A Deep Dive into Deep Learning-based Side-channel Analysis

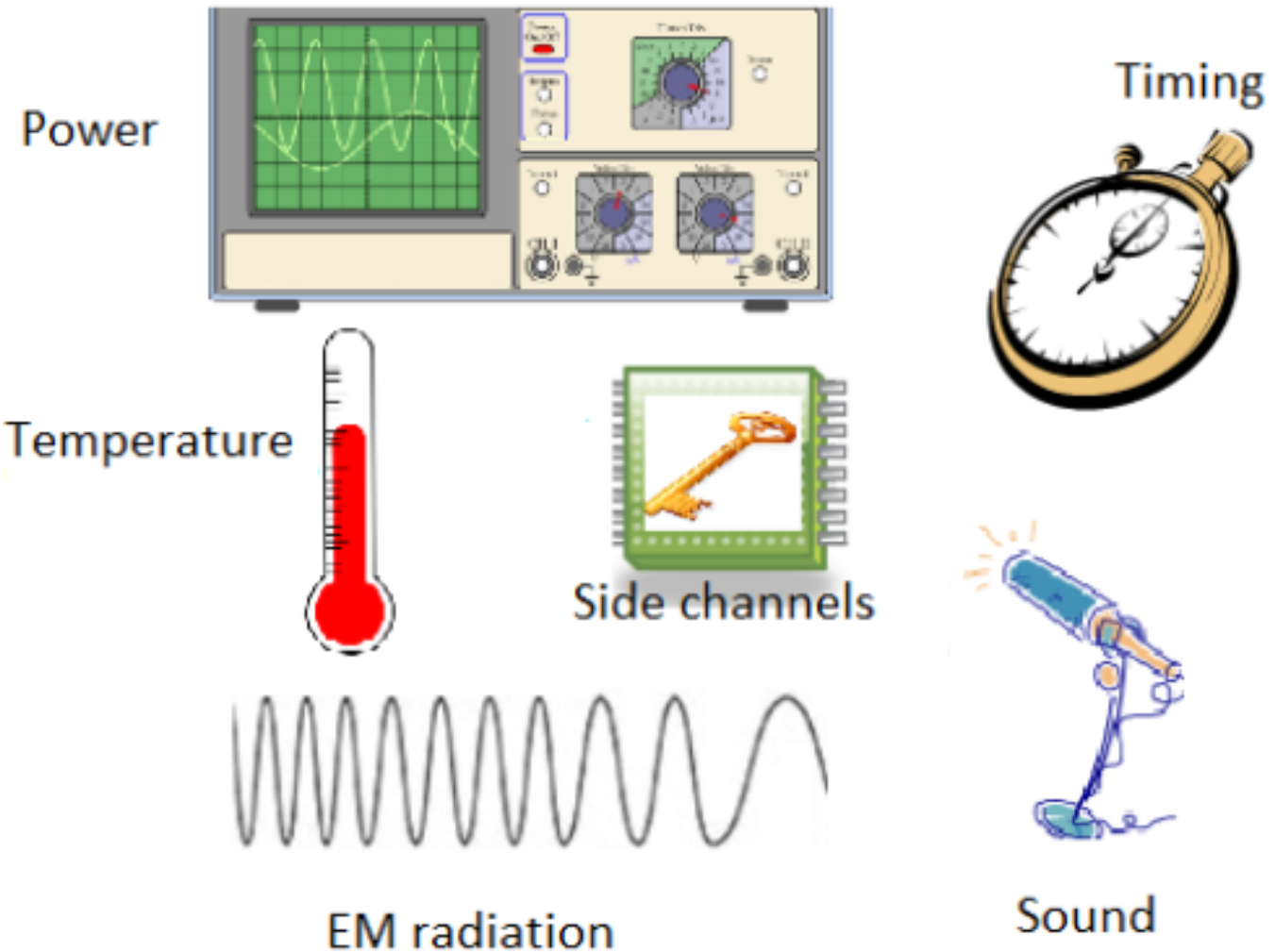
Stjepan Picek

SAC Summer School, 12.08.2025

Outline

Outline

Side Channels



Timing

- One of the earliest side-channel attacks due to easy measurements collection.
- Can also be exploited remotely.
- Exploit some not foreseen effects of caches to crypto implementations.
- Applied to symmetric and asymmetric cryptography.

Timing

- Software for PIN code verification

```
Input: 4-digit PIN code
Output: PIN verified or rejected
Process CheckPIN (pin[4])
int pin_ok=0;
if (pin[0]==5)
   if (pin[1]==9)
      if (pin[2]==0)
         if (pin[3]==2)
            pin_ok=1;
         end
      end
   end
end
return pin_ok;
EndProcess
```

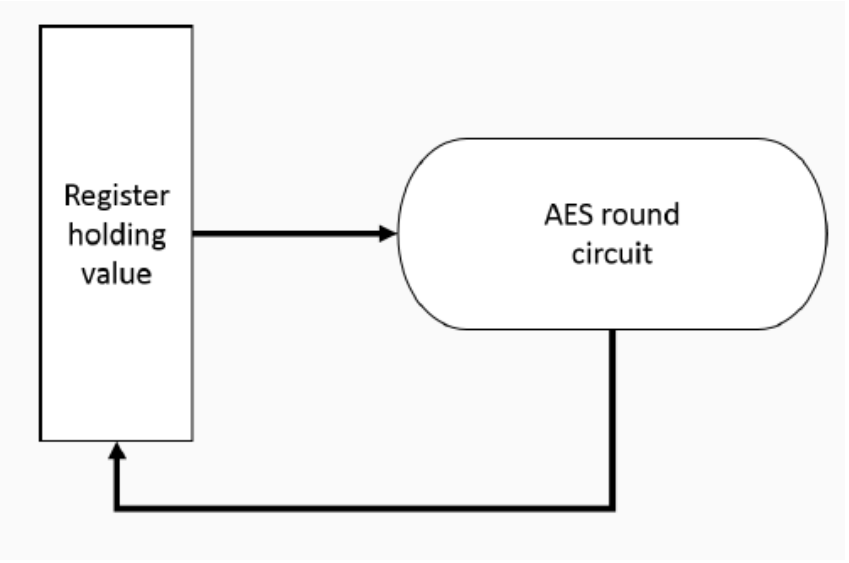
Power Consumption

- CMOS is one of the most popular technologies for chip design.
- CMOS circuits exhibit several types of leakage.
- Charge and discharge of the CMOS load capacitance leads to side-channel leakage (dynamic power consumption).
- Power analysis attack exploits the fact that the dynamic power consumption depends on the data and instructions being processed.
- Dynamic power consumption is produced by CMOS transitions from state 0 to 1 and from state 1 to 0.

How to Model the Leakage

- We use the number of transitions to model the leakage.
- The Hamming distance model counts the number of $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions.
- Example 1: A register R is storing the result of an AES round and initial value v_0 gets overwritten with 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.
- Modeled as HammingDistance(v0, v1) = $HammingWeight(v_0 \oplus v_1)$.
- Common leakage model for hardware implementations (FPGA, ASIC).

How to Model the Leakage

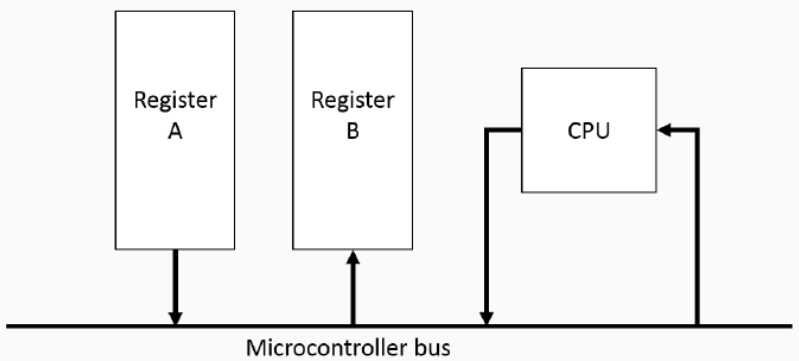


How to Model the Leakage

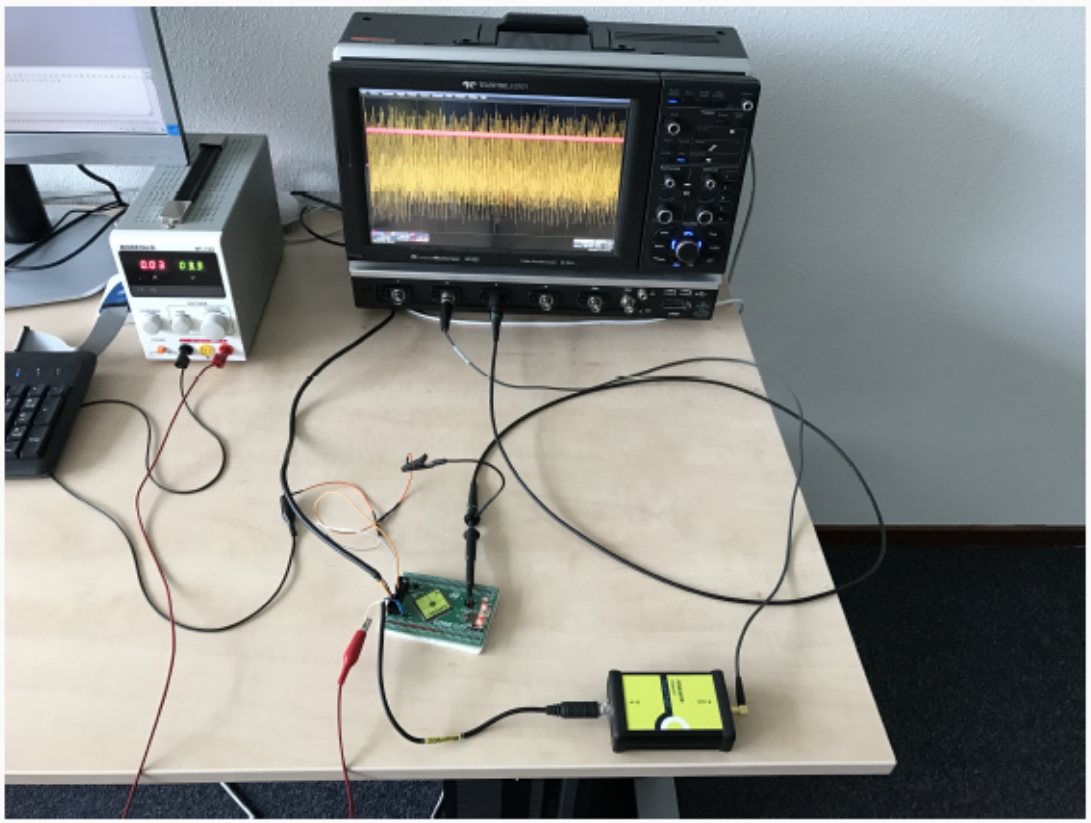
- Example 2: In a microcontroller, a register A contains value v_0 and an assembly instruction moves the content of register A to B.
- This instruction transfers v_0 from A to B via the CPU, using the bus.
- Typically the bus is precharged at all bits being zeros or one (busInitialValue).
- The power consumption of the instruction can be modeled as $HammingDistance(busInitialValue, v_0) = HammingWeight(v_0 \oplus 0) = HW(v_0)$.
- Common leakage model for software implementations (AVR/ARM).

How to Model the Leakage

```
mov rB, rA
```



Measurement Setup



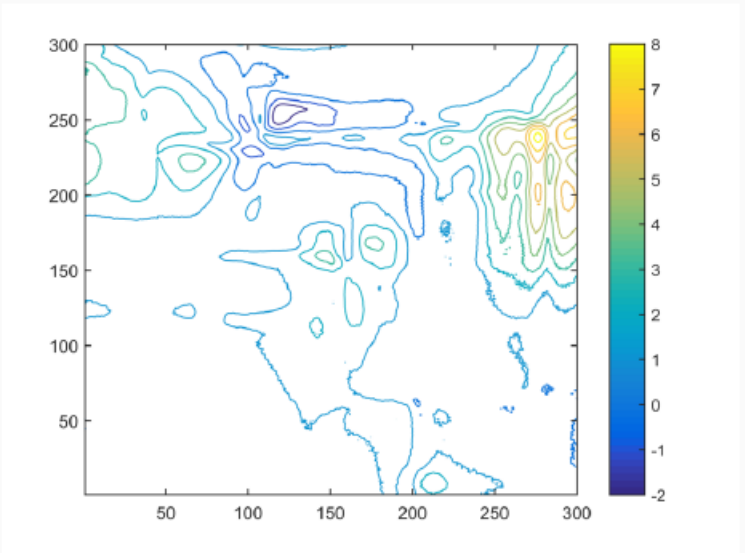
EM Side Channel: Probing

- Observing a power signal in more complex systems can be messy.
- Complicated SoCs with multiple peripherals.
- Countermeasures trying to flatten the power consumption signal.
- Use an electromagnetic probe instead.
- A probe is used to access the power consumption with less board modifications.
- Smaller probes can focus on interesting locations and ignore interference from unrelated electrical components.

EM Side Channel: Decapsulation and Microprobing

- To improve spatial resolution of analysis use a micrometer-sized antenna.
- To exploit more leakage decapsulate the chip using chemicals.
- EM enables side-channel attacks both in high proximity scenarios and distance scenarios.
- The main side channel for SoCs, FPGAs, contactless cards due to their complexity and communication methods.

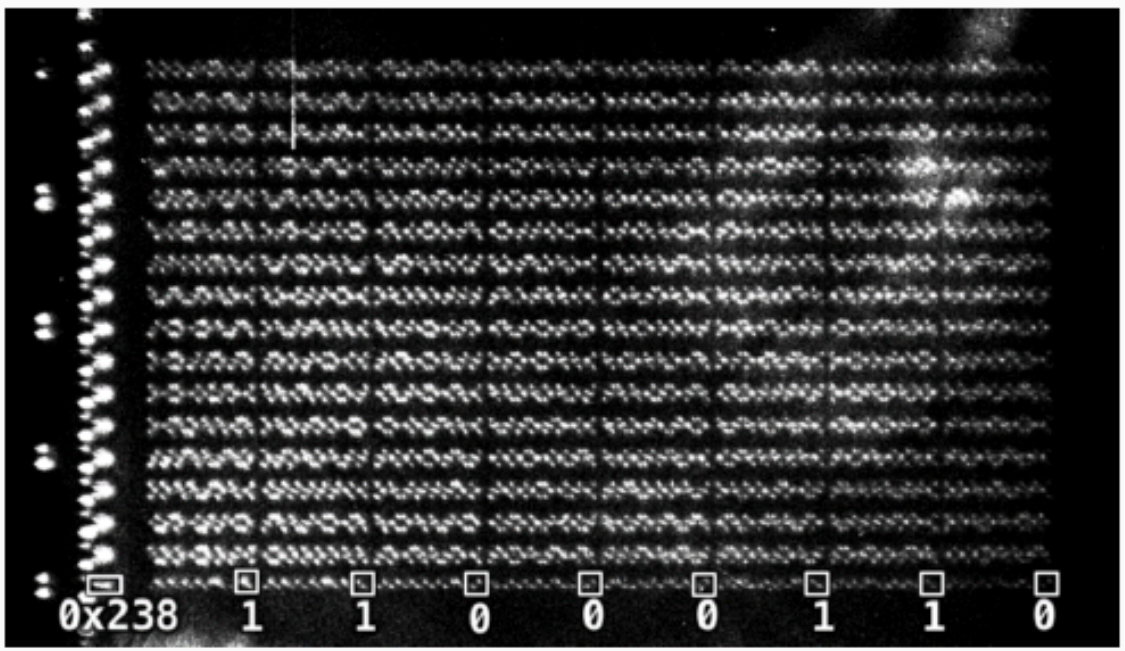
EM Side Channel: Decapsulation and Microprobing



Optical Emission

- Accessing the chip SRAM cells emits photons that can be detected by a high-resolution camera.
- Visual inspection can reveal the memory location accessed.
- The memory location maps to a specific value (e.g., in the AES LUT), i.e., it maps directly to $Sbox(in \oplus key)$.
- Since the input in is known, knowledge of the memory location reveals the key.

Optical Emission



Sound Emission

- Attacking a CPU by listening to the high-pitched (10 to 150 KHz) sounds produced as it decrypts data.

Cryptographic Theory vs Physical Reality

- Cryptographic algorithms are (supposed to be) theoretically secure.
- Implementations leak in physical world.

Implementation Attack Categories

- Side-channel attacks.
- Faults.
- Microprobing.

Taxonomy of Implementation Attacks

- Active vs passive.
- Active:
  1. Active: the key is recovered by exploiting some abnormal behavior.
  2. Insertion of signals.
- Passive:
  1. The device operates within its specifications.
  2. Reading hidden signals.

Implementation Attacks

Implementation attacks

Implementation attacks do not aim at the weaknesses of the algorithm, but on its implementation.

- **Side-channel attacks** (SCAs) are passive, non-invasive attacks.
- SCAs represent one of the most powerful category of attacks on crypto devices.

Examples of Implementation Attacks

- KeeLoq: eavesdropping from up to 100 m.
- PS3 hack due to ECDSA implementation failed.
- Attacks on Mifare Classic, Atmel CryptoMemory.
- Spectre and Meltdown.
- Google Titan.
- EUCLEAK.

The Goals of Attackers

- Secret data.
- Location.
- Reverse engineering.
- Theoretical cryptanalysis.
- ...

Physical Security in the Beginning

- Tempest – already known in 1960s that computers generate EM radiation that leaks information about the processed data.
- 1965: MI5 used a microphone positioned near the rotor machine used by Egyptian embassy to deduce the positions of rotors.
- 1996: first academic publication on SCA – timing.
- 1997: Bellcore attack.
- 1999: first publication of SCA – power.
- 2002: Template attack.
- 2016: Deep learning-based SCA.

Analysis Capabilities

- Direct attacks:
  1. Simple side-channel analysis.
  2. Differential side-channel analysis.
  3. Higher order attacks.
  4. ...
- Two-stage (profiling) attacks:
  1. Template attack.
  2. Stochastic models.
  3. Machine learning-based attacks.
  4. ...

Trade-offs and SCA

- Implementation attacks are very powerful and realistic threat.
- Many devices offer limited resources, which means there are limited resources for countermeasures.
- Optimizations can often open additional avenues for attacks.

