initialized with random weights.
If we give the MLP the input $[2, 8]$, what is the output?



## Forward propagation example

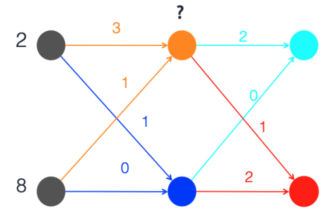
Example: a simple MLP with three layers. Each layers has two nodes. The network has been initialized with random weights.
If we give the MLP the input $[2, 8]$, what is the output?



(1) we compute the value of the orange node, as follows:

## Forward propagation example

Example: a simple MLP with three layers. Each layers has two nodes. The network has been initialized with random weights.
If we give the MLP the input $[2, 8]$, what is the output?



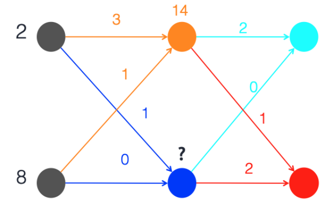
3*2+ 1*8= 14

(1) we compute the value of the orange node, $n^l_j$ as follows: $n^l_j = \sum_i w_i * n^{l-1}_i + b_i$, where:

- $l$ = represents current layer;
- $n^l_j$ = represents node $j$ in layer $l$;
- $w_i$ = represents the weight connecting $n^{l-1}_i$ to $n^l_j$
- $b_i$ = bias (optional)

## Forward propagation example

Example: a simple MLP with three layers. Each layers has two nodes. The network has been initialized with random weights.
If we give the MLP the input $[2, 8]$, what is the output?



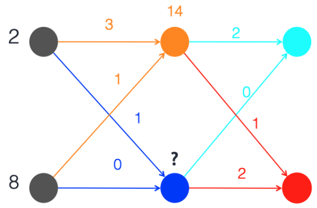
3*2+ 1*8= 14

(1) we compute the value of the orange node, the result is 14;

(2) we compute the value of the blue node, same as above;

## Forward propagation example

Example: a simple MLP with three layers. Each layers has two nodes. The network has been initialized with random weights.
If we give the MLP the input $[2, 8]$, what is the output?



$3 * 2 + 1 * 8 = 14$

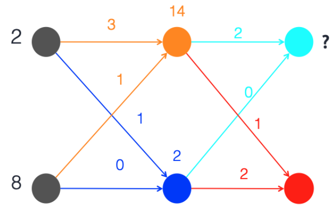
$1 * 2 + 8 * 0 = 2$

(1) we compute the value of the orange node, the result is 14;

(2) we compute the value of the blue node, the same way;

## Forward propagation example

Example: a simple MLP with three layers. Each layers has two nodes. The network has been initialized with random weights.
If we give the MLP the input $[2, 8]$, what is the output?



3*2+ 1*8= 14

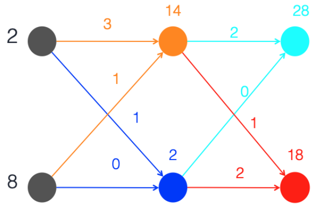
1*2+ 8*0= 2

(1) we compute the value of the orange node, the result is 14;

(2) we compute the value of the blue node, the same way;

## Forward propagation example

Example: a simple MLP with three layers. Each layers has two nodes. The network has been initialized with random weights.
If we give the MLP the input [2, 8], what is the output?



3*2+ 1*8= 14

1*2+ 8*0= 2

14*2+ 2*0=28

14*1+ 2*2=18

(1) we compute the value of the orange node, the result is 14;
(2) we compute the value of the blue node, the result is 2;
(3) we compute the value of the two output nodes, and we obtain the result 28 and 18 respectively;

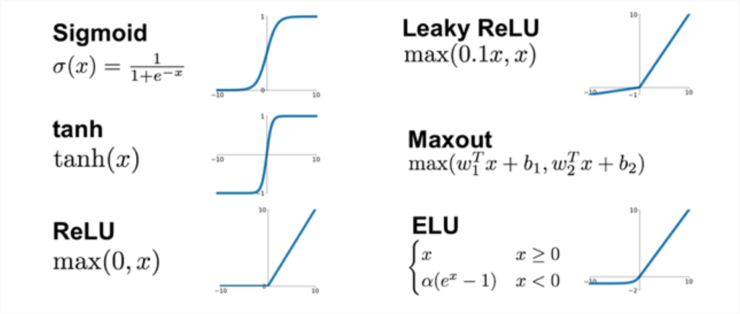
Answer: if we give the network the input [2, 8], the output is [28, 18].

