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

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

[1] Leiden University, The Netherlands
[2] Technical University of Darmstadt, Germany
[3] Radboud University, The Netherlands

Abstract. Deep neural networks have proven effective for second-order profiling side-channel attacks, even in a black-box setting with 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 what relevant features deep neural networks use, and where in the networks relevant features are learned and irrelevant features are compressed. Using our method, evaluators can determine what secret masks are being exploited by the network, which allows for more detailed feedback on the implementations. This paper opens new perspectives for understanding the role of different neural network layers in profiling side-channel attacks.[4]

Keywords: Side-channel Analysis, Deep Learning, Countermeasures, Explainability, Perceived Information, Information Bottleneck Theory, Activation Patching

1 Introduction

Side-channel attacks (SCA) represent powerful non-invasive attacks that exploit unintentional leakages of confidential information from electronic devices [22].

[4] Source code for reproducing our results is available in an anonymized repository at `https://anonymous.4open.science/r/explainability-D555/`

During cryptographic executions, the devices leak information through different side channels, such as power consumption [19], electromagnetic emission [34], or execution time [18]. Commonly used SCAs can be broadly categorized as profiled (e.g., Template Attacks [7]) and non-profiled attacks (e.g., Simple Power Analysis [18] and Differential Power Analysis [19]).

Deep learning (DL-SCA) has drawn significant interest recently [33] due to its strong attack capabilities in breaking protected cryptographic implementations [21,5,17], e.g., software/hardware AES protected by first-order (Boolean) masking [44]. 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 attack traces to break the target [3]. Today, we can break the same dataset with a single attack trace and neural network architectures with only a few hidden layers [32]. Unfortunately, despite all of the advances in enhancing attack efficiency, we still lack the knowledge to understand how neural networks break a target due to the complexity of the architecture. Although past research put effort into the understanding of some aspects of deep learning models [25,16,47], 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 [33].

We argue that the capacity to explain the behavior of a machine learning model is at least as important as enhancing attack performance. A deeper understanding of the neural network enables evaluators to 1) understand what leakage the model is exploiting, 2) bolster the security of devices, and 3) ultimately develop devices that are resilient against even the most advanced threats. Following this, we advocate for the necessity of an **Ex**plainable **D**eep **L**earning-based **S**ide-**C**hannel **A**nalysis framework, denoted by **ExDL-SCA**. The primary objectives of ExDL-SCA are to 1) elucidate the sources of (exploitable) information leakage and 2) explain where, in a profiling model, masking countermeasures are defeated. This approach aims to bridge the gap between model transparency and enhanced security in side-channel analysis. To reach those goals, we propose the following questions to be answered with ExDL-SCA:

1. **Where**: explain the contribution of different layers in a neural network.
2. **What**: explain what happens with relevant or irrelevant features and if the information is processed (fitting, compression) to the subsequent layers of a neural network.

This paper proposes an information-theoretic-based explainability methodology that infers how much information the black-box model[5] learns from secret masks. The starting point of our methodology is Information Bottleneck Theory (IB) [41,38], which has sparked much interest from the 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

[5] Secret masks are not supplied during the training of a black-box profiling model and are only considered for post-hoc explainability.

suggests that deep neural network training undergoes two phases: fitting and compression. The fitting phase is supposedly fast and is characterized by hidden layers trying to maximize information about $\mathcal{X}$. At the same time, compression is slower, and it is responsible for the generalization ability of the model. In the compression phase, the network starts to compress noise and other irrelevant features while preserving only relevant features from input training data $\mathcal{X}$. 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, computing $I(\mathcal{X}, T)$ is particularly challenging for discrete and high-dimensional representations [14,36], which constrains the estimation of the compression phase during training. To address this, we adapt the Information Bottleneck (IB) framework to the Perceived Information metric [4], enabling a more precise explanation of the fitting, compression, and generalization phases across different hidden layers. For the first time, we provide security evaluators with specific insights into **where** and **what** each neural network layer learns from high-order leakages during training. This methodology allows evaluators to accurately assess what a profiling model learns—or fails to learn—from the implemented countermeasures at a layer-specific level.
Our main contributions are:

1. We discuss explainability in the context of DL-SCA and recognize the **where** and **what** 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.
4. We conclusively show that neural networks learn to use and combine separate sharing, i.e., with separate randomness of the same sensitive value, to improve model predictions.
5. We validate that the secret masks determined to contribute to model predictions by our method causally contribute to the predictions by executing activation patching experiments.

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 $\mathtt{k}_i$.[6] 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 (e.g., $s(.)$ 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 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.

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 $C(\mathtt{fi}, \mathtt{ks}, \mathtt{st})$ 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, $AP(\mathtt{ps}, \mathtt{st})$ is an average pooling layer with pooling size `ps` and stride `st`, $FC(\mathtt{ne})$ is a fully connected layer with `ne` neurons and $S(\mathtt{c})$ 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.

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

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

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

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

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.[7] 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:

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

where $\diamond$ can indicate a Boolean [6], arithmetic [12], multiplicative [12], or affine [10] 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 the 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 `S-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 `S-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

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

assume that to implement second-order profiling, a neural network must learn the following mapping that also includes a leakage function $\mathcal{L}$:

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

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 $\texttt{S-box}(\texttt{d}_i \oplus \texttt{k}_i) \oplus \texttt{m}_1$ and $\texttt{m}_1$. 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 $g(\texttt{S-box}(\texttt{d}_i \oplus \texttt{k}_i) \oplus \texttt{m}_1, \texttt{m}_1)$.

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

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

From Eq. (7), we can verify that

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

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 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 the network will automatically learn the leakage model properties by implementing the mapping from Eq. (6).

2.3 Mechanistic Interpretability

As deeper models are deployed in a wider range of tasks, a precise understanding of how these models make decisions is increasingly important [15]. Recently, mechanistic interpretability has emerged as a promising subfield that attempts to reverse-engineer neural networks and their behavior. Several key techniques have been proposed that result in significant gains in finding detailed explanations of what neural networks are learning [29]. The main hypothesis underpinning mechanistic interpretability research is that neural networks consist of understandable *features* [30]. These features are the abstract concepts or attributes of the input that models use to make predictions. Features can be low-level, e.g., "there is a curve in the top-right of this image", or more high-level, e.g., "there is a cat in this image". We note that "features" is a term for any information about inputs a model might use to make decisions. While determining what features are used

by a model is a reasonable method to form hypotheses on what a network is learning, to fully understand it, we also need to know how the model is using these features. The mechanisms the model weights implement to manipulate features are referred to as *circuits* [29]. These, again, can be low-level, e.g., how a model is composing several curves into shapes, but circuits also exist at much higher levels of abstraction, e.g., how a model decides whether there is a cat in an image. The goal of mechanistic interpretability is then to find these features and circuits in trained models and give post-hoc explanations that accurately describe model behavior based on the models' internals. In the following sections, we will provide background on two main tools in mechanistic interpretability.

Probing Probing [1] is a method for analyzing the internal representations of neural networks. The main idea is to annotate a dataset with features that are hypothesized to be relevant to a trained model's predictions, save the activations, i.e., outputs of a layer in the model, during the forward pass of the model on this dataset, and finally train a probe, i.e., a small (linear) classifier, to predict the annotated features from these activations. When the probe converges and achieves reasonable accuracy levels (some information about), the feature is represented in the activations. By probing for the evolution of features at different points in training and different intermediate layers, it is possible to determine when and where the model learns to use certain features. Recently, probing has been used to gain insights into several large models. In [26], the authors explored when in the training of AlphaZero [39] knowledge about certain chess concepts is acquired.[8] More recently, in [20,28], probes were used to discover a representation of a world model, i.e., board state, of a simple board game (Othello) in a toy model trained to predict the next move.

Activation Patching Activation Patching[9] is a technique used to understand the behavior of a model by intervening in its internal computations. It involves modifying or "patching" the activations (i.e., the outputs of neurons in a layer) during the forward pass of the network to observe how changes affect the model's predictions or behavior [11]. This technique is useful for understanding the network's internal decision-making process and gaining insight into which layers or neurons are responsible for certain outputs or behaviors. Furthermore, it allows us to test hypotheses about if/how a model uses a feature by changing it during the forward pass. Concretely, for a model that is trained to do addition between two group elements $a, b \in G$, a reasonable hypothesis is that the model is somehow combining a and b in some internal layer [8]. To verify that the model implements recombination before/after a certain layer, during the forward pass, we can replace the activations before that layer that correspond to a with the

[8] From basic concepts like material imbalance to more complex positional concepts used in classical chess engines.

[9] Also referred to as causal tracing [43] or interchange intervention [11].

activations for $a' \in G, a' \neq a$.[10] Then, if the intervention changes the output from ab to $a'b$, there is strong evidence that the model uses this feature, and the combination happens after the intervention. If the output does not change, the recombination might have happened before this layer, or the model might not faithfully implement the group operation. Some concrete examples of using this type of intervention to find/verify circuits are the editing of factual information in language models [27], finding circuits that do grammatical operations in language models [43], and changing the internal board representations [20,28] to alter the predicted next moves.

