I Know What Your Layers Did: Layer-wise Explainability of Deep Learning Side-channel Analysis

Guilherme Perin[1], Lichao Wu[2], and Stjepan Picek[3]

[1] Leiden University, The Netherlands
[2] Delft University of Technology, The Netherlands
[3] Radboud University, The Netherlands

Abstract. Masked cryptographic implementations can be vulnerable to higher-order attacks. For instance, deep neural networks have proven effective for second-order profiling side-channel attacks even in a black-box setting (no prior knowledge of masks and implementation details). While such attacks have been successful, no explanations were provided for understanding why a variety of deep neural networks can (or cannot) learn high-order leakages and what the limitations are. In other words, we lack the explainability on neural network layers combining (or not) unknown and random secret shares, which is a necessary step to defeat, e.g., Boolean masking countermeasures.
In this paper, we use information-theoretic metrics to explain the internal activities of deep neural network layers. We propose a novel methodology for the explainability of deep learning-based profiling side-channel analysis (denoted ExDL-SCA) to understand the processing of secret masks. Inspired by the Information Bottleneck theory, our explainability methodology uses perceived information to explain and detect the different phenomena that occur in deep neural networks, such as fitting, compression, and generalization. We provide experimental results on masked AES datasets showing where, what, and why deep neural networks learn relevant features from input trace sets while compressing irrelevant ones, including noise. This paper opens new perspectives for understanding the role of different neural network layers in profiling side-channel attacks.

Keywords: Side-channel Analysis, Deep learning, Countermeasures, Explainability, Perceived Information, Information Bottleneck Theory

1 Introduction

Side-channel attacks (SCA) represent powerful non-invasive attacks that exploit unintentional leakages of confidential information from electronic devices [24]. During cryptographic executions, the devices leak information through different side channels, such as power consumption [21], electromagnetic emission [32], or execution time [20]. Depending on the application, attacks revealing secret information can lead to serious consequences for the security industry [33]. Thus, chip manufacturers are interested in assessing the vulnerabilities of their designs before putting them on the market or into a certification process. To address this goal, security evaluators deploy different side-channel attacks that can be categorized as profiling (e.g., Template Attacks [10]) and non-profiling attacks (e.g., Simple Power Analysis [20] and Differential Power Analysis [21]).

Deep learning (DL-SCA) has drawn significant interest from researchers in the SCA domain [31]. Powerful attacks against protected cryptographic implementations have been demonstrated [23,7,19], especially against datasets containing first-order (Boolean) masked AES software or hardware [39] implementations. Moreover, the SCA community constantly manages to improve the performance of deep learning. For example, the first publication using the ASCAD dataset with the fixed key required around 400 traces to break the target [3]. Today, we can break the same dataset with a single attack trace and simpler neural network architecture [29]. Unfortunately, despite all of the advances in enhancing the attack efficiency, we still lack the knowledge to understand *why* neural networks break a target.[4] Although past research put effort into the understanding of some aspects of deep learning models [27,17,42], there is still an evident lack of knowledge about, for instance, deep neural networks learning to defeat masking countermeasures. We refer interested readers to several recognized challenges in deep learning-based SCA, especially the one connected to the explainability of the masking countermeasure processing [31].

We argue that the ability to explain why a machine learning model behaves in a certain way is (at least) as important as improving the attack performance. With a deeper understanding of the neural network, an evaluator can: 1) mount more powerful attacks, 2) improve the security of devices, and 3) ultimately offer devices that are resilient against the strongest attacks. To conclude, we require **Ex**plainable **D**eep **L**earning-based **S**ide-**C**hannel **A**nalysis methodology: **ExDL-SCA**. ExDL-SCA has two main goals: 1) explain where the leakage comes from and 2) explain how a profiling model defeats different countermeasures. To reach those goals, we propose the following questions to be answered with ExDL-SCA:

1. **Where**. By answering this question, we can explain the contribution of different layers in a neural network.
2. **What**. By answering this question, we can explain what happens with relevant or irrelevant features and if the information is processed (fitting, compression) to the subsequent layers of a neural network.
3. **Why**. By answering this question, we can explain why a neural network behaves in a certain way, e.g., breaking a target or failing to do so.

This paper proposes an explainability methodology that infers how much information the black-box model learns from secret masks. Secret masks are not

[4] Deep neural networks usually have complex architectures that are difficult to interpret and explain. By interpretable machine learning, we consider designing machine learning models that are inherently interpretable or answering the question of how the model works [25]. By explainability (explainable AI - XAI), we consider how to provide post hoc explanations of the black-box models.

supplied during the training of a black-box profiling model and are only considered for explainability. For that, we consider information-theoretic methods. The starting point of our explainability methodology is Information Bottleneck Theory (IB) [37,35], which has sparked a lot of interest from deep learning community as a potential theoretical framework to explain how deep neural networks achieve enormous success in many different applications. In essence, IB theory suggests that deep neural network training undergoes two different phases: fitting and compression. The fitting phase is supposedly fast and is characterized by hidden layers trying to maximize information about $\mathcal{X}$ while compression is slower, and it is responsible for the generalization ability of the model. According to [35], during the fitting phase, the model already shows generalization, which is enhanced during the compression phase. When and how these phases happen in a specific model depends on the model architecture, hyperparameters, and the dataset's characteristics. The compression phase is particularly important, as in this phase, the network starts to compress noise and other irrelevant features while preserving only relevant features from input training data $\mathcal{X}$. For that, the IB theory requires the computation of mutual information between (usually) high dimensional input data $\mathcal{X}$ (such as side-channel measurements) and (potentially) high dimensional intermediate network representations T (the output of a hidden layer), i.e., $I(\mathcal{X}, T)$. However, as we will explain in this paper, computing $I(\mathcal{X}, T)$ is particularly hard for discrete and high dimensional representations [14,34], which limits the estimation of the compression phase during training. As a solution, we adapt the IB framework to the Perceived Information [5] metric, which allows us to precisely explain fitting, compression, and generalization phases in different hidden layers. Thus, we verify, during training, **where**, **what**, and **why** every neural network layer learns from high-order leakages. Thus, our explainability methodology allows security evaluators to verify, with more specific information from hidden layers, what a profiling model learns (or not) from the implemented countermeasures. Our main contributions are:

1. We discuss explainability in the context of DL-SCA and recognize the three questions that need to be answered to provide (the core of) explainable DL-SCA.
2. We define a new methodology to quantify the information learned by hidden neural network layers during profiling. Our method allows an evaluator to measure how the input information leakage is learned and conveyed layer by layer in a deep neural network. Furthermore, our method can show in what layer the information bottleneck is inherently implemented to compress irrelevant input information (such as noise) and preserve relevant leakages to break masking countermeasures.
3. We provide experimental results on publicly available datasets and different neural network architectures. All our results indicate that the information bottleneck theory is a valid method to explain the different phases of deep neural network training. Then, we apply our explainability methodology to a combination of masking and desynchronization countermeasures, showing our approach to work even if different (multiple) countermeasures are used.

This paper is organized as follows. We start by providing background information in Section 2 while related works are discussed in Section 3. Section 4 introduces our novel explainability methodology, and Section 5 discusses the compression in deep networks with practical examples on different datasets. Experimental results with different protected AES datasets and neural network architectures are provided in Section 6. Finally, conclusions and future work directions are provided in Section 7.

2 Background

2.1 Notations and Terminology

We refer to $\mathcal{X}$ as a set of side-channel measurements, with $\mathtt{x}_i$ being the i-th observation of $\mathcal{X}$. $\mathcal{X}_p$ is a set of profiling side-channel measurements and $\mathcal{X}_a$ is the attack set with lengths n_p and n_a, respectively. Each side-channel measurement $\mathtt{x}_i$ represents the side-channel leakages of a cryptographic operation having input data $\mathtt{d}_i$ and encryption key $\mathtt{k}_i$.[5] We refer to $\mathcal{Y}$ as the set of hypothetical leakage values (or labels) for $\mathcal{X}$ where $\mathtt{y}_i = f(\mathtt{d}_i, \mathtt{k}_i)$ is one element of $\mathcal{Y}$, and $s(.)$ denotes a leakage selection function (i.e., $s(.)$ can be represented by an `S-box` operation in the first encryption AES round). In the case of masking countermeasures, $\mathtt{m}_r$ refers to (r)-th secret share. Alternatively, the term $\mathcal{Y}_f$ refers to the set of labels representing a feature in side-channel measurements (e.g., a secret share related to key byte index j).

With respect to information-theoretic notions, we refer to $\mathtt{p}(\mathtt{x}_i)$ as the probability of observing $\mathtt{x}_i$ and $\mathtt{p}(\mathtt{y}_i|\mathtt{x}_i)$ as the probability of observing $\mathtt{y}_i$ given $\mathtt{x}_i$. $H(\mathcal{X})$ is the entropy of $\mathcal{X}$ while $H(\mathcal{Y}|\mathcal{X})$ gives the conditional entropy of $\mathcal{Y}$ given $\mathcal{X}$. The mutual information between $\mathcal{X}$ and $\mathcal{Y}$ is given by $I(\mathcal{X};\mathcal{Y})$.

For neural network representations, we refer to T as an encoding providing an intermediate representation of $\mathcal{X}$ in a neural network (e.g., T could represent the feature map output of a convolution layer or the activations output of a fully-connected layer) and $\mathtt{t}_i$ is an observation of T. The term L refers to the number of hidden layers (excluding the output layer from the counting). The index of a hidden layer is given by l. The term X^l indicates the predicted output of a hidden layer l when the input data to the network is $\mathcal{X}$. Finally, $\widehat{\mathcal{Y}}$ is the output prediction from a neural network. Figure 1 provides an example of a convolutional neural network with representations.

In this paper, the term *sample* refers to the point of interest $\mathtt{x}_i[j]$ in a side-channel measurement $\mathtt{x}_i$. The term *feature* refers to the meaning of some information contained in $\mathcal{X}$. For example, when $\mathcal{X}$ represents the set of side-channel measurements from the AES encryption process, the leakage of an intermediate byte in each measurement $\mathtt{x}_i$, given by a label set $\mathcal{Y}$, is a feature of $\mathcal{X}$.

We also define specific notations for neural networks. Convolutional neural networks (CNNs) have a layer-wise structure according to the Eq. (1), where

[5] Instead of the encryption function and plaintext, it is also possible to consider decryption function and ciphertext, but for simplicity, we consider encryption only.

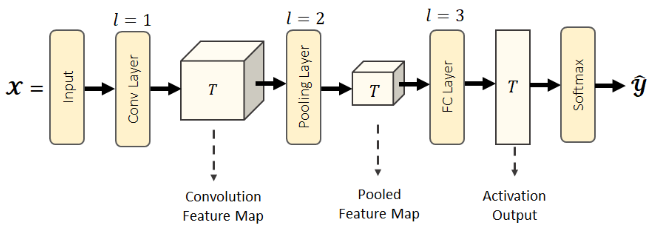

Fig. 1: Convolutional neural network intermediate representation.

$C(\texttt{fi}, \texttt{ks}, \texttt{st})$ denote a convolution layer with fi filters, kernel size ks, and stride st, A is the activation layer (which can be RE in case of relu, SE in case of selu, or E in case of elu), BN is a batch normalization layer, $AP(\texttt{ps}, \texttt{st})$ is an average pooling layer with pooling size ps and stride st, $FC(\texttt{ne})$ is a fully-connected layer with ne neurons and $S(\texttt{c})$ is a Softmax layer with c output neurons. The superscripts n_c and n_{fc} indicate the number of convolution blocks and fully-connected layers, respectively.

$$\mathcal{X} \rightarrow [C(\texttt{fi}, \texttt{ks}, \texttt{st}) \rightarrow A \rightarrow BN \rightarrow AP(\texttt{ps}, \texttt{st})]^{n_c} \rightarrow [FC(\texttt{ne}) \rightarrow A]^{n_{fc}} \rightarrow S(\texttt{c}) \rightarrow \widehat{\mathcal{Y}}. \tag{1}$$

Similarly, a multilayer perceptron (MLP) is defined according to the following layer-wise notation:

$$\mathcal{X} \rightarrow [FC(\texttt{ne}) \rightarrow A]^{n_{fc}} \rightarrow S(\texttt{c}) \rightarrow \widehat{\mathcal{Y}}. \tag{2}$$

2.2 Deep Learning-based Profiling SCA Against Masked Implementations

In classification applications, a neural network model represents a function that maps input data $\mathcal{X}$ into a finite number of output class probabilities $\hat{\mathcal{Y}}$. The mapping is performed by a function $f(\mathcal{X}, \theta) \rightarrow \hat{\mathcal{Y}}$, where θ is a set of parameters learned during the training phase by minimizing a loss function.[6] The learned mapping between input side-channel traces $\mathcal{X}$ and outputs probabilities $\hat{\mathcal{Y}}$ depends on the estimated number of classes presented in $\mathcal{X}$. This number of classes, $|\mathcal{Y}|$, is derived from a leakage function that indicates the hypothetical leakage value in a side-channel measurement.