Question: Do you see any limitations for our machine? What type of problems can such a machine solve?

## Activation functions

Question: How can we make the machine solve non-linear problems?

Answer: make use of activation functions. Example of common activation functions:



**Sigmoid**
$\sigma(x) = \frac{1}{1+e^{-x}}$

**tanh**
$\tanh(x)$

**ReLU**
$\max(0, x)$

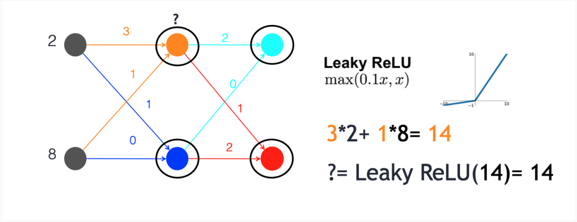
**Leaky ReLU**
$\max(0.1x, x)$

**Maxout**
$\max(w_1^T x + b_1, w_2^T x + b_2)$

**ELU**
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

## Solving non-linear problems

Forward propagation with activation functions:

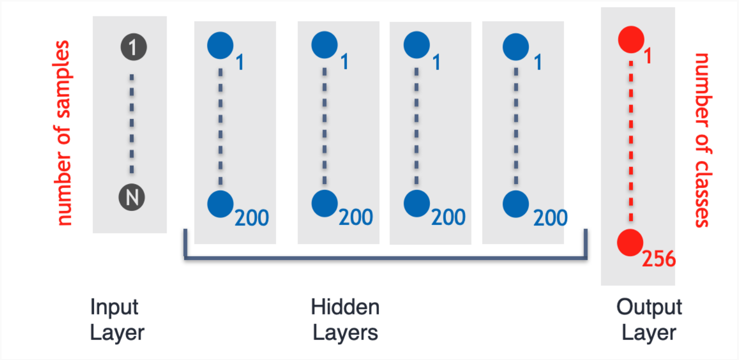


We compute the value of node $n_j^l$ as follows: $n_j^l = \sigma(\sum_i w_i * n_i^{l-1} + b_i)$, where:

- $\sigma$ is the activation function;
  - **(SCA)** common activation functions for hidden layers: `tanh`, `relu`, `selu`;
  - **(SCA)** for the output layer, we use `softmax` activation as we are dealing with a multi-class classification problem;
- $l$ = represents current layer;
- $n_j^l$ = represents node $j$ in layer $l$;
- $w_i$ = represents the weight connecting $n_i^{l-1}$ to $n_j^l$
- $b_i$ = bias (optional)

## Model Selection

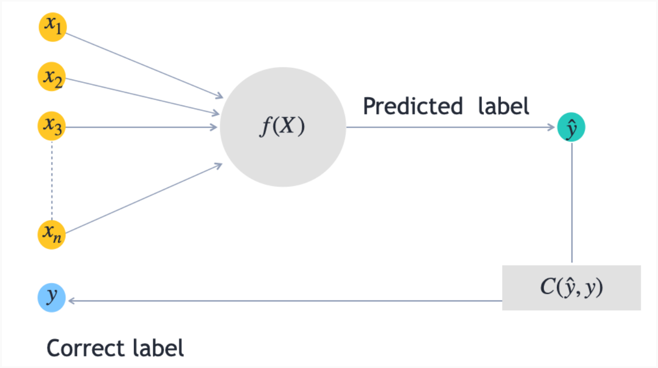
An example of a simple MLP architecture used for side-channel analysis:



Source: Guilherme Perin, Lichao Wu, Stjepan Picek, AISY – *Deep Learning-based Framework for Side-channel Analysis*, 2021; `https://eprint.iacr.org/2021/357`