SCA Countermeasures

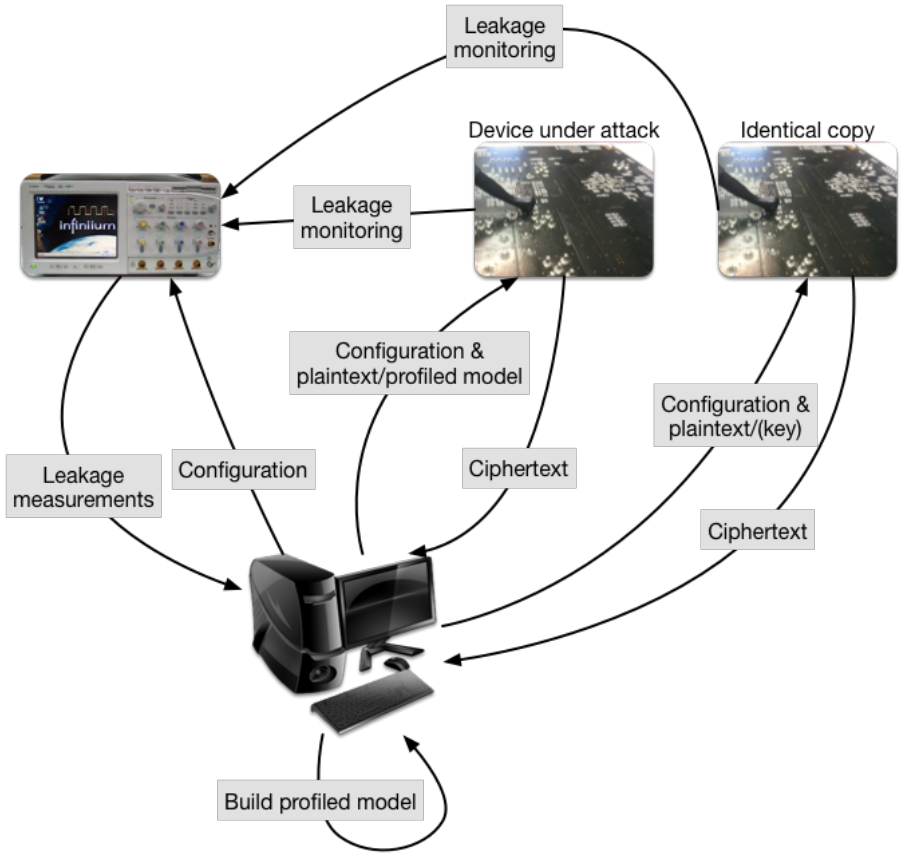
- The aim is to destroy the link between intermediate values and power consumption.
- There are two main categories of countermeasures for SCA:
  1. Masking.
  2. Hiding.
- Can be on any level, i.e., transistor level, program level, algorithmic level, protocol level.

Outline

Profiling Attacks

- Profiling attacks have a prominent place as the most powerful among side-channel attacks.
- Within profiling phase the adversary estimates leakage models for targeted intermediate computations, which are then exploited to extract secret information in the actual attack phase.
- Template Attack (TA) is the most powerful attack from the information theoretic point of view.
- Some **machine learning** (ML) techniques also belong to the profiling attacks.

Profiling Attacks



- Profiling attacks are more complicated than the direct attacks.
- The attacker must have a copy of the device to be attacked.

Types of Machine Learning

- Supervised learning.
- Unsupervised learning.
- Reinforcement learning.

Supervised Learning

- Supervised learning - available data include information how to correctly classify at least a part of data.
- Common tasks are classification and regression.

Unsupervised Learning

- Unsupervised learning - input data does not tell the algorithm what the clusters should be.
- Common tasks are clustering, density estimation, and dimensionality reduction.

Reinforcement Learning

- Take actions based on current knowledge of the environment.
- Receive feedback in the form of rewards.
- Learn based on received rewards and update behavior (policy) in order to maximize the expected reward (utility).

Machine Learning Basic Components

- Model.
- Loss function.
- Optimization procedure to minimize the empirical error.

Underfitting and Overfitting

- Overfitting – if a model is too complex for the problem, then it can learn the detail and noise in the training data so it negatively impacts the performance of the model on new data $\rightarrow$ a model that models the training data too good.
- Underfitting – if a model is too simple for the problem, then it cannot generalize to new data.
- Simple model $\rightarrow$ high bias.
- Complex model $\rightarrow$ high variance.

Deep Learning

- Stacked neural networks, i.e., networks consisting of multiple layers.
- Layers are made of nodes.

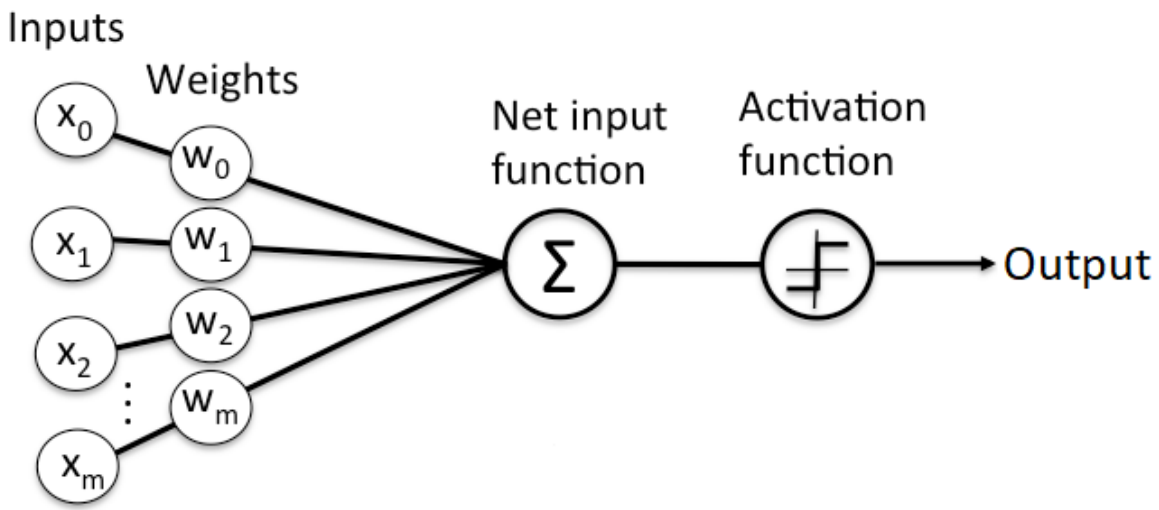


Figure: Perceptron.

Multilayer Perceptron

- One input layer, one output layer, at least one hidden layer.

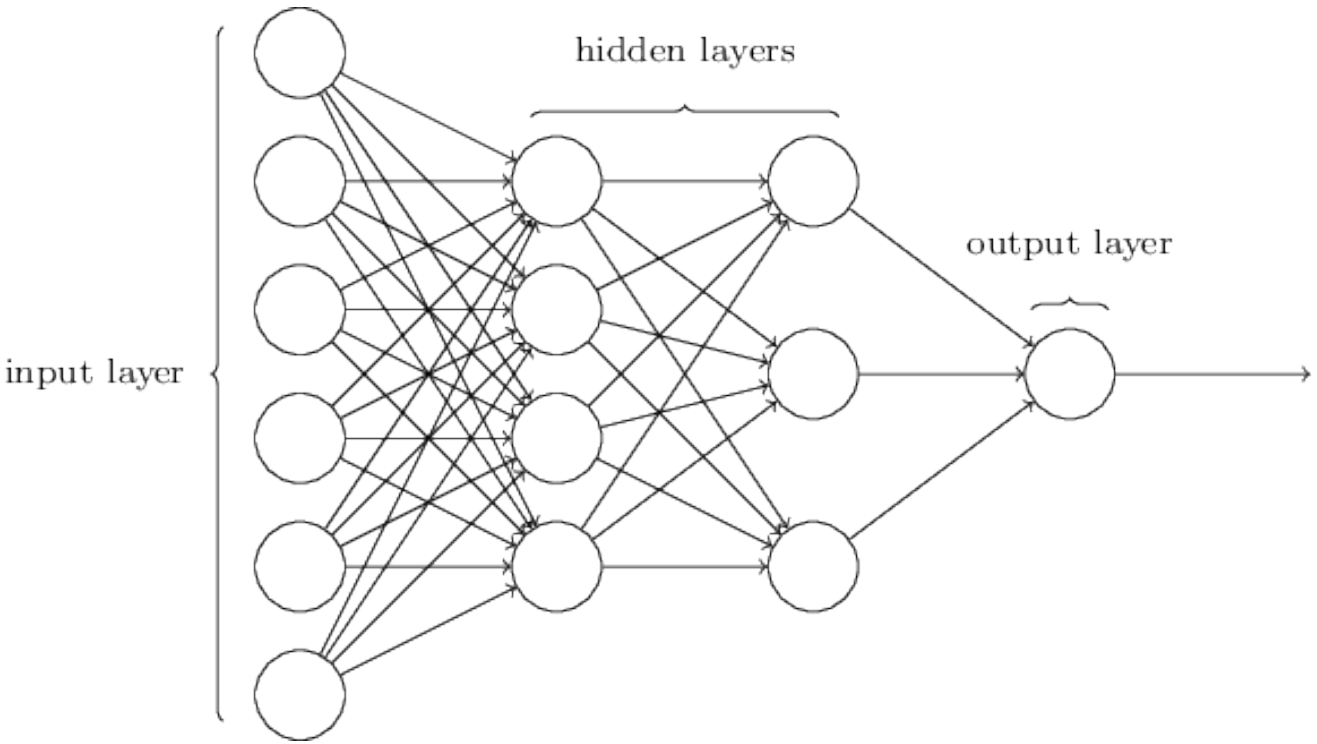


Figure: Multilayer perceptron.

Deep Learning

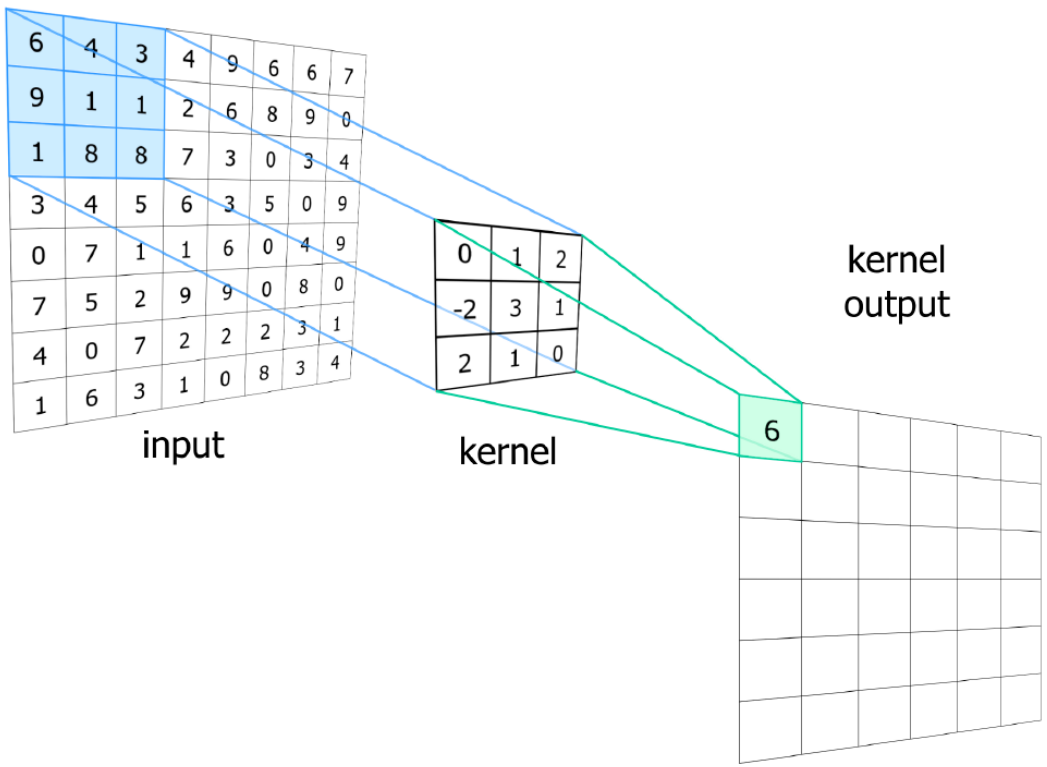
- By adding more hidden layers, we arrive at deep learning.
- Some definitions say everything more than one hidden layer is deep learning.
- A field existing for a number of years but one that gained much attention in the last decade.
- Sets of algorithms that attempt to model high-level abstractions in data by using model architectures with multiple processing layers, composed of a sequence of scalar products and non-linear transformations.
- In many tasks, deep learning is not necessary since machine learning performs well.

Convolutional Neural Networks

- CNNs represent a type of neural network first designed for 2-dimensional convolutions.
- They are primarily used for image classification, but lately, they have proven to be powerful classifiers in other domains.
- From the operational perspective, CNNs are similar to ordinary neural networks: they consist of a number of layers where each layer is made up of neurons.
- CNNs use three main types of layers: convolutional layers, pooling layers, and fully-connected layers.

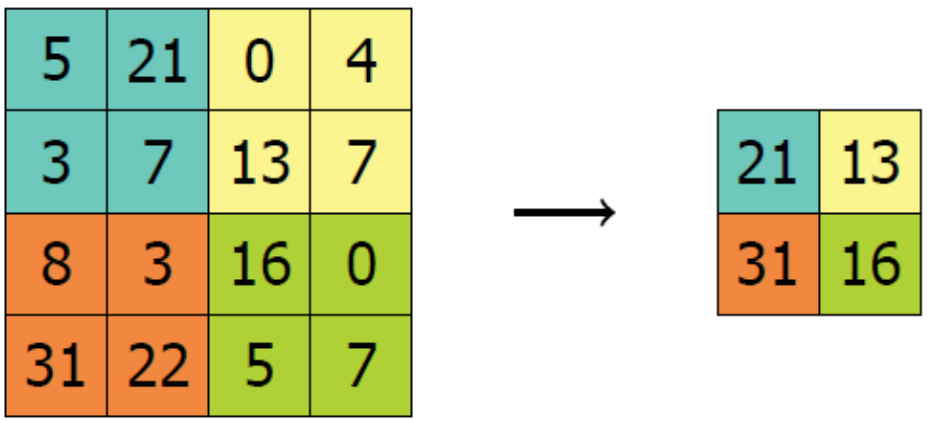
Convolutional Neural Networks - Convolution Layer

Convolutional layer: on this layer, during the forward computation phase, the input data are convoluted with some filters. The output of the convolution is commonly called a feature map. It shows where the features detected by the filter can be found on the input data.



CNN - Pooling

Max (average) pooling layer: sub-sampling layer. The feature map is divided into regions and the output of this layer is the concatenation of the maximum (average) values of all these regions.



Activation Functions

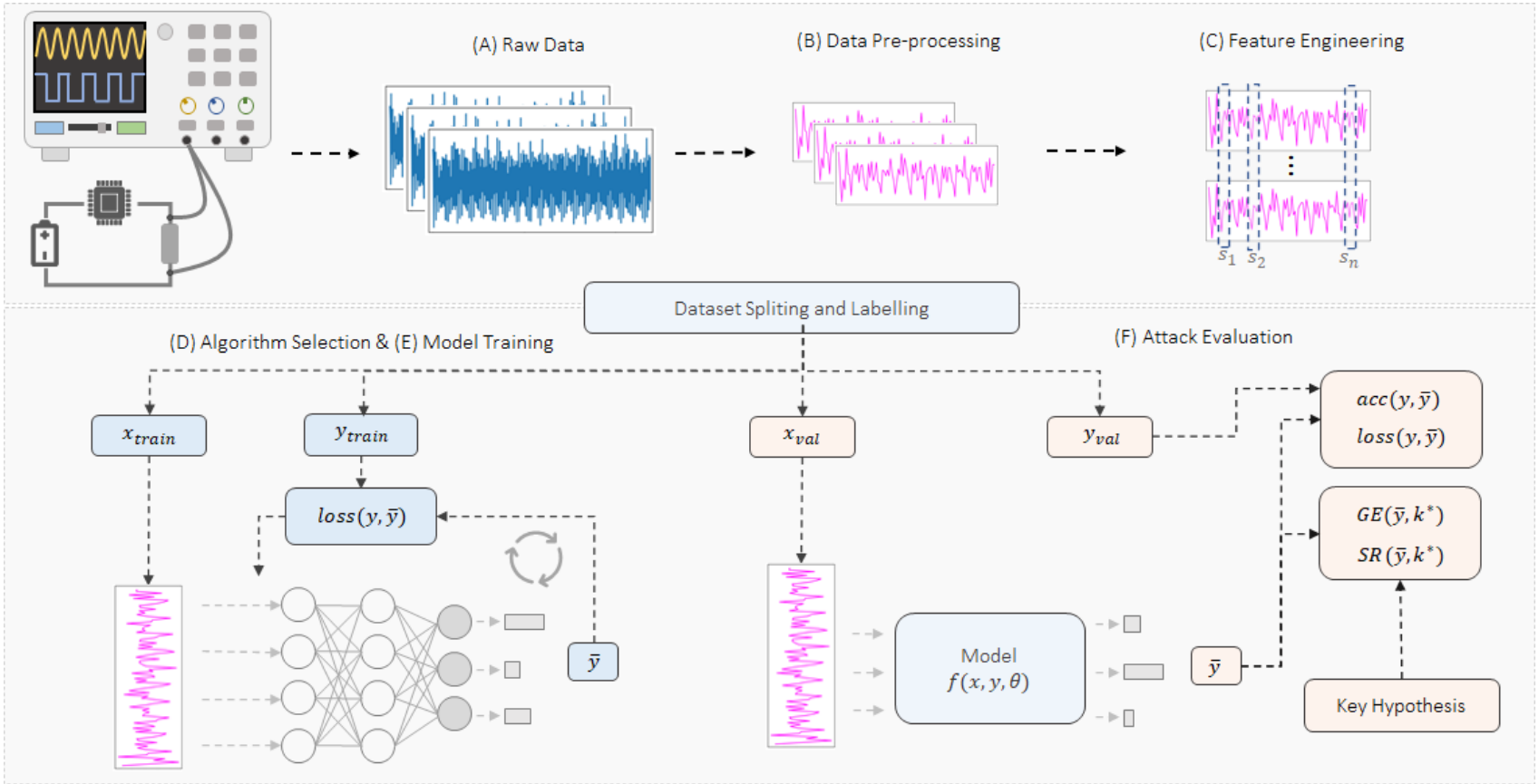
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Figure: Activation functions.

Backpropagation

- A method used in artificial neural networks to calculate the gradient needed for the calculations of weights in the network.
- The calculation of the gradient proceeds backward through the network: the gradient of the final layer of weights being calculated first.
- Steepest descent – an algorithm for finding the minimum of a function.