To protect against side-channel attacks, masking is implemented to break the statistical dependence between side-channel measurements (e.g., power consumption) and hypothetical leakage values. For an m-order masking scheme, an intermediate byte b in a cryptographic algorithm is protected as follows:

$$\texttt{b}_m = \texttt{b} \diamond \texttt{m}_1 \diamond \texttt{m}_2 \cdots \diamond \texttt{m}_m, \tag{3}$$

[6] In this paper, we always consider categorical cross-entropy as the loss function since it is commonly used in deep learning-based SCA.

where $\diamond$ can indicate a Boolean [9], arithmetic [12], multiplicative [12], or affine [11] operation.

The leakage function $g(\cdot) = \mathcal{L}$ defined for a second-order profiling SCA is supposed to learn how to combine two unknown variables $\mathtt{m}_1$ and $\mathtt{m}_2$ according to:

$$\mathcal{L} = g(\mathtt{m}_1, \mathtt{m}_2) = \mathtt{m}_1 \diamond \mathtt{m}_2, \tag{4}$$

where g is a function mathematically combining two variables through operation $\diamond$.

In our analysis, $\mathtt{m}_1$ will always be given by an 8-bit mask share randomly generated for each encryption execution while $\mathtt{m}_2$ will be given by an 8-bit masked $\mathtt{S\text{-}box}$ output of the first AES encryption round, i.e., $\mathtt{m}_2 = \mathtt{S\text{-}box}(\mathtt{d}_i \oplus \mathtt{k}_i) \oplus \mathtt{m}_1$. For instance, the leakage function for a second-order attack on a masked AES implementation is defined as:

$$\mathcal{L} = g(\mathtt{m}_1, \mathtt{S\text{-}box}(\mathtt{d}_i \oplus \mathtt{k}_i) \oplus \mathtt{m}_1) = \mathtt{m}_1 \oplus \mathtt{S\text{-}box}(\mathtt{d}_i \oplus \mathtt{k}_i) \oplus \mathtt{m}_1 = \mathtt{S\text{-}box}(\mathtt{d}_i \oplus \mathtt{k}_i), \tag{5}$$

where $\diamond = \oplus$.

A side-channel measurement $\mathtt{x}_i$ containing second-order leakages must embed leakage of information from the treatment of masked $\mathtt{S\text{-}box}$ output ($\mathtt{S\text{-}box}(\mathtt{d}_i \oplus \mathtt{k}_i) \oplus \mathtt{m}_1$) and, at least, the loading of mask share $\mathtt{m}_1$ from memory. We can finally assume that to implement second-order profiling, a neural network must learn the following mapping that also includes a leakage function $\mathcal{L}$:

$$F(\mathcal{X}, \mathcal{L}, \theta) = F(\mathcal{X}, g(\mathtt{S\text{-}box}(\mathtt{d}_i \oplus \mathtt{k}_i) \oplus \mathtt{m}_1, \mathtt{m}_1), \theta) \to \hat{\mathcal{Y}}. \tag{6}$$

This means that a neural network can learn a mapping from side-channel traces $\mathcal{X}$ to output class probabilities $\hat{\mathcal{Y}}$ that represents (ideally) the *xor* between two random 8-bit variables $\mathtt{S\text{-}box}(\mathtt{d}_i \oplus \mathtt{k}_i) \oplus \mathtt{m}_1$ and $\mathtt{m}_1$. In essence, the leakage function represented by learned parameters θ defines how continuous variables or input features ($\mathcal{X}$) (i.e., raw or pre-processed trace samples) are converted into hypothetical discrete leakage values $g(\mathcal{X}) \xrightarrow{\mathcal{L}} \hat{\mathcal{Y}}$, where $\hat{\mathcal{Y}}$ can also be seen as the set of predicted labels. As a consequence, a neural network that can learn second-order leakages defines a mapping with an intermediate function that can be given by one or more hidden layers, which learns how to implement $g(\mathtt{S\text{-}box}(\mathtt{d}_i \oplus \mathtt{k}_i) \oplus \mathtt{m}_1, \mathtt{m}_1)$.

The trained neural network, therefore, implements the following path:

$$\mathcal{X} \to T_1 \to \cdots \to T_L \xrightarrow{Softmax} \hat{\mathcal{Y}} \equiv \mathcal{X} \to g(\mathtt{S\text{-}box}(\mathtt{d}_i \oplus \mathtt{k}_i) \oplus \mathtt{m}_1, \mathtt{m}_1) \xrightarrow{Softmax} \hat{\mathcal{Y}}. \tag{7}$$

From Eq. (7), we can immediately verify that

$$T_1 \to \cdots \to T_L \equiv g(\mathtt{m}_1, \mathtt{m}_2). \tag{8}$$

A loss function assesses the overall variation between expected (ground truth) labels $\mathcal{Y}$ and predicted labels $\hat{\mathcal{Y}}$. Common hypothetical leakage models for side-channel analysis include Identity, Hamming weight, Hamming distance, or simply bit-level models (e.g., the least or most significant bits). Besides the predefined number of classes for classification, the trained neural network has no other information on converting input features into labels. Therefore, training a model assumes that the network will automatically learn the leakage model properties by implementing the mapping from Eq. (6).

Following the same principle, for a third-order neural network-based profiling SCA against a second-order masking scheme, the leakage function that the trained model is expected to learn is given by:

$$\mathcal{L} = g(\mathtt{m}_1, \mathtt{m}_2, \mathtt{m}_3) = \mathtt{m}_1 \diamond \mathtt{m}_2 \diamond \mathtt{m}_3 = (\mathtt{S\text{-}box}(\mathtt{d}_i \oplus \mathtt{k}_i) \diamond \mathtt{m}_1 \diamond \mathtt{m}_2) \diamond \mathtt{m}_1 \diamond \mathtt{m}_2 = \mathtt{S\text{-}box}(\mathtt{d}_i \oplus \mathtt{k}_i). \tag{9}$$

In this paper, we provide empirical experiments to analyze the performance of deep neural networks against a first-order Boolean masking scheme and leave the analysis of high-order masking, including other masking schemes, for future work. Nevertheless, the explainability methodology proposed in Section 4 applies to different masking schemes.

3 Related Work

Explainable AI (XAI) and explainable machine learning (XML) are very active research domains, see, e.g., [6,15,2,22,30,18]. Still, despite the developments and techniques proposed over the years, no widely accepted approach allows explainability for diverse machine learning tasks. Indeed, in 2015, DARPA formulated an XAI program "with the goal to enable end users to better understand, trust, and effectively manage artificially intelligent systems". The program started in 2017 and ended in 2021. Among the results, it was concluded that "There currently is no universal solution to XAI." and "One of the challenges in developing XAI is measuring the effectiveness of an explanation. DARPA's XAI effort has helped develop foundational technology in this area, but much more needs to be done" [16].

Interpretability and explainability in deep learning profiling SCA have received little attention. A larger focus has been put on neural network optimization to solve the difficult task of hyperparameter tuning. Nevertheless, the efforts to build various methodologies could be considered interpretability research since the authors (tried to) provide guidelines to build good neural networks, which intuitively means they could interpret what models do [45,40]. Moreover, Zaid et al. used weight visualization and feature maps to understand what features are more important [45].

In [28], the authors considered information bottleneck theory to derive an early stopping mechanism for a deep learning-based profiling attack by maximizing mutual information $I(T;\mathcal{Y})$ for the output layer. In this case, the authors ignored $I(\mathcal{X};T)$ and $I(T;\mathcal{Y})$ for hidden layers and only focused on the output layer. Note that the approach that we propose in Section 4 is completely different from this early stopping technique. Indeed, [28] only measures the information in the softmax layer (i.e., predictions) while our technique measures information in hidden layers.

The more “direct” attempts at interpretability and explainability can be divided into approaches that concentrate on the input layer and approaches that concentrate on the inner (hidden) layers. The techniques that concentrate on the input layer try to recognize the most important features (or, what is the influence of each feature on the performance of a neural network). Visualization techniques were the first attempt to explain what side-channel trace features have more impact on neural network decisions. Masure et al. provided visualization results through input gradient from the input network layer. They verified that neural networks automatically detect the time location of secret shares even in the presence of desynchronization countermeasures [26]. The input gradients analysis implements the so-called sensitivity analysis of loss function concerning input features or side-channel samples [26]. In [17], the authors compared different visualization techniques, e.g., Layer-wise Relevance Propagation and Occlusion, in profiling SCA and considered them as side-channel attack distinguishers. More recently, Golder et al. explored even more visualization techniques like Integrated Gradient and SmoothGrad [13].

To explain the behavior of hidden layers in profiling SCA, the authors of [38] considered Singular Vector Canonical Correlation Analysis (SVCCA) to explain what neural network layers learn from different side-channel traces. Unfortunately, the authors only managed to reach interpretability on a coarse level as even diverse datasets (side-channel dataset and image dataset) had more similarity than two side-channel datasets. Wu et al. proposed the adoption of ablation techniques to explain how different neural network configurations perform in the presence of diverse hiding countermeasures [42]. While these results are very interesting, we note they cannot explain the processing of masks. Furthermore, the approach is rather involved and gives results that are (potentially) difficult to interpret. Yap et al. used the Truth Table Deep Convolutional Neural Network approach to obtain the rules and decisions that the neural networks learned when retrieving the secret key from the cryptographic primitive [43]. With this approach, the authors located the points of interest responsible for neural network learning. Still, the authors mentioned the approach being inferior to the feature map of gradient visualization if the exact position of POI is needed. Moreover, the approach does not work with desynchronization/jitter countermeasure. Finally, Zaid et al. designed a novel machine learning generative model called Conditional Variational AutoEncoder [44]. With this approach, the authors showed the weights of the neural networks to provide an equation of the traces corresponding to the leakage.

The related works concentrate either on inputs (features) not allowing to explain the internal working of neural networks, or they are constrained concerning computational complexity and/or type of countermeasures that can be analyzed. Consequently, despite the progress obtained in the last few years, a technique to quantify the occurrence and propagation of high-order leakages in hidden layers and how (if) the masking countermeasures are defeated are still open questions.

4 Explainability Methodology for Profiling SCA - ExDL-SCA

In this section, we describe our novel explainability methodology. The process allows us to quantify how much information each intermediate network representation obtains about input features present in training measurements. As we apply our explainability methodology to the AES masked implementation, features in input measurements are given by different secret shares (i.e., mask and masked `S-Box` output bytes).

The proposed solution works by adding extra calculations during the deep neural network training. The model is trained with profiling traces $\mathcal{X}_p$ labeled in a black-box way with $\mathcal{Y}_p$. At the end of each training epoch, we predict the model with both profiling and attack traces $\mathcal{X}_p$ and $\mathcal{X}_a$, respectively. By doing so, the output of each hidden layer l, given by T, is saved as encoded versions of profiling and attack sets, i.e., $\mathcal{X}_p^l$ and $\mathcal{X}_a^l$, respectively. The shapes of $\mathcal{X}_p^l$ and $\mathcal{X}_a^l$ depend on the output layer dimensions. $\mathcal{X}_p^l$ and $\mathcal{X}_a^l$ are activation outputs from a hidden layer l when the model predicts with profiling and attack sets, respectively.

4.1 Relevant and Irrelevant Features

Information bottleneck theory is implemented by deep neural networks as hidden layer representations provide compressed information contained in input data. Real-world data, such as side-channel measurements, contain information (i.e., features) that can be defined as relevant or irrelevant to the classification. If the information bottleneck principle is aligned with the learned deep neural network function $F(\mathcal{X}, \mathcal{L}, \theta)$, there will be at least one hidden layer that can compress irrelevant features while preserving (or learning) the relevant ones.

In our case, the profiling function $F(\mathcal{X}, \mathcal{L}, \theta)$ learns from side-channel measurements $\mathcal{X}$ that contain several features. When this function needs to learn high-order leakages, the learned leakage function $\mathcal{L}$ needs to detect, learn, and combine at least two features from $\mathcal{X}$ representing the two secret shares. Normally, the measured side-channel interval includes leakages representing the processing of several intermediate data from a cryptographic algorithm plus noise. This way, it is expected that an efficient model $F(\mathcal{X}, \mathcal{L}, \theta)$ is the one that learns or preserves as much information as possible about the two secret shares to be combined while compresses as much as possible the rest of the information.