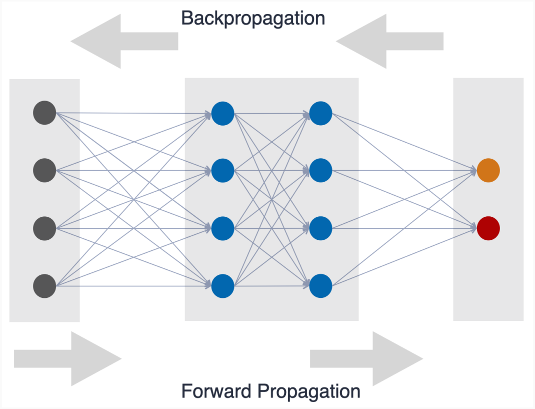
## Cost functions

Cost function is a measure of how well the network did in predicting the labels of the input value $(x_1, x_2, x_3, .., x_n)$.



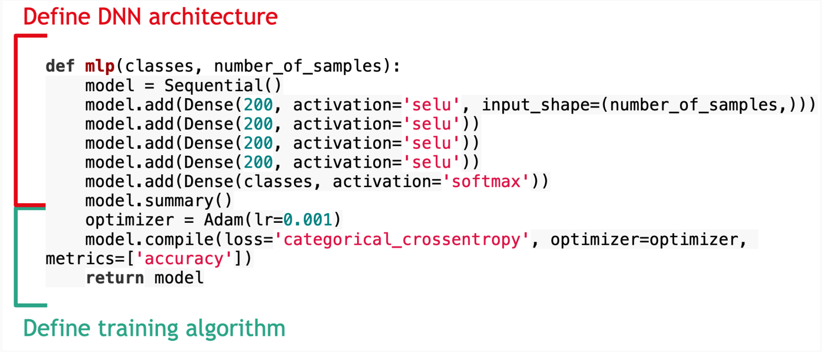
- Mean Square Error(MSE): $C(\hat{y}, y) = \frac{1}{c}\sum_{i=1}^{c}(\hat{y}_i - y_i)^2$
- Categorical Cross-Entropy (CE): $C(\hat{y}, y) = -\sum_{i=1}^{c} y_i \log(\hat{y}_i)$, use this for SCA;
- Custom cost-functions for SCA exist, e.g. ranking loss, cross-entropy ration, etc;

## How does a machine "learn"?



- Optimizer iterative algorithm which minimize the cost function, updates the parameters $\theta$ in response to the output of the cost function (e.g. Adaptive Moment Estimation (ADAM))
- Learning rate hyper-parameter which determines the step size at each iteration while moving toward a minimum of the cost function.
- Metrics recorded after each epoch, shows the performance of the model during training;

## Example code

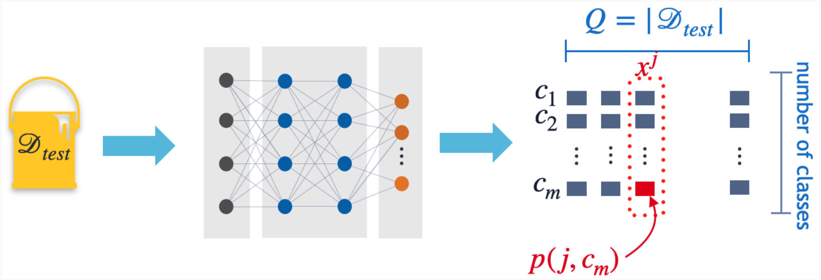


```python
def mlp(classes, number_of_samples):
    model = Sequential()
    model.add(Dense(200, activation='selu', input_shape=(number_of_samples,)))
    model.add(Dense(200, activation='selu'))
    model.add(Dense(200, activation='selu'))
    model.add(Dense(200, activation='selu'))
    model.add(Dense(classes, activation='softmax'))
    model.summary()
    optimizer = Adam(lr=0.001)
    model.compile(loss='categorical_crossentropy', optimizer=optimizer,
metrics=['accuracy'])
    return model
```