3 Related Work

Interpretability and explainability in deep learning profiling SCA have received relatively 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 attempt to provide guidelines to build good neural networks, which intuitively means they could interpret what models do [51,45]. Moreover, Zaid et al. used weight visualization and feature maps to understand what features are more important [51]. In [31], 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. Note that the approach we propose in Section 4 differs from this early stopping technique. Indeed, [31] 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 the influence of each feature on the performance of a neural network). Visualization techniques were the first attempt to explain what side-channel trace sample points 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 [24]. The input gradients analysis implements the so-called sensitivity analysis of loss function concerning input features or side-channel samples [24]. In [16], 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]. Finally, Schamberger et al. [37] and Yap et

[10] We obtain the activations for a' by running another forward pass of the model with the a' as input.

al. [49] investigated occlusion techniques to understand better the influence of specific features on the DNN decision-making process.

To explain the behavior of hidden layers in profiling SCA, the authors of [42] 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 and image datasets) 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 [47]. 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 the neural networks learned when retrieving the secret key from the cryptographic primitive [48]. 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 countermeasures. Furthermore, Masure et al. [23] defined a neural network layer that explicitly recombines mask shares. However, this approach requires knowledge about the masking scheme implemented on the device. Finally, Zaid et al. adapted Conditional Variational AutoEncoder [50] to model side-channel traces. With this approach, the authors used the weights of the neural networks to provide an equation of the traces corresponding to the leakage.

The related works concentrate 1) on inputs (features) that do not allow for the explanation of the internal working of neural networks or 2) 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 is still unknown.

4 Explainability Methodology for Profiling SCA - ExDL-SCA

This section describes our 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 adds extra calculations during the deep neural network training. The model is trained with profiling traces $\mathcal{X}_p$ labeled in a black-box

way[11] 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. $\mathcal{X}_p^l$ and $\mathcal{X}_a^l$ are activation outputs[12] from a hidden layer l when the model predicts with profiling and attack sets, respectively. The shapes of $\mathcal{X}_p^l$ and $\mathcal{X}_a^l$ depend on the output layer dimensions.

4.1 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 to learn and combine high-order leakages from input traces automatically. 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 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 quantify efficiently, through a function $q(.)$, how much information a hidden layer representation T has about a certain feature $\mathcal{Y}_f$, i.e., $\mathcal{I}(T, \mathcal{Y}_f)$. We use Perceived Information [4], a lower bound on Mutual Information, as it is relatively straightforward to compute from a classification model predictions and can thus be easily used to quantify the information a trained probe captures for a feature. Alternatives and their drawbacks are discussed in Appendix A.

Perceived Information (PI) Estimation [4] 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 probe $q(.)$ with X_p^l in each layer l to measure how much information T has about an input feature $\mathcal{Y}_f$. The probe $q(.)$ 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

[11] i.e., without mask knowledge

[12] We refer to these as activations throughout this paper as this is the terminology generally used in mechanistic interpretability research

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 use it to predict encoded attack traces X_a^l to obtain class probabilities associated with each selected input feature. 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:

$$H(X_p^l|\mathcal{Y}_f) = \sum_{z=1}^{|\mathcal{Y}_f|} \mathtt{p}(\mathtt{y}_f = z) \sum_{i=1}^{n_p} \mathtt{p}(\mathtt{x}_i^l|\mathtt{y}_f = z) \log_2 \mathtt{p}(\mathtt{y}_f = z|\mathtt{x}_i^l), \tag{9}$$

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. (9):

$$H(X_a^l|\mathcal{Y}_f) = \sum_{z=1}^{|\mathcal{Y}_f|} \mathtt{p}(\mathtt{y}_f = z) \sum_{i=1}^{n_a} \mathtt{p}(\mathtt{x}_i^l|\mathtt{y}_f = z) \log_2 \mathtt{p}(\mathtt{y}_f = z|\mathtt{x}_i^l), \tag{10}$$

where $\mathtt{y}_f$ are labels obtained from intermediate values processed in attack traces $\mathcal{X}_a$. Following the approach proposed in [4], 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*(see Eq. (11) from [4]):

$$\widehat{PI}(X_a^l|\mathcal{Y}_f) = H(\mathcal{Y}_f) + \sum_{z=1}^{|\mathcal{Y}_f|} \mathtt{p}(\mathtt{y}_f = z) \frac{1}{n_{\mathtt{y}_f=z}} \sum_{i=1}^{n_a} \log_2 \hat{\mathtt{p}}(\mathtt{y}_f|\mathtt{x}_i^{l,\mathtt{y}_f=z}), \tag{11}$$

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.[13] 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. (11) *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.*

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. (11). We validate that minor hyperparameter variations do not significantly affect the results in Appendix D. The magnitude of input feature information (such as

[13] 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.

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 $PI(T^l, \mathcal{Y}_f) \geq PI(T^{l+1}, \mathcal{Y}_f)$ always holds for the initial part of the network training. As training progresses, this relationship might not hold anymore. While this may be counterintuitive, as information about a share in a later layer implicitly needs to also be in the earlier layer, the reasons behind this are clear: the shallow MLP $q(.)$ we fit to obtain a PMF for a feature cannot always extract all of the information present in the activations. Later in training, when the network has (possibly) implemented a bottleneck, it is easier for $q(.)$ to extract information in deeper layers as there is less irrelevant information and noise. Note that computing perceived information to estimate information in hidden representations with the classifier $q(.)$ makes our explainability method independent of (although inspired by) IB theory.

4.2 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 1 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 7 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. Instead of training $q(.)$ multiple times (one time for each different input feature $\mathcal{Y}_f$), we 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 $\mathcal{Y}_f$. These quantities are considered for a perceived information calculation as in Eq. (11).

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

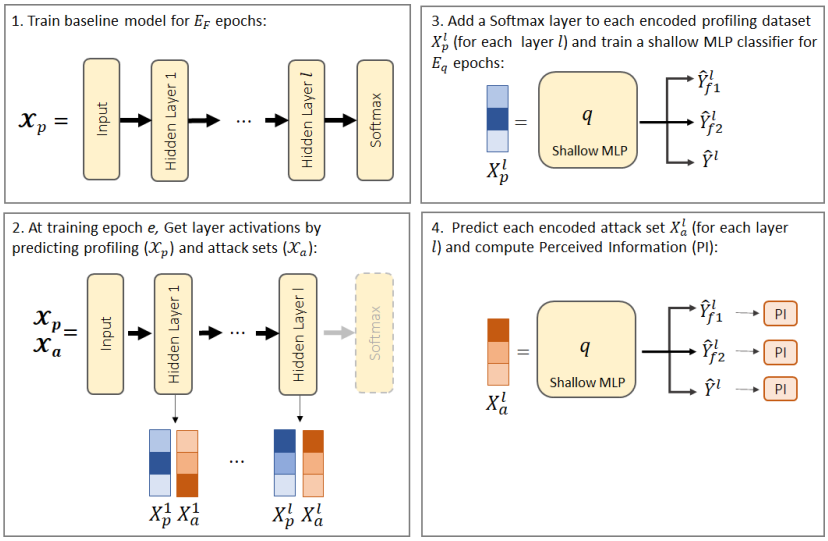


Fig. 1: Methodology for mask share fitting explainability. Here, q represents a shallow MLP with multi-label classification.

two secret shares and becomes capable of implementing a second-order attack. Algorithm 1 (Appendix B) 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 $\mathcal{X}$.

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 Appendix A, measuring the level of compression with mutual information $I(\mathcal{X}; T)$ is difficult and depends on the neural network architectural choices (e.g., activation functions). Therefore, the solution adopted in our work relies on quantifying perceived information of specific input features in hidden layer representations with a separate classifier. This way, our estimation of information in hidden representation is not limited by specific choices such as saturating (e.g., *tanh*) or non-saturating (e.g., *relu* or *elu*) activation functions.

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 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 7 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:

$$\begin{aligned} \mathtt{x}_i[2j] &= HW(\texttt{S-box}(\mathtt{d}_j \oplus \mathtt{k}_j) \oplus \mathtt{m}_j) \\ \mathtt{x}_i[2j+1] &= HW(\mathtt{m}_j), \end{aligned} \tag{12}$$

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. HW returns the Hamming weight of the intermediate variable. In total, we generate 100 000 simulated profiling traces, and 5 000 simulated attack traces.

In the first experiment, we define a 4-layer MLP with the following layer-wise structure:[14]

$$\mathcal{X} \rightarrow [FC(100) \rightarrow E]^4 \rightarrow S(9) \rightarrow \widehat{\mathcal{Y}}.$$

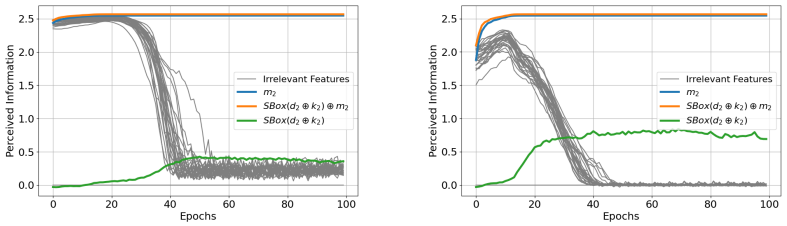
The learning rate and batch size are set to 0.0025 and 400, respectively. Different values for these two hyperparameters would also provide the optimal results we require for our explanations. This MLP is trained with the `Adam` optimizer for 100 epochs. The simulated profiling and attack traces $\mathcal{X}_p$ and $\mathcal{X}_a$ are labeled with the Hamming weight leakage model according to the intermediate values related to the second key byte $j = 2$, i.e., $\mathcal{Y} = \texttt{S-box}(\mathtt{d}_2 \oplus \mathtt{k}_2)$. This is why the output layer $S(9)$ has 9 neurons.