Here, input (relevant or irrelevant) features from $\mathcal{X}$ are defined by a set of labels $\mathcal{Y}_f$ that represents such features. To simplify, a input feature is always represented by an intermediate byte being processed by the cryptographic algorithm (e.g., a mask or a masked `S-box` output byte), which the leakages of this intermediate byte are included in the attacked side-channel interval. Because we only consider first-order masked AES implementation in our experiments, $\mathcal{Y}_f$ will always refer to three possible features: the secret j-th mask, $\mathtt{m}_j$, the j-th masked `S-box` output byte, $\mathtt{S\text{-}box}(\mathtt{d}_j \oplus \mathtt{k}_j) \oplus \mathtt{m}_j$, and the j-th `S-box` output byte without masking, $\mathtt{S\text{-}box}(\mathtt{d}_j \oplus \mathtt{k}_j)$.

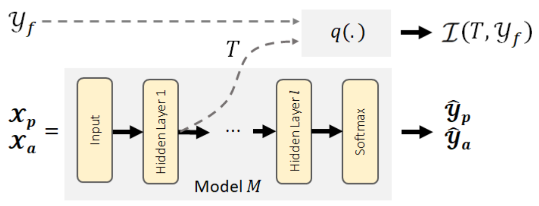
$\mathcal{Y}_f$
$q(.)$
$\mathcal{I}(T, \mathcal{Y}_f)$
T
$\boldsymbol{\mathcal{X}_p}$
$\boldsymbol{\mathcal{X}_a}$
=
Input
Hidden Layer 1
...
Hidden Layer l
Softmax
Model M
$\hat{\boldsymbol{\mathcal{Y}}}_p$
$\hat{\boldsymbol{\mathcal{Y}}}_a$

Fig. 2: Quantifying feature information $\mathcal{I}$ in a hidden layer representation T with a function $q(.)$, where $\mathcal{Y}_f$ is a label set representing an input feature.

4.2 Quantitative Measures of Side-channel Leakages in Hidden Layer Representations

When training a deep neural network as a profiling model against masked implementations, one expects a model that can automatically learn and combine high-order leakages from input traces. If the attacked side-channel trace interval includes the processing of several intermediate bytes from a cryptographic implementation, including the masks, each of these bytes can be considered as input features. An intermediate network representation that is given by the output activations of a certain hidden layer should be able to compress irrelevant features while preserving information about relevant ones. The irrelevant features are assumed to be noise by the neural network and, therefore, should not be propagated to the next layers. The relevant features, in this case, would be the intermediate bytes representing the masks and masked target operation (e.g., masked `S-box` output byte in the first encryption round), which should be combined by some intermediate network representation as well.

To understand where (e.g., in which layer) and when (e.g., in which training epoch) the model implements feature compression and feature learning, we need to define an information metric $\mathcal{I}$. Such an information metric must be able to efficiently quantify, through a function $q(.)$, how much information a hidden layer representation T has about a certain feature $\mathcal{Y}_f$, i.e., $\mathcal{I}(T, \mathcal{Y}_f)$. This process is illustrated in Figure 2.

Following, we describe possible ways of quantifying information in a hidden layer representation and discuss the main drawbacks and advantages of each method.

Mutual Information (MI) Estimation The most obvious way to measure how much information a hidden network layer compresses (and preserves) from $\mathcal{X}$ would be to compute the mutual information between $\mathcal{X}$ and T, $I(\mathcal{X};T)$, as proposed by [35]. The main problem is that directly applying MI estimation to compute $I(\mathcal{X};T)$ requires computing mutual information between two high dimensional data $\mathcal{X}$ (side-channel traces) and T (layer representation of T). This way, an accurate estimation of MI requires exponentially more data, especially for histogram-based mutual estimation, as will be explained next. Since mutual information is symmetric, $H(\mathcal{X}) - H(\mathcal{X}|T) = H(T) - H(T|\mathcal{X})$, $I(\mathcal{X};T)$ can be computed according to:

$$I(\mathcal{X};T) = H(T) - H(T|\mathcal{X}) = H(T) - \sum_{i=1}^{n_p} \mathtt{p}(\mathtt{x}_i) \sum_{j=1}^{n_p} \mathtt{p}(\mathtt{t}_j|\mathtt{x}_i) \log_2 \mathtt{p}(\mathtt{t}_j|\mathtt{x}_i). \quad (10)$$

As discussed in [34], the histogram-based estimation of mutual information requires a correct selection of the number of bins. When the number of bins is too large to keep the precision of T, every input $\mathtt{x}_i$ yields a different activation pattern $\mathtt{t}_j$ in each hidden layer. In other words, due to the high dimensionality of $\mathcal{X}$ and T, it is often impossible to have two input traces $\mathtt{x}_i$ that generate two identical intermediate network representations $\mathtt{t}_i$. This will result in $H(T|\mathcal{X}) = 0$ because the conditional probabilities $\mathbf{p}(\mathtt{t}_j|\mathtt{x}_i)$ become 1 for all $\mathtt{x}_i$ and all $\mathtt{t}_j$ and, consequently, $I(\mathcal{X};T) = H(T)$. This result would wrongly indicate the compression of input $\mathcal{X}$ during training, as we would only be computing the entropy of T. A possible solution to obtain $H(T|\mathcal{X}) > 0$ is to select a smaller number of bins. Nevertheless, this would add too much noise to the mutual information calculation, also leading to wrong estimations (e.g., see Appendix A in [28]). Similarly, computing mutual information between a high-dimensional T and an one-dimensional feature $\mathcal{Y}_f$, $I(T;\mathcal{Y}_f)$, could often result in $I(T;\mathcal{Y}_f) = H(\mathcal{Y}_f)$, which is equivalent to obtaining the entropy of $\mathcal{Y}_f$. Therefore, measuring mutual information $I(a,b)$ when either a or b is high-dimensional data is ill-posed when estimating the amount of information that T has about $\mathcal{X}$ or $\mathcal{Y}_f$.

Mutual Information Neural Estimation (MINE) In [1], the authors proposed a neural estimation of mutual information that consists in a neural network having two inputs and a single output neuron. The mutual information is computed thanks to a loss function that provides a neural information value in the output. Having a hidden layer representation of T, one could quantify the mutual information between T and a secret share $\mathcal{Y}_f$, $I(T,\mathcal{Y}_f)$ by implementing MINE. The process sounds intuitive. However, it provides three main disadvantages for our explainability solution: (1) using MINE to quantify $I(T,\mathcal{Y}_f)$ implies finding neural network hyperparameters for each different model to explain (MINE neural network could also suffer from overfitting issues), (2) the convergence of MINE could require an excessive number of training epochs that could render our explainability process very time-consuming, and (3) the MINE structure is designed to receive two inputs $\mathcal{X}$ and $\mathcal{Y}$ to estimate the mutual information between these two inputs only, which means that computing mutual information between $\mathcal{X}$ and multiple input features in a single training is not possible, which implies an even more complex analysis.

Moreover, we implemented experiments with MINE, and, in various cases, we verified the exploding gradient problem often happens.[7] This could be solved

[7] When large error gradients accumulate and result in very large updates to neural network model weights during training.

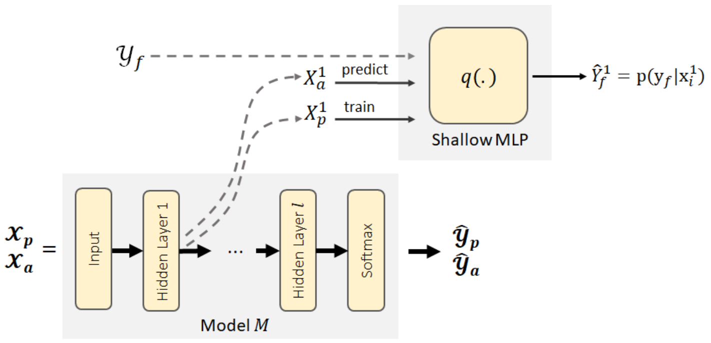

Fig. 3: Training and predicting classifier M_s^l after obtaining encoded representations X_p^l and X_a^l from the first hidden layer.

with regularization, gradient clipping, or by carefully adjusting the MINE network hyperparameters. The authors of [36] also observed these limitations and proposed to add a clipping function to the MINE loss function estimator. Nevertheless, this solution adds new hyperparameters to be selected, which increases the complexity of MINE as an estimator for information on hidden network representations.

Perceived Information (PI) Estimation [5] We propose an alternative solution to measure the amount of information a hidden layer learns from input data. In our method, we are interested in estimating the amount of information from secret shares that can be extracted by each hidden layer. Thus, by taking the intermediate representations $T = X_p^l$ (obtained when predicting the network with training data $\mathcal{X}_p$) from all hidden layers, at the end of each training epoch, we train a separate classifier $q(.)$ with X_p^l in each layer l to measure how much information T has about an input feature $\mathcal{Y}_f$. The classifier $q(.)$ can be any supervised classification method. In our case, we use a shallow MLP classifier with an output softmax layer. This shallow MLP classifier is also a multi-output classifier, which allows us to estimate the information from several input features in a hidden layer representation T with a single training of $q(.)$. After training this shallow MLP classifier with X_p^l, we predict it with encoded attack traces X_a^l to obtain class probabilities associated with each selected input feature. Figure 3 illustrates this process. The shallow MLP classifier provided output class probabilities $\mathtt{p}(\mathtt{y}_f|\mathtt{x}_i^l)$ for each input feature $\mathcal{Y}_f$.

After obtaining the conditional class probabilities $\mathtt{p}(\mathtt{y}_f|\mathtt{x}_i^l)$ from $q(.)$, perceived information becomes a convenient approach as it measures the amount of information learned by a profiling model concerning specific leakage model resulting in a set of labels $\mathcal{Y}_f$). Initially, we need to estimate the conditional entropy $H(\mathcal{Y}_f|X_p^l)$, which is given by:

$$H(X_p^l|\mathcal{Y}_f) = \sum_{z=1}^{|\mathcal{Y}_f|} \mathtt{p}(\mathtt{y}_f = z) \sum_{i=1}^{n_p} \mathtt{p}(\mathtt{x}_i^l|\mathtt{y}_f = z) \log_2 \mathtt{p}(\mathtt{y}_f = z|\mathtt{x}_i^l), \tag{11}$$

where $\mathtt{x}_i^l$ is the i-th observation of X_p^l and $\mathtt{y}_f$ is one element of $\mathcal{Y}_f$. First, we replace X_p^l by attack (or test) encoded representations X_a^l in Eq. (11):

$$H(X_a^l|\mathcal{Y}_f) = \sum_{z=1}^{|\mathcal{Y}_f|} \mathtt{p}(\mathtt{y}_f = z) \sum_{i=1}^{n_a} \mathtt{p}(\mathtt{x}_i^l|\mathtt{y}_f = z) \log_2 \mathtt{p}(\mathtt{y}_f = z|\mathtt{x}_i^l), \tag{12}$$

where $\mathtt{y}_f$ are labels obtained from intermediate values processed in attack traces $\mathcal{X}_a$. Following the approach proposed in [5], we can replace the true probability mass function (PMF), $\mathtt{p}(\mathtt{y}_f|\mathtt{x}_i^l)$, by $\hat{\mathtt{p}}(\mathtt{y}_f|\mathtt{x}_i^{l,\mathtt{y}_f})$ and compute the perceived information by *sampling* according to (see Eq. (11) from [5]):

$$\widehat{PI}(X_a^l|\mathcal{Y}_f) = H(\mathcal{Y}_f) + \sum_{z=1}^{|\mathcal{Y}_f|} \mathtt{p}(\mathtt{y}_f = z) \frac{1}{n_{\mathtt{y}_f = z}} \sum_{i=1}^{n_a} \log_2 \hat{\mathtt{p}}(\mathtt{y}_f|\mathtt{x}_i^{l,\mathtt{y}_f = z}), \tag{13}$$

where $\hat{\mathtt{p}}(y_f|x_a^{l,\mathtt{y}_f}))$ gives the probability that an encoded attack trace $\mathtt{x}_i^{l,\mathtt{y}_f=z}$ is labeled with class $\mathtt{y}_f$ when it is labeled with this same class.[8] The term $n_{\mathtt{y}_f=z}$ gives the number of attack traces that are labeled with class $\mathtt{y}_f = z$.

The metric from Eq. (13) *becomes an indirect estimation of how much information the encoded hidden layer representation $T = X_p^l$ (obtained by predicting the neural network with profiling set $\mathcal{X}_p$) contains from a specific byte being manipulated by a cryptographic algorithm, such as a secret share in masked implementations.*

Obviously, the quantity $\widehat{PI}(X_a^l|\mathcal{Y}_f)$ depends on a properly built classifier $q(.)$, which leads to the more precise estimation of perceived information values from Eq. (13). This is analyzed in Section 5.2.