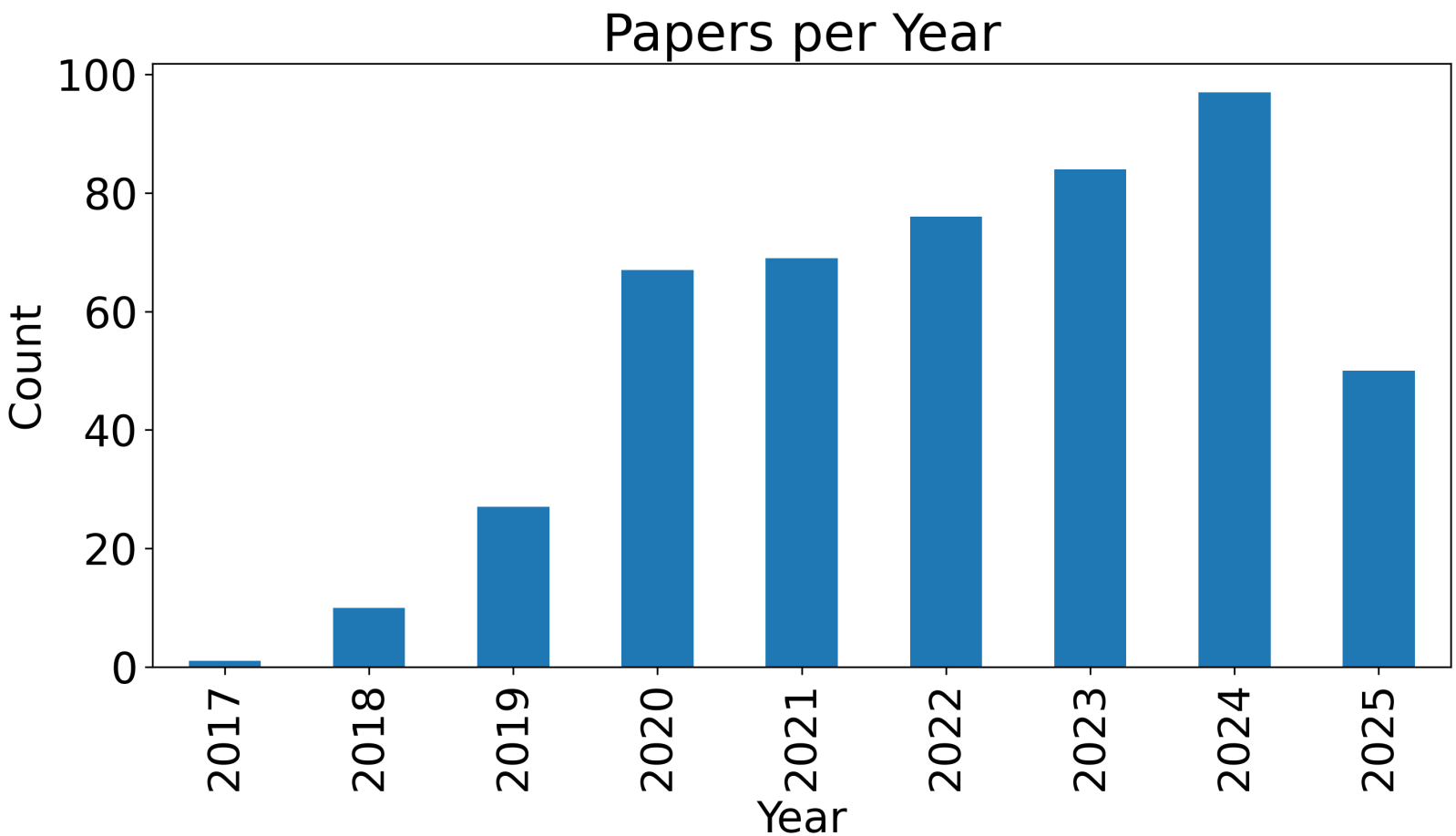
Machine Learning Process Flow



For Beginners

- A Practical Tutorial on Deep Learning-based Side-channel Analysis
- `https://github.com/marinakrcek/DLSCA-tutorial`

Deep Learning Publications



Outline

Hyperparameter Tuning

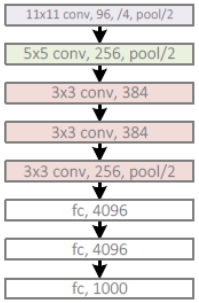
- Hyperparameter tuning is extremely important.
- Different algorithms have different hyperparameters.
- Neural networks have many hyperparameters.
- Random search.
- Grid search.
- Advanced techniques.
- Methodologies.

Different Neural Network Types in SCA

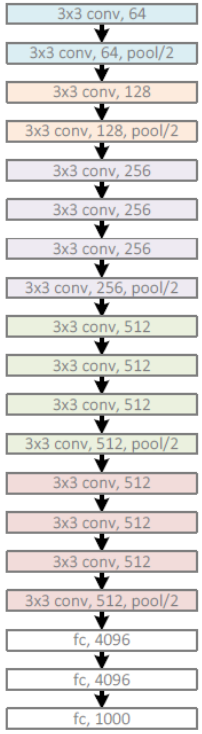
- Autoencoder.
- Recurrent neural network.
- Residual neural network.
- Generative Adversarial Network.
- Transformers.
- ...

More Complex Architectures

AlexNet, 8 layers
(ILSVRC 2012)

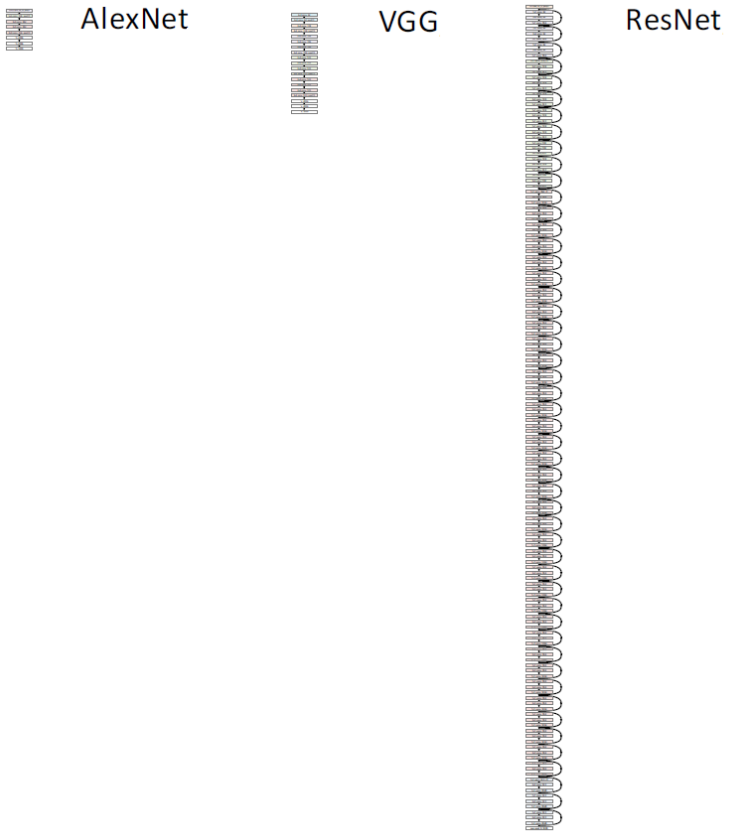


VGG, 19 layers
(ILSVRC 2014)



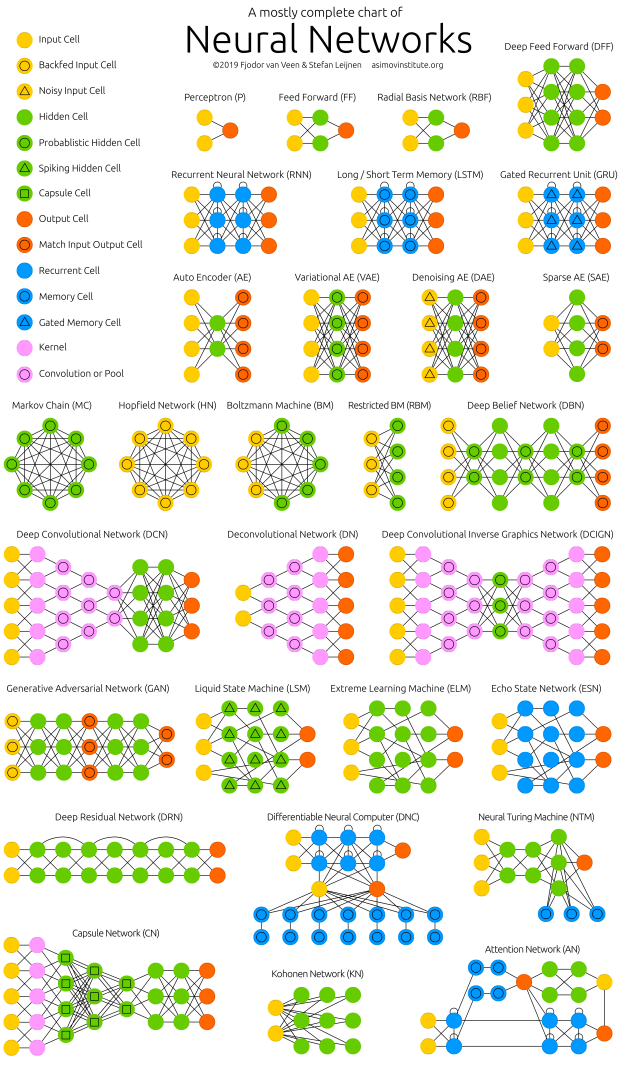
GoogleNet, 22 layers
(ILSVRC 2014)

More Complex Architectures



- More complex architectures means more complex hyperparameter tuning.

Neural Networks



ML vs. SCA Metrics

- Training process is assessed based on ML metrics while in the attack phase we care about SCA metrics.
- Does good ML performance mean good SCA performance?
- How about poor ML performance?
- Can we use SCA metrics in the training phase?

New Metrics

- Key guessing vector: search success, guessing entropy estimation algorithm (GEEA), Cross Entropy Ratio (CER), comparing the success rate discrepancy on the training/validation sets to judge the generalization capacity of a model, area of hit, Label Correlation (LD), (simplified) Leading Degree(LD).
- Mutual information (MI): MI transferred to the output layer, Perceived information (PI), Hypothetical Information (HI), Ranking Loss (RkL), Ensembling Loss (EL), efficient cross-entropy (ECE) and efficient PI (EPI), latent (LPI).

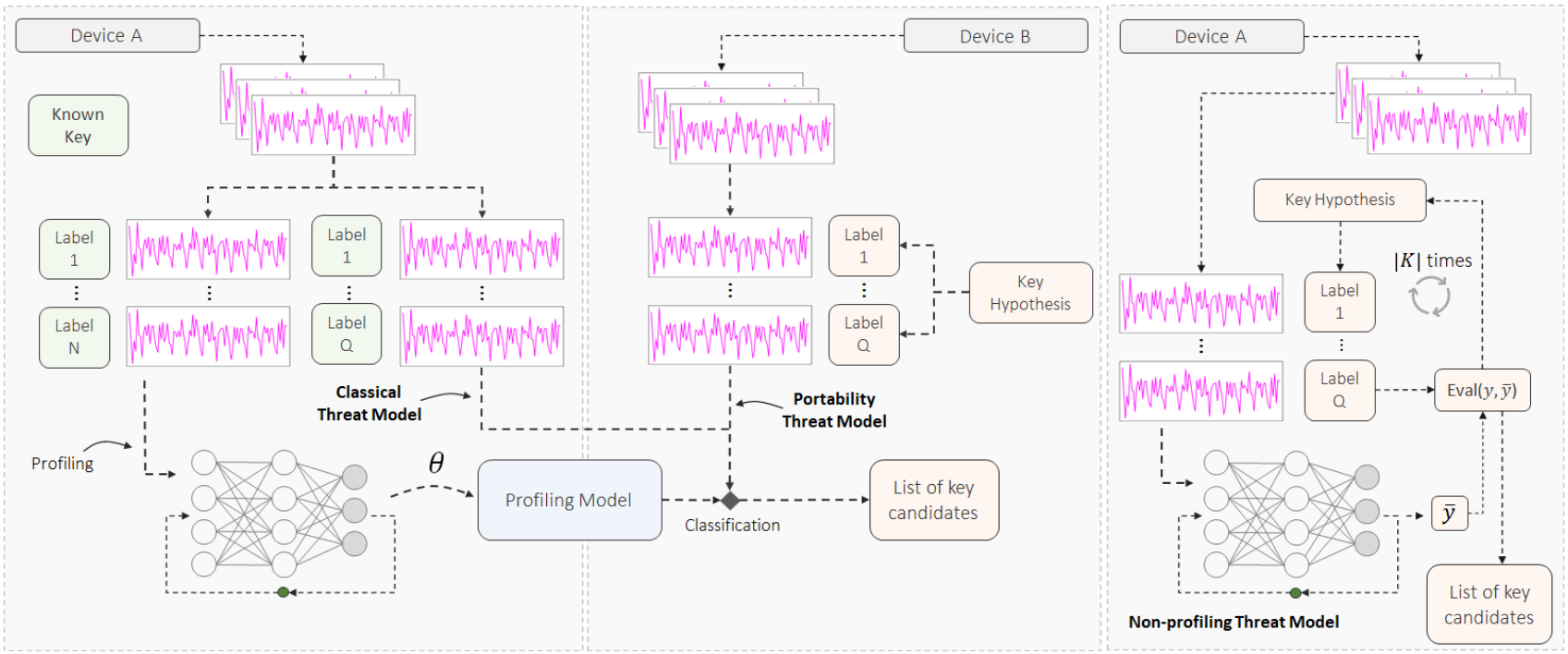
Many Techniques Actually Work

- Simple hyperparameter search.
- Data Augmentation.
- Various types of architectures.
- Small architectures.
- Custom metrics and neural network elements.
- ...

Datasets

Dataset	Platform	Traces (Features)	Keys	Implementation	Countermeasures
DPAv1 [Gui07] (2008)	home-made acquisition platform		Fixed	DES	
DPAv2 [CDD+14] (2010)	SASEBO GII (FPGA)	1 000 000 (3 253) power traces	Random keys	AES128	
DPAv4 [NSGD12]link (2013)	Atmel ATMega-163 (Software)	100 000 (435 002)	1 fixed key	AES256	First-order masking (RSM)
DPAv4.2 [BBD+14]link (2014)	Atmel ATMega-163 (Software)	80 000 (1 704 400)	16 different keys	AES128	First-order masking (RSM)
AES_HD [BJP20a]link(2020)	SASEBO GII (FPGA)	50 000 (1 250)	1 fixed key	AES128	
AES_HD_Ext [BJP20b] (2020)	SASEBO GII (FPGA)	500 000 (1 250)	1 fixed key	AES128	
AES_HD_MM [Nor]link (2014)	SASEBO GII (FPGA)	5 600 000 (3 125)	1 fixed key	AES128	Boolean + Affine Masking
AES_RD [CK09]link(2009)	Atmel AVR (Software)	50 000 (3 500)	1 fixed key	AES128	Hiding (Random Delay Interrupt)
ASCADf [BPS+20] link (2018)	Atmega (Software)	60 000 (100 000)	1 fixed key	AES128	First-order boolean masking
ASCADv1 [BPS+20]link (2018)	Atmega (Software)	300 000 (250 000)	Random keys + 1 fixed key	AES128	First-order boolean masking
ASCADv2 [MS23] link(2021)	STM32 (Software)	810 000 (1 000 000)	Random + 1 fixed key	AES128	Affine Masking + Hiding (Shuffling)
CHES_CTF 2018 [Ris18] link(2018)	STM32 (Software)	42 000 (650 000)	Random Keys + 3 fixed keys	AES128	First-order BM (XOR)
Portability [BCH+20] (2020)	Atmega (Software)	50 000 (600)	4 fixed keys	AES128	
SMAesH [CM24]link(2023)	Xilinx Artix-7 and Spartan-6 (FPGAs)	6×2^{24} (4 450/4 400), power	Random + Fixed key	AES128	First-order masking
eShard [VTM23]link (2021)	STM32F446 (Software)	90 000/10 000 (1 400), EM	Fixed key	AES128	Boolean masking (+ simulated shuffling)
CrossEM [NNR+24] link(2024)	AVR XMEGA (sofwtare) STM32F3(Software)	3 million EM traces from 9 probe locations over 4 targets, 150 000 (5 000)	Fixed key	AES128	Random delay
X-DeepSCA [DGD+19]link (2020)	Atmega	(3 000), power		AES128	
DFS_DESYNCH [GLMZ25]link (2025)	RISC-V CPU (Software)	256 000 (200 000), power	16 keys	AES128	DFS desynchronization
GE_wars [BBJ+]link (2025)	ARM Cortex-A72 (Software)	500K profiling, 100K attack traces (7 000), EM	Random keys + fixed key	AES128	Jitter and noise, no masking
Chameleon [GCZ25]link (2025)	RISC-V CPU (Software)	256, 256, 512, 512, and 1 024 (134 217 550), power	Segment traces to detect crypto	AES128	Dynamic frequency scaling, random delay, morphing, and chaffing
Keccak (and AES) [UXT+21]link (2021)	(Software & hardware)			Keccak, AES	Unprotected and masked AES, unprotected Keccak
AES_PTv2 [?]link (2022)	STM32F4 and Riscue Piñata	600 000 (× 4 devices), power/EM	random and fixed key	AES128	unprotected, and masked AES
Ed25519 (**WolfSSL**) [WPB19]link (2019)	STM32 (Software)	6 400 (1 000), power	Random Ephemeral keys	EdDSA	
Curve25519 (μNaCl) REASSURE [CBT20]link (2020)	STM32F4 (Software)	5 997 (5 500)	Random Ephemeral keys	Curve25519 NaCl Montgomery Ladder scalar multiplication	CSWAP, Coord./Scalar Rand.
Curve25519 [NCOS17] (2016)	STM32 (Software)	300 (8 000)	Random keys	EdDSA	CSWAP, Coord./Scalar Rand.
Curve25519 [NCOS17] (2016)	STM32 (Software)	300 (1 000)	Random keys	EdDSA	CSPOINTER, Coord./Scalar Rand.
SCAAML ECC CM0 [BIK+24]link (2024)	NXP K82F (Hardware accelerated)	57 344/16,384 (1 600 000), power	Random Ephemeral keys	ECC scalar multiplication	
SCAAML ECC CM1 [BIK+24]link (2024)	NXP K82F (Hardware accelerated)	194 544/16,384 (5 000 000), power	Random Ephemeral keys	ECC scalar multiplication	Additive blinding
SCAAML ECC CM2 [BIK+24]link(2024)	NXP K82F (Hardware accelerated)	112 880/16,384 (10 000 000), power	Random Ephemeral keys	ECC scalar multiplication	multiplicative blinding
SCAAML ECC CM3 [BIK+24]link(2024)	NXP K82F (Hardware accelerated)	112 880/16,384 (17 500 000), power	Random Ephemeral keys	ECC scalar multiplication	Combination of CM1 + CM2
Spook [BBC+21]link (2020)	Software/Hardware	200 000/100 000 †	Fixed/random	Clyde-128	ISW masking ⋆
ASCON [RBBWP24] link(2024)	STM32F4(Software)	50 000/10 000 (772), power	Random/Fixed	Ascon-128 v1.2	
ASCON protected [RBBWP24] link(2024)	STM32F4(Software)	500 000/60 000 (1 408), power	Random/Fixed	Ascon-128 v1.2	Domain Oriented Masking
Xoodyak [EMW+24] (2024)	Sakura-G FPGA (Hardware)	200 000/50 000 (5 000), power	Random/Fixed	Xoodyak	
PRESENT (and AES) [LO23]link (2023)	Sakura-G (FPGA)	50 000 (10 000/2 500), power		PRESENT, AES	
PRESENT [MWM21b]link (2021)	SAKURA-G (Spartan-6 FPGA)	1 000-50 000 000 (2 000-200 000), voltage drop over a 1 Ω shunt resistor		PRESENT	(Un)Protected, (Mis)Aligned, Randomized Clock
Dilithium [dzyxxxb25]link (2025)	Cortex M4	60 000, power		Dilithium	
Dilithium [Fan25]link (2025)		POIs, power		Dilithium	First and second-order masking
Kyber [RYJ+25b]link (2025)	STM32F3 (ARM Cortex-M4)	100 000 (50 000), power		Kyber pair-pointwise multiplication in NTT Domain	
BLISS [MWG+22]link (2022)	Cortex-M4 (software)	1 532/1 450 492 power traces	random key + fixed key	GALACTICS (BLISS)	

Threat Models



Threat Models

- White-box - Scheme-Aware - Black-Box.
- Extra Reference Device.
- Portability.
- Non-Profiled Supervised DLSCA.
- Weakly Profiling DLSCA.
- Collision-based DLSCA.
- Blind DLSCA.
- Leakage Assessment Using Deep Learning.

Tools

- Python.
- scikit-learn.
- TensorFlow/PyTorch.
- Keras.