In Figure 2, 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. For the shallow MLP classifier q to compute perceived information from hidden layers, we always use 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 100 epochs.

Since $\mathcal{X}_p$ is labeled with $\mathcal{Y} = \texttt{S-box}(\mathtt{d}_2 \oplus \mathtt{k}_2)$, we expect that the network layers will fit the relevant features corresponding to key byte 2, i.e., $\mathtt{m}_2$ and $\texttt{S-box}(\mathtt{d}_2 \oplus \mathtt{k}_2) \oplus \mathtt{m}_2$, and compress the rest of the features. In Figure 2, we clearly see that in layers 2-4, the information related to irrelevant features decreases around epoch 10. Concurrently, we also see an increase in the perceived information for the label $\mathcal{Y}$. This indicates that IB theory is a solid explanation for the models' behavior. Indeed, later in training, the bottleneck moves from

[14] Hyperparameters are chosen arbitrarily as the simulation is noise-free and essentially anything works.

later layers to the first layer. Again, the start of compression of irrelevant features in the first layer, around epoch 35, is accompanied by an increase in the perceived information for the label $\mathcal{Y}$ in the first layer. Note that the PI values for masks reach 2.56 bits, the maximum value we can obtain under the Hamming weight leakage model.



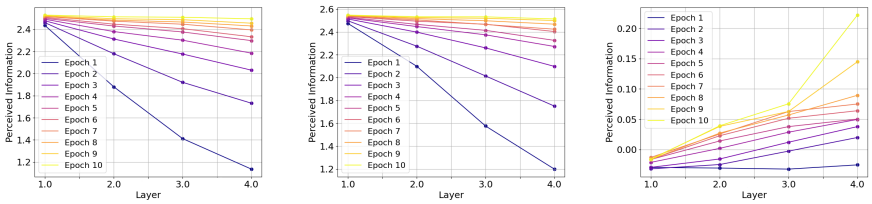
(a) Layer 1 (b) Layer 2

(c) Layer 3 (d) Layer 4

Fig. 2: The compression of irrelevant features with a 4-layer MLP. Blue and orange lines indicate the perceived information from relevant features. Irrelevant features associated with the other key bytes are given as gray lines.

When applying the explainability method to simulated results, we can immediately confirm that during the first training epochs, the information about input features is higher in the first layers and weaker in layers closer to the network output, which confirms that the relation $PI(T^l, \mathcal{Y}_f) \geq PI(T^{l+1}, \mathcal{Y}_f)$ holds. As the training progresses, the network layers start to learn the information from input features, and they might eventually become higher in the middle layers than in the first layers. This scenario is better illustrated in Figure 3c, where the values are the same as shown previously in Figure 2. Figure 3c also shows that $PI(T^l, Sbox(d_2 \oplus k_2))$ grows more aggressively in the deeper layers, which is expected as these layers tend to maximize information about target labels, successfully indicating the combination of two secret shares. We note that this information is also present in earlier layers but that the probes are too shallow to (fully) recombine masks themselves. This allows for a reasonable measure of where the model is doing the mask recombination. Note that as the probe can potentially combine masks itself, especially in later layers when the network is

closer to the prediction, $PI(T^{l+1}, \mathcal{Y}_f)$ we estimate is not a perfect indication of where mask recombination happens.[15]



(a) m_2 (b) $Sbox(d_2 \oplus k_2) \oplus m_2$ (c) $Sbox(d_2 \oplus k_2)$

Fig. 3: The evolution of perceived information values in hidden layers. Initially, when training starts, the relation for input features $PI_{layer1} \geq PI_{layer2} \geq PI_{layer3} \geq PI_{layer4}$ always holds. During training, this relation changes as the relevant input features are transferred to the deeper layers.

6 Validating Results Using Activation Patching

The method presented in Section 4 gives us a solid basis to form hypotheses about the usage of specific secret shares and where in the network mask recombination is happening. However, assessing the validity of these hypotheses is still difficult. While finding the relevant features that relate to tasks in, e.g., language models is highly nontrivial, in our case, it is relatively straightforward. The features that contribute to model predictions should correspond to secret shares that get recombined to the target value at some point in the network, see Eq. (7). If we can find the part of the activations that corresponds to a mask m, then we can replace those parts of the activations with those corresponding to a specific mask value and observe the effects on the output behavior of the network. Concretely, if we take a network that implements $g(\texttt{S-box}(\texttt{d}_i \oplus \texttt{k}_i) \oplus \texttt{m}, \texttt{m})$ and patch it to replace $\texttt{m}$ with 0[16], then the network now implements $g(\texttt{S-box}(\texttt{d}_i \oplus \texttt{k}_i) \oplus \texttt{m}, 0)$, which for the Boolean masking schemes we consider equals $\texttt{S-box}(\texttt{d}_i \oplus \texttt{k}_i) \oplus \texttt{m}$. Thus, if we replace the mask value before recombination, the prediction should change to $\texttt{S-box}(\texttt{d}_i \oplus \texttt{k}_i) \oplus \texttt{m}$. If we replace it after recombination, it should remain the same as it was originally.

7 Experimental Results

For the shallow MLP classifier q, we always use a 1-layer MLP with 100 neurons, where the activation function is `elu`, the learning rate is 0.001, the batch

[15] Our results against `ESHARD` in Figures 8 and 10 provide a practical example of this.

[16] We note that changing $\texttt{m}$ to 0 results in easier mask recombination that amplifies the effect of the patches. This is by design.

size is 400, and the optimizer is `Adam`. The classifier q is always trained for 20 epochs. The precise details for the activation patching experiments provided in this section can be found in Appendix F.

7.1 Datasets

For our experimental results, we considered the trace sets from the ASCAD and `ESHARD` databases, where mask shares are also provided in the metadata. For the considered datasets, we target trace intervals with second-order leakages of multiple key bytes to ensure we include several other intermediate bytes representing irrelevant features during training in different hidden layers. Additionally, we simulate the effect of a hiding countermeasure (desynchronization) for the ASCAD dataset with the results given in Appendix E.

`ASCADr` This dataset contains 300 000 traces,[17] 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 software 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 (following the procedure from [32]). 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. Note that these traces also contain significant leakages of the masked inputs of the `S-box` [9], which we also include in our analysis.

`ESHARD` The `ESHARD` dataset consists of side-channel measurements collected from a software-masked AES-128 implementation running on an ARM Cortex-M4 device. The AES implementation is protected with a first-order Boolean masking scheme with a shared masked `S-box`. In this work, we consider a trimmed version of the dataset that is publicly available[18] and includes the processing of the mask and all masked `S-box` operations in the first encryption round without shuffling. This dataset contains 100 000 measurements, split into groups of 90 000, 5 000, and 5 000 for profiling, validation, and attack sets, respectively.

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

[18] `https://gitlab.com/eshard/nucleo_sw_aes_masked_shuffled`

7.2 Reading the Plots

Ex-DLSCA plots 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. (11)) during training for different features given by specific label sets. For the evaluated datasets, 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 `ESHARD`, the target interval includes second-order leakages from all key bytes, and we target key byte 2.

The main idea is to illustrate the compression of irrelevant features by bottleneck layers. Thus, in all plots, we provide perceived information results for mask 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 those key bytes. The perceived information values for target key bytes (key byte 2) are shown with colored lines (blue color for the mask $\mathtt{m}_j/\mathtt{m}_{out}$ and orange color for $\mathtt{S\text{-}box}(\mathtt{k}_j \oplus \mathtt{d}_j) \oplus \mathtt{m}_{j/out}$) while for the rest of key bytes, we show the results with gray lines. For `ASCADr` mint lines represent $\mathtt{m}_{in}$ and purple lines represent the masked `S-box` input $\mathtt{k}_j \oplus \mathtt{d}_j \oplus \mathtt{m}_{in}$. 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, orange, mint, and purple 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.

Activation Patching Plots The main takeaway from the activation patching plots is to assess whether patching the mask value to be 0 in a specific layer has the expected effect on the model prediction. When mask recombination happens after the layer we patch in, then we expect "Patched Activation/Patched Label" to be high and "Patched Activation/Original Label" to be low, as changing the mask before the recombination happens should affect the output. Conversely, if mask recombination happens before the layer we patch in, we do not expect predictions to change, resulting in "Patched Activation/Patched Label" being low and "Patched Activation/Original Label" being similar to the original PI value as changing the mask value has no effect on the prediction. Note that we always plot the original PI with unaltered activations under "Original Activation/Original Label" as a reference.