The magnitude of input feature information (such as secret shares) is expected to be higher in the first hidden layers, where the relation $PI(T^l, \mathcal{Y}_f) \geq PI(T^{l+1}, \mathcal{Y}_f)$ for two subsequent layers l and $l+1$ is always preserved. However, we experimentally verified that the relation $PI(T^l, \mathcal{Y}_f) \geq PI(T^{l+1}, \mathcal{Y}_f)$ does not always hold unless the bottleneck layer is the first layer. The reasons for that are intuitive: (1) in the hidden layers closer to input, the intermediate representation T is closer to input data, and, for this reason, it is expected to contain more information about noise and irrelevant features. In a deeper layer, where a bottleneck layer is possibly successfully implemented by the network, the intermediate representation T compresses more aggressively irrelevant features while

[8] Here, we assume a known-key attack setting analysis. For an evaluator, the attack set is always the validation set, and the corresponding validation key bytes are always known.

preserving the relevant ones. When the classifier $q(.)$ measures the information present in this bottleneck layer, it can quantify more precisely the information about relevant features, possibly leading to higher perceived information quantities in these layers concerning previous layers.

4.3 ExDL-SCA Steps

After explaining how we estimate perceived information between input features represented by labels and intermediate network representations T from a hidden layer l, we can define the complete structure of our explainability methodology. Figure 4 provides the four steps that form our methodology:

1. In Step ①, we define a baseline neural network F to be trained for E epochs with profiling traces $\mathcal{X}_p$ that are labeled as $\mathcal{Y}$ (without the knowledge of secret mask shares). Our experimental results in Sections 5 and 6 are obtained from software AES 128 implementations and, therefore, (black-box) labels $\mathcal{Y}$ are generated from `S-box` output bytes in the first encryption round, i.e., $\texttt{S-box}(\texttt{d}_j \oplus \texttt{k}_j)$.
2. In Step ②, we extract the intermediate representations T from hidden layers. Thus, at the end of training epoch e, the trained model F predicts both profiling ($\mathcal{X}_p$) and attack ($\mathcal{X}_a$) sets. For each hidden layer l, we obtain output activations that are taken as encoded versions of input profiling and attack sets, i.e., X_p^l and X_a^l, respectively.
3. In Step ③, we take encoded profiling sets X_p^l from all hidden layers l and build a shallow MLP classifier, denoted as $q(.)$, which is trained with X_p^l for E_q epochs. There are no pre-processing steps such as dimensionality reduction, which could be computationally intensive in the case a large output dimension from a hidden layer produces a very large encoded set X_p^l. Therefore, using machine learning algorithms (e.g., Support Vector Machine or Decision Trees) or even Gaussian Template Attacks to implement the $q(.)$ classifier would require feature selection and/or dimensionality reduction, which could render the process impractical due to significant overheads. Instead of training $q(.)$ multiple times (one time for each different input feature $\mathcal{Y}_f$), one could implement a multi-label classifier to speed up the process.
4. Finally, in Step ④, the shallow MLP classifier $q(.)$ predicts encoded attack set X_a^l, producing output class probabilities $\widehat{\mathcal{Y}_f^l} = \texttt{p}(\texttt{y}_f|\texttt{x}_i^l)$ for each different input feature $\mathcal{Y}_f$ (see Figure 3). These quantities are considered for a perceived information calculation in Eq. (13).

This way, we can estimate the amount of information that an encoded representation X_p^l, in a hidden layer l, can have about input information $\mathcal{Y}_f$ for every training epoch in a black-box model. Note that $\mathcal{Y}_f$ can also be the true labels $\mathcal{Y}$, which allows us also to measure the moment when the network F combines two secret shares and becomes capable of implementing a second-order attack. Algorithm 1 (Appendix A) provides the steps necessary to implement our explainability methodology. The method $\texttt{LayerPredict}(L_F, \mathcal{X})$ returns the output activations from a layer of index l when this layer predicts some input $\mathcal{X}$.

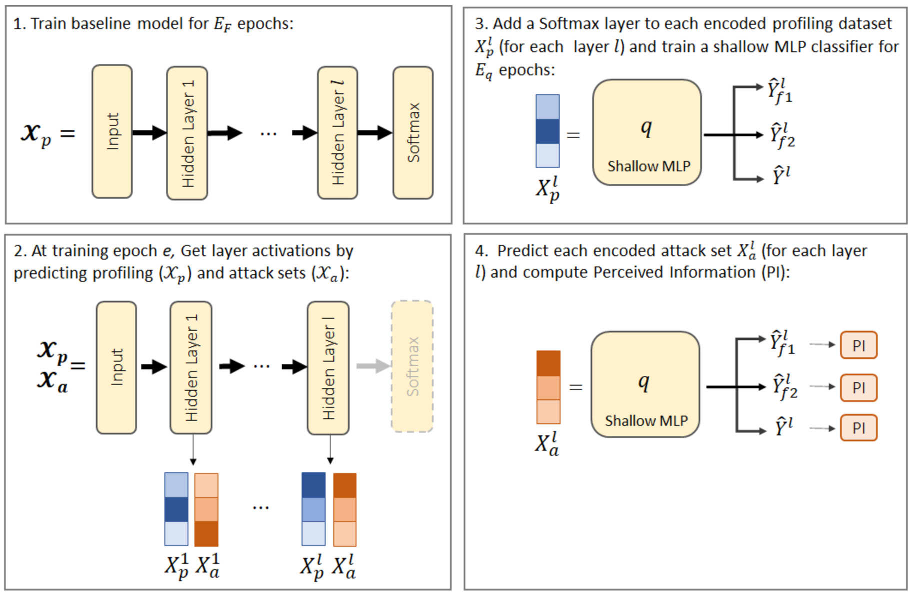
1. Train baseline model for E_F epochs:
$\mathcal{X}_p$ =
Input
Hidden Layer 1
Hidden Layer l
Softmax
3. Add a Softmax layer to each encoded profiling dataset X_p^l (for each layer l) and train a shallow MLP classifier for E_q epochs:
X_p^l
q
Shallow MLP
$\hat{Y}_{f1}^l$
$\hat{Y}_{f2}^l$
$\hat{Y}^l$
2. At training epoch e, Get layer activations by predicting profiling ($\mathcal{X}_p$) and attack sets ($\mathcal{X}_a$):
$\mathcal{X}_p$ = $\mathcal{X}_a$
Input
Hidden Layer 1
Hidden Layer l
Softmax
X_p^1 X_a^1
X_p^l X_a^l
4. Predict each encoded attack set X_a^l (for each layer l) and compute Perceived Information (PI):
X_a^l
q
Shallow MLP
$\hat{Y}_{f1}^l$
$\hat{Y}_{f2}^l$
$\hat{Y}^l$
PI
PI
PI

Fig. 4: Methodology for mask share fitting explainability. Here, q represents a shallow MLP with multi-label classification.

4.4 On the Granularity of the Explainability Approach

One could ask why developing an explainability approach that considers the granularity level of a neural network layer. For instance, previous approaches even considered specific neurons (filters in a feature map). Our analysis showed that multiple neurons (filters in a feature map) are responsible for the successful deep learning-based SCA. Thus, connecting specific neurons (filters in a feature map) to the outcome of the analysis is difficult due to the ability of neural networks to find (many) good functions representing the leakage.
Looking at the level of multiple layers would allow the analysis. Still, it would not provide information about what happens in every layer, which is especially relevant if we consider that state-of-the-art neural networks in SCA contain only a few layers [29].

5 Compression in Deep Neural Networks

The information bottleneck theory suggests that one of the main aspects of learning from noisy datasets is the compression of $\mathcal{X}$ to an intermediate representation during training to eliminate the noise and preserve relevant information about $\mathcal{Y}$. For the reasons explained in Section 4.2, measuring the level of compression with mutual information $I(\mathcal{X};T)$ is difficult and, therefore, the solution adopted in our work relies on quantifying perceived information of specific input features in hidden layer representations.

5.1 Compression of Irrelevant (Key Byte) Features in First-order Masked Datasets

We use our explainability methodology to show how deep neural network layers compress the information about irrelevant features present in training set $\mathcal{X}_p$ while preserving relevant features. When targeting a single key byte in a first-order masked AES dataset, relevant features become the two secret shares associated with the target key byte. In contrast, irrelevant features are all the information corresponding to other key bytes and noise components. We provide results for a noise-free simulated dataset, in which all features are well defined so that every sample represents a feature (with zero noise) and there are no samples that would represent noise. Section 6 provides experiments on real side-channel measurements from the first-order Boolean masked AES implementations.

Our simulated traces contain leakages from `S-box` outputs in the first AES encryption round. Each trace $\mathtt{x}_i$ contains 32 samples, and each of these samples is generated according to the following equations:

$$\begin{aligned} \mathtt{x}_i[2j] &= \mathtt{S\text{-}box}(\mathtt{d}_j \oplus \mathtt{k}_j) \oplus \mathtt{m}_j \\ \mathtt{x}_i[2j+1] &= \mathtt{m}_j, \end{aligned} \tag{14}$$

where $j \in [0, 15]$ denotes the j-th key byte index and $\mathtt{m}_j$ is the mask share associated with the j-th key byte.

In the first experiment, we define a 4-layer MLP with the following layer-wise structure (values were randomly chosen, as basically any hyperparameters combination provided optimal results, which does not justify implementing a tuning process):

$$\mathcal{X} \rightarrow [FC(50) \rightarrow E]^4 \rightarrow S(256) \rightarrow \widehat{\mathcal{Y}}.$$

The learning rate and batch size are set to 0.001 and 400, respectively. This MLP is trained with the `Adam` optimizer for 100 epochs. The simulated profiling and attack traces $\mathcal{X}_p$ and $\mathcal{X}_a$ are labeled with leakages from the second key byte $j = 2$, i.e., $\mathcal{Y} = \mathtt{S\text{-}box}(\mathtt{d}_2 \oplus \mathtt{k}_2)$.

In Figure 5, we plot the perceived information values obtained for all hidden layers. The perceived information values are computed concerning all 32 input features (16 masks plus 16 masked `S-box` output bytes) contained in simulated traces. With respect to the shallow MLP classifier q to compute perceived information from hidden layers, we always consider a 1-layer MLP with 50 neurons, where the activation function is `elu`, the learning rate is 0.001, the batch size is 200, and the optimizer is `Adam`. This classifier q is always trained for 10 epochs.

Because $\mathcal{X}_p$ is labeled with $\mathcal{Y} = \mathtt{S\text{-}box}(\mathtt{d}_2 \oplus \mathtt{k}_2)$, we expect that the network layers will fit the relevant features corresponding to key byte 2, i.e., $\mathtt{m}_2$ and $\mathtt{S\text{-}box}(\mathtt{d}_2 \oplus \mathtt{k}_2) \oplus \mathtt{m}_2$, and compress the rest of the features. In Figure 5, we see that the first layer still contains information from several irrelevant features from other key bytes (other than key byte 2) in the first training epochs. When we move our analysis to the second layer, we recognize a bottleneck implemented by the MLP, in which irrelevant features start to be aggressively compressed. In contrast, the relevant ones are kept with relatively more information (i.e., higher

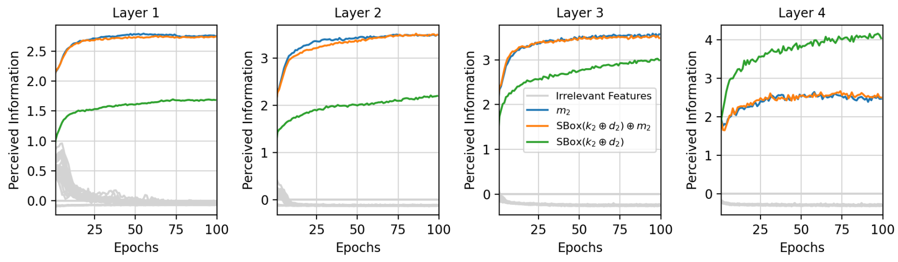

Fig. 5: The compression of irrelevant features with a 4-layer MLP. Blue and orange lines indicate the perceived information from relevant features, which are the mask $\mathtt{m}_2$ and the masked S-Box output $\mathtt{S\text{-}box}(\mathtt{k}_2 \oplus \mathtt{p}_2) \oplus \mathtt{m}_2$, related to the key byte 2. Irrelevant features associated with the rest of the key bytes are given as gray lines.

perceived information magnitude). Then, when we look at the third layer, we see a bottleneck layer compressing all irrelevant features (all of them show negative perceived information) while trying to maximize the relevant ones to improve the prediction of $\mathcal{Y}$. Indeed, in the third layer, the perceived information for input features $\mathtt{m}_2$ and $\mathtt{S\text{-}box}(\mathtt{d}_2 \oplus \mathtt{k}_2) \oplus \mathtt{m}_2$ results in the higher values when we compare with the perceived information values obtained for these same input features in previous and subsequent layers. Finally, in the fourth layer, the network already has enough information from relevant features to provide a good generalization to $\mathcal{Y}$, which means that the focus of this layer is not maximizing second-order leakages, but actually optimizing the recombination of high-order leakages into $\mathcal{Y}$. As a result, the third layer is the one containing more information about input relevant features while the fourth layer is the one containing more information about target output labels. This is a clear example of an information bottleneck being implemented by a deep neural network, and applying our explainability methodology allows us to explain what happens in this model during training.