Source: Guilherme Perin, Lichao Wu, Stjepan Picek, *AISY – Deep Learning-based Framework for Side-channel Analysis*, 2021;for an overview on a DNN framework for SCA;

## Computing a score function

The test set $\mathcal{D}_{test}$ with $Q$ traces, where $x^j \in \mathcal{D}_{test}$ is a trace. The number of classes is denoted with $m$. [6]



The result of the DNN evaluation is a matrix of size $m \times Q$, where $p(j, c_m)$ is the probability of classifying trace $x^j$ with label $c_m$.
The score function for class $c_i$ given $Q$ traces in the test set is calculated as:

$$s(c_i, Q) = \sum_{j=1}^{Q} \log p(x^j, c_i)$$

[6] which matches the number of possible values for the target intermediate.

## Open problems

- What is an optimal network?
  - optimize attack: extract the key with as little traces as possible?
  - optimize training effort: reuse a network which was trained on one dataset to attack another;
- Explainable AI or what are the features which caused the decision of the DNN:
  - is an attack is NOT successful, why is this the case, i.e. the implementation is strong enough or the model is not effective?
  - is an attack is successful, what is the cause of a leak, i.e. how to improve the implementation?
- Can DNN successfully attack hardware implementations?
- Can DNN successfully attack high-order crypto implementations?
- What is the best metric to evaluate the performance of a DNN used for SCA?
- Can SCA attacks be successful in an unsupervised scenario?

## Screen Gleaning

## TEMPEST: Cause and History

Oscillating electric currents create EM radiation in the RF range and those signal drive the video display of various screens.

- Bell Labs noted this vulnerability for teleprinter communications during World War II producing 75% of the plaintext being processed from a distance of 24 $m$
- Van Eck phreaking: In 1985 published the first unclassified analysis of the security risks of emanations from computer monitors using just 15$ equipment+TV set
- Van Eck phreaking was used to successfully compromise ballot secrecy for electronic voting in Brazil
- NSA published TEMPEST Fundamentals in 1982 referring to spying on systems through leaking emanations, including radio or el. signals, sounds and vibrations
- TEMPEST covers both methods to spy and to shield equipment against such spying

## Motivation and Outcomes

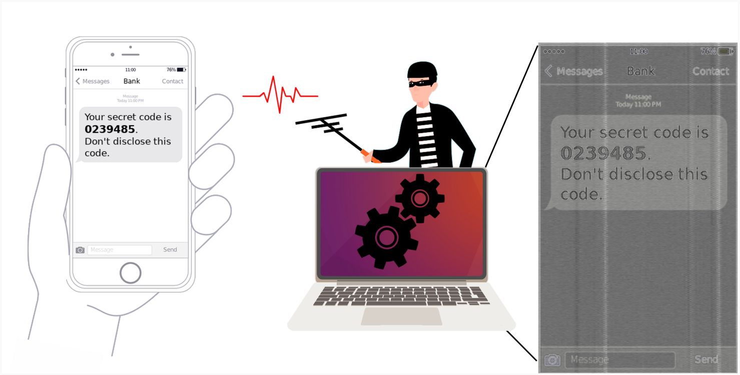
Motivation:

- TEMPEST attack is known for a long time but **no methodology** has been established to evaluate it **on mobile devices**
- Using TEMPEST the adversaries can reconstruct the images displayed through leaking emanations

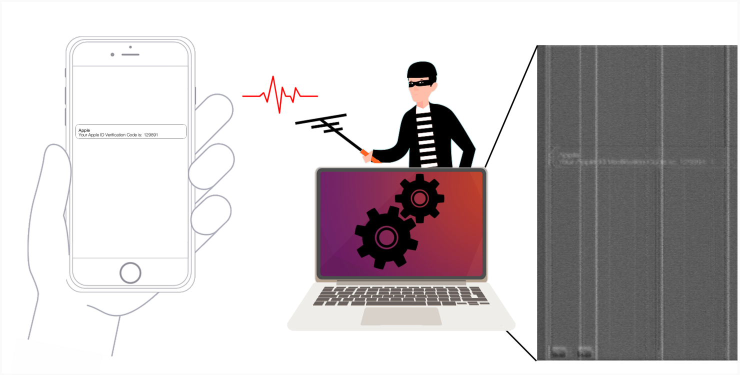
In this work we:

- Introduce Screen Gleaning, a new electromagnetic TEMPEST attack targeting mobile phones
- Demonstrate the attack and its portability to different targets using machine learning
- Provide a testbed and parameterized attacker model for further research

## Screen gleaning (Theory)



## Screen gleaning (Practise)



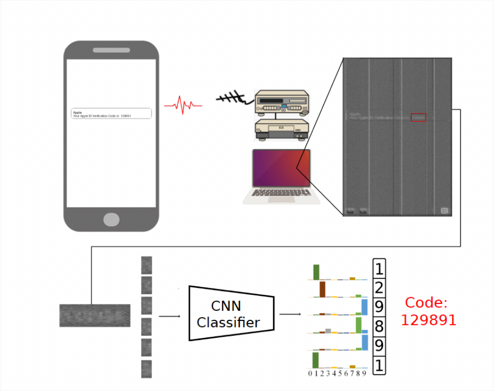
The signal we observe is, in most cases, not interpretable to the human eye.

## Attacker model: Motivating story and assumptions

*Alice keeps her phone on a stack of magazines on her desk (face down) to block the visual line of sight to the screen. Eve has hidden an antenna under the top magazine to read the security code via electromagnetic emanations of the phone.*

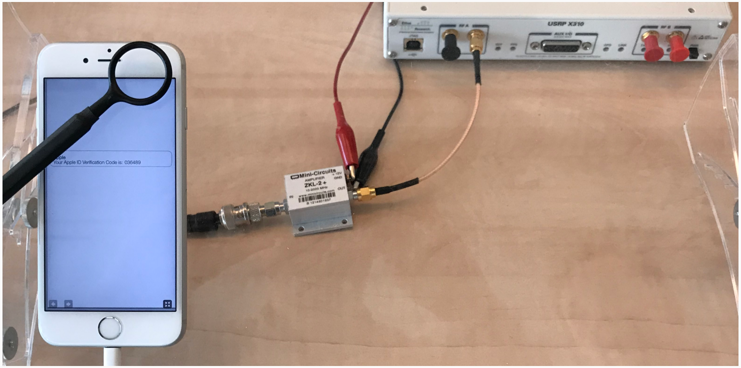
- The set of symbols displayed on the phone is finite and known (digits 0-9)
- The attacker has access to a profiling device that is "similar" to the target device
- The attacker can collect electromagnetic traces from the target device (representing the image displayed on the screen)

## Attacker model: setup



- The target emits EM signal intercepted by an antenna connected to a software-defined radio (SDR)
- The leaked information is collected and reconstructed as a gray-scale image (emage)
- From emage, the 6-digit security code is cropped and fed into a CNN classifier for recognition

## Screen gleaning setup

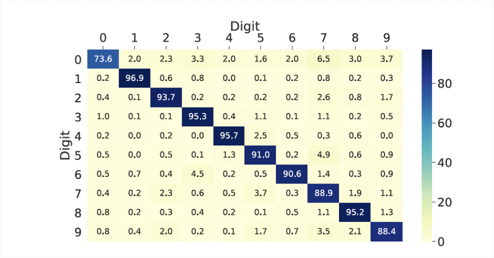


## The security code

Use of authentication code like to extract all digits.