7.3 `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. Figure 4 shows a six-layer MLP with the following layer-wise structure:

$$\mathcal{X} \rightarrow [FC(\mathtt{100}) \rightarrow E]^6 \rightarrow S(\mathtt{256}) \rightarrow \widehat{\mathcal{Y}}. \tag{13}$$

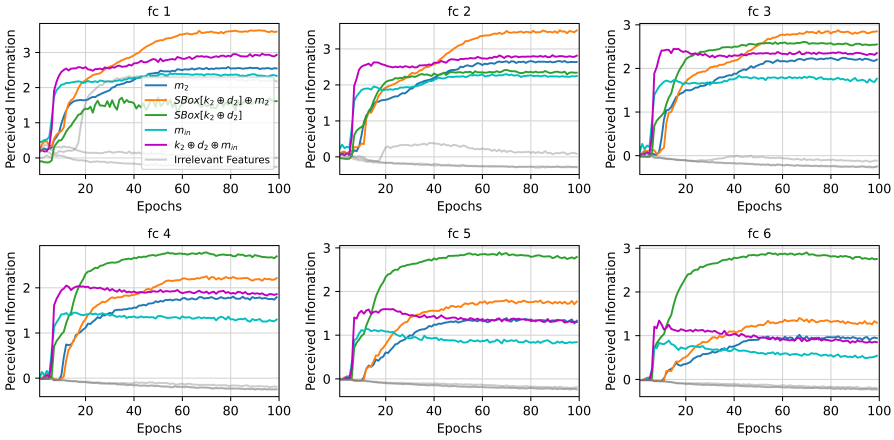


Fig. 4: Perceived information values for various features $\mathcal{Y}_f$ from a six-layer MLP trained with the `ASCADr` dataset.

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 concerning $\mathcal{Y} = \texttt{S-box}(\texttt{k}_j \oplus \texttt{d}_j)$. One of the most interesting observations in Figure 4 is that, especially in the later layers, we see early generalization with only input shares. Then, a bit later in training, we see in `fc_3` that increases in PI for the `S-box` output shares coincide with further increases in PI for the label, indicating that the network first learns to classify from input shares, and later learns to combine this with the leakage from output shares.

1. **Where**. Our results indicate that compression of $\mathcal{X}$ happens in the first hidden layer in this 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 than 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. Indeed, for `ASCADr`, we see that both input and output shares are exploited and combined to improve generalization.

When we attempt to patch different shares for `ASCADr` and MLP, we see in Figure 5 that only patches in the first layer can change the label to a specific targeted value. When we intervene in later layers, we see that changing has a very

limited impact on the predictions, and PI remains positive for the original label for patches to both mask values. This closely matches the results in Figure 4, which indicate compression is complete after the first layer. Furthermore, we note that the primary driver for prediction, especially early on in the generalization of the model, seems to be the `S-box` input shares[19] (full lines in Figure 5), which closely match the initial spike in PI shown in Figure 4 where PI for the `S-box` inputs and the label increase early, after which PI for the output shares also starts slowly increasing. This early generalization without output shares is matched in our ability to change the label to a targeted value with patches using the input shares (orange line in Figure 5). We see that, as the PI for output shares increases (blue and orange lines Figure 4), our ability to effectively patch with input masks decreases until after epoch 20, we cannot get positive PI for the patched label anymore.

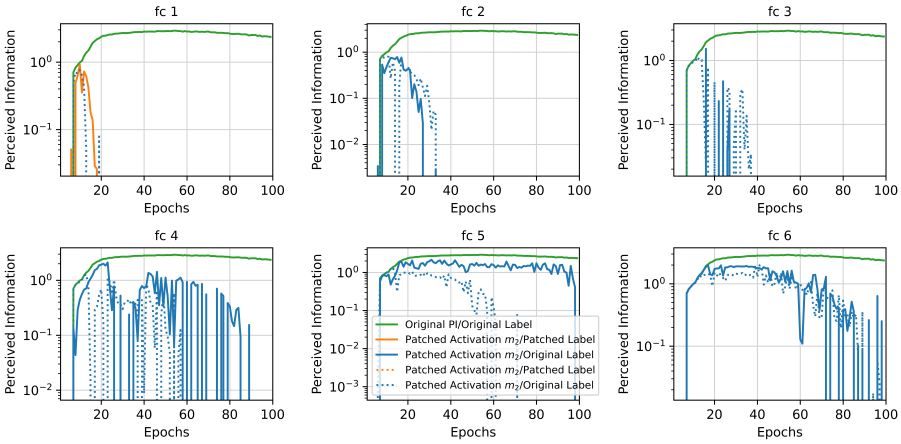


Fig. 5: Perceived information values at the model output for patching activations in `ASCADr` MLP.

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 Supplementary material C (Table 2). Figure 6 shows the results obtained from a CNN with four convolution layers and two fully connected layers with the following structure:

$$\mathcal{X} \rightarrow [C(\texttt{fi}, \texttt{40}, \texttt{15}) \rightarrow SE \rightarrow BN \rightarrow AP(\texttt{2}, \texttt{2})]^4 \rightarrow [FC(\texttt{20}) \rightarrow SE]^2 \rightarrow S(\texttt{256}) \rightarrow \widehat{\mathcal{Y}}, \tag{14}$$

[19] As for the input shares, we patch the `S-box` input mask $\texttt{m}_{in}$, the patched label, in this case, is $\texttt{S-box}(\texttt{d}_j \oplus \texttt{k}_j \oplus \texttt{m}_{in} \oplus 0)$

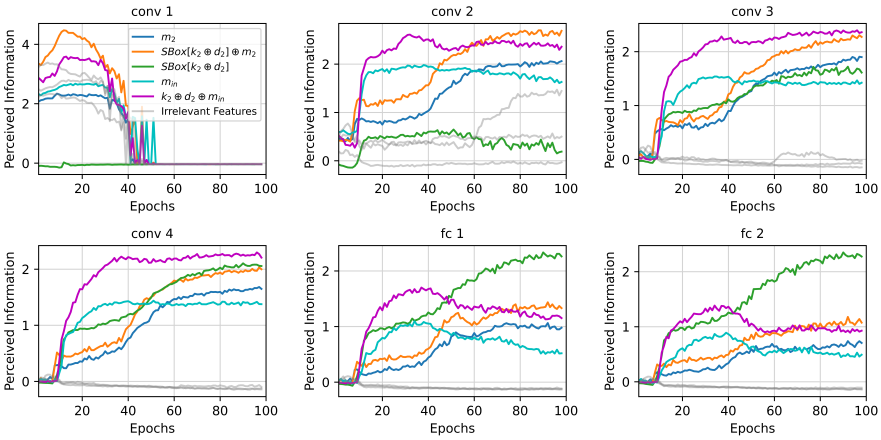


Fig. 6: Perceived information values for various features $\mathcal{Y}_f$ from from CNN layers (Eq. (14)) trained with the `ASCADr` dataset.

where `fi` is set to 12, 24, 36, and 48 for the four convolution layers. The learning rate for this model is `1e-3`, 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 approximately the first 50 epochs. For this layer, after epoch 50, it seemingly starts to happen for all features, but this seems to be the result of the probes not being able to extract the features from the larger representations in `conv_1`. 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_2` already generalized to Y. The subsequent layers 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. Furthermore, we see that the model initially, for epochs 8-10, learns to utilize the `S-box` output shares to get a small, positive PI for $\mathcal{Y}$ in layer `fc_2`. Then, the model predictions significantly improve after it includes the input shares, after which we see a steady increase in PI for $\mathcal{Y}$ and the output shares, as we observed for MLP in Figure 4.

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 implements fitting and compression phases, characterized by increased perceived information values obtained from features (i.e., secret shares) related

to the target key byte. The bottleneck is completed by the fourth convolutional layer, where we only see positive PI for relevant shares.

2. **What**. We again see that the model learns to combine both `S-box` input and output shares. Indeed, we initially see a minor increase in PI for the label as the PI for output shares increases, after which we see an increase in PI for input shares. Finally, just as for the MLP, the PI for output shares slowly increases to improve generalization.

When we look at the patching results in Figure 7, we see that patches after the compression is finished in the second convolutional layer result in the original label being the main prediction early on in training. We also see that we can get positive PI for the patched label when we replace $\mathtt{m}_{in}$ in the first convolutional layer. Besides this, there is a (very) minor spike in PI for the patched label after patching the `S-box` output mask in the second convolutional layer, which occurs in the short period where PI in Figure 6 has not yet increased for the input masks but has started to increase for the output shares and label (around epoch 8-10). When we patch in later layers, the effects of the patches eventually start to decrease. As expected, the PI of patched models with original labels starts approaching the original PI, indicating that the mask recombination has been completed after irrelevant features are fully compressed in the last convolutional layer.