5.2 Defining and Training the Classifier q

In the previous section, we described the main reasons to use perceived information as the metric to quantify information from an input feature $\mathcal{Y}_f$ in an intermediate network representation T. This requires the definition and the training of a classifier q to obtain class probabilities necessary to compute perceived information in Eq. (13). Next, we provide performance analysis on q.

In this work, the classifier q is implemented with a shallow MLP network. The output layer is given by a softmax layer as we want to obtain class probabilities for perceived information calculation. This classifier requires the definition of several additional hyperparameters to ensure we obtain a consistent perceived information estimation. Although it is common knowledge that hyperparameter tuning is usually a difficult problem in profiled SCA, here, we demonstrate that

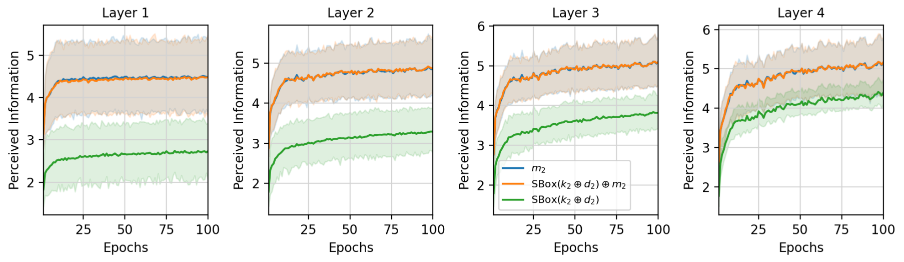

Fig. 6: Variation of measured perceived information for different hyperparameter combinations for classifier $q(T, \mathcal{Y}_f)$.

designing this shallow MLP classifier for perceived information estimation is not difficult, as performing hyperparameter tuning does not provide a wide variety of results. Figure 6 shows an example when we compute perceived information with 32 different hyperparameter tuning combinations. We applied this analysis to the 4-layer MLP considered in the previous section with a simulated AES dataset. This grid search takes the following hyperparameter values and generates all possible combinations:

- Layers: $[1, 2]$
- Neurons: $[50, 100]$
- Batch size: $[200, 400]$
- Epochs: $[10, 20]$
- Activation Function: $[\texttt{elu}, \texttt{selu}]$

The learning rate is set to 0.001, and the optimizer is always `Adam`. In this case, we only compute perceived information for the two relevant input features $\mathtt{m}_2$ (blue line) and $\texttt{S-box}(\mathtt{d}_2 \oplus \mathtt{k}_2) \oplus \mathtt{m}_2$ (orange line) plus the perceived information of the target label $\mathcal{Y} = \texttt{S-box}(\mathtt{d}_2 \oplus \mathtt{k}_2)$ (green line). The main line indicates the average perceived information for 32 hyperparameter combinations, and the shadow interval indicates the variation we obtain with these same combinations. Note how the variation, although more significant in the first layer, is much less significant in the last layers, which means that varying the hyperparameters for the shallow MLP implementing q does not provide a significant impact on results.

6 Experimental Results

In this section, we provide experimental results on various CNN and MLP configurations. For the shallow MLP classifier q, we always consider a 1-layer MLP with 100 neurons, where the activation function is `elu`, the learning rate is 0.001, the batch size is 400, and the optimizer is `Adam`. This classifier q is always trained for 20 epochs. Note that in this paper, we consider the Identity leakage model only.

6.1 Datasets

For our experimental results, we select, among several publicly available datasets, two masked AES datasets. We decided to consider the trace sets from the ASCAD and DPAv4 databases, whose keys in profiling and attack phases are different, and mask shares are also provided in the metadata. Additionally, we simulate the effect of a hiding countermeasure (desynchronization) on those datasets. For both considered datasets, we target trace intervals with second-order leakages of multiple key bytes to make sure that we also include several other intermediate bytes representing irrelevant features during training in different hidden layers.

`ASCADr` This dataset contains 300 000 traces collected from a software implementation of AES 128,[9] where the first 200 000 measurements have random keys and are considered for profiling while 100 000 measurements contain a fixed key and, from this second set, we consider 5 000 for the attack phase. Each measurement contains 250 000 samples. This dataset was collected from an AES 128 implementation featuring a first-order Boolean masking countermeasure. In previous works, a trimmed version of this dataset containing measurements with 1 400 samples is commonly adopted, which contains second-order leakages related to the third key byte only. For our experiments, we start from raw measurements containing 250 000 samples and select the interval from sample 70 000 until sample 90 000. Then, we apply a window resampling with a resampling window of 20 samples and a step of 10 samples, resulting in traces with 2 000 samples. These trimmed and resampled measurements contain second-order leakages related to the third key byte in the first encryption round but also include leakages from other key bytes.

`DPAv4.2` The `DPAv4.2` dataset contains side-channel measurements obtained from a masked AES 128 software implementation.[10] The countermeasure is based on RSM (*Rotation `S-box` Masking*). The original `DPAv4.2` contains 80 000 traces subdivided into 16 groups of 5 000 traces. Each group is defined with a separate and fixed key. Each measurement has 1 704 046 samples. In this work, we conduct our analyses on an interval resulting from the concatenation of two intervals from the original dataset. The main idea is to combine two trace intervals containing second-order leakages from several key bytes, including the attacked one. The second-order leakages include masked `S-box` output bytes and the corresponding masks. The first interval ranges from sample 265 000 until sample 280 000, while the second interval starts at sample 305 000 and finishes at sample 315 000. Thus, concatenating these two intervals results in measurements with 25 000 samples. We apply a resampling process with a resampling window of 10 and step of 5 to the concatenated intervals, resulting in 5 000 samples per measurement.

[9] `https://github.com/ANSSI-FR/ASCAD/tree/master/ATMEGA_AES_v1/ATM_AES_v1_variable_key`

[10] `http://www.dpacontest.org/v4/42_doc.php`

6.2 Reading the Plots

Here, the main goal is to demonstrate how different hidden layers fit, compress, or generalize concerning different features. Plots provided in this section show the evolution of perceived information (Eq. (13)) during training for different features given by specific label sets. For both evaluated datasets (`ASCADr` and `DPAv4.2`), we deploy profiling and attack phases over trace intervals that include leakages from different key bytes. For `ASCADr`, the evaluated interval includes second-order leakages from key bytes 2, 4, 5, and 11, in which the target is key byte 2. For `DPAv4.2`, the target interval includes second-order leakages from key bytes 0, 4, 5, 9, and 12, and we target key byte 0. The main idea is to illustrate the compression of irrelevant features by bottleneck layers. Thus, in all plots, we provide perceived information results for masks shares (given by $\mathtt{m}_j$ in the plot's legend, where j is the key byte index) and masked `S-Box` output byte (given by $\mathtt{S\text{-}box}(\mathtt{k}_j \oplus \mathtt{d}_j) \oplus \mathtt{m}_j$ in plot's legend) for all these key bytes. The perceived information values for target key bytes (key byte 2 for `ASCADr` and key byte 0 for `DPAv4.2`) are shown with colored lines (blue color for $\mathtt{m}_j$ and orange color for $\mathtt{S\text{-}box}(\mathtt{k}_j \oplus \mathtt{d}_j) \oplus \mathtt{m}_j$) while for the rest of key bytes, we show the results with gray lines. The perceived information for the actual black-box attack labels $\mathcal{Y} = \mathtt{S\text{-}box}(\mathtt{k}_j \oplus \mathtt{d}_j)$ is represented by a green line plot. Therefore, blue and orange lines indicate relevant features, while gray lines indicate irrelevant features. Every subfigure shows results for a specific hidden layer, and the x-axis indicates the training epochs of the main F model.

6.3 `ASCADr`

Multilayer Perceptron We select various MLP architectures that implement successful key recovery on `ASCADr` from a random search (see Appendix 1 for details). We consider a profiling MLP model successful when it reaches guessing entropy equal to 0 for the correct key $\mathtt{k}_2$ after processing up to 5 000 attack traces. Our hyperparameter search process allows us to find successful models with a different number of hidden layers. Nevertheless, by applying our explainability methodology, we observe common behavior for these models regardless of the number of hidden layers.

Figure 7 shows an example of a six-layer MLP with the following layer-wise structure:

$$\mathcal{X} \rightarrow [FC(\mathtt{100}) \rightarrow E]^6 \rightarrow S(\mathtt{256}) \rightarrow \widehat{\mathcal{Y}}. \tag{15}$$

For this model, the learning rate is set to `5e-4` and weights are initialized with `random_uniform` method. The attacked interval includes the second-order leakages from four different key bytes, including the target one. We immediately verify that the first hidden layer still contains information from irrelevant features represented by key bytes different from the target one. From the second layer, we see that the irrelevant information is highly compressed, and the outer layer generalizes better to $\mathcal{Y}$, as shown by the green line representing perceived information with respect to $\mathcal{Y} = \mathtt{S\text{-}box}(\mathtt{k}_j \oplus \mathtt{d}_j)$. Another interesting fact from

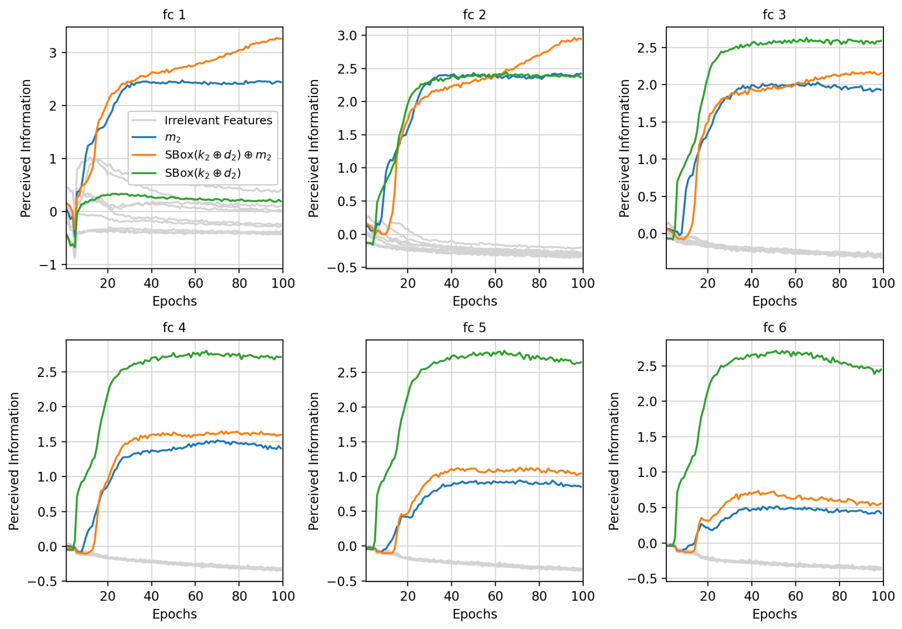

Fig. 7: Perceived information values from a six-layer MLP trained with the `ASCADr` dataset. Blue and orange lines represent the secret shares $\mathcal{Y}_f = \mathtt{m}_2$ (the mask) and $\mathcal{Y}_f = \mathtt{S\text{-}Box}(\mathtt{d}_2 \oplus \mathtt{k}_2) \oplus \mathtt{m}_2$ (the masked `S-Box` output byte), respectively. Green line represents the black-box labels $\mathcal{Y} = \mathtt{S\text{-}Box}(\mathtt{d}_2 \oplus \mathtt{k}_2)$.

this specific model is that outer layers achieve higher perceived information values of $\widehat{PI}(X_a^l; \mathcal{Y})$ during training even before achieving higher values of perceived information with respect to secret shares. This happens because previous layers already implemented unmasking and transmitted this information to the next layer.

1. **Where**. Our empirical 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.
2. **What**. We verified that to 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.
3. **Why**. Our analyses indicate that MLP follows IB theory as the bottleneck is usually implemented already by the first hidden layer. In essence, all intermediate network representations follow the IB theory, but the bottleneck (compression of irrelevant features) is usually more evident in the first layers.

Convolutional Neural Network We again deployed a random search to select various CNN architectures that implement successful key recovery on the `ASCADr` dataset. Details about our CNN random search process can be found in Appendix B (Table 2). The number of convolution layers in the search space ranges from one to four, and CNN models may contain one or two fully-connected layers.