- display code
- sample leakage
- analyze the leakage
- interpret the results

## Security code results



| Digit | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 73.6 | 2.0 | 2.3 | 3.3 | 2.0 | 1.6 | 2.0 | 6.5 | 3.0 | 3.7 |
| 1 | 0.2 | 96.9 | 0.6 | 0.8 | 0.0 | 0.1 | 0.2 | 0.8 | 0.2 | 0.3 |
| 2 | 0.4 | 0.1 | 93.7 | 0.2 | 0.2 | 0.2 | 0.2 | 2.6 | 0.8 | 1.7 |
| 3 | 1.0 | 0.1 | 0.1 | 95.3 | 0.4 | 1.1 | 0.1 | 1.1 | 0.2 | 0.5 |
| 4 | 0.2 | 0.0 | 0.2 | 0.0 | 95.7 | 2.5 | 0.5 | 0.3 | 0.6 | 0.0 |
| 5 | 0.5 | 0.0 | 0.5 | 0.1 | 1.3 | 91.0 | 0.2 | 4.9 | 0.6 | 0.9 |
| 6 | 0.5 | 0.7 | 0.4 | 4.5 | 0.2 | 0.5 | 90.6 | 1.4 | 0.3 | 0.9 |
| 7 | 0.4 | 0.2 | 2.3 | 0.6 | 0.5 | 3.7 | 0.3 | 88.9 | 1.9 | 1.1 |
| 8 | 0.8 | 0.2 | 0.3 | 0.4 | 0.2 | 0.1 | 0.5 | 1.1 | 95.2 | 1.3 |
| 9 | 0.8 | 0.4 | 2.0 | 0.2 | 0.1 | 1.7 | 0.7 | 3.5 | 2.1 | 88.4 |

**Figure:** Confusion matrix of the inter-session accuracy of the security.

| **Digits** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | All |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Acc. (%)** | 87.2 | 86.8 | 97.4 | 75.8 | 99.1 | 97.4 | 95.1 | 93.1 | 82.5 | 86.1 | 89.8 |

**Table:** Accuracy with respect to different digits (0-9) and overall accuracy in our security code attack.

|  | **6 digits** | $\geq$ 5 **digits** | $\geq$ 4 **digits** |
|---|---|---|---|
| **Acc. (%)** | 50.5 | 89.5 | 99.0 |

**Table:** Accuracy of predicting partial security code correctly.

## Security code results

- Attack on different phones of the same model
  E.g., cross-device accuracy of 61.5%, where the classifier is trained and tested on two distinct iPhone 6.
- Attack on different phone of different model
  E.g., accuracy of 74.0% on Huawei Honor 6X.
- Attack at a greater distance (through a magazine)
  E.g., accuracy of 65.8% on Huawei Honor 6X through 200 pages.

Z. Liu, Niels Samwel, L. Weissbart, Z. Zhao, D. Lauret, L. Batina, M. Larson, Screen Gleaning: A Screen Reading TEMPEST Attack on Mobile Devices Exploiting an Electromagnetic Side Channel, NDSS 2021.

## Conclusions

- Screen gleaning is a new TEMPEST attack that uses an antenna and SDR to capture an electromagnetic side channel, i.e., emanations leaking from a mobile phone
- We demonstrated the effectiveness of it on three different phones with an example of the recovery of a security code
- We introduced 5-dimension attacker model that can be extended further
- We proposed a testbed providing a standard setup in which screen gleaning can be tested further with different attacker models

## Further reading

- Jon Krohn, *Deep Learning Illustrated - A visual, interactive guide to artificial intelligence*, Addison-Wesley, 2020; provides a beginner friendly introduction to deep learning;
- Houssem Maghrebi and Thibault Portigliatti and Emmanuel Prouff, *Breaking Cryptographic Implementations Using Deep Learning Techniques*, SPACE 2016; https://eprint.iacr.org/2016/921
- Source: Guilherme Perin, Lichao Wu, Stjepan Picek, *AISY–Deep Learning-based Framework for Side-channel Analysis*, 2021; an overview on a DNN framework for SCA; https://eprint.iacr.org/2021/357
- Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu and Lejla Batina, *SoK: Deep Learning-based Physical Side-channel Analysis*, 2021; an overview of the developments and remaining challenges when using DNNs for SCA; https://eprint.iacr.org/2021/1092

## Conclusions

- Screen gleaning is a new TEMPEST attack that uses an antenna and SDR to capture an electromagnetic side channel, i.e., emanations leaking from a mobile phone
- We demonstrated the effectiveness of it on three different phones with an example of the recovery of a security code
- We introduced 5-dimension attacker model that can be extended further
- We proposed a testbed providing a standard setup in which screen gleaning can be tested further with different attacker models

## Location-based leakage

## Motivation

- Registers, memory or other storage units exhibit identifiable and data-independent leakage when accessed
- Exploits dependence between the secret key and the location of the activated component