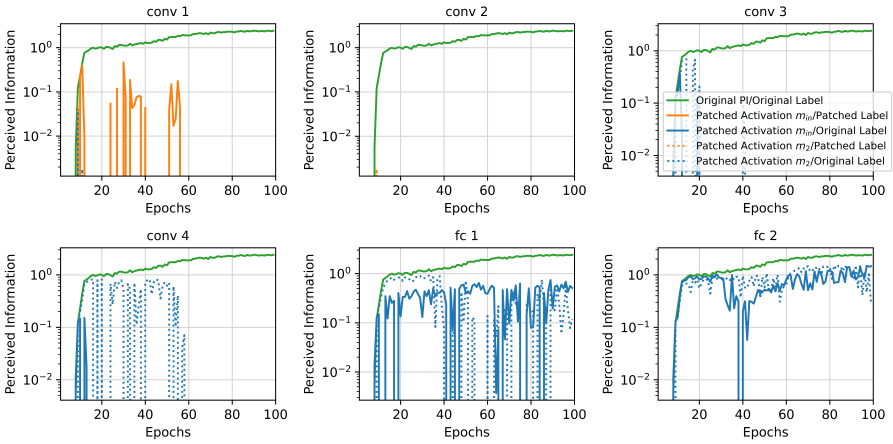


Fig. 7: Perceived information values at model output for patching activations in `ASCADr` CNN.

7.4 `ESHARD`

Multilayer Perceptron We again consider random search to select an MLP architecture that performs well with the `ESHARD` dataset. The found model reaches

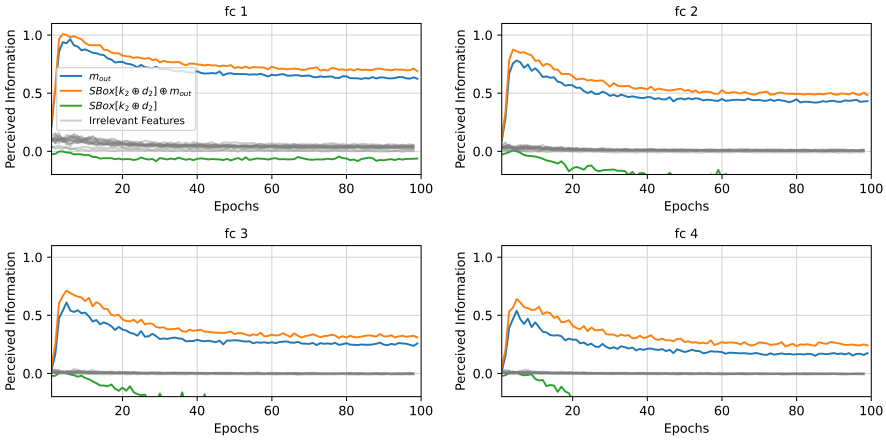


Fig. 8: Perceived information values for varying $\mathcal{Y}_f$ from a four-layer MLP layers (Eq. (15)) trained with the `ESHARD` dataset.

GE of 0 for the correct key $\mathtt{k}_2$ after processing up to 5 000 attack traces. For this dataset, we consider the Hamming weight leakage model. Figure 8 shows the results of a four-layer MLP with the following layer-wise structure:

$$\mathcal{X} \rightarrow [FC(\mathtt{40}) \rightarrow \mathtt{Re}]^4 \rightarrow S(\mathtt{9}) \rightarrow \widehat{\mathcal{Y}}. \tag{15}$$

For this model, the learning rate is set to `0.0025`, and weights are initialized with `he_uniform` method. To reduce overfitting during training, we add a regularization $l1 = 0.000075$ to all hidden layers. The attacked interval includes the second-order leakages from all key bytes, including the target one. We verify that all hidden layers contain information from irrelevant features represented by key bytes different from the target one. In the third and fourth layers, the information regarding irrelevant features is smaller than the previous layers. This model generalizes and can successfully recover the key. The green lines indicate perceived information with respect to $\mathcal{Y} = \mathtt{S\text{-}box}(\mathtt{k}_j \oplus \mathtt{d}_j)$ and it is positive until epoch 10 in layers `fc_3` and `fc_4`. After that, we observe overfitting and performance degradation, but the model still recovers the key from the attack traces.

1. **Where**. For this MLP model, we observe that the compression of irrelevant features already happens mostly in the first two layers. Moreover, the compression is more aggressive in the last two layers, and the PI for the label is additionally more clear in the latter layers, indicating that the mask recombination is happening across layers `fc_2` and `fc_3`.
2. **What**. The explainability method allows us to detect more precisely that after epochs 8-10, the training does not improve the model's generalization, while the PI for relevant shares remains reasonably high. The decrease in PI for relevant shares as training progresses then results in performance

degradation, but as the shares are still present, key recovery is possible. We note that this is potentially part of the explanation for why models that overfit (negative PI, high validation/test loss) can still recover the key.

In Figure 9, we can clearly see that only the patches in the first layer change the label. Indeed, the PI for the new label is about 10 times larger than the original PI. Patches in the second layer result still result in negative PI for the original label, as expected, given that compression is still happening for the irrelevant features in Figure 8. When we patch after the third layer, the effects of the patch are limited, and PI for the original label remains (approximately) the same as with the original activations. Patching later activations does not seem to impact the predictions significantly as PI for the original label remains (approximately) the same as with original activations, indicating that masks have already been recombined.

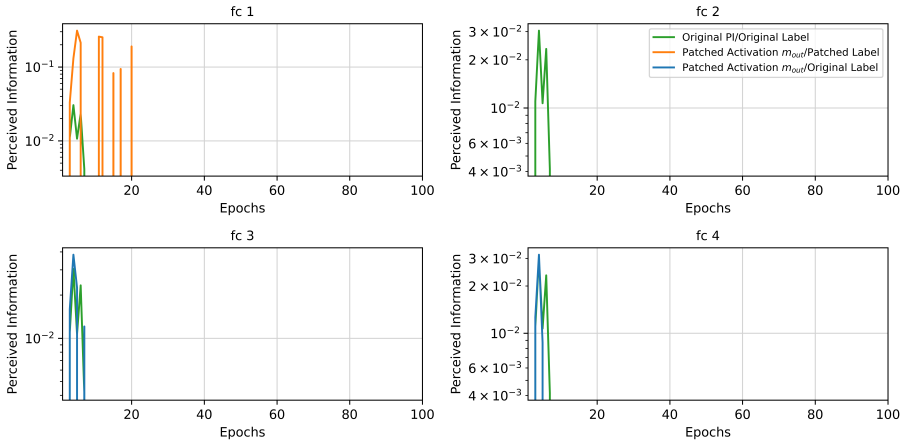


Fig. 9: Perceived information values at model output for patching activations in `ESHARD` MLP.

Convolutional Neural Network With another random search, we found a CNN architecture that can successfully recover the key for the `ESHARD` dataset. Figure 10 shows the results obtained from a CNN with two convolution layers and two fully connected layers with the following structure:

$$\mathcal{X} \rightarrow [C(\texttt{fi}, 10, 5) \rightarrow SE \rightarrow BN \rightarrow AP(2,2)]^4 \rightarrow [FC(100) \rightarrow SE]^2 \rightarrow S(9) \rightarrow \widehat{\mathcal{Y}}, \tag{16}$$

where `fi` is set to 16 and 32 for the two convolution layers. The learning rate for this model is `0.0025`, and trainable weights are initialized with `he_uniform` method. Again, we add a regularization $l1 = 0.000075$ to all hidden layers. The first layer, `conv_1`, fits information from relevant and irrelevant features, as it

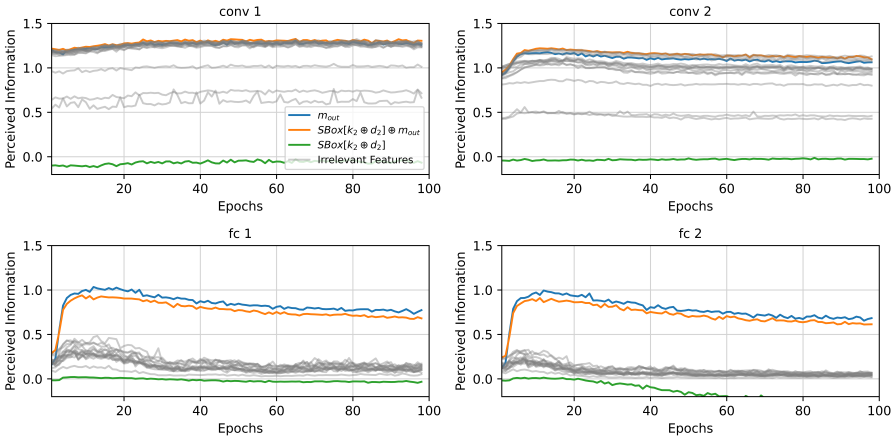


Fig. 10: Perceived information values for varying $\mathcal{Y}_f$ from the CNN layers (Eq. (16)) trained with the ESHARD dataset.

does not show any compression of irrelevant features. The compression becomes visible in the second convolution layer (conv_2). The fc_1 layer already shows high compression levels of irrelevant features while it preserves the relevant ones, and even further compression of irrelevant features occurs in the fc_2.