Tools

- Brisfors and Forsmark developed a python-based tool called *DLSCA* that allows deep learning-based SCA `https://github.com/brisfors/DLSCA`.
- The tool allows running multilayer perceptron architecture for attacks on AES128 and plotting the results (key rank, guessing entropy).
- While the authors mention it is not difficult to add new functionalities, there has not been any development in the last few years.

SCARED

- The company eShard developed a Python library called *scared* that allows various types of SCA `https://github.com/eshard/scared/`.
- The library receives regular updates and provides various functionalities, but there are no deep learning-based functionalities available in the repository (there is TA, which is a profiling SCA).
- Examining recent posts, *scared* library does offer deep learning functionalities.

SCAAML

- Google recently published their python-based deep learning framework for SCA called *SCAAML* `https://github.com/google/scaaml`.
- The framework is actively developed but offers (at the moment) limited functionality.
- The framework provides one CNN architecture designed to attack TinyAES (the architecture is tuned and the best-performing one over more than 1 000 tested ones).
- To evaluate the attack performance, it is possible to use the (average) key rank.

Inspector

- Riscure offers a tool called *Inspector* that also offers deep learning capabilities.
- Since the *Inspector* tool is a commercial one, there are no publicly available versions of it.
- What can be deduced based on the available information is that Inspector offers MLP and CNN architectures, regularization, data augmentation, and hyperparameter tuning.

AISY

- The AISY framework is intended for the deep learning-based SCA.
- **Easy to use**. AISY framework allows very easy execution of deep learning in profiling side-channel attacks. The framework is built on top of *Keras* library (integrated in *TensorFlow* library) and users familiar to basic *Keras*'s functionalities can easily extent the framework.
- **Integrated Database**. AISY framework comes with the option to store all analysis results in an SQLite database. Standard libraries are implemented in the framework, and users can easily add custom tables to the framework.

AISY

- **Web application**. AISY framework is also integrated with *Flask* python-based web framework. A web application is integrated with a web-based user interface. The web application provides a user-friendly way to visualize analysis, plots, results, and tables.
- **One-click Script Generation**. A user can generate the full script used to produce results stored in the web application database.
- `https://github.com/AISyLab/AISY_Framework`

SCALib

- The Side-Channel Analysis Library (SCALib) is a Python package that contains state-of-the-art tools for side-channel evaluation.
- It focuses on providing efficient implementations of analysis methods widely used by the side-channel community and maintaining a flexible and simple interface.
- `https://scalib.readthedocs.io/en/stable/`

Generalization of Function Approximation

- While we use machine learning metrics to drive the training, we are interested in results as observed through SCA metrics.
- Ideally, we should always train a neural network until it achieves the maximum quality in generalization to the validation set.
- Underfitting, generalization, and overfitting phases.

Generalization of Function Approximation

- In SCA, the generalization phase is directly related to the key recovery, and it may start very soon after the training starts because a low accuracy can already represent the turning point from underfitting to generalization.
- Can a low accuracy (sometimes close to random guessing) still be associated with this *good enough* generalization phase?

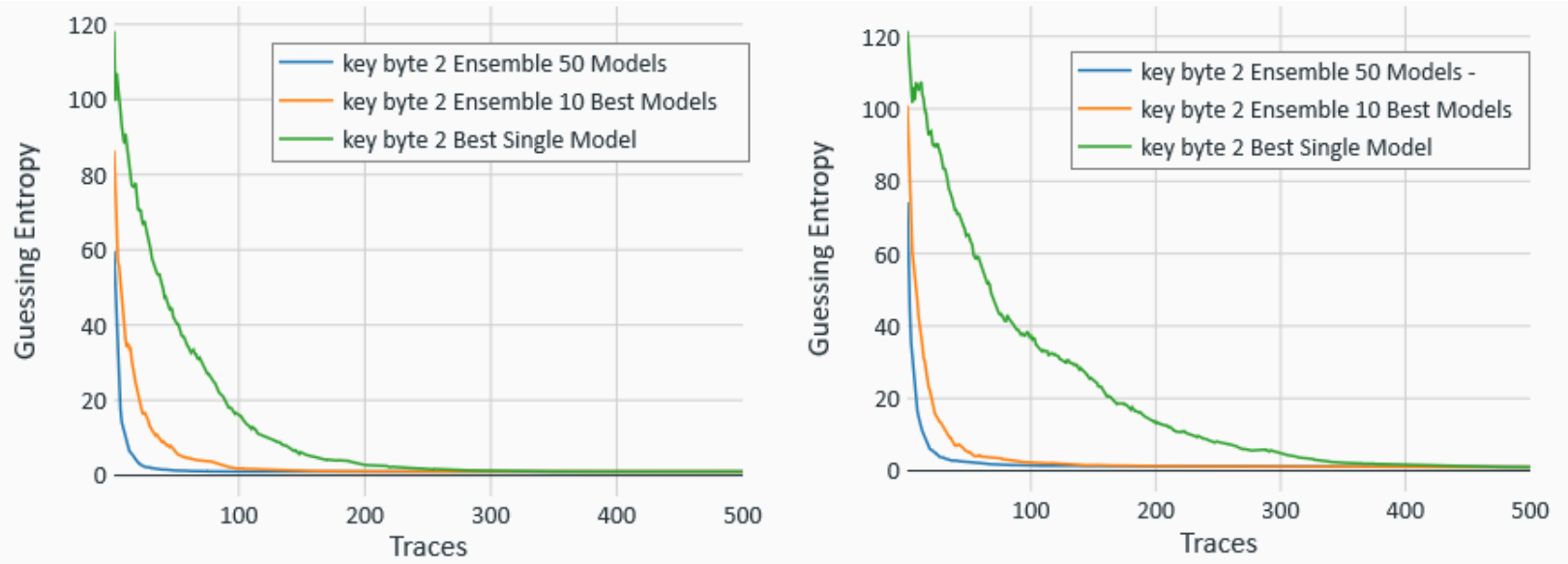
How to Improve Generalization?

- There are many ways to improve generalization (more powerful classification methods, better hyperparameter tuning, regularization, etc.).
- We can also do something simpler!
- Commonly, in the experimental phase, one runs a number of evaluations to find the best hyperparameters.
- Can we somehow use multiple results?
- It sounds reasonable to take the most out of the hyperparameter tuning phase and explore whether one can use more than a single machine learning model obtained during the tuning phase.

Bagging

- Create many subsamples of the dataset with replacement (meaning that the two sample values are independent, i.e., their covariance equals 0).
- Train a classifier for each subsample.
- Calculate the average prediction from each classifier.

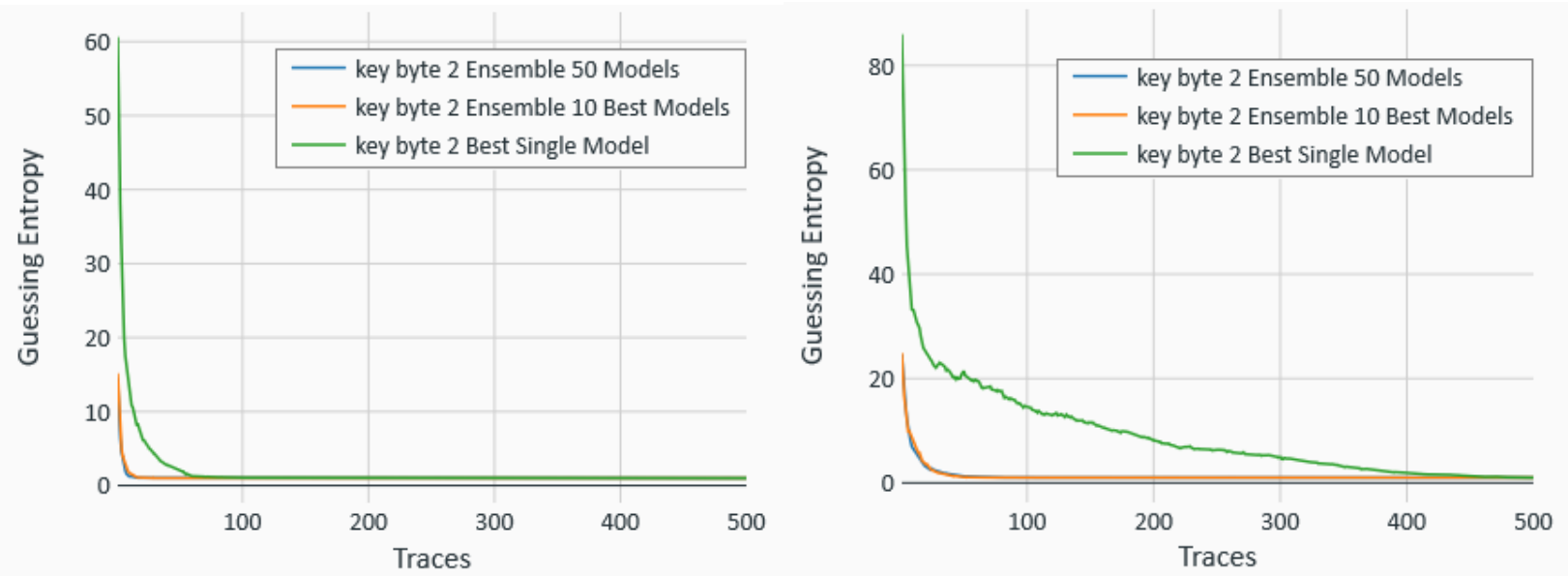
Deep Learning Ensembles



(a) MLP results. (b) CNN results.

Figure: Guessing entropy for ASCAD for the Hamming weight leakage model.

Deep Learning Ensembles

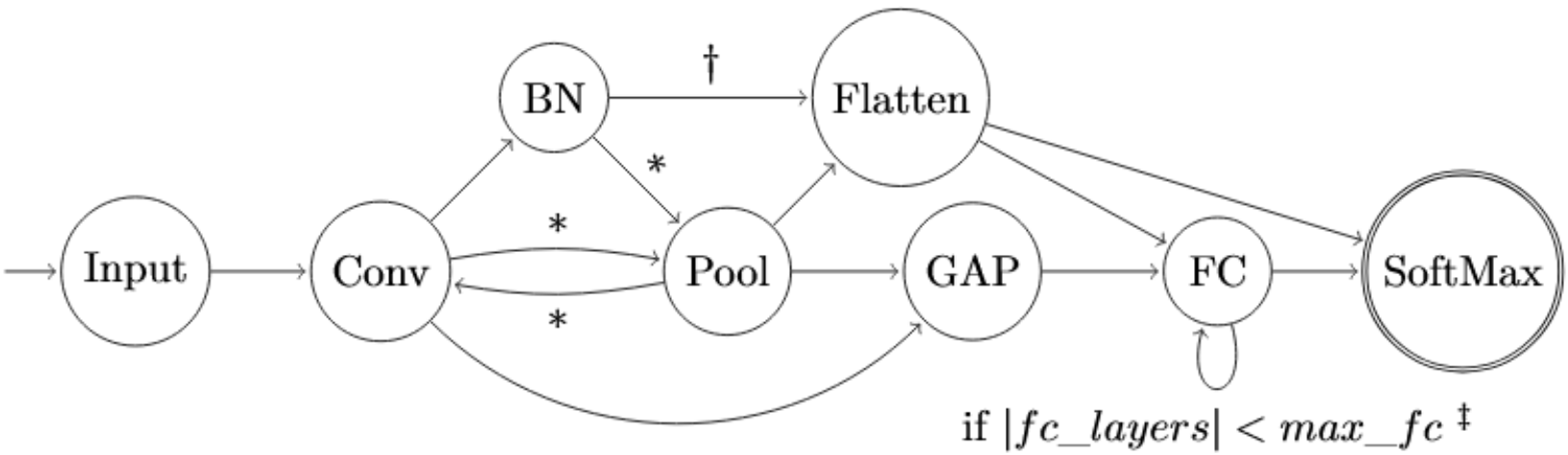


(a) MLP results. (b) CNN results.

Figure: Guessing entropy for ASCAD for the Identity leakage model.

Reinforcement Learning

- Reinforcement learning attempts to teach an agent how to perform a task by letting the agent experiment and experience the environment, maximizing some reward signal.
- `https://github.com/AISyLab/Reinforcement-Learning-for-SCA`



Reinforcement Learning

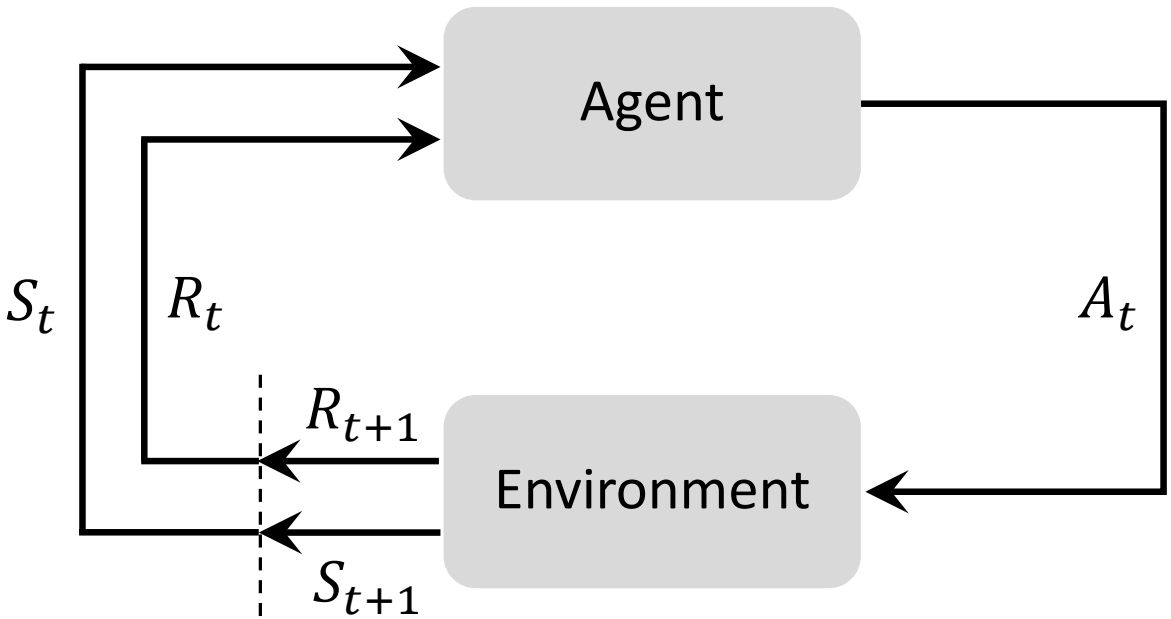
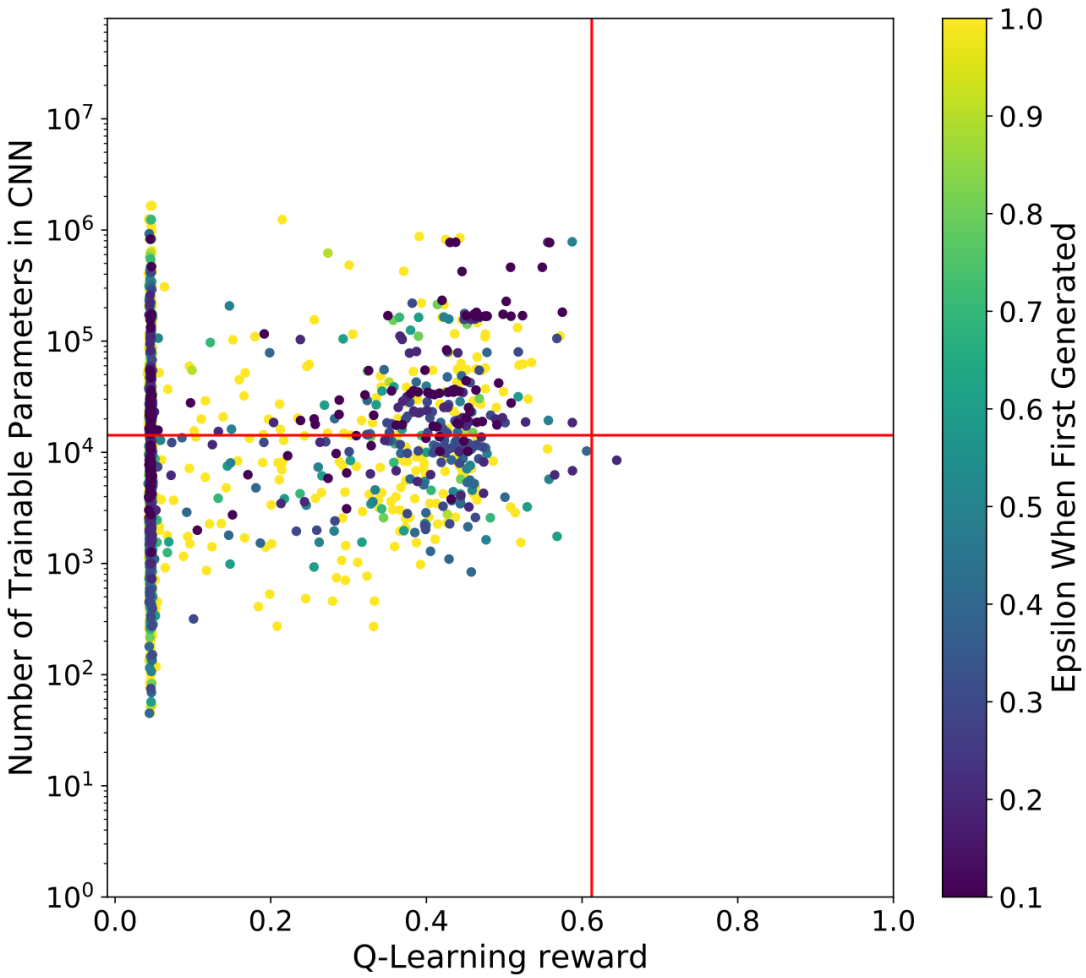
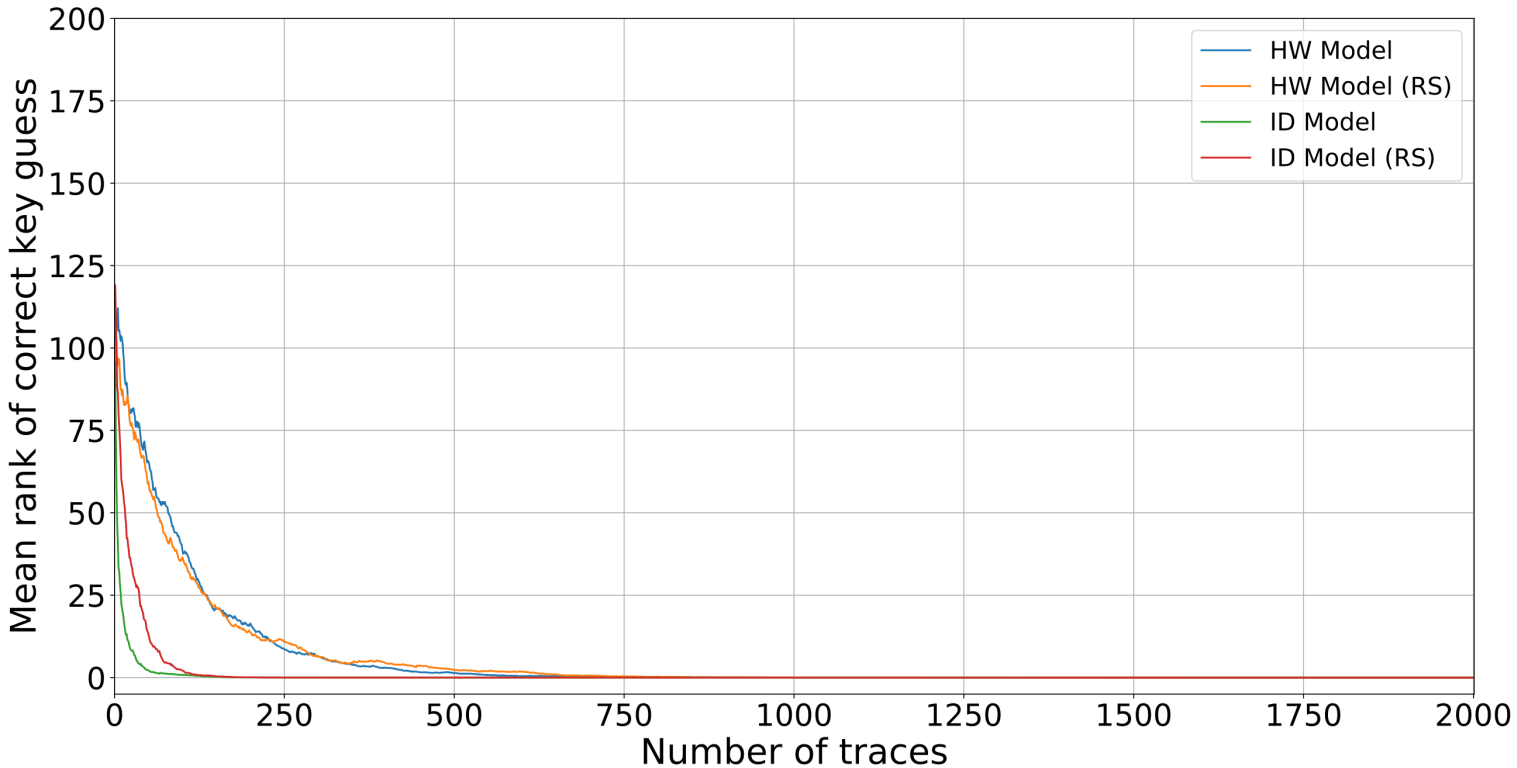


Figure: The q-learning concept where an agent chooses an action A_t, based on the current state S_t, which affects the environment. This action is then given a reward R_{t+1} and leads to state S_{t+1}.