**Algorithm 1:** Montgomery ladder

```
Input: $P, k = (k_{x-1}, k_{x-2}, ..., k_0)_2$
Output: $Q = k \cdot P$
  $R_0 \leftarrow P$
  $R_1 \leftarrow 2 \cdot P$
  for $i = x - 2$ downto 0 do
    $b = 1 - k_i$
    $R_b = R_0 + R_1$
    $R_{k_i} = 2 \cdot R_{k_i}$
  end for
  return $R_0$
```

## Location-based leakage revisited

- Distinguishing the activity of small regions
- Exploiting the spatial dependencies of crypto algorithms
- Different than localized and address leakages
- Forward Neural Networks classifiers exploiting location-based side-channel on the SRAM of a ARM Cortex-M4
- 2 SRAM regions of 128 bytes each can be distinguished with 100% success rate and 256 SRAM byte-regions with 32% success rate

## Adversarial model

- Implementation of a key-dependent crypto operation
- Adversary aims to infer which part of the table is active
- Location leakage that is caused by switching circuitry and is observable via EM emissions on the die surface

## Experimental setup

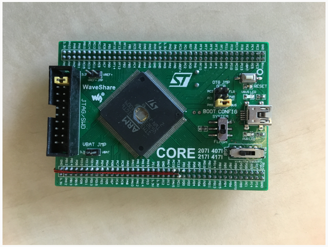


**Figure:** Modified Pinata ARM STM32F417IG device.

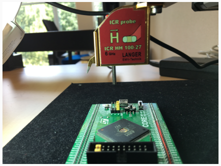


**Figure:** Decapsulated Pinata with Langer microprobe on top.

## Setup details

- Decapsulated Piñata with ARM Cortex-M4 in 90 *nm* technology
- ICR HH 100-27 Langer microprobe $d = 100\mu m$
- Rectangular grid of 300 $\times$ 300 measurement spots
- Sampling rate of 1 Gs/sec resulting in 170$k$ samples
- Near-field probe with positioning accuracy of 50 $\mu m$
- Sequential accesses to a cont. region of 16 KBytes in the SRAM using ARM assembly

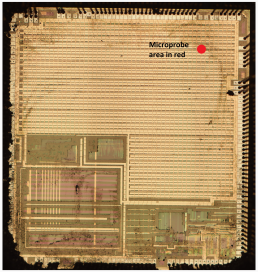


**Figure:** ARM Cortex-M4 after removal of the plastic layer.

## Experiment

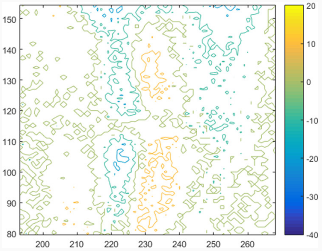


**Figure:** Distinguishing two 8 KByte regions of the SRAM. Yellow region = stronger leakage from class 1, blue = stronger from class 2.

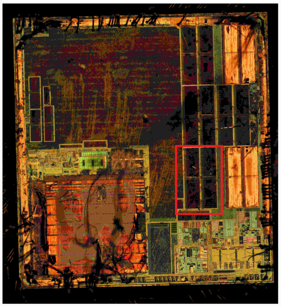
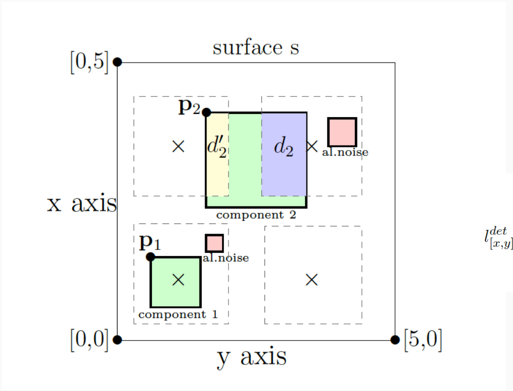