Figure 8 shows the results obtained from a CNN with four convolution layers and two fully-connected layers with the following structure:

$$\mathcal{X} \rightarrow [C(\texttt{fi}, \texttt{40}, \texttt{15}) \rightarrow SE \rightarrow BN \rightarrow AP(\texttt{2}, \texttt{2})]^4 \rightarrow [FC(\texttt{20}) \rightarrow SE]^2 \rightarrow S(\texttt{256}) \rightarrow \widehat{\mathcal{Y}}, \tag{16}$$

where `fi` is set to 12, 24, 36, and 48 for the four convolution layers. The learning rate for this model is `1e-4`, and trainable weights are initialized with `glorot_normal` method. The first layer, `conv_1`, fits information from relevant and irrelevant features, as perceived information values are positive. For this layer, after epoch 20, compression starts to happen for all features. Layer `conv_2` shows the fitting of input features, including irrelevant ones, and layer `conv_3` shows compression of irrelevant features while preserving and learning relevant ones. Note how `conv_3` already generalized to Y. The subsequent layers (`conv_4`, `fc_1`, and `fc_2`) also show compression of irrelevant leakages from key bytes other than the target one. Prediction to $\mathcal{Y}$ (given by positive values of $\widehat{PI}(X_a^l; \mathcal{Y})$), is already seen in `conv_4`, and in `fc_1` and `fc_2` layers, this generalization to $\mathcal{Y}$ is even stronger. What we see in this figure is a general behavior observed for various successful CNN models on the `ASCADr` dataset.

1. **Where**. We verify that the first convolution layer is usually unable to implement compression of irrelevant features to keep the relevant ones (we observed that when the model achieves good levels of generalization, the first convolution layer tends to compress the input information, including the features related to the target key byte). The second convolution layer usually implements fitting and compression phases, which is characterized by the increase of perceived information values obtained from features (i.e., secret shares) related to the target key byte. At the same time, there are reduced perceived information values for features related to the remaining key bytes that the model is not supposed to learn. When CNN has more than two convolution layers, the other convolution layers implement the bottleneck more efficiently. Fully-connected layers provide higher $\widehat{PI}(X_a^l; \mathcal{Y})$, indicating generalization to $\mathcal{Y}$.
2. **What**. The bottleneck is implemented at least in the second hidden layer. From this moment, the network starts to generate an intermediate representation that mostly preserves the information from secret shares related to the target key byte.
3. **Why**. Fitting and compression happen in CNN models because this type of architecture also follows the IB principle. Hidden layers implement the bottleneck, which provides conditions for the model to generalize as relevant features are fit by the layers.

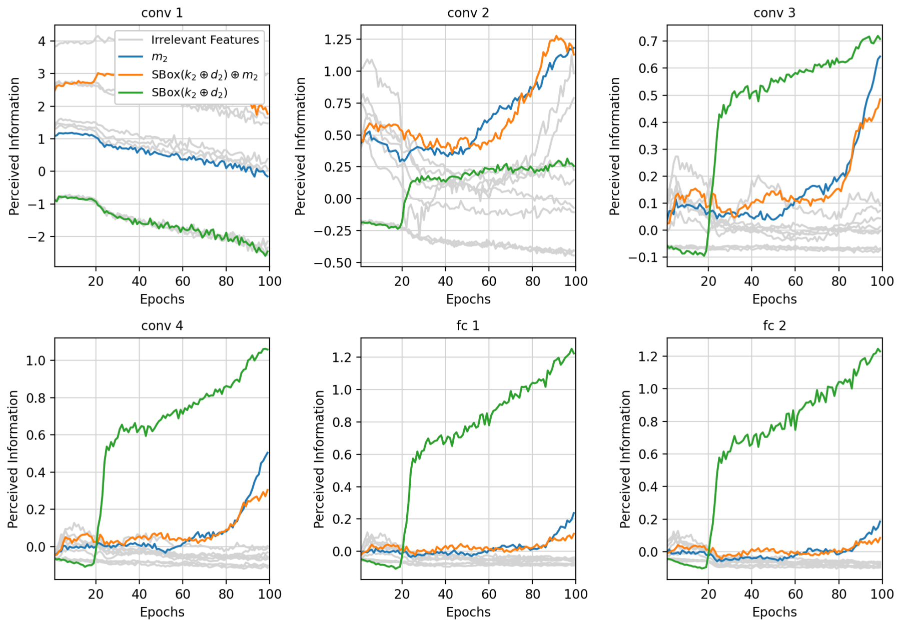

Fig. 8: Perceived information values from CNN layers (Eq. (16)) trained with the `ASCADr` dataset. Blue and orange lines represent the secret shares $\mathcal{Y}_f = \texttt{m}_2$ (the mask) and $\mathcal{Y}_f = \texttt{S-Box}(\texttt{d}_2 \oplus \texttt{k}_2) \oplus \texttt{m}_2$ (the masked `S-Box` output byte), respectively. Green line represents the black-box labels $\mathcal{Y} = \texttt{S-Box}(\texttt{d}_2 \oplus \texttt{k}_2)$.

Convolutional Neural Networks with Desynchronized Dataset In this section, we evaluate CNNs with our explainability methodology when the model is trained with a desynchronized `ASCADr` dataset. To produce misalignment, traces are randomly shifted by up to 50 samples (see [41] for details on how to simulate the desynchronization effect). To circumvent the desynchronization effect [7], the CNN is trained with data augmentation that implements random shifts (again up to 50 samples). For each epoch, we generate 200 000 augmented profiling traces (double the number of the profiling traces). We apply our hyper-parameter search until we generate at least 100 successful CNN models able to reduce the guessing entropy of the correct key to 0. In Figure 9, we provide an example result from a CNN model with the following layer-wise structure:

$$\mathcal{X} \rightarrow [C(\texttt{fi}, 30, 15) \rightarrow E \rightarrow BN \rightarrow AP(2,2)]^3 \rightarrow [FC(200) \rightarrow E]^2 \rightarrow S(256) \rightarrow \widehat{\mathcal{Y}}, \tag{17}$$

where filters `fi` are set to 16, 32, and 48 for the three convolution layers. The learning rate is set to `1e-4`, and weights are initialized with `random_uniform` method.

In Figure 9, we see how the first two convolution layers `conv_1` and `conv_2` process relevant and irrelevant features. These layers mostly fit all features, with-

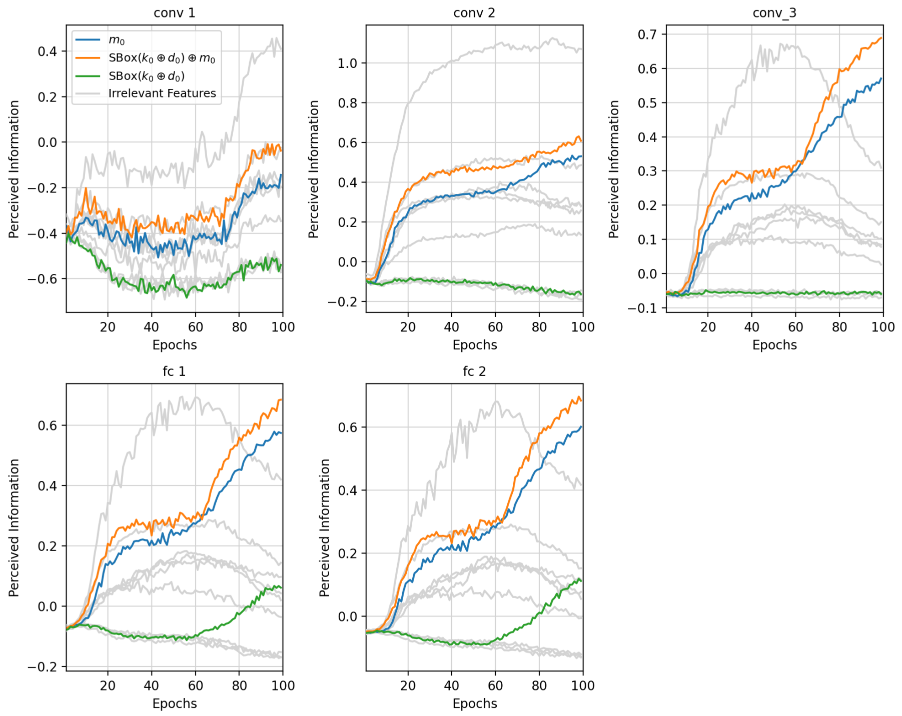

Fig. 9: Perceived information values from a CNN trained with the desynchronized `ASCADr` dataset.

out compression. Layers `conv_3`, `fc_1`, and `fc_2` start to implement the compression of irrelevant features related to key bytes different from $\mathtt{k}_2$, more specifically after epoch 50. Looking at layers `conv_3`, `fc_1`, and `fc_2`, we conclude that these three layers implement bottlenecks, but the perceived information values suggest that this model should be trained for more epochs, as we see a growing trend for relevant features and generalization (given by $\widehat{PI}(X_a^l; \mathcal{Y})$) while perceived information values related to irrelevant features are continuously decreasing.

1. **Where**. Bottleneck layers are usually implemented by layers closer to the output layer. When the CNN contains more than two convolution layers, we observed that the bottleneck happens from the third convolution layer.
2. **What**. The bottleneck is implemented less efficiently when the network is trained on a more noisy dataset. Irrelevant features (and probably other sources of noise) are preserved until the last hidden layer with some level of compression. Relevant features, which are necessary to defeat masking, are usually (but not always) preserved more intensively, allowing the model to implement a second-order attack successfully.

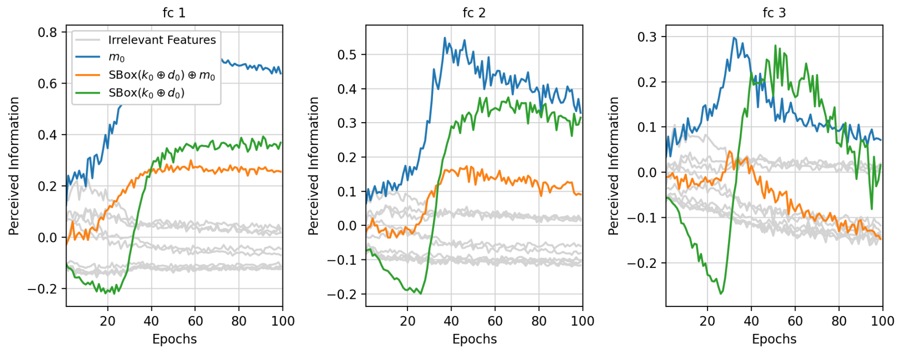

Fig. 10: Perceived information values from a MLP trained with the `DPAv4.2` dataset.

3. **Why**. Our results suggest that the first convolution layers work on bypassing desynchronization effects, becoming less able to separate relevant from irrelevant features.

6.4 `DPAv4.2`

Multilayer Perceptron We apply the same hyperparameter search process from Appendix B (Table 1) for the `DPAv4.2` dataset to find successful MLP models. Again, for these successful models, we observe a similar trend when applying our explainability methodology. Figure 10 shows results for the following layer-wise MLP structure:

$$\mathcal{X} \rightarrow [FC(\mathtt{20}) \rightarrow E]^3 \rightarrow S(\mathtt{256}) \rightarrow \widehat{\mathcal{Y}}. \quad (18)$$

Here, the learning rate is set to `5e-4`, and weights are initialized with `glorot_normal` method. We apply `l1` regularization to all hidden layers with a regularization value of $l1$ = `1e-4`. This figure shows a common trend observed from multiple successful MLP models we found with our hyperparameter search process. The first two hidden layers (when models contain more than two hidden layers, at least) already show the capacity to differentiate between relevant and irrelevant features, with the compression of irrelevant ones. The perceived information values with respect to $\mathtt{m}_0$ are significantly higher than those with respect to $\mathtt{S\text{-}box}(\mathtt{k}_0 \oplus \mathtt{d}_0) \oplus \mathtt{m}_0$ in all hidden layers, which is a consequence of higher SNR values for the mask inside the attacked interval. For the last hidden layer, `fc_3`, perceived information values with respect to $\mathtt{m}_0$ become negative after epoch 40, which is also reflected in the values of $\widehat{PI}(X_a^l; \mathcal{Y})$ that indicate prediction to $\mathcal{Y}$ and overfitting.

1. **Where**. MLP models for `DPAv4.2` implement the bottleneck already in the first fully-connected layers. As also observed for `ASCADr` dataset, the first hidden layer already implements generalization as soon as the compression phase happens.

2. **What**. The bottleneck, implemented already in the first hidden layers, separates relevant from irrelevant features and shows generalization ability. Because `DPAv4.2` is a smaller dataset in terms of profiling traces, overparameterized MLP models tend to overfit the relevant features, resulting in a decrease of $\widehat{PI}(X_a^l;\mathcal{Y})$ values in outer layers.
3. **Why**. With MLPs, the first hidden layer is already able to implement the bottleneck and the generalization to $\mathcal{Y}$, suggesting that a single hidden layer would be enough to implement the profiling model successfully. Our experiments required the usage of `l1` or `l2` regularization to find better-performing models, indicating that overparameterized MLPs are problematic for this dataset.

