

# Glimpsing the Leakage Flow for Deep Learning-Based Side-Channel Analysis using Stochastic Leakage Model

Trevor Yap<sup>1,2</sup>[0000–0001–8651–574X] and Shivam Bhasin<sup>1,2</sup>[0000–0002–6903–5127]

<sup>1</sup> Temasek Laboratories, Nanyang Technological University, Singapore

<sup>2</sup> National integrated Centre For Evaluation, Nanyang Technological University,  
Singapore  
trevor.yap@ntu.edu.sg  
sbhasin@ntu.edu.sg

**Abstract.** Deep learning has become the dominant method for profiled Side-Channel Analysis (SCA), successfully recovering secret keys even in the presence of strong countermeasures. Despite its efficacy, Deep Neural Networks (DNNs) are predominantly black-box in nature. Evaluators do not know what information leakages are extracted and utilized across the network’s layers. Yet, understanding this leakage flow is vital for designing more resilient cryptographic implementations. Existing explainability techniques often do not provide any explanation of the leakages at specific points within a layer’s feature representation.

In this work, we introduce GlimpseDNN, a novel model-agnostic framework designed to provide layer-by-layer visualizations that reveal how DNNs process physical leakage traces to achieve secret key recovery. GlimpseDNN utilizes the stochastic leakage model to provide characterization of the underlying leakage in the intermediate features of each layer. A key advantage of the stochastic leakage model is its dual capability: it can learn complex leakage models and is inherently interpretable through its learned weights. We validate the GlimpseDNN framework on a simulated traces and various public datasets, including ASCADf, ASCADr, ASCADf\_desync50 and ASCADf\_desync100. In particular, we show that even when leakages fail to be captured in the original traces, the GlimpseDNN framework manages to find these leakages by considering the deeper layers of the well-trained DNN. This provides evaluators an alternative to glimpse the underlying leakage for identifying specific implementation vulnerabilities used by DNN, ultimately aiding in the design of more robust, side-channel-resistant hardware.

**Keywords:** Side-Channel Analysis · Neural Network · Explainability · Pearson Correlation · Stochastic Leakage Model

## 1 Introduction

Side-Channel Analysis (SCA) aims to recover secret data by exploiting physical characteristics of a device, such as timing delays [19], power consumption [20],

or electromagnetic emissions [1]. Such analyses are routinely conducted in laboratories to assess the security of various Internet-of-Things (IoT) applications. Typically, SCA is studied under two scenarios: profiled and non-profiled settings. In the profiled setting, an attacker has access to an identical device that can be used to train a distinguisher or model. In contrast, in the non-profiled setting, the attacker depends solely on the physical leakages from the target device, without access to a comparable reference device. In this work, we focus on the profiled setting.

Over the past decade, the usage of Deep Neural Networks (DNNs) in side-channel analysis has become very prominent due to their ability to recover secrets without extensive preprocessing, even in the presence of both hiding and masking countermeasures [15, 8]. This has become a widely studied field known as Deep Learning-based SCA. However, DNNs are predominantly black-box in nature. While we understand the mathematical equations that define their architectures and the training process (e.g., backpropagation), we often do not understand what the underlying weights represent or how the network processes the traces to recover the secret keys. Since DNNs comprise many layers, there is a need to develop methods that explain what type of leakage flows through DNNs from one