1. **Where**. Our results clearly show that the compression of irrelevant features starts happening in the first fully connected layer. Note that the PI for irrelevant features is somewhat lower in fc_2 than fc_1, indicating that the compression is only fully completed in the final layer.
2. **What**. The model clearly learns to preserve the relevant shares, even after it overfits, although the PI for the relevant features starts marginally decreasing. The compression of irrelevant features also reduces PI for irrelevant features after the PI for the label is negative. Again, we note that PI for relevant features remains reasonably high while the model overfits, which is quite surprising, and this is potentially part of the explanation for why the model can still extract the key. Additional results and discussion around the overfitting dynamics for ESHARD models are in Appendix D.1.

As can be seen in Figure 10, the compression of irrelevant features seems to only be fully complete after the second fully connected layer. When we look at the patching results in Figure 11, we see that patches in the first three layers effectively change the label to the target value, indicating that mask recombination only happens after the irrelevant features are suppressed. Indeed, even when the model normally overfits, these patches result in positive PI for a changed label. Only if we patch in the last fully connected layer, we see some positive PI for the original labels. This again confirms the results based on mask recombination only happening after the compression of irrelevant features.

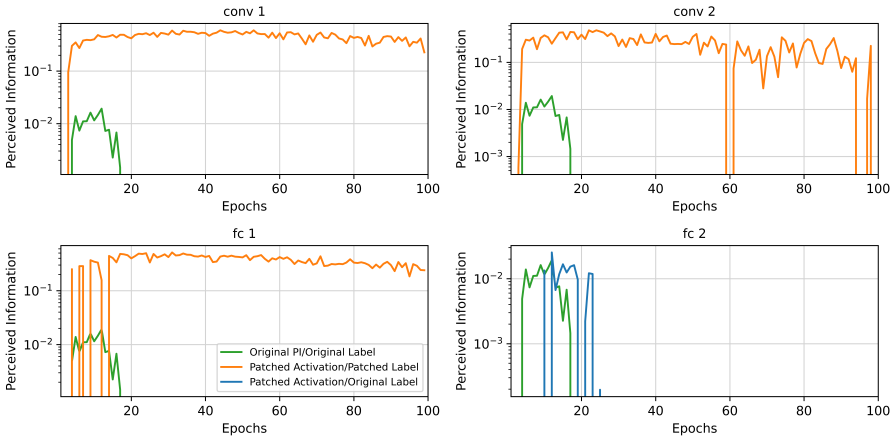


Fig. 11: Perceived information values at model output for patching activations in ESHARD CNN.

7.5 Discussion

We have presented a methodology for the post-hoc analysis of neural networks that break cryptographic implementations. We show that models learn to compress irrelevant features across several layers and showcase a method for demonstrating how information related to features evolves in each layer during training. These results clearly allow us to evaluate what features, or mask shares, are contributing to the models' predictions and are, therefore, contributing to exploitable leakage. Notably, the answers to **what** the networks are learning are strongly supported by both IB theory and empirical validation. The networks clearly implement a bottleneck and compress irrelevant features throughout training. We further validate that the features that are not compressed in the networks are causally related to the networks' outputs by patching activations to change the network predictions to a target value. The answers to **where** networks are recombining shares are somewhat harder to verify. Indeed, it seems quite clear that the intuition behind recombination happening after irrelevant features are compressed is empirically supported by patching results. However, as the patching method is not extremely precise, pinpointing the exact layer where masks are recombined can be difficult.

8 Conclusions and Future Works

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 [41], and through the lens of perceived information [4], we provide a method to visualize **what** every hidden network layer learns from high-order leakages and which

layer(s) (i.e., **where**) effectively perform 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 and validated that these answers are correct by testing the generated hypotheses about **what** masks the networks are recombining **where**.

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 to efficiently regularize the model and discard noise by keeping the information from relevant features. Finally, the patching method introduced in Section 6 seems promising for future work. Possible directions include refining the share localization to allow for more precise interventions, automating part(s) of the analysis, and a more systematic evaluation of what can be done with these interventions.

References

1. Alain, G., Bengio, Y.: Understanding intermediate layers using linear classifier probes. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings. OpenReview.net (2017), https://openreview.net/forum?id=HJ4-rAVtl
2. 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
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., 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
5. 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

6. 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
7. 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)
8. Chughtai, B., Chan, L., Nanda, N.: A toy model of universality: Reverse engineering how networks learn group operations. In: Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., Scarlett, J. (eds.) International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA. Proceedings of Machine Learning Research, vol. 202, pp. 6243–6267. PMLR (2023), https://proceedings.mlr.press/v202/chughtai23a.html
9. Egger, M., Schamberger, T., Tebelmann, L., Lippert, F., Sigl, G.: A second look at the ASCAD databases. In: Balasch, J., O'Flynn, C. (eds.) Constructive Side-Channel Analysis and Secure Design - 13th International Workshop, COSADE 2022, Leuven, Belgium, April 11-12, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13211, pp. 75–99. Springer (2022). https://doi.org/10.1007/978-3-030-99766-3_4, https://doi.org/10.1007/978-3-030-99766-3_4
10. 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
11. Geiger, A., Lu, H., Icard, T., Potts, C.: Causal abstractions of neural networks. In: Ranzato, M., Beygelzimer, A., Dauphin, Y.N., Liang, P., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual. pp. 9574–9586 (2021), https://proceedings.neurips.cc/paper/2021/hash/4f5c422f4d49a5a807eda27434231040-Abstract.html
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., 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
16. 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
17. 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)
18. 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)
19. 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
20. Li, K., Hopkins, A.K., Bau, D., Viégas, F., Pfister, H., Wattenberg, M.: Emergent world representations: Exploring a sequence model trained on a synthetic task. In: The Eleventh International Conference on Learning Representations (2023), https://openreview.net/forum?id=DeG07_TcZvT
21. 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)
22. 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/
23. Masure, L., Cristiani, V., Lecomte, M., Standaert, F.: Don't learn what you already know scheme-aware modeling for profiling side-channel analysis against masking. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2023**(1), 32–59 (2023). https://doi.org/10.46586/TCHES.V2023.I1.32-59, https://doi.org/10.46586/tches.v2023.i1.32-59
24. 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
25. 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

26. McGrath, T., Kapishnikov, A., Tomasev, N., Pearce, A., Hassabis, D., Kim, B., Paquet, U., Kramnik, V.: Acquisition of chess knowledge in alphazero. CoRR **abs/2111.09259** (2021), https://arxiv.org/abs/2111.09259
27. Meng, K., Bau, D., Andonian, A., Belinkov, Y.: Locating and editing factual associations in GPT. In: Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A. (eds.) Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022 (2022), http://papers.nips.cc/paper_files/paper/2022/hash/6f1d43d5a82a37e89b0665b33bf3a182-Abstract-Conference.html
28. Nanda, N., Lee, A., Wattenberg, M.: Emergent linear representations in world models of self-supervised sequence models. In: Belinkov, Y., Hao, S., Jumelet, J., Kim, N., McCarthy, A., Mohebbi, H. (eds.) Proceedings of the 6th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@EMNLP 2023, Singapore, December 7, 2023. pp. 16–30. Association for Computational Linguistics (2023). https://doi.org/10.18653/V1/2023.BLACKBOXNLP-1.2, https://doi.org/10.18653/v1/2023.blackboxnlp-1.2
29. Olah, C., Cammarata, N., Schubert, L., Goh, G., Petrov, M., Carter, S.: Zoom in: An introduction to circuits. Distill (2020). https://doi.org/10.23915/distill.00024.001, https://distill.pub/2020/circuits/zoom-in
30. Olah, C., Mordvintsev, A., Schubert, L.: Feature visualization. Distill (2017). https://doi.org/10.23915/distill.00007, https://distill.pub/2017/feature-visualization
31. 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
32. 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
33. 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
34. 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)
35. Rijsdijk, J., Wu, L., Perin, G., Picek, S.: Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(3), 677–707 (2021).

https://doi.org/10.46586/tches.v2021.i3.677-707, https://doi.org/10.46586/tches.v2021.i3.677-707

36. 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-
37. Schamberger, T., Egger, M., Tebelmann, L.: Hide and Seek: Using Occlusion Techniques for Side-Channel Leakage Attribution in CNNs: An Evaluation of the ASCAD Databases, p. 139–158. Springer Nature Switzerland (2023). https://doi.org/10.1007/978-3-031-41181-6_8, http://dx.doi.org/10.1007/978-3-031-41181-6_8
38. 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
39. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T.P., Simonyan, K., Hassabis, D.: Mastering chess and shogi by self-play with a general reinforcement learning algorithm. CoRR **abs/1712.01815** (2017), http://arxiv.org/abs/1712.01815
40. 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
41. Tishby, N., Pereira, F.C.N., Bialek, W.: The information bottleneck method. CoRR **physics/0004057** (2000), http://arxiv.org/abs/physics/0004057
42. 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
43. Wang, K.R., Variengien, A., Conmy, A., Shlegeris, B., Steinhardt, J.: Interpretability in the wild: a circuit for indirect object identification in GPT-2 small. In: The Eleventh International Conference on Learning Representations (2023), https://openreview.net/forum?id=NpsVSN6o4ul
44. 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
45. 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
46. 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