Reinforcement Learning



Reinforcement Learning



Feature Selection for Deep Learning SCA

Scenario	**Knowledge of r mask share**	**POI selection and pre-processing**	**Noisy/non-leaking samples**
RPOI	Yes	Main SNR peaks of r and s_r. No pre-processing required.	No
OPOI	Yes	Minimum trace interval including SNR peaks of r and s_r. No pre-processing required.	Reduced
NOPOI	No	No POI selection and pre-processing is required.	All available

Table: Possible feature selection scenarios for deep learning-based SCA with the synchronized measurements.

Feature Selection for Deep Learning SCA

Dataset	RPOI	OPOI	NOPOI	Total
ASCADf	up to 1 000 SNR peaks from NOPOI interval	[45 400, 46 100]	[0, 100 000]	100 000
ASCADr	up to 1 000 SNR peaks from NOPOI interval	[80 945, 82 345]	[0, 250 000]	250 000
DPAv4.2	up to 1 000 SNR peaks from NOPOI interval	[170 000, 174 000] + [206 000, 210 000]	[250 000, 400 000]	1 700 000
CHES CTF	-	[0, 10 000] + [120 000, 150 000]	[0, 150 000]	650 000

Table: Selected intervals for each feature selection scenario. '-' denotes that we did not explore that specific setting.

Feature Selection for Deep Learning SCA

Table: Points of interest, minimum number of attack traces to get guessing entropy equal to 1, model search success (when GE=1), and number of trainable parameters for all datasets and feature selection scenarios.

Dataset	Neural Network Model	Feature Selection Scenario	Amount of POIs (HW/ID)	Attack Traces (HW/ID)	Search Success (%) (HW/ID)	Trainable Parameters (HW/ID)
ASCADf	MLP	RPOI	200/100	5/**1**	99.22%/96.86%	82 209/429 256
ASCADf	CNN	RPOI	400/200	5/**1**	99.23%/99.08%	499 533/158 108
ASCADf	MLP	OPOI	700/700	480/104	82.80%/68.80%	16 309/10 266
ASCADf	CNN	OPOI	700/700	744/87	55.53%/35.33%	594 305/62 396
ASCADf	MLP	NOPOI	2 500/2 500	7/**1**	74.50%/39.00%	2 203 009/5 379 256
ASCADf	CNN	NOPOI	10 000/10 000	7/ **1**	15.40%/2.45%	545 693/439 348
ASCADf	CNN	NOPOI desync	10 000/10 000	532/36	2.44%/2.64%	268 433/64 002
ASCADr	MLP	RPOI	200/20	3/**1**	99.23%/100%	565 209/639 756
ASCADr	CNN	RPOI	400/30	5/**1**	100%/100%	575 369/636 224
ASCADr	MLP	OPOI	1 400/1 400	328/129	71.40%/37.25%	31 149/34 236
ASCADr	CNN	OPOI	1 400/1 400	538/78	47.92%/23.95%	270 953/87 632
ASCADr	MLP	NOPOI	25 000/25 000	6/ **1**	44.39%/7.02%	5 243 209/12 628 756
ASCADr	CNN	NOPOI	25 000/25 000	7/ **1**	19.17%/4.35%	369 109/721 012
ASCADr	CNN	NOPOI desync	25 000/25 000	305/73	0.71%/1.04%	22 889/90 368

Public-key Crypto and SCA

- To mitigate side-channel attacks, real-world implementations of public-key cryptosystems adopt state-of-the-art countermeasures based on randomizing private or ephemeral keys.
- Usually, for each private key operation, a "scalar blinding" is performed using 32 or 64 randomly generated bits.
- Nevertheless, horizontal attacks based on a single trace still pose serious threats to protected ECC or RSA implementations.
- If the secrets learned through a single-trace attack contain too many wrong (or noisy) bits, the cryptanalysis methods for recovering the remaining bits become impractical due to time and computational constraints.

Public-key Crypto and SCA

- By attacking several single traces, an attacker may recover several partially correct random private keys.
- This information is then used to label each sub-trace (trace interval representing the processing of a single private key bit) and use them as elements in a training set to train a neural network.
- Assuming that each recovered private key contains more than 50% of correct bits (just above a random guess), the trained neural network can significantly improve the number of correct bits in a random private key related to a single trace.
- The target is protected ECC implementations in software (protected μNaCl).

Deep Learning-based Iterative Framework

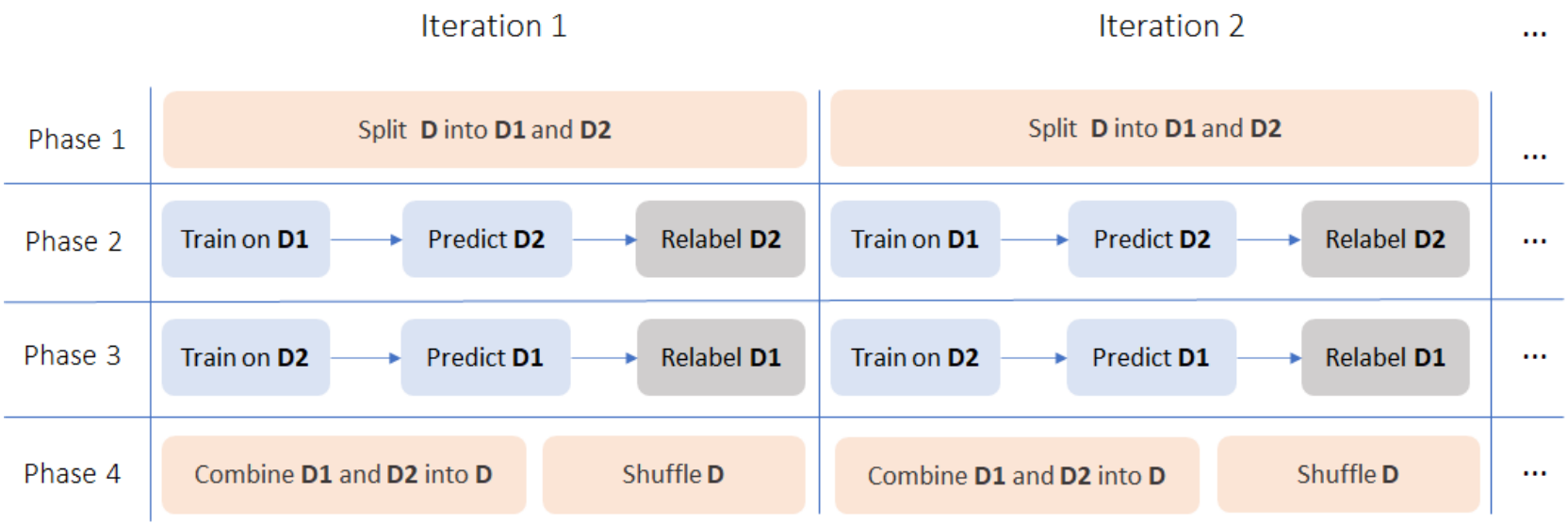


Figure: Proposed iterative framework

Deep Learning-based Iterative Framework

- The procedure continues iteratively until a successful attack is achieved.
- In every step of this iterative process, it is expected that the amount of noisy labels decreases as a result of deep neural networks learning side-channel leakages from the limited correct labels in the training set.
- The higher the error bits in the initial training set, the more iterations we expect to need to reach a successful attack.

Deep Learning-based Iterative Framework

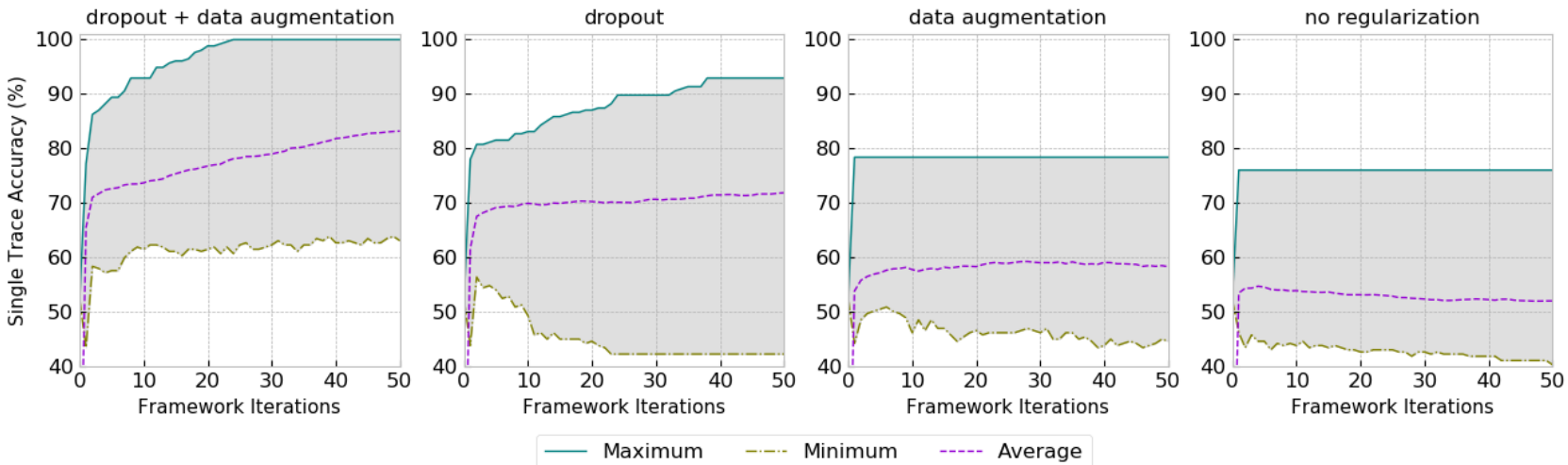


Figure: Minimum, maximum, and average single trace accuracy with iterative framework on *cswap-arith* dataset.

Deep Learning-based Iterative Framework

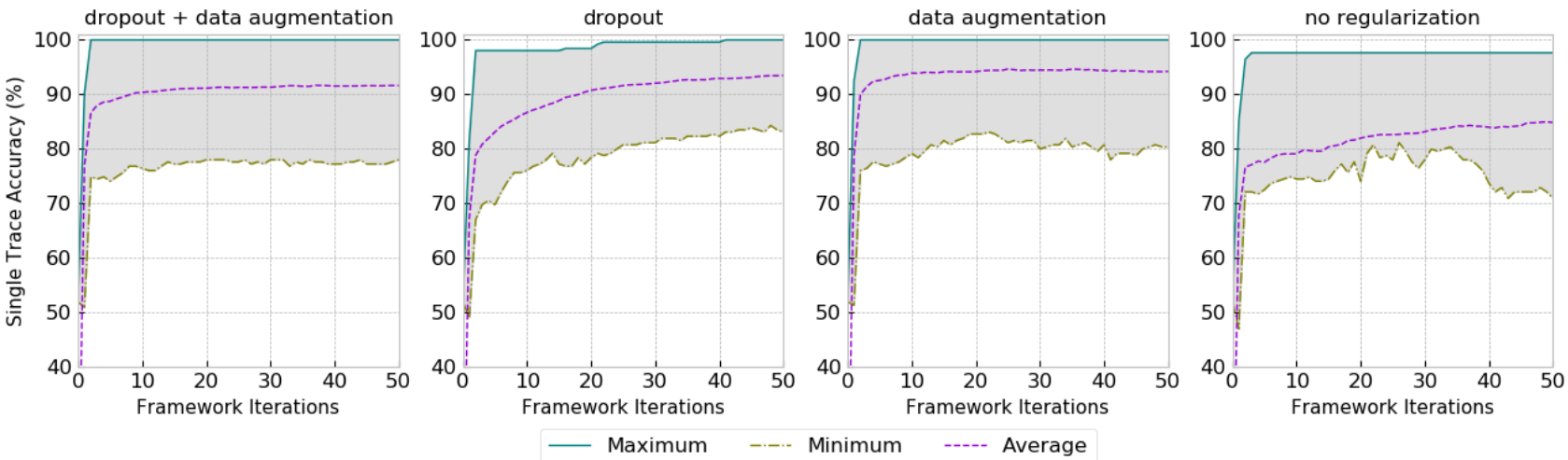


Figure: Minimum, maximum, and average single trace accuracy with iterative framework on *cswap-pointer* dataset.

Outline

Side-channel Analysis and Grammatical Evolution

- A neuroevolution framework that could replace random search to efficiently generate SCA models.

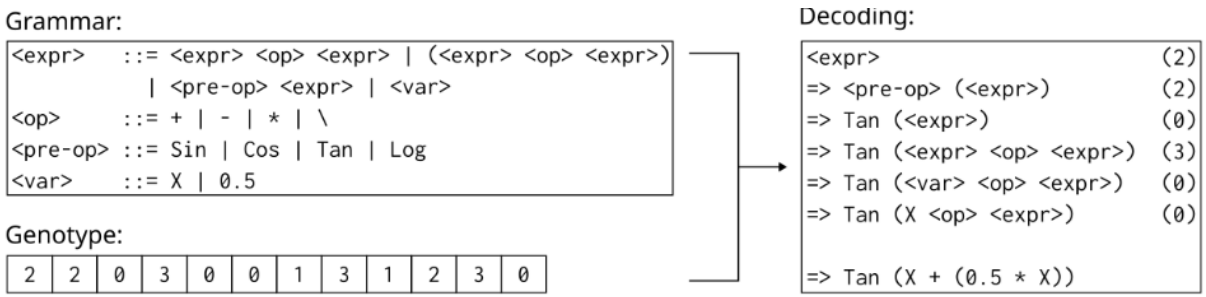
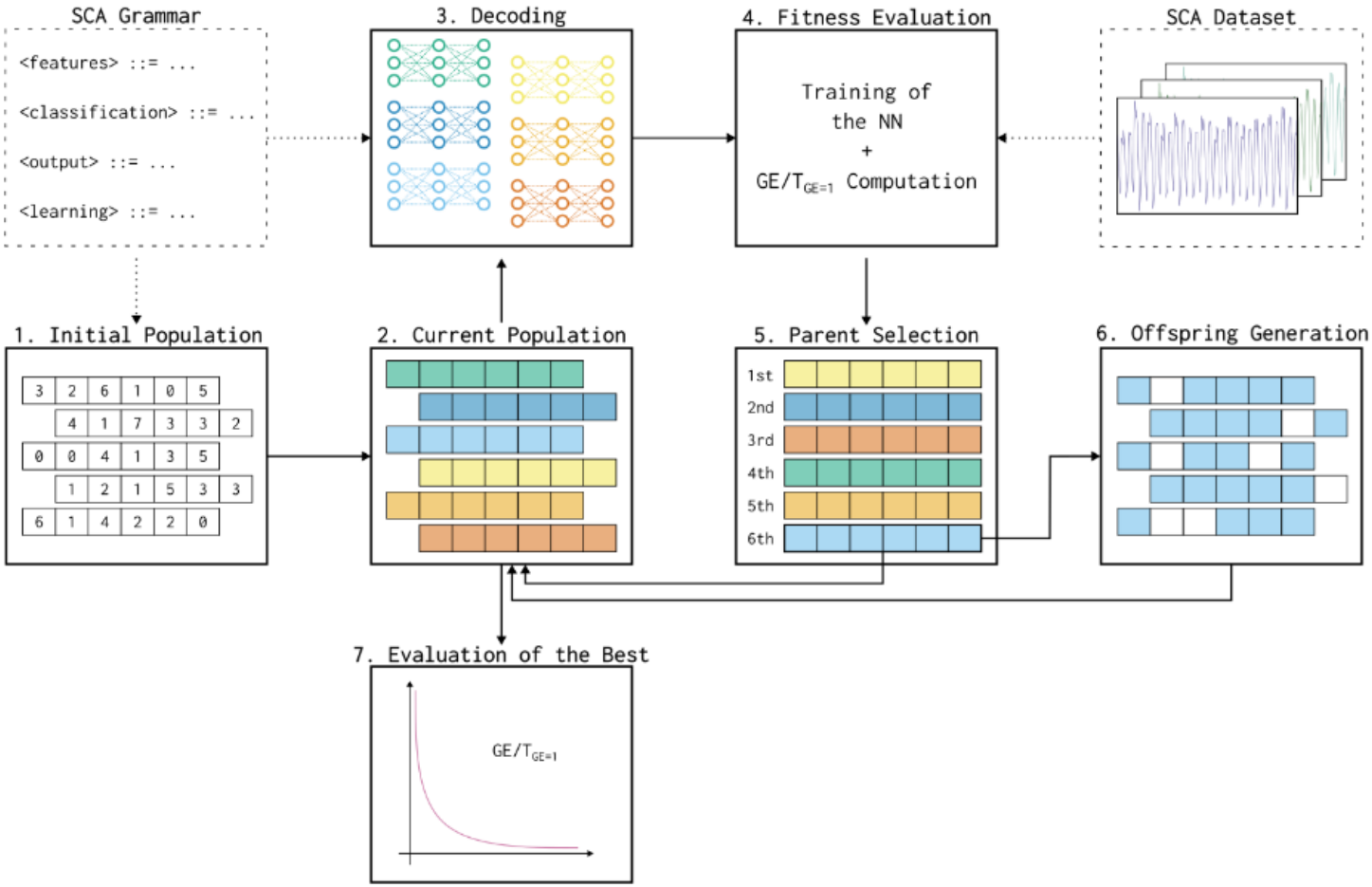


Figure 1: An example of GE decoding, using a grammar for evolving expressions.

```
<features> ::= <convolution> | <pooling>
<convolution> ::= layer:conv1d [num-filters,int,1,4,512]
<pooling> ::= <pool-type> [kernel-size,int,1,2,20] [stride,int,1,2,20]
<pool-type> ::= layer:pool-avg1d | layer:pool-max1d
...
<classification> ::= <fully-connected> | <dropout>
...
<learning> ::= <rmsprop> <early-stop> [batch_size,int,1,50,500]
             | ...
...
```

Side-channel Analysis and Grammatical Evolution



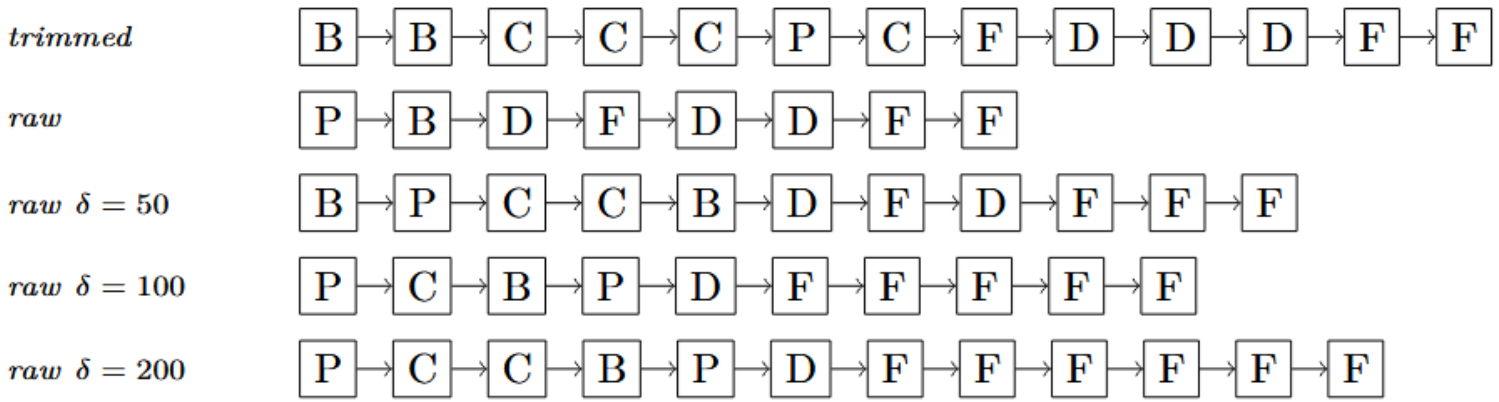
Side-channel Analysis and Grammatical Evolution

```
<features> ::= <convolution>
             | <pooling>
             | <batch-norm>
<convolution> ::= layer:conv1d [num-filters,int,1,4,512]
    ↪ [filter-shape,int,1,2,80] [stride,int,1,30,50] <activation-function>
<pooling> ::= <pool-type> [kernel-size,int,1,2,20] [stride,int,1,2,20]
<pool-type> ::= layer:pool-avg1d | layer:pool-max1d
<batch-norm> ::= layer:batch-norm
<classification> ::= <fully-connected> | <dropout>
<fully-connected> ::= layer:fc <activation-function> <regularizer>
    ↪ [num-units,int,1,10,1000]
<regularizer> ::= <regularizer-type> [regrate,float,1,0.00001,0.05]
<regularizer-type> ::= reg:l1 | reg:l2 | reg:none
<dropout> ::= layer:dropout [rate,float,1,0.05,0.5]
<activation-function> ::= act:relu | act:selu
<output> ::= layer:fc act:softmax num-units:256 reg:none
<learning> ::= <optimizer> [lr,float,1,0.0001,0.001] <early-stop>
    ↪ [batch_size,int,1,100,1000] epochs:100
<optimizer> ::= learning:adam | learning:rmsprop
<early-stop> ::= [early_stop,int,1,5,20]
```

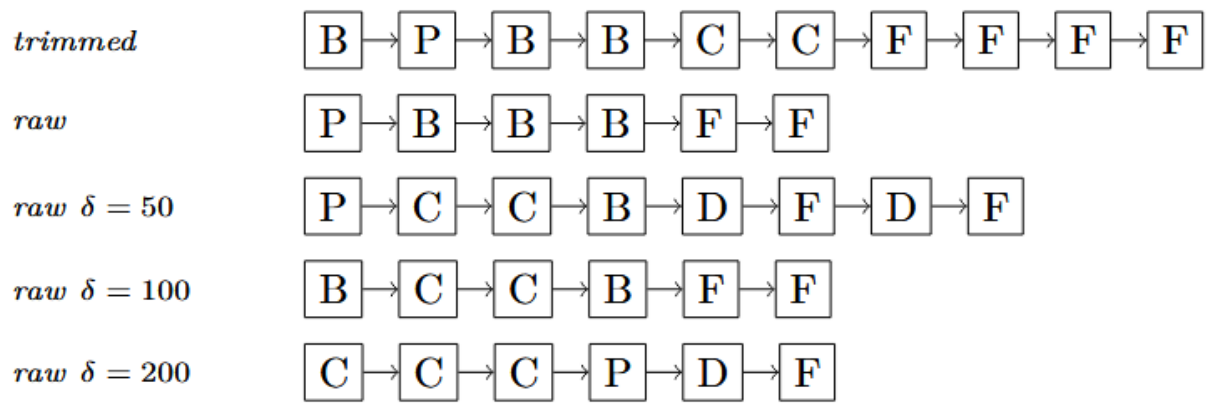
Side-channel Analysis and Grammatical Evolution

Dataset	$T_{GE=1}$					
	SCAGE	RS	[PWP22]	[HCM23]	[AGF22]	[LP24]
trimmed	23	85	78	-	120	-
raw $\delta_{max} = 0$	1	1	1	5	-	-
raw $\delta_{max} = 50$	1	2	-	-	-	33
raw $\delta_{max} = 100$	1	x	73	5	-	251
raw $\delta_{max} = 200$	1	x	-	4	-	44

Side-channel Analysis and Grammatical Evolution

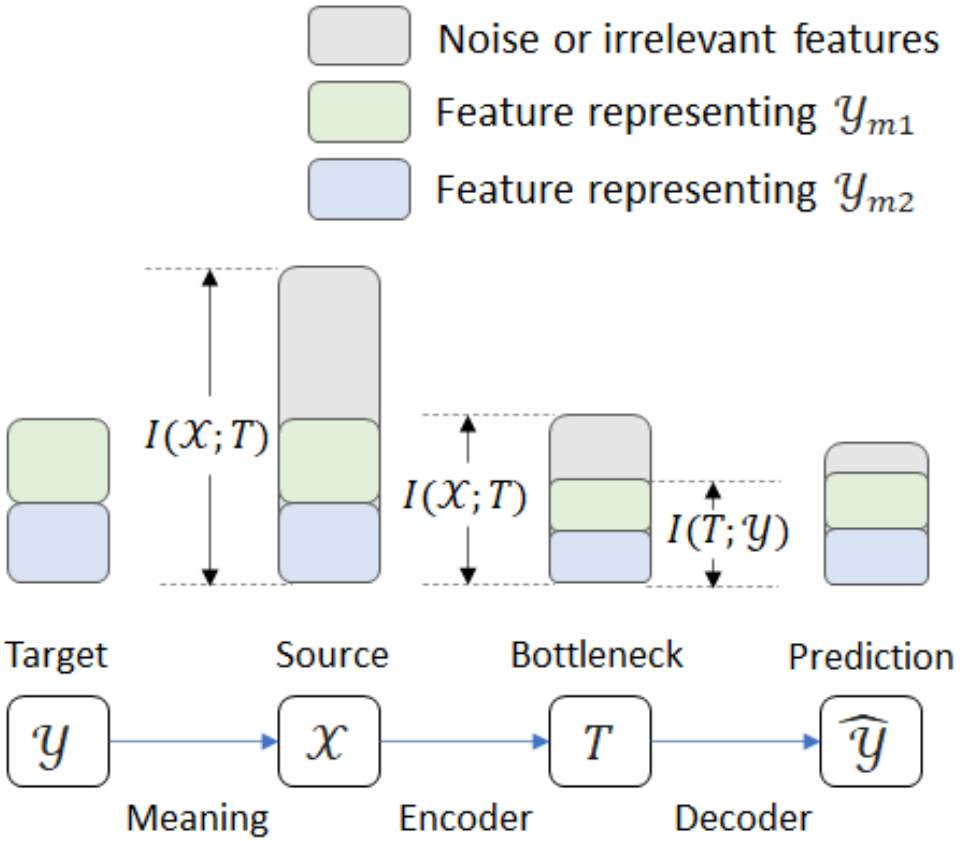


(a) ASCADf NN architectures

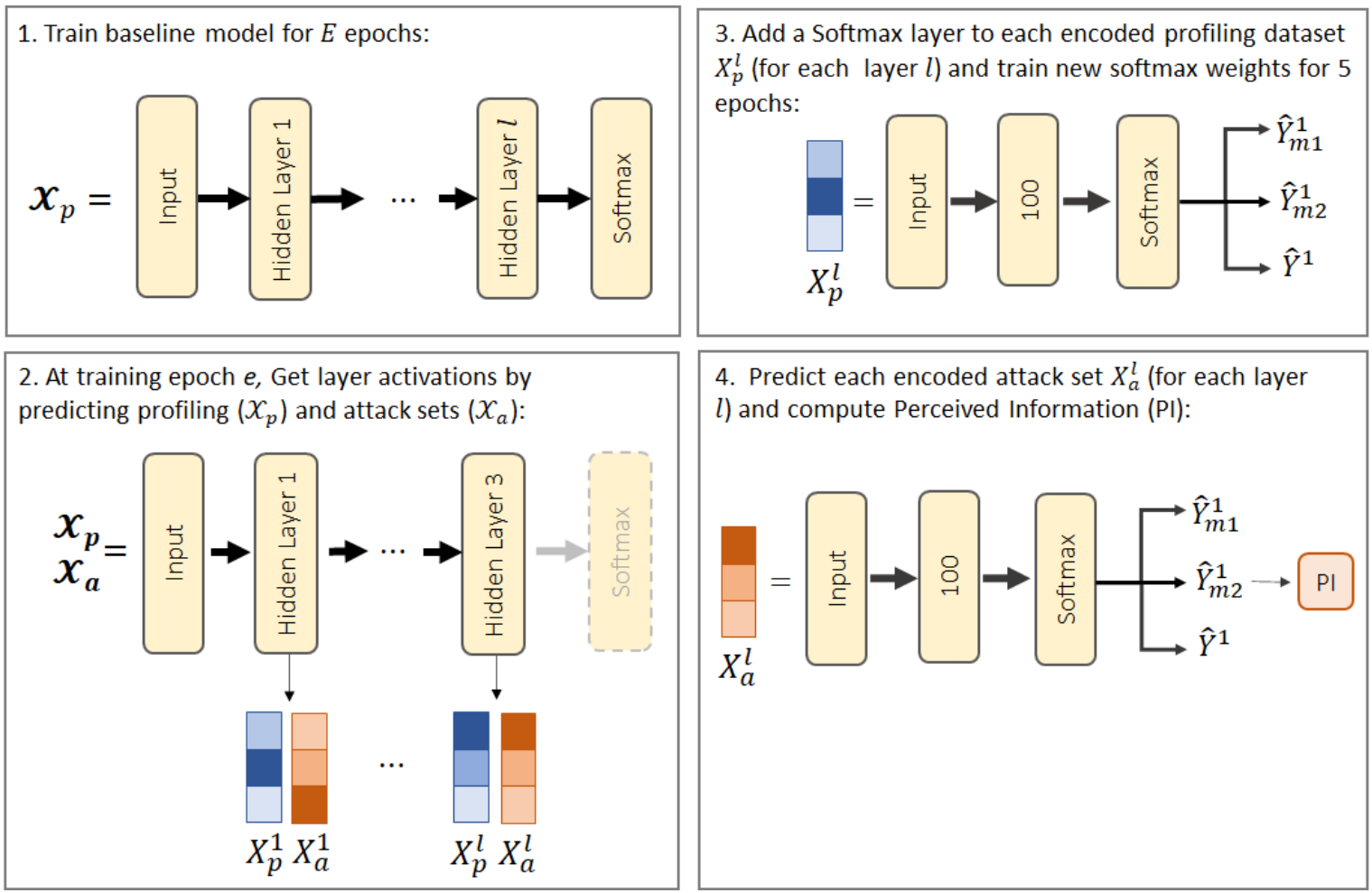


(b) ASCADr NN architectures

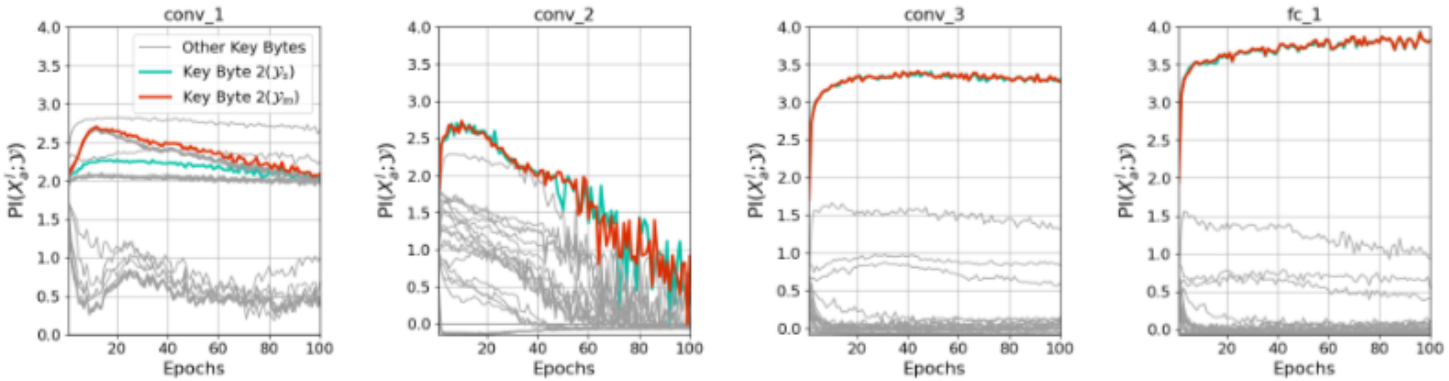
Information Bottleneck for SCA



Explainability Approach



Explainability Approach



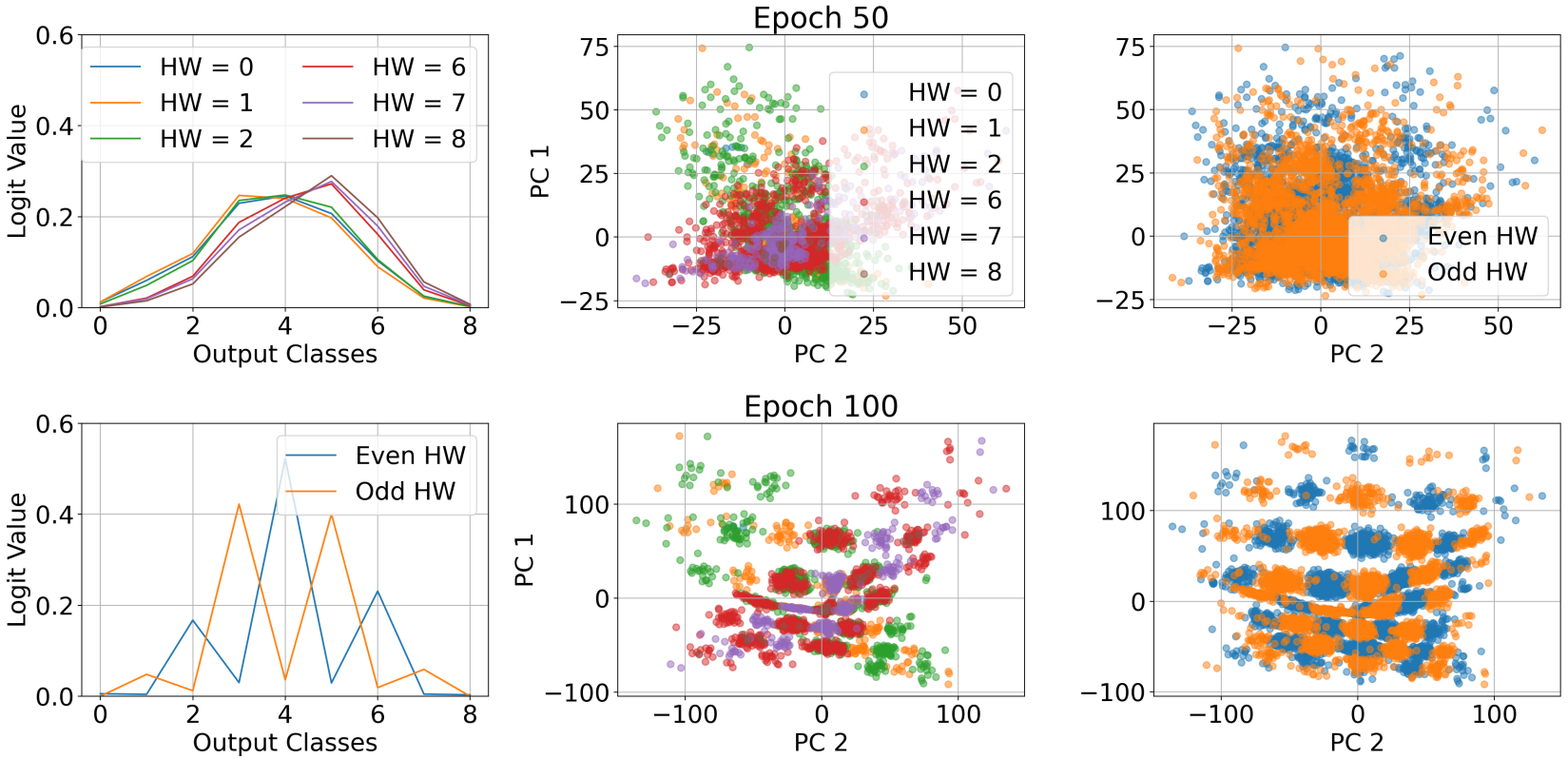
(a) conv_1 (b) conv_2 (c) conv_3 (d) fc_1

Figure : The compression of irrelevant features with CNN. Red and green lines indicate the perceived information from relevant features, which are the mask $\mathcal{Y}_m[2]$ and the masked S-Box output $\mathcal{Y}_s[2]$ labels related to key byte 2. Irrelevant features associated with the rest of the key bytes are illustrated as gray lines.

AI Explainability and SCA

- **Where**. Results indicate that compression of $\mathcal{X}$ mostly happens in the first hidden layer in any MLP configuration. Generalization to $\mathcal{Y}$ is stronger in hidden layers closer to the output layer, and this conclusion comes from higher $\widehat{PI}(X_a^l; \mathcal{Y})$ values obtained for the outer layer in comparison to hidden layers closer to the input layer.
- **What**. Tto generalize to $\mathcal{Y}$, the first hidden layer compresses noise and irrelevant features and transmits information from relevant secret shares to the subsequent hidden layers. This also suggests that hidden layers perform unmasking by combining the two secret shares.