Convolutional Neural Network We find successful CNN models for `DPAv4.2` by applying the hyperparameter search process from Appendix B (Table 2). We identified several successful CNN models that reduce the guessing entropy of the target key byte to 0 with up to 5 000 attack traces. Figure 11 shows an example result for the CNN with the following structure:

$$\mathcal{X} \to [C(\mathtt{fi},\mathtt{40},\mathtt{10}) \to E \to BN \to AP(\mathtt{2},\mathtt{2})]^4 \to [FC(\mathtt{100}) \to E]^2 \to S(\mathtt{256}) \to \widehat{\mathcal{Y}}, \tag{19}$$

where filters `fi` are set to 4, 8, 12, and 16 for the four convolution layers. For this model, the learning rate is set to `1e-3`, and weights are initialized with `random_uniform` method. Similar to what we observed for `ASCADr` CNN case, for `DPAv4.2`, the first convolution layers fit relevant and irrelevant features without clear compression. From the convolution layer `conv_3`, we verify the occurrence of the bottleneck where relevant features are learned while irrelevant ones are compressed, as shown in Figure 11. The subsequent layers `conv_4`, `fc_1`, and `fc_2` show significant levels of generalization to $\mathcal{Y}$, as we confirm by observing higher values of $\widehat{PI}(X_a^l;\mathcal{Y})$ (green lines). This suggests that convolution layers also implement classification, as this task is usually attributed to fully-connected layers in the DL-SCA literature [8].

1. **Where**. We commonly observed that the first convolution layer shows the fitting of input features while compression happens for relevant and irrelevant ones. The bottleneck layer usually happens from the second or third convolution layer. Generalization to $\mathcal{Y}$ happens with more intensity in hidden layers closer to the output layer.
2. **What**. Compression of irrelevant features by a bottleneck layer tends to happen as soon as the first convolution layer starts to compress all features, including relevant ones. Note that here, we are estimating compression from the full feature map obtained from each convolution layer. To verify if some of the features are not compressed in the first layer, compression should be estimated per filter in a feature map. In terms of generalization to $\mathcal{Y}$, convolution layers closer to the fully-connected layer show positive perceived information values for $\widehat{PI}(X_a^l;\mathcal{Y})$ as they received already relevant features from the previous layers.

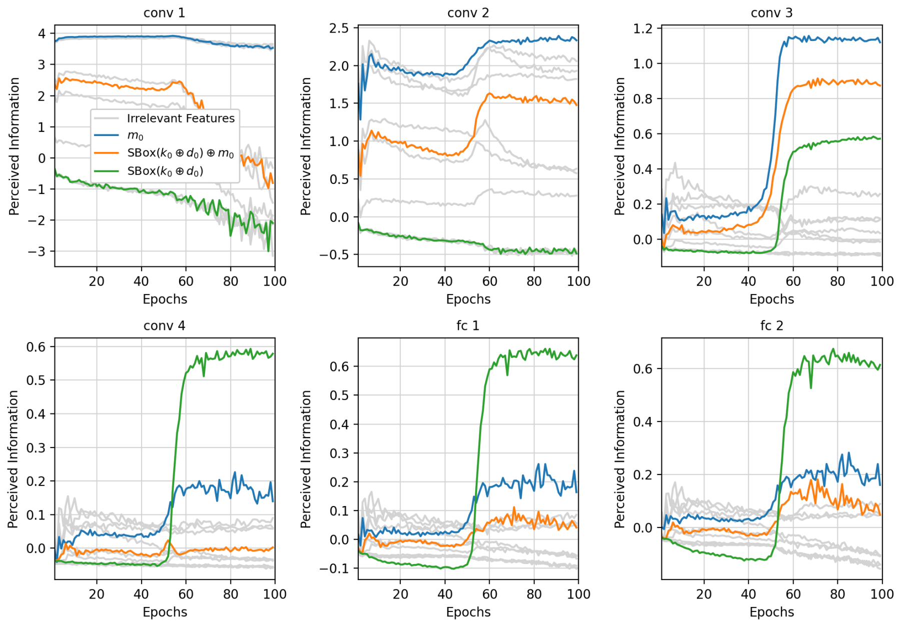

Fig. 11: Perceived information values from a CNN trained with the `DPAv4.2` dataset.

3. **Why**. As our selected CNN models can successfully implement second-order attacks, we verify that hidden layers can automatically locate relevant features from side-channel measurements by differentiating them from irrelevant ones and noise. This suggests that the information bottleneck principle is happening in these models.

Convolutional Neural Networks with Desynchronized Dataset We apply again the random hyperparameter search process from Table 2 to the desynchronized `DPAv4.2` dataset. We apply random shifts to measurements with up to 50 samples and the resulting desynchronized dataset has a very low SNR. Training is performed with data augmentation, in which we generate 70 000 augmented traces per epoch. Again, we use double the number of the profiling traces in data augmentation. Data augmentation is based on random shifts for up to 50 samples.

Figure 12 shows explainability results obtained from a CNN with the following layer-wise structure, which is selected among successful CNN models obtained with the random search:

$$\mathcal{X} \to [C(\mathtt{fi}, 20, 15) \to RE \to BN \to AP(2,2)]^2 \to [FC(20) \to RE]^1 \to S(256) \to \widehat{\mathcal{Y}}. \quad (20)$$

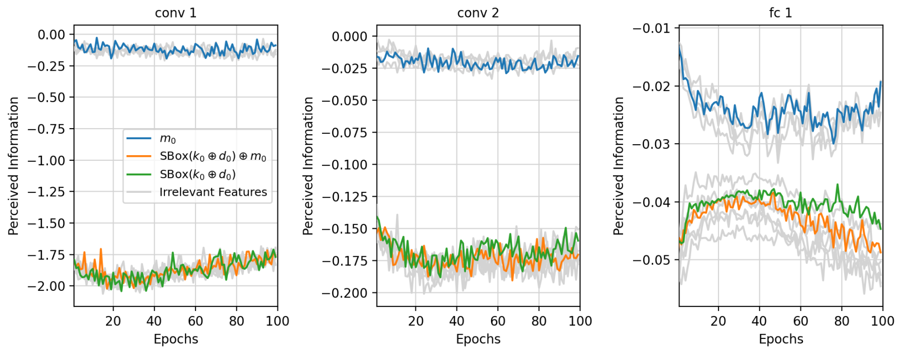

Fig. 12: Perceived information values from a CNN trained with the desynchronized `DPAv4.2` dataset.

Here, filters `fi` are set to 4 and 8 for the two convolution layers. Because the resulting desynchronized dataset has very low SNR, compression and fitting phases are not clearly highlighted in perceived information results. In fact, layer `fc_1` seems to implement the bottleneck as we can observe, from Figure 12, that some irrelevant features are compressed while still not compressing all irrelevant ones. For several models, although they result in successful key recovery with up to 5 000 attack traces, perceived information values are negative, suggesting a sub-optimal profiling model [4].

1. **Where**. Compression usually happens in hidden layers closer to the output layer. This conclusion aligns well with the observations from [42], as there, the authors mentioned that desynchronization makes the deeper layers more involved in processing.
2. **What**. Due to desynchronization effects, the evaluated dataset results in very low SNR. Therefore, we observed fewer bottleneck effects in hidden layers, as some irrelevant features are still preserved in hidden layers.
3. **Why**. Although the evaluated CNNs can successfully implement a key recovery with less than 5 000 attack traces, noise levels are higher, making the bottleneck less efficient.

7 Conclusions and Future Works

Masking countermeasures are powerful protections against profiling SCA, especially when higher levels of noise are present in side-channel measurements [4]. Recent research works have shown that deep learning techniques are effective in defeating masking and hiding countermeasures (see Sections 1 and 3) and in reducing their protective effects in side-channel traces [41]. Therefore, it is of great interest for security engineers and evaluators to understand to what extent the implemented protections are sufficient against different adversaries.

The proposed explainability methodology (ExDL-SCA) brings more clarity to understanding the effect of masking countermeasures against different deep learning-based profiling attacks. Inspired by the information bottleneck principle [37], and through the lens of perceived information [5], we provide a method to visualize **what** every hidden network layer learns from high-order leakages and **which one** of them effectively performs the unmasking operation when the high-order leakages are recombined. We applied our methodology to real side-channel measurements and verified how hidden layers successfully implement a bottleneck to fit relevant features associated with high-order leakages from the target key byte while compressing irrelevant ones. Furthermore, we answered the main explainability questions in all experimental results scenarios.

We evaluated the mask share learnability of deep learning models when the desynchronization countermeasures are implemented. Applying our explainability methodology to CNNs trained with desynchronized datasets allows us to understand how convolution layers deal with this type of hiding countermeasure. In particular, we verify how compression in convolution layers plays an important role for the model to be able to generalize and bypass the countermeasures.

This research opens several new research directions for deep learning-based SCA. In future works, we will investigate and quantify compression and generalization phases in profiling attacks. The main goal is to find the optimal trade-off between these two phenomena in deep neural networks. Furthermore, we will investigate how to tune specific neural network hyperparameters to achieve satisfactory compression of noise and irrelevant features, leading to more efficient profiling attacks against more noisy datasets. In particular, we will research a new way to create the best possible bottleneck that can efficiently regularize the model and discard noise by keeping the information from relevant features. Finally, we will consider the explainability methodology to differentiate between a deep learning model that fails to learn existing leakages (which could transmit a false sense of security) from a deep learning model considered for a successful security evaluation that fails against a protected implementation (which results in a correct security estimation).

A Algorithm for the Mask Shares Explainability

Algorithm 1 provides the required steps to implement the explainability methodology presented in Section 4 when the masking scheme contains two secret shares.

B Random Hyperparameter Search

Tables 1 and 2 provide the ranges for random search for MLP and CNN architectures, respectively. For each scenario (dataset and architecture), we ran 1 000 hyperparameter search attempts, which was enough to find at least fifty successful models.

Algorithm 1 Steps for mask share fitting explainability.

```
1: procedure 2-SHARE MASK EXPLAINABILITY(model $F$, shallow MLP model $q$, number of layers $L$ in model $F$, number of epochs $E_F$ for model $F$, number of epochs $E_q$ for model $q$, profiling set $\mathcal{X}_p$, attack set $\mathcal{X}_a$, label set representing an input feature {}).
2:     for $e = 1$ to $E_f$ do
3:         $F_e \leftarrow$ TrainOneEpoch($F$, $\mathcal{X}_p$)                      ▷ Step 1 in Figure 4
4:         for $l = 1$ to $L$ do
5:             $L_F \leftarrow$ GetLayer($F_e$, $l$)                              ▷ Step 2 in Figure 4
6:             $X_p^l \leftarrow$ LayerPredict($L_F$, $\mathcal{X}_p$)                     ▷ Step 2 in Figure 4
7:             $X_a^l \leftarrow$ LayerPredict($L_F$, $\mathcal{X}_a$)                     ▷ Step 2 in Figure 4
8:             $q^l \leftarrow$ Train($q$, $E_q$, $X_p^l$, $\mathcal{Y}_f$)                    ▷ Step 3 in Figure 4
9:             $\hat{\mathcal{Y}}_f^l \leftarrow$ Predict($q^l$, $X_a^l$)                        ▷ Step 4 in Figure 4
10:            $\widehat{PI}(X_a^l; \hat{\mathcal{Y}}_f^l) =$ PerceivedInformation($X_a^l$, $\hat{\mathcal{Y}}_f^l$)   ▷ Step 4 in Figure 4
11:        end for
12:    end for
13: end procedure
```

Hyperparameter	**Options**
Optimizer	`Adam`
Dense Layers	2, 3, 4, 5, 6
Neurons	20, 40, 50, 100, 150, 200, 300, 400
Activation Function	`elu`, `selu`, `relu`
Learning Rate	0.005, 0.0025, 0.001, 0.0005, 0.00025, 0.0001, 0.00005, 0.000025, 0.00001
Batch Size	400
Epochs	100
Weight Initialization	`random_uniform`, `glorot_uniform`, `he_uniform`, `random_normal`, `glorot_normal`, `he_normal`
Regularization	`None`, `l1` or `l2`
`l1` or `l2`	0.005, 0.0025, 0.001, 0.0005, 0.00025, 0.0001, 0.00005, 0.000025, 0.00001
Total Search Space	**174 960**

Table 1: Hyperparameter search options and ranges for MLPs.