47. Wu, L., Won, Y.S., Jap, D., Perin, G., Bhasin, S., Picek, S.: Ablation analysis for multi-device deep learning-based physical side-channel analysis. IEEE Transactions on Dependable and Secure Computing pp. 1–12 (2023). https://doi.org/10.1109/TDSC.2023.3278857
48. Yap, T., Benamira, A., Bhasin, S., Peyrin, T.: Peek into the black-box: Interpretable neural network using sat equations in side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems **2023**(2), 24–53 (Mar 2023). https://doi.org/10.46586/tches.v2023.i2.24-53, https://tches.iacr.org/index.php/TCHES/article/view/10276
49. Yap, T., Picek, S., Bhasin, S.: Beyond the last layer: Deep feature loss functions in side-channel analysis. In: Proceedings of the 2023 Workshop on Attacks and Solutions in Hardware Security. p. 73–82. ASHES '23, Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3605769.3623996, https://doi.org/10.1145/3605769.3623996
50. Zaid, G., Bossuet, L., Carbone, M., Habrard, A., Venelli, A.: Conditional variational autoencoder based on stochastic attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems **2023**(2), 310–357 (Mar 2023). https://doi.org/10.46586/tches.v2023.i2.310-357, https://tches.iacr.org/index.php/TCHES/article/view/10286
51. 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

A Alternative MI Estimation Methods

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 [38]. 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 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:

$$I(\mathcal{X};T) = H(T) - H(T|\mathcal{X}) = H(T) - \sum_{i=1}^{n_p} \mathtt{p}(\mathtt{x}_i) \sum_{j=1}^{n_p} \mathtt{p}(\mathtt{t}_j|\mathtt{x}_i) \log_2 \mathtt{p}(\mathtt{t}_j|\mathtt{x}_i). \quad (17)$$

As discussed in [36], 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 [31]). 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 to estimate the amount of information that T has about $\mathcal{X}$ or $\mathcal{Y}_f$.

Mutual Information Neural Estimation (MINE) In [2], 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 requirements: (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.
We implemented experiments with MINE, and, in multiple cases, we verified that the exploding gradient problem happens.[20] This could be solved with regularization, gradient clipping, or carefully adjusting the MINE network hyperparameters. The authors of [40] 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.

B 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. Note that the algorithm can be easily adapted to any number of secret shares.

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

Algorithm 1 Steps for mask share fitting explainability.

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 {}).
  for $e = 1$ to E_f **do**
    $F_e \leftarrow$ `TrainOneEpoch`$(F, \mathcal{X}_p)$ ▷ Step 1 in Figure 1
    for $l = 1$ to L **do**
      $L_F \leftarrow$ `GetLayer`(F_e, l) ▷ Step 2 in Figure 1
      $X_p^l \leftarrow$ `LayerPredict`$(L_F, \mathcal{X}_p)$ ▷ Step 2 in Figure 1
      $X_a^l \leftarrow$ `LayerPredict`$(L_F, \mathcal{X}_a)$ ▷ Step 2 in Figure 1
      $q^l \leftarrow$ `Train`$(q, E_q, X_p^l, \mathcal{Y}_f)$ ▷ Step 3 in Figure 1
      $\hat{\mathcal{Y}}_f^l \leftarrow$ `Predict`(q^l, X_a^l) ▷ Step 4 in Figure 1
      $\widehat{PI}(X_a^l; \hat{\mathcal{Y}}_f^l) =$ `PerceivedInformation`$(X_a^l, \hat{\mathcal{Y}}_f^l)$ ▷ Step 4 in Figure 1
    end for
  end for
end procedure

C 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.

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.

D Impact of Hyperparameters for Probe q

The probe 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

Hyperparameter	Options
Optimizer	`Adam`
Dense Layers	1, 2
Convolution Layers	1, 2, 3, 4
Neurons	20, 50, 100, 200
Filters	4, 8, 12, 16 (×) 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.

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, see, e.g., [35], we demonstrate that 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 12 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. The difference is that this time, trace simulations are made for the Identity leakage model, which allows us to obtain PI values up to 8 bits. Perceived information values obtained with this MLP model are lower than 8 because this model is not optimal and does not reach 100% of classification accuracy. This grid search takes the following hyperparameter values and generates all possible combinations. We consider a) layers: $[1, 2]$, b) neurons: $[50, 100]$, c) batch sizes: $[200, 400]$, d) epochs: $[10, 20]$, and e) activation functions: $[\texttt{elu}, \texttt{selu}]$.

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 $\texttt{m}_2$ (blue line) and $\texttt{S-box}(\texttt{d}_2 \oplus \texttt{k}_2) \oplus \texttt{m}_2$ (orange line) plus the perceived information of the target label $\mathcal{Y} = \texttt{S-box}(\texttt{d}_2 \oplus \texttt{k}_2)$ (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. This means that varying the hyperparameters for the shallow MLP implementing q does not significantly impact results.

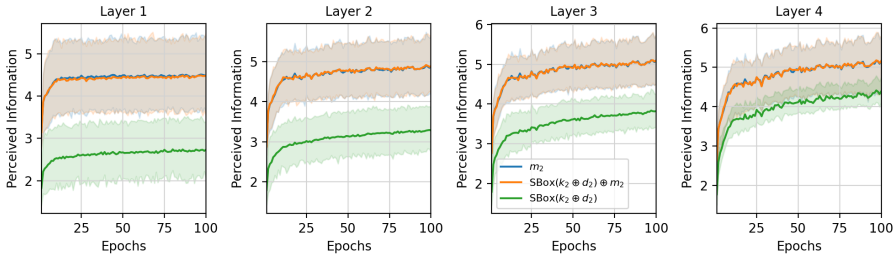


Fig. 12: Variation of measured perceived information for different hyperparameter combinations for classifier $q(T, \mathcal{Y}_f)$.

D.1 On Overfitting in ESHARD Models

Our analysis of the ESHARD models clearly indicates that the model overfits. During training, after an initial decrease in PI, the models start memorizing training examples, and the PI for the attack set becomes negative. However, even after the model is overfitting for most of its training epochs, it can still recover the key. This raises some questions about what is happening in the models. Our explanation in Section 7 ids that the relevant shares are still present and the model, therefore, still has some generalization ability. While this is a reasonable explanation that is supported by the patching results in Figure 11, it raises further questions about how the model is learning. Why do our patches result in positive PI being maintained throughout training?

The first point is that for ESHARD, the leakage closely follows the HW distribution.[21] As we patch to set $\mathtt{m}_{out} = 0$, the model can directly pass the HW of the output share to its prediction. When the HW of the mask is, e.g., 4, the combination is less certain. To illustrate, if both mask and masked output have HW 4, we cannot determine the HW of the recombination with certainty. This then leads to a further point about the dynamics of model overfitting. If the model has learned a reasonable function for recombining shares, it can improve its training loss further by memorizing specific examples. Here, we propose that it first learns to memorize the "harder cases" where the HW of the mask leaves more uncertainty (i.e., $HW(\mathtt{m}) \in \{2, 3, 4, 5, 6\}$). To give some evidence for this, we plot PI for both ESHARD models across training for easy/hard attack traces in Figure 13. We see that PI for easy cases is significantly higher, but this is an obvious outcome of these cases being easier. The main point is that the PI for easy cases starts decreasing significantly later, indicating that overfitting starts with hard cases. While this insight does not have any direct applications, it improves our understanding of the dynamics of training models in noisy environments. Furthermore, it is a step towards answering why models that overfit can still (sometimes) extract keys.

[21] For ASCADr, the leakage is biased towards specific bits in the byte.

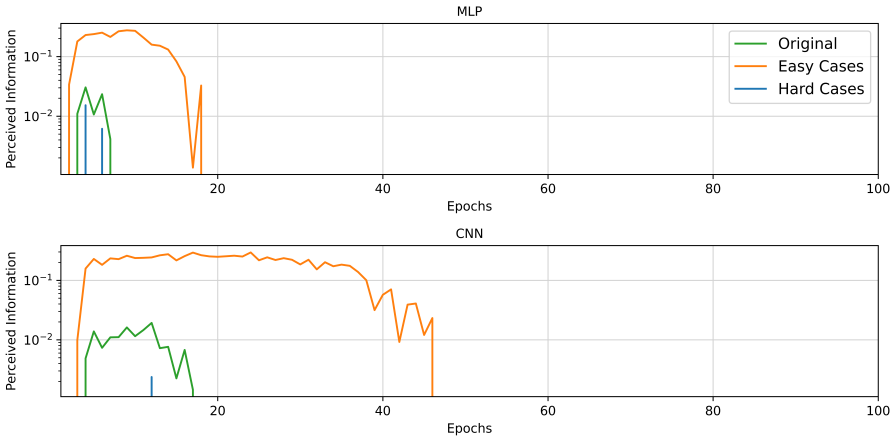


Fig. 13: PI for ESHARD models across training for easy ($HW(\mathtt{m}) \in \{0, 1, 7, 8\}$)) and hard ($HW(\mathtt{m}) \in \{2, 3, 4, 5, 6\}$) cases.