Mechanistic Interpretability



Non-profiled Deep Learning-based SCA

- Non-profiling deep learning-based SCA was first proposed by B. Timon in 2019, with an approach called Differential Deep Learning Analysis (DDLA).
- Although its performance is better than conventional non-profiling attacks such as CPA, it is mainly criticized for practical limitations.
- To attack one key byte, DDLA needs to train a deep neural network 256 times (one network for each key hypothesis for commonly attacked byte-oriented cipher like AES) to brute force all possible key bytes.
- Such an attack may easily become impractical, considering a dataset with millions of measurements.

Non-profiled Deep Learning-based SCA

- First approaches to improve the performance of unsupervised DLSCA concentrated on parallel network architectures.
- While it decreased time, it increased memory.
- More recent approaches consider multi-output learning.
- Multi-output classification (MOC) and multi-output regression (MOR).
- MOC is faster but does not work for the ID leakage model.

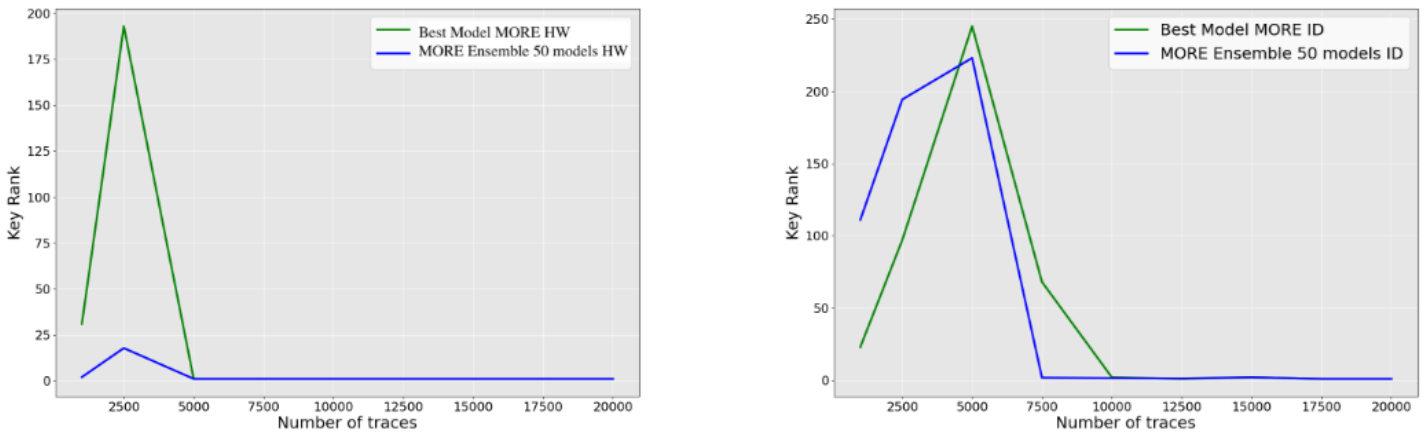
Non-profiled Deep Learning-based SCA

- **Classification** problems involve categorizing input data into discrete classes or categories.
- In these problems, the output is a discrete value representing the class or category that an input data point belongs to.
- Various algorithms can be used for classification, such as logistic regression, decision trees, support vector machines, and neural networks.
- **Regression** problems involve predicting continuous output values based on input data.
- In these problems, the output is a continuous numeric value, often representing a measurement or quantity.
- Common regression algorithms include linear regression, polynomial regression, ridge regression, and support vector regression.

Multi-output Regression (MOR) for SCA

- Rather than training 256 models, where each aims to classify hypothetical labels with probabilities accurately, MOR utilizes the concept of multi-output regression, which seeks to *regress* the prediction outputs to the actual label values.
- A model is trained to map input leakage traces to the actual values of all possible $y_i(k)$, which denotes the key-related intermediate data (label) given a specific key byte k.
- The most likely key k^* is determined by identifying the smallest loss measured by MSE.

MORE



(a) `ASCADf`, Key rank, HW leakage (b) `ASCADf`, Key rank, ID leakage

Figure: ASCAD fixed key.

Plaintext/Ciphertext-based Non-profiling SCA

- Supervised deep learning-based SCA learns a mapping based on known plaintexts and keys.
- Then, the adversary estimates the conditional probability given a leakage trace with the unknown key.
- In unsupervised setting, we do not know the key.
- But, the key is commonly fixed for all traces.
- The label $\mathsf{I}(k, d_i)$ and d_i would satisfy:

$$d_i \longmapsto \mathsf{I}(k, d_i). \tag{1}$$

$$\mathsf{I}(k, d_i) = \mathsf{map}_k(d_i), \tag{2}$$

Plaintext/Ciphertext-based Non-profiling SCA

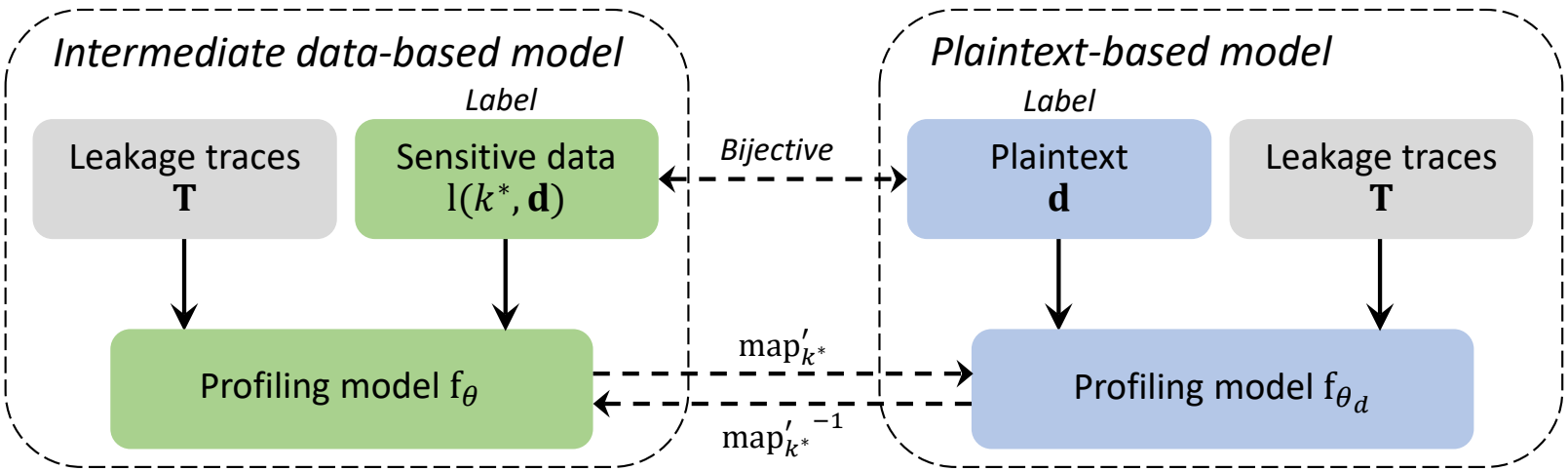


Figure: The relationship between intermediate data-based model and plaintext-based model.

Plaintext/Ciphertext-based Non-profiling SCA

- For supervised DLSCA, if a profiling model is generalized well on the leakage traces, the probability of the incorrect value is closely correlated with the correct label.
- In unsupervised setting, we can still estimate the label distance, providing us with plaintext/ciphertext distribution.

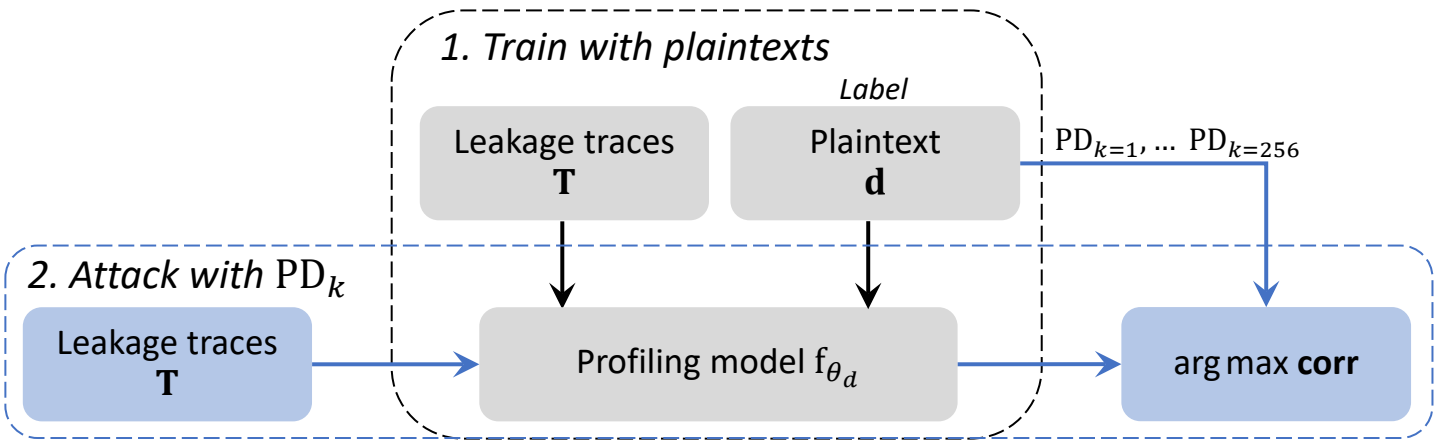


Figure: Attack scheme of the Plaintext Labeling Deep Learning (PLDL).

$$k^* = \arg\max_k \text{corr}(\text{PD}_k^{\text{M}}(\mathbf{d}),\ \text{f}_{\boldsymbol{\theta}_d}(\mathbf{T})),\ k \in \mathcal{K}. \qquad (3)$$

Plaintext/Ciphertext-based Non-profiling SCA

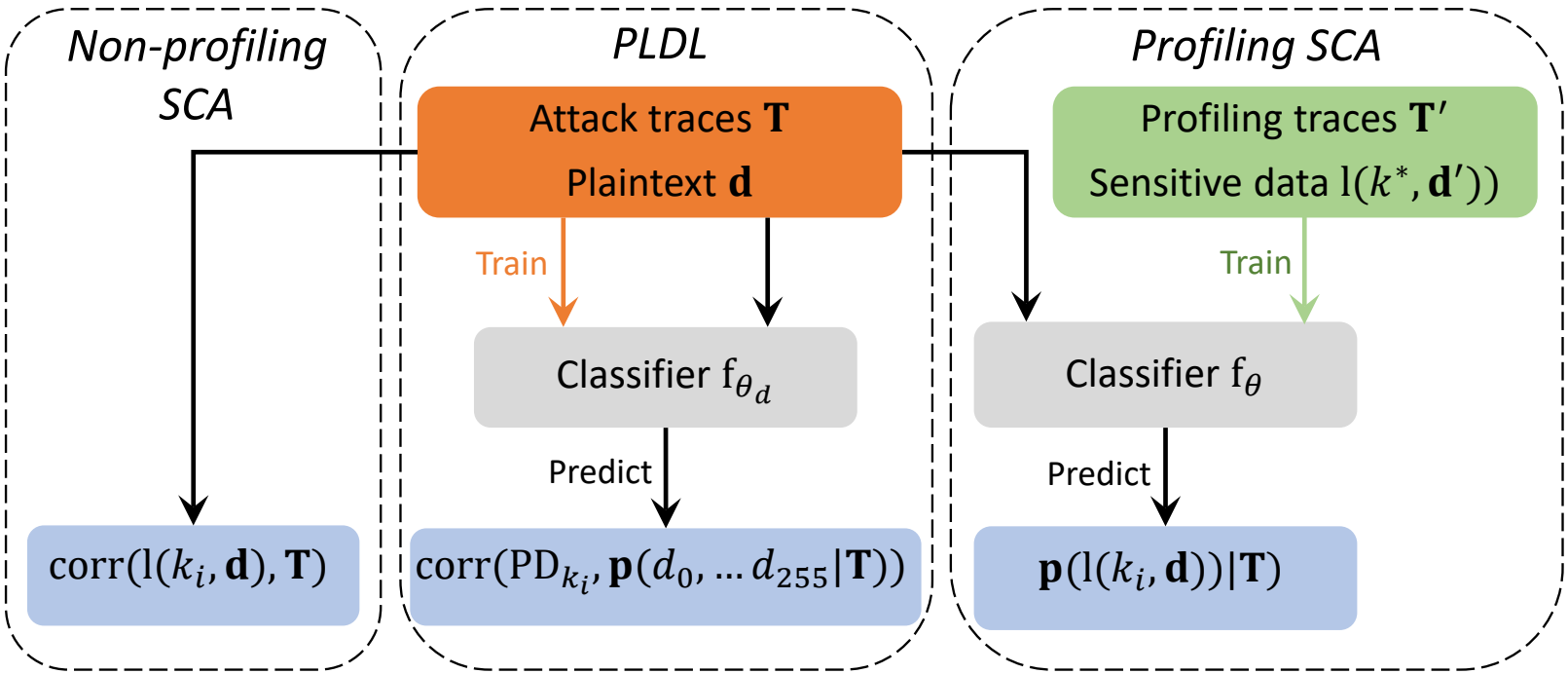


Figure: A demonstration of non-profiling SCA, PLDL, and profiling SCA.

Plaintext/Ciphertext-based Non-profiling SCA

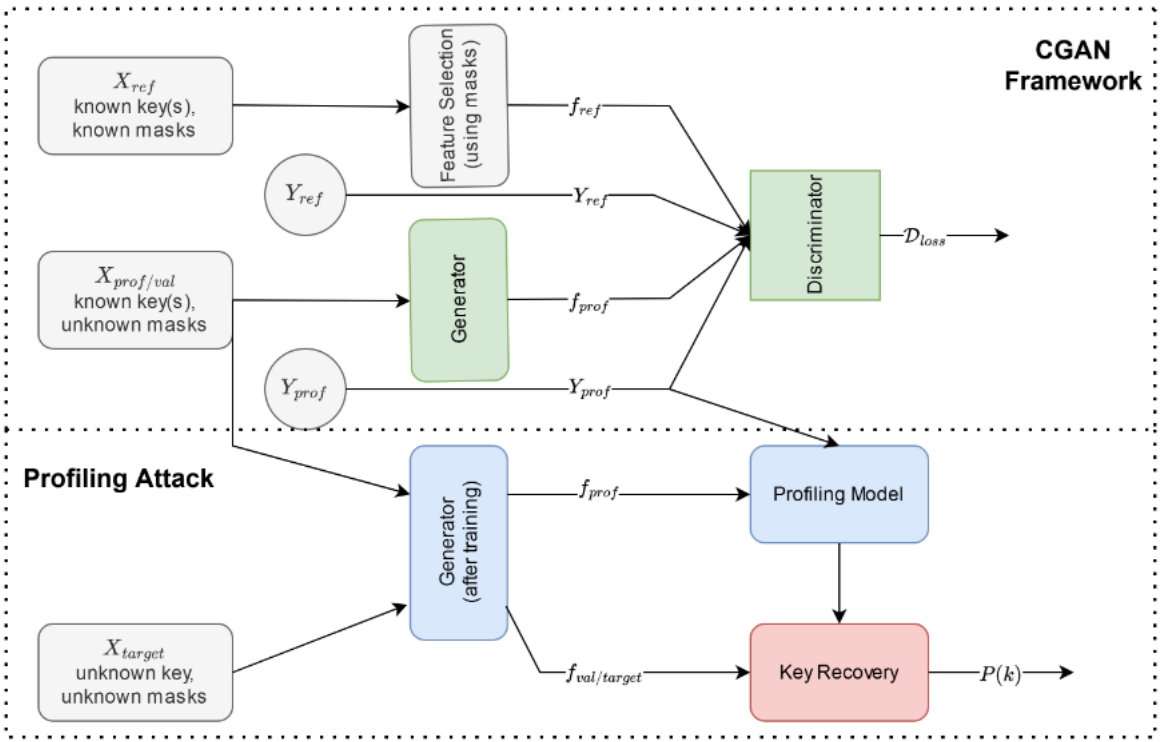
Table: Performance benchmark with non-profiling attacks.

Dataset	CPA	MOR	DDLA	PLDL
ASCAD_F	KR161/KR47	1 957/638	KR7/309	**8**/**111**
ASCAD_R	KR64/KR8	KR28/KR9	27 266/KR48	**20**/**19**
CHES_CTF	KR139/KR220	KR6/KR31	KR54/KR85	**6 121**/**KR2**
AES_RD	KR2/KR31	KR33/3 112	2 541/KR2	**1**/**57**
AES_HD	KR19/KR145	5 593/KR10	KR26/KR20	**60**/**KR6**

From Black Box to White Box

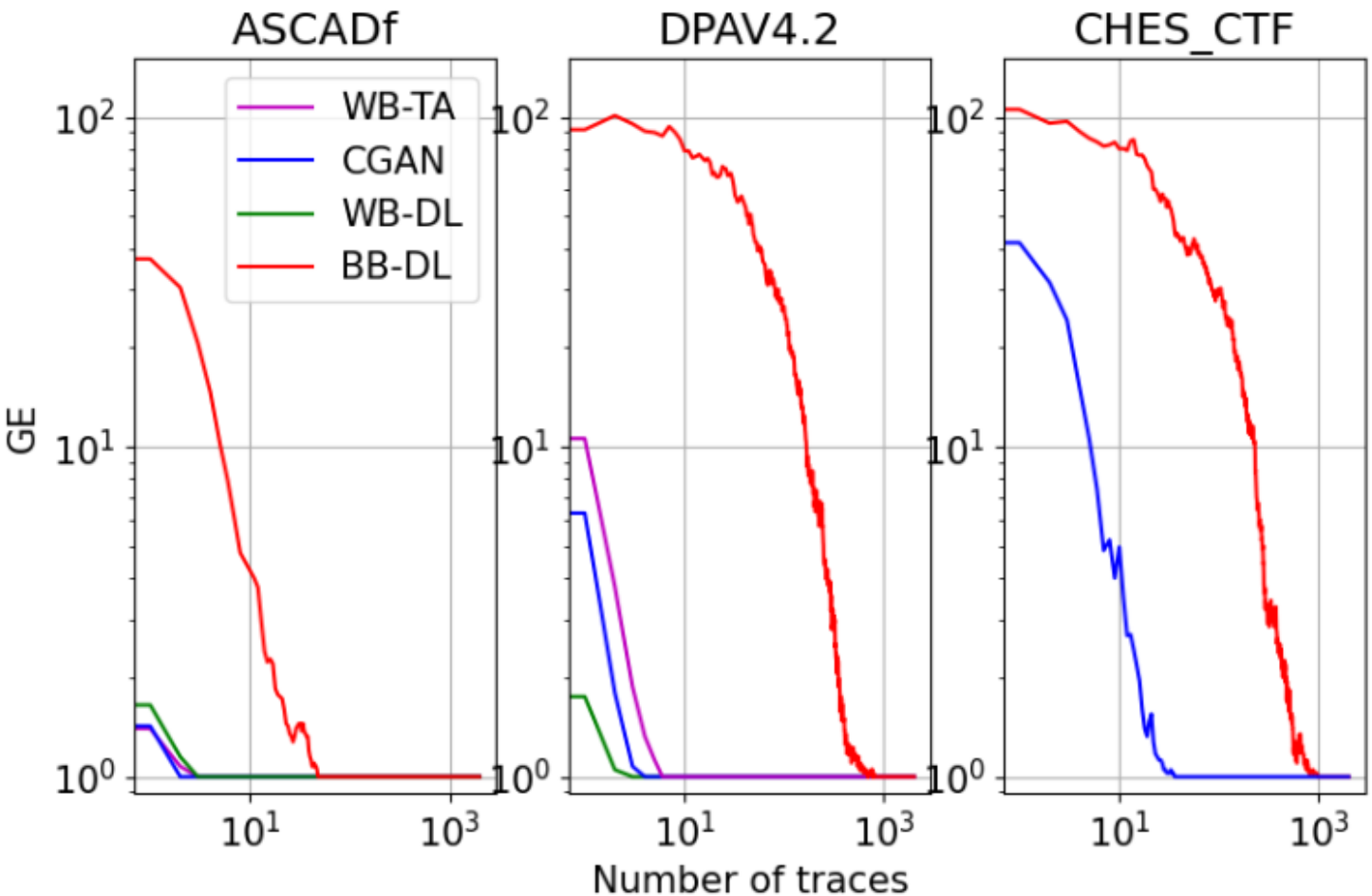
- The adversary possesses a similar implementation that can be used as a white-box reference design.
- We create an adversarial dataset by extracting features or points of interest from this reference design.
- These features are then utilized for training a conditional generative adversarial network (CGAN) framework, enabling a generative model to extract features from high-order leakages in protected implementation without any assumptions about the masking scheme or secret masks.

CGAN Framework



Proposed CGAN-SCA framework. X_{ref} is the reference dataset with known secrets (i.e., masks and keys), X_{prof} is the profiling set with known key(s) and unknown masks, and X_{target} is the target device with unknown key and mask. Y_{ref} and Y_{prof} are corresponding labels to the X_{ref} and X_{prof} datasets, respectively.

CGAN Profiling and Attack



GE results for key-byte 2 for various targets and methods (ref: ASCADr)

Leakage Model-flexible DLSCA

- A deep learning approach with a flexible leakage model, referred to as the multi-bit model.
- Instead of trying to learn a pre-determined representation of the target intermediate data, we utilize the concept of the stochastic model to decompose the label into bits.
- Then, the deep learning model is used to classify each bit independently.
- This versatile multi-bit model can adjust to existing leakage models like the Hamming weight and Most Significant Bit while also possessing the flexibility to adapt to complex leakage scenarios.

Leakage Model-flexible DLSCA

- Stochastic model approximates the linear portion of function to be learned using base functions but fails to encompass non-linear parts.
- Furthermore, it neglects potential multivariate key-dependent noise terms.
- These two constraints limit the discriminative power when identifying different leakages, leading to mediocre performance when, for instance, dealing with low numbers of side-channel traces

Leakage Model-flexible DLSCA

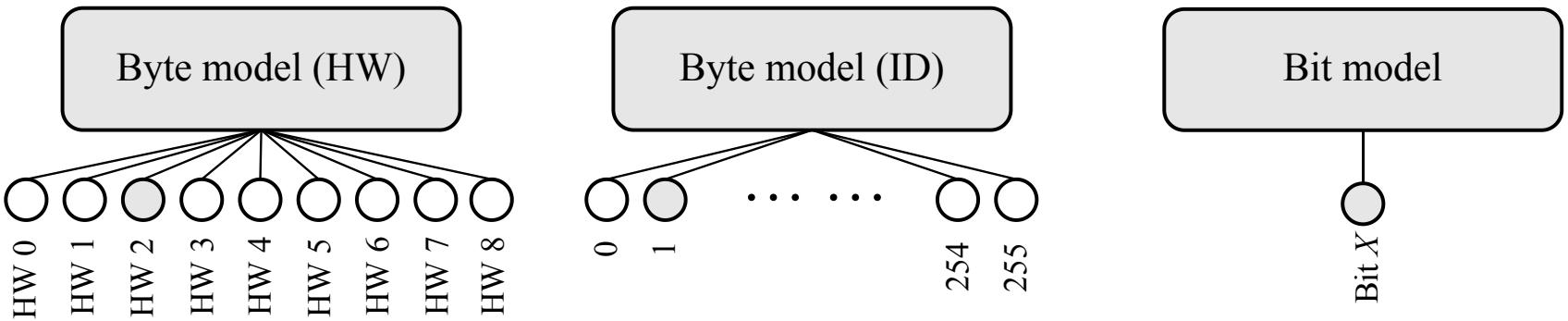


Figure: Conventional DLSCA models.

Leakage Model-flexible DLSCA

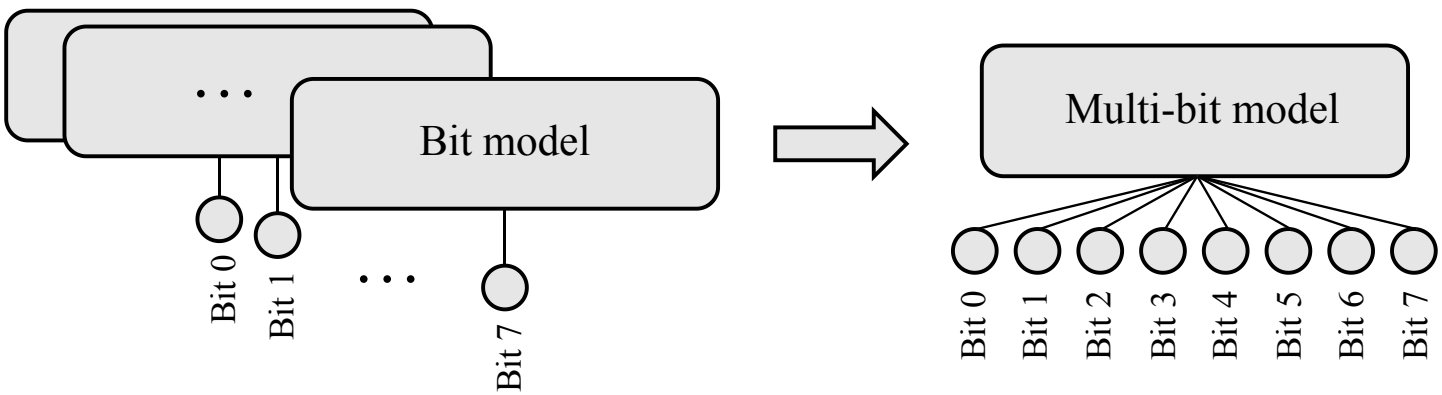


Figure: Multi-bit model.

Leakage Model-flexible DLSCA

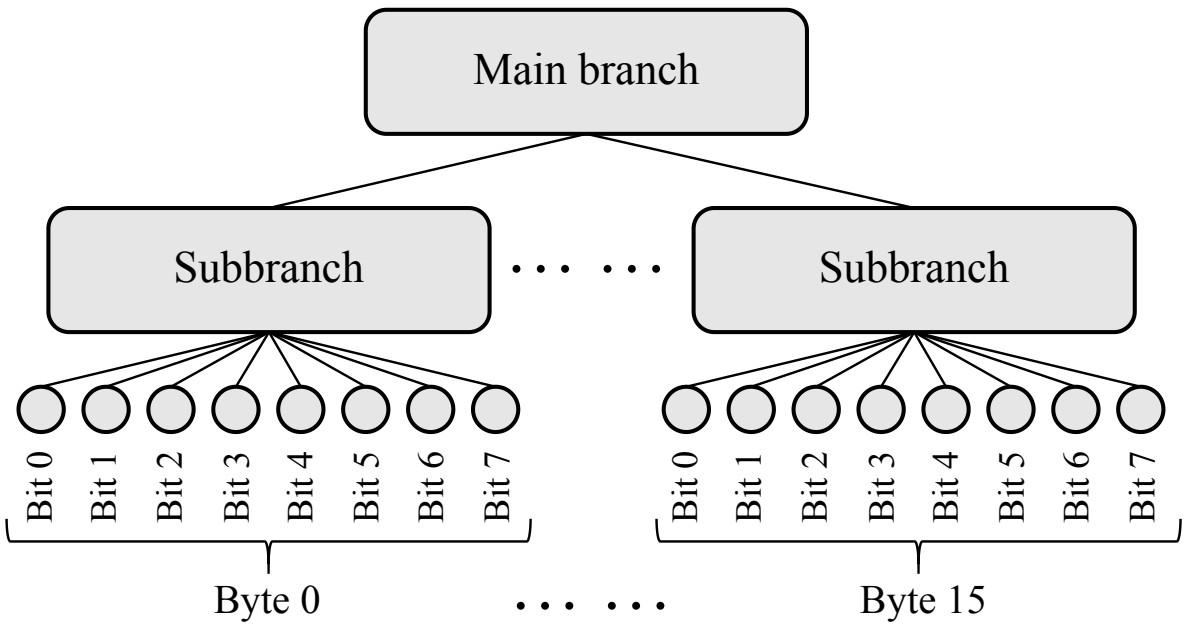


Figure: Multi-byte multi-bit DLSCA.

Leakage Model-flexible DLSCA

T_{GE0} of each subkey for the ASCAD_F dataset.

	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}
MMB	1	4	7	20	9	4	22	10	135	15	76	18	219	123	9	69
MB	1	4	5	19	9	7	23	17	185	6	37	35	59	67	9	51
HW	4	4	494	282	419	372	535	590	x	628	1 402	792	x	x	202	x
ID	2	2	x	290	66	74	2 415	111	x	126	1 272	x	x	x	23	x
LSB	17	17	1 552	43	63	81	208	57	2 683	49	138	x	x	1 084	44	61
MSB	27	33	x	1 351	1 079	983	984	866	x	623	x	517	x	x	3 544	x

Figure: Multi-byte multi-bit DLSCA Results.

Collision-based DLSCA

- Side-channel collision attack (SCCA) is considered a non-profiling SCA (as it does not rely on a profiling device) but follows a different attack principle.
- It exploits data inter-dependence leaked during cryptographic procedures by targeting the collision of an internal state, which is more likely to coincide between two cryptographic operations.

Collision-based DLSCA

- Concretely, an adversary monitors the side-channel information while the system processes different inputs and then searches for repeated leakage patterns signifying a collision event.
- When a collision is detected, the adversary uses this information to infer insights about the inter-dependencies of different key sections or the algorithm's internal state.

Collision-based DLSCA

- For instance, let us consider the SubBytes operation of the Advanced Encryption Standard (AES) with the same substitution box (Sbox).
- The same data has been processed if two different Sbox operations lead to an identical side-channel pattern.

Collision-based DLSCA

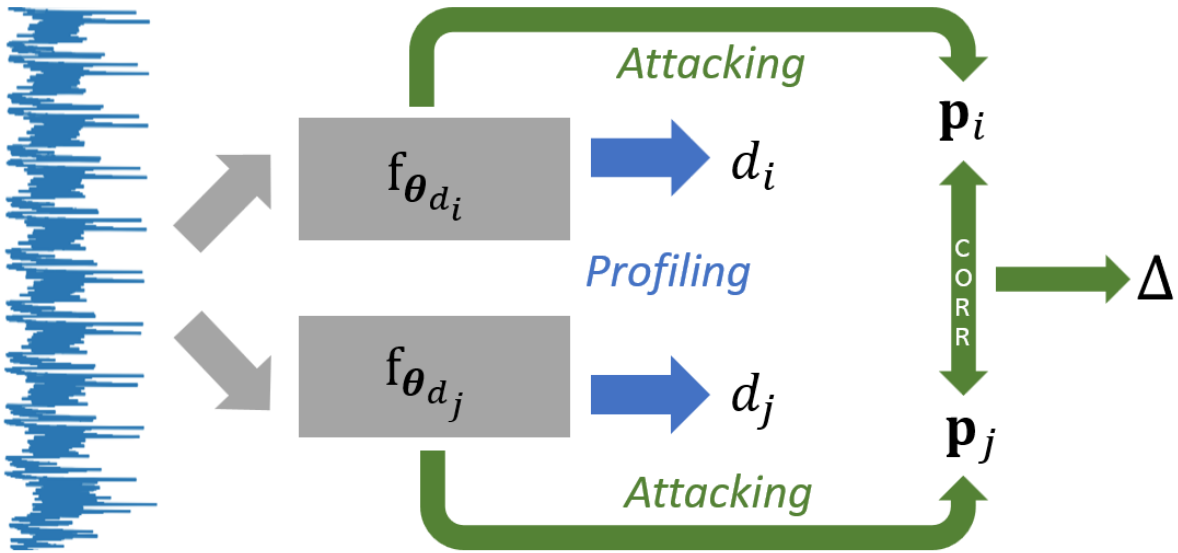


Figure: Plaintext-based correlation.

Collision-based DLSCA

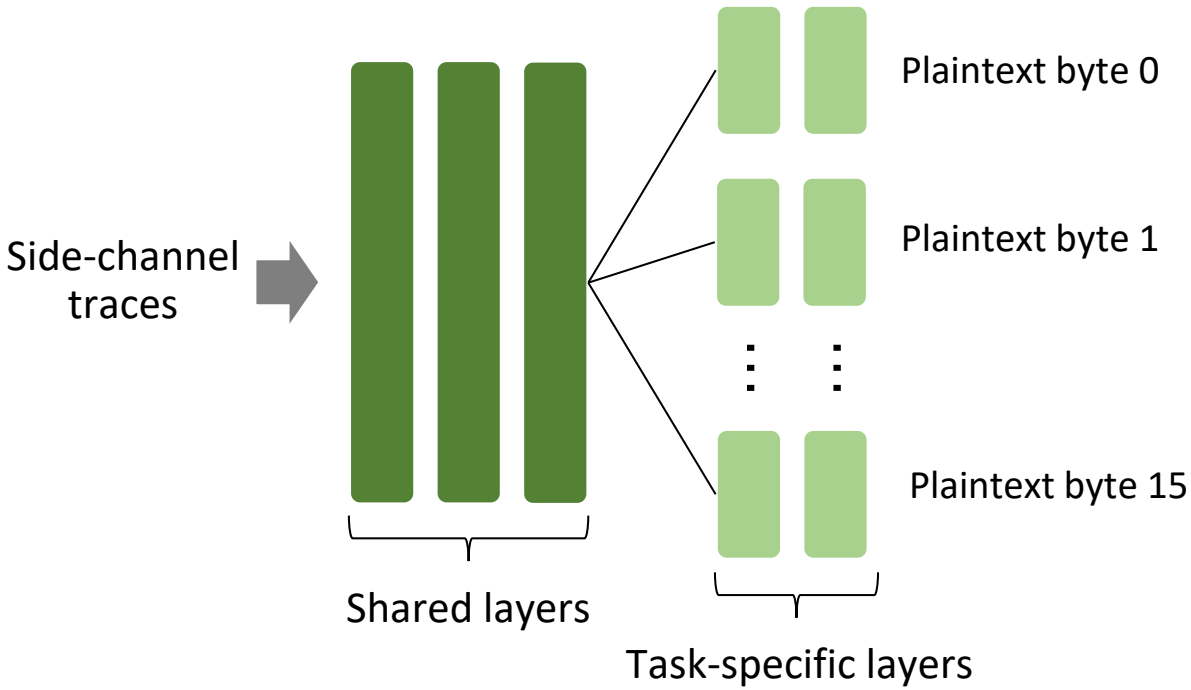


Figure: Deep learning architecture of plaintext-based SCCA.

Collision-based DLSCA

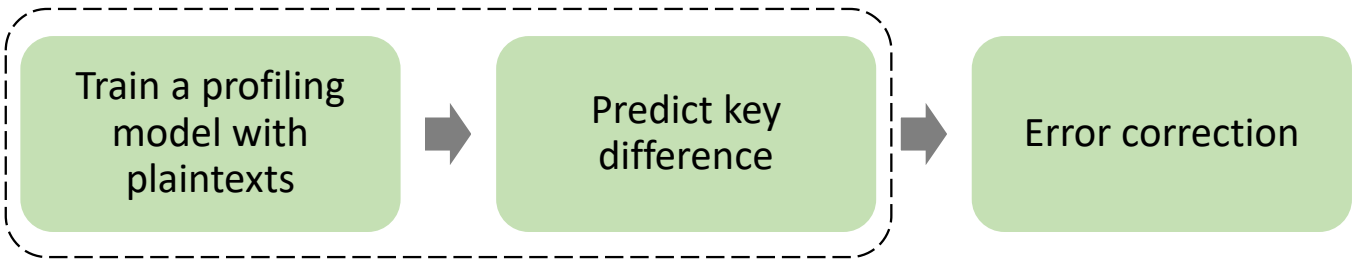


Figure: Plaintext-based SCCA.

Collision-based DLSCA

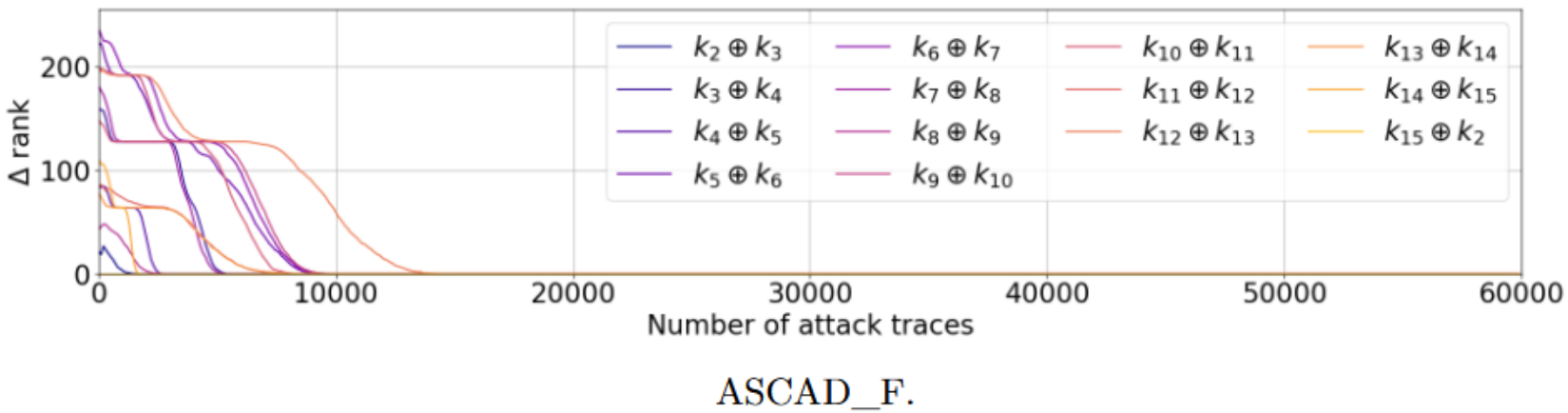


Figure: Example of results.

Collision-based DLSCA

	Attack setting	Samples	OSR	Computation time
DDLA	Known leakage offset	Refined leakage interval	100%	3h 14min
DDLA	Black box	Full Set	93%	5h 23min
DL-SCCA	Known leakage offset	Full Set	0%/100%	23min
DL-SCCA	Black box	Full Set	0%/76%	3h 47min
This work	Black box	Full Set	100%/100%	5min

Figure: Performance comparison.

Outline

Conclusions

- Deep learning-based SCA is rather active domain (and does not show signs of slowing down).
- As there are so many works available, it is challenging to recognize what and when to use.
- The results are very good but there are potential issues.
- The big challenges are unsupervised deep learning-based SCA and explainable AI for SCA.

And More Challenges

- Saturation of the domain.
- New (ciphers, hardware, better countermeasures) targets.
- Lack of good datasets.
- Generative AI.
- "Just" an application of deep learning.
- Real-world applicability.
- ...

Questions?

Thank you for your attention!

stjepan.picek@ru.nl