References

1. Belghazi, M.I., Baratin, A., Rajeswar, S., Ozair, S., Bengio, Y., Hjelm, R.D., Courville, A.C.: Mutual information neural estimation. In: Dy, J.G., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018. Proceedings of Machine Learning Research, vol. 80, pp. 530–539. PMLR (2018), `http://proceedings.mlr.press/v80/belghazi18a.html`
2. Belle, V., Papantonis, I.: Principles and practice of explainable machine learning. Frontiers in Big Data **4** (Jul 2021). https://doi.org/10.3389/fdata.2021.688969, `https://doi.org/10.3389/fdata.2021.688969`

Hyperparameter	**Options**
Optimizer	`Adam`
Dense Layers	1, 2
Convolution Layers	1, 2, 3, 4
Neurons	20, 50, 100, 200
Filters	4, 8, 12, 16 ($\times$) Conv. Layer Index
Kernel Size	2, 4, 6, 8, 10, 20, 30, 40
Strides	2, 3, 4, 5, 10, 15, 20
Pooling Size	2
Pooling Stride	2
Activation Function	`elu`, `selu`, `relu`
Learning Rate	0.005, 0.0025, 0.001, 0.0005, 0.00025, 0.0001, 0.00005, 0.000025, 0.00001
Batch Size	400
Epochs	100
Weight Initialization	`random_uniform`, `glorot_uniform`, `he_uniform`, `random_normal`, `glorot_normal`, `he_normal`
Regularization	`None`
Total Search Space	**903 168**

Table 2: Hyperparameter search options and ranges for CNNs.

3. Benadjila, R., Prouff, E., Strullu, R., Cagli, E., Dumas, C.: Deep learning for side-channel analysis and introduction to ASCAD database. J. Cryptographic Engineering **10**(2), 163–188 (2020). https://doi.org/10.1007/s13389-019-00220-8, `https://doi.org/10.1007/s13389-019-00220-8`
4. Bronchain, O., Durvaux, F., Masure, L., Standaert, F.: Efficient profiled side-channel analysis of masked implementations, extended. IEEE Trans. Inf. Forensics Secur. **17**, 574–584 (2022). https://doi.org/10.1109/TIFS.2022.3144871, `https://doi.org/10.1109/TIFS.2022.3144871`
5. Bronchain, O., Hendrickx, J.M., Massart, C., Olshevsky, A., Standaert, F.: Leakage certification revisited: Bounding model errors in side-channel security evaluations. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11692, pp. 713–737. Springer (2019). https://doi.org/10.1007/978-3-030-26948-7_25, `https://doi.org/10.1007/978-3-030-26948-7_25`
6. Burkart, N., Huber, M.F.: A survey on the explainability of supervised machine learning. Journal of Artificial Intelligence Research **70**, 245–317 (Jan 2021). https://doi.org/10.1613/jair.1.12228, `https://doi.org/10.1613%2Fjair.1.12228`
7. Cagli, E., Dumas, C., Prouff, E.: Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without preprocessing. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10529, pp. 45–68. Springer (2017). https://doi.org/10.1007/978-3-319-66787-4_3, `https://doi.org/10.1007/978-3-319-66787-4_3`

8. Cao, P., Zhang, C., Lu, X., Gu, D.: Cross-device profiled side-channel attack with unsupervised domain adaptation. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(4), 27–56 (2021). https://doi.org/10.46586/tches.v2021.i4.27-56, `https://doi.org/10.46586/tches.v2021.i4.27-56`
9. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M.J. (ed.) Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1666, pp. 398–412. Springer (1999). https://doi.org/10.1007/3-540-48405-1_26, `https://doi.org/10.1007/3-540-48405-1_26`
10. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski, B.S., Koç, ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002. pp. 13–28. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
11. Fumaroli, G., Martinelli, A., Prouff, E., Rivain, M.: Affine masking against higher-order side channel analysis. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers. Lecture Notes in Computer Science, vol. 6544, pp. 262–280. Springer (2010). https://doi.org/10.1007/978-3-642-19574-7_18, `https://doi.org/10.1007/978-3-642-19574-7_18`
12. Genelle, L., Prouff, E., Quisquater, M.: Thwarting higher-order side channel analysis with additive and multiplicative maskings. In: Preneel, B., Takagi, T. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6917, pp. 240–255. Springer (2011). https://doi.org/10.1007/978-3-642-23951-9_16, `https://doi.org/10.1007/978-3-642-23951-9_16`
13. Golder, A., Bhat, A., Raychowdhury, A.: Exploration into the explainability of neural network models for power side-channel analysis. In: Proceedings of the Great Lakes Symposium on VLSI 2022. p. 59–64. GLSVLSI '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3526241.3530346, `https://doi.org/10.1145/3526241.3530346`
14. Goldfeld, Z., Polyanskiy, Y.: The information bottleneck problem and its applications in machine learning. CoRR **abs/2004.14941** (2020), `https://arxiv.org/abs/2004.14941`
15. Gunning, D., Stefik, M., Choi, J., Miller, T., Stumpf, S., Yang, G.Z.: Xai-explainable artificial intelligence. Science Robotics **4**(37), eaay7120 (2019). https://doi.org/10.1126/scirobotics.aay7120, `https://www.science.org/doi/abs/10.1126/scirobotics.aay7120`
16. Gunning, D., Vorm, E., Wang, J.Y., Turek, M.: Darpa's explainable ai (xai) program: A retrospective. Applied AI Letters **2**(4) (Dec 2021). https://doi.org/10.1002/ail2.61, `https://doi.org/10.1002/ail2.61`
17. Hettwer, B., Gehrer, S., Güneysu, T.: Deep neural network attribution methods for leakage analysis and symmetric key recovery. In: Paterson, K.G., Stebila, D. (eds.) Selected Areas in Cryptography - SAC 2019 - 26th International Conference, Waterloo, ON, Canada, August 12-16, 2019, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11959, pp. 645–666. Springer (2019). https://doi.org/10.1007/978-3-030-38471-5_26, `https://doi.org/10.1007/978-3-030-38471-5_26`

18. Holzinger, A.: From machine learning to explainable ai. In: 2018 World Symposium on Digital Intelligence for Systems and Machines (DISA). pp. 55–66 (2018). https://doi.org/10.1109/DISA.2018.8490530
19. Kim, J., Picek, S., Heuser, A., Bhasin, S., Hanjalic, A.: Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 148–179 (2019)
20. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Proceedings of CRYPTO'96. LNCS, vol. 1109, pp. 104–113. Springer-Verlag (1996)
21. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1666, pp. 388–397. Springer (1999). https://doi.org/10.1007/3-540-48405-1_25, https://doi.org/10.1007/3-540-48405-1_25
22. Linardatos, P., Papastefanopoulos, V., Kotsiantis, S.: Explainable AI: A review of machine learning interpretability methods. Entropy **23**(1), 18 (Dec 2020). https://doi.org/10.3390/e23010018, https://doi.org/10.3390/e23010018
23. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: International Conference on Security, Privacy, and Applied Cryptography Engineering. pp. 3–26. Springer (2016)
24. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer (December 2006), ISBN 0-387-30857-1, http://www.dpabook.org/
25. Marcinkevics, R., Vogt, J.E.: Interpretability and explainability: A machine learning zoo mini-tour. CoRR **abs/2012.01805** (2020), https://arxiv.org/abs/2012.01805
26. Masure, L., Dumas, C., Prouff, E.: Gradient visualization for general characterization in profiling attacks. In: Polian, I., Stöttinger, M. (eds.) Constructive Side-Channel Analysis and Secure Design - 10th International Workshop, COSADE 2019, Darmstadt, Germany, April 3-5, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11421, pp. 145–167. Springer (2019). https://doi.org/10.1007/978-3-030-16350-1_9, https://doi.org/10.1007/978-3-030-16350-1_9
27. Masure, L., Dumas, C., Prouff, E.: A comprehensive study of deep learning for side-channel analysis. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(1), 348–375 (2020). https://doi.org/10.13154/tches.v2020.i1.348-375, https://doi.org/10.13154/tches.v2020.i1.348-375
28. Perin, G., Buhan, I., Picek, S.: Learning when to stop: A mutual information approach to prevent overfitting in profiled side-channel analysis. In: Bhasin, S., Santis, F.D. (eds.) Constructive Side-Channel Analysis and Secure Design - 12th International Workshop, COSADE 2021, Lugano, Switzerland, October 25-27, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12910, pp. 53–81. Springer (2021). https://doi.org/10.1007/978-3-030-89915-8_3, https://doi.org/10.1007/978-3-030-89915-8_3
29. Perin, G., Wu, L., Picek, S.: Exploring feature selection scenarios for deep learning-based side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems **2022**(4), 828–861 (Aug 2022). https://doi.org/10.46586/tches.v2022.i4.828-861, https://tches.iacr.org/index.php/TCHES/article/view/9842

30. Petch, J., Di, S., Nelson, W.: Opening the black box: The promise and limitations of explainable machine learning in cardiology. Canadian Journal of Cardiology **38**(2), 204–213 (2022). https://doi.org/https://doi.org/10.1016/j.cjca.2021.09.004, https://www.sciencedirect.com/science/article/pii/S0828282X21007030
31. Picek, S., Perin, G., Mariot, L., Wu, L., Batina, L.: Sok: Deep learning-based physical side-channel analysis. ACM Comput. Surv. **55**(11) (feb 2023). https://doi.org/10.1145/3569577, https://doi.org/10.1145/3569577
32. Quisquater, J.J., Samyde, D.: Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In: Attali, I., Jensen, T. (eds.) Smart Card Programming and Security. pp. 200–210. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
33. Roche, T., Lomné, V., Mutschler, C., Imbert, L.: A side journey to titan. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 231–248. USENIX Association (Aug 2021), https://www.usenix.org/conference/usenixsecurity21/presentation/roche
34. Saxe, A.M., Bansal, Y., Dapello, J., Advani, M., Kolchinsky, A., Tracey, B.D., Cox, D.D.: On the information bottleneck theory of deep learning. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. OpenReview.net (2018), https://openreview.net/forum?id=ry_WPG-A-
35. Shwartz-Ziv, R., Tishby, N.: Opening the black box of deep neural networks via information. CoRR **abs/1703.00810** (2017), http://arxiv.org/abs/1703.00810
36. Song, J., Ermon, S.: Understanding the limitations of variational mutual information estimators. In: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net (2020), https://openreview.net/forum?id=B1x62TNtDS
37. Tishby, N., Pereira, F.C.N., Bialek, W.: The information bottleneck method. CoRR **physics/0004057** (2000), http://arxiv.org/abs/physics/0004057
38. van der Valk, D., Picek, S., Bhasin, S.: Kilroy was here: The first step towards explainability of neural networks in profiled side-channel analysis. In: Bertoni, G.M., Regazzoni, F. (eds.) Constructive Side-Channel Analysis and Secure Design - 11th International Workshop, COSADE 2020, Lugano, Switzerland, April 1-3, 2020, Revised Selected Papers. Lecture Notes in Computer Science, vol. 12244, pp. 175–199. Springer (2020). https://doi.org/10.1007/978-3-030-68773-1_9, https://doi.org/10.1007/978-3-030-68773-1_9
39. Won, Y., Hou, X., Jap, D., Breier, J., Bhasin, S.: Back to the basics: Seamless integration of side-channel pre-processing in deep neural networks. IEEE Trans. Inf. Forensics Secur. **16**, 3215–3227 (2021). https://doi.org/10.1109/TIFS.2021.3076928, https://doi.org/10.1109/TIFS.2021.3076928
40. Wouters, L., Arribas, V., Gierlichs, B., Preneel, B.: Revisiting a methodology for efficient cnn architectures in profiling attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(3), 147–168 (Jun 2020). https://doi.org/10.13154/tches.v2020.i3.147-168, https://tches.iacr.org/index.php/TCHES/article/view/8586
41. Wu, L., Picek, S.: Remove some noise: On pre-processing of side-channel measurements with autoencoders. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(4), 389–415 (2020). https://doi.org/10.13154/tches.v2020.i4.389-415, https://doi.org/10.13154/tches.v2020.i4.389-415

42. Wu, L., Won, Y., Jap, D., Perin, G., Bhasin, S., Picek, S.: Explain some noise: Ablation analysis for deep learning-based physical side-channel analysis. IACR Cryptol. ePrint Arch. p. 717 (2021), `https://eprint.iacr.org/2021/717`
43. Yap, T., Benamira, A., Bhasin, S., Peyrin, T.: Peek into the black-box: Interpretable neural network using sat equations in side-channel analysis. Cryptology ePrint Archive, Paper 2022/1247 (2022), `https://eprint.iacr.org/2022/1247`, `https://eprint.iacr.org/2022/1247`
44. Zaid, G., Bossuet, L., Carbone, M., Habrard, A., Venelli, A.: Conditional variational autoencoder based on stochastic attack. Cryptology ePrint Archive, Paper 2022/232 (2022), `https://eprint.iacr.org/2022/232`, `https://eprint.iacr.org/2022/232`
45. Zaid, G., Bossuet, L., Habrard, A., Venelli, A.: Methodology for efficient cnn architectures in profiling attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(1), 1–36 (Nov 2019). https://doi.org/10.13154/tches.v2020.i1.1-36, `https://tches.iacr.org/index.php/TCHES/article/view/8391`