**Figure:** Red rectangle shows the location where the highest differences were observed.

## Parameters

| Parameter | Description | Unit | Our example |
|---|---|---|---|
| S | chip surface area | $u^2$ | $\leq 6$ $mm^2$ (whole chip) |
| O | probe area | $u^2$ | 0.03 $mm^2$ |
| G | scan grid dimension | – | 300 |
| A | component areas | vector with 1D entries | – |
| P | component positions | vector with 2D entries | – |

## Location Leakage Model

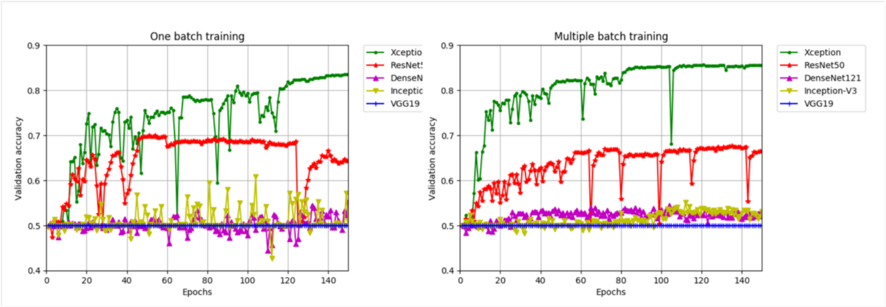


$$l^{det}_{[x,y]}|\mathbf{v}^i = \begin{cases} 0, & \text{if comp. } i \text{ is not captured at [x,y]} \\ d_i,\ 0 < d_i < a_i, & \text{if comp. } i \text{ is partially captured at [x,y]} \\ a_i, & \text{if comp. } i \text{ is fully captured at [x,y]} \end{cases}$$

**Figure:** Vectors $p_1$, $p_2$ show the position of two components whose areas ($a_1$, $a_2$) are solid black-line rectangles.

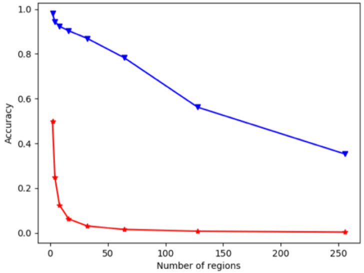
## Location-based attacks: CNNs

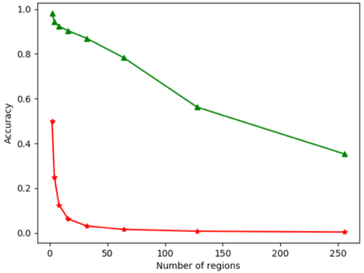
- Popular pre-trained networks Convolution Neural Network classifier – 2 regions, 128 bytes each
- Single-trace attacks improved compared to templates
- All 5 CNNs were trained in 2 ways



## Location-based Attacks: MLPs validation vs attack

- Custom Multi Layer Perceptrons in Riscure Inspector
- Single-trace attacks improve further
- Even smaller regions could be attacked





Left: validation accuracy for 2, 4, 8, 16, 32, 64, 128, and 256 partitions vs random guess.

Right: The attack success rates for the test traces for 2, 4, 8, 16, 32, 64, 128, and 256 partitions; the exact accuracy values are 96%, 91%, 90%, 88%, 83%, 75%, 57%, and 32%, respectively

## Summary on location-based SCA

- Location-based leakages could reveal secret keys, even for SCA protected implementations
- Even simple spatial model to capture location-based leakages can be effective
- Successful location-based attacks demonstrated on a modern ARM Cortex-M4 using standard template attacks and deep learning
- Attacks apply to PKC and AES implementations

## Lessons learned

- Deep learning can be useful for some (exotic) use cases of SCA but we should not use a cannon to kill a fly
- AI-assisted SCA attacks could be more powerful e.g. in the presence of countermeasures, noise etc.
- But, in many SCA evaluations “classical” techniques could be more efficient
- Screen gleaning takeaway: DL is often used for image recognition, which can have relevant implications for privacy: discrimination vs generalization scenario
- The role of AI in leakage detection and assessment should be investigated in detail

## The end

*Thank you for your attention!*

`https://cescalab.cs.ru.nl/`

## The end

*Thank you for your attention!*

`https://cescalab.cs.ru.nl/`