E Results with ASCADr Desynchronized Dataset

To produce misalignment, traces are randomly shifted by up to 50 samples (see [46] for details on how to simulate the desynchronization effect). To circumvent the desynchronization effect [5], 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 hyperparameter search until we generate at least 100 successful CNN models able to reduce the GE of the correct key to 0. In Figure 14, we provide an example result from a CNN model with the following layer-wise structure:

$$\mathcal{X} \rightarrow [C(\mathtt{fi}, \mathtt{30}, \mathtt{15}) \rightarrow E \rightarrow BN \rightarrow AP(\mathtt{2}, \mathtt{2})]^3 \rightarrow [FC(\mathtt{200}) \rightarrow E]^2 \rightarrow S(\mathtt{256}) \rightarrow \widehat{\mathcal{Y}}, \tag{18}$$

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 14, we see how the first two convolution layers conv_1 and conv_2 process relevant and irrelevant features. These layers mostly fit all features without 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 PI 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 PI values related to irrelevant features are continuously decreasing.

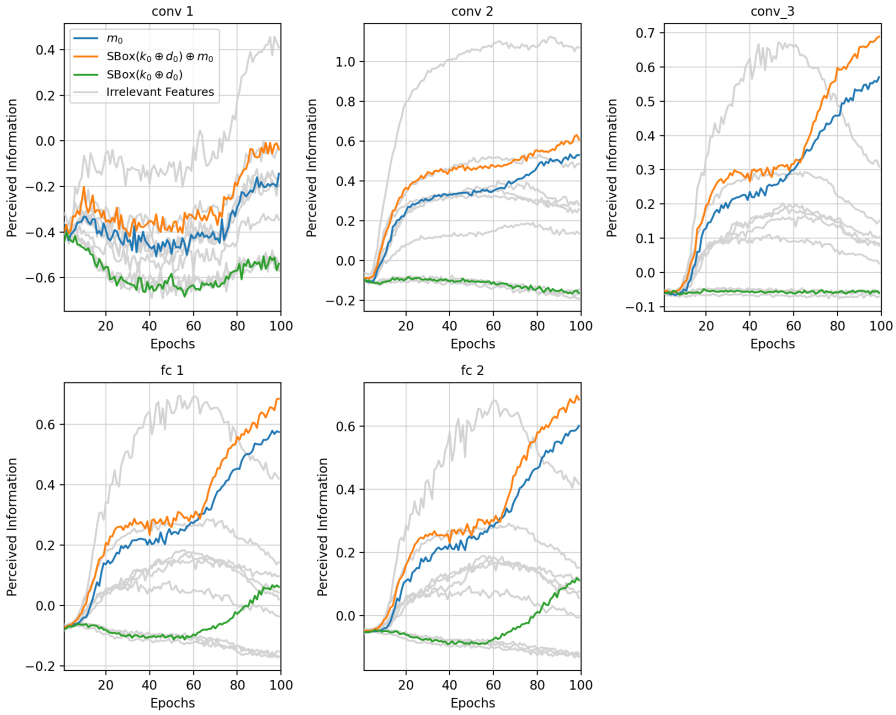


Fig. 14: Perceived information values from a CNN trained with the desynchronized `ASCADr` dataset.

1. **Where**. Bottleneck layers are usually implemented by layers closer to the output layer. When a CNN has 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 noise sources) 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.

F Experimental Setup Activation Patching

To apply effective activation patches for a secret mask `m`, we developed the following procedure, which is visualized in Figure 15:

1. Take activations at layer l, X_p^l, X_a^l.

2. Use Principal Component Analysis (PCA) to simplify the number of components.[22]
3. Find component(s) corresponding to $\mathtt{m}$ using Signal-to-Noise Ratio, selecting components that have $SNR >$ **mean(SNR)**.
4. Replace selected component(s) in attack set with average of the selected component(s) for X_p^l where $\mathtt{m} = 0$.
5. Invert the PCA for the patched components to get new activations $X_a^{\prime l}$.
6. Observe the effect on PI with output of model for both original label $\mathtt{S\text{-}box}(\mathtt{d}_j \oplus \mathtt{k}_j)$ and target label $\mathtt{S\text{-}box}(\mathtt{d}_j \oplus \mathtt{k}_j) \oplus \mathtt{m}$.

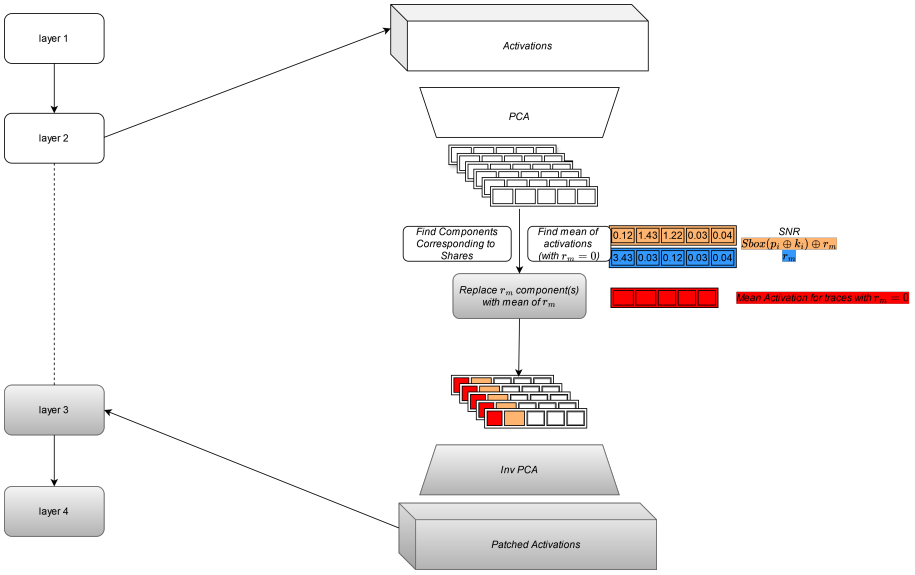


Fig. 15: Diagram of setup for activation patching.

G A Cautionary Note

While we can visualize the evolution of perceived information for selected shares, and this allows for a significantly better understanding of both the post-exploitation analysis and the training dynamics of the networks, it does rely on evaluator expertise to determine what masks/shares should be included in the analysis. Indeed, for `ASCADr` in particular, we showcase that both `S-box` input and output shares are used by the models to extract the key, but in our initial experiments, we did not include the input masks. We showcase the corresponding plots for the

[22] Using PCA is inspired by the ablation experiments that remove irrelevant components to verify the faithfulness of the found algorithms in [8].

MLP model[23] in Figure 16. In this figure, the plots look reasonable: unnecessary features get compressed, and the relevant features do not. However, when we know the plots are incomplete, from the `fc_2` layer onward, we can clearly see that PI for the label increases *before* PI for the output shares. However, this can be straightforwardly explained by the model using input shares.

We include this appendix to emphasize that while the explanations generated by our method are useful in explaining model behavior, they require a critical examination. If there is some odd behavior in the plots, an evaluator should carefully consider whether additional exploitable leakage could be present and, when possible, include more features in the analyses than presumed necessary.

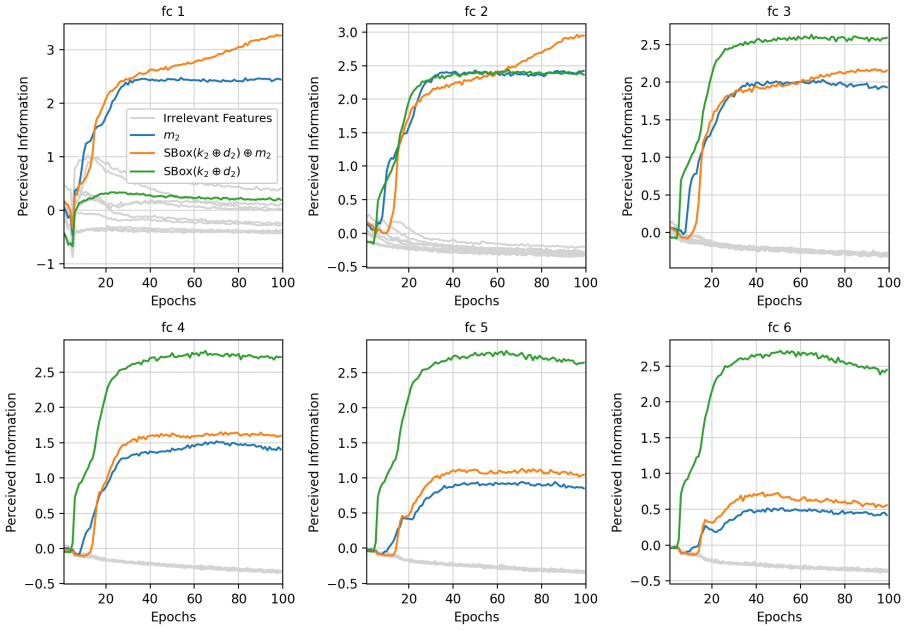


Fig. 16: Old PI evolution for `ASCADr` MLP.

[23] Same hyperparameters as in Eq. (13) but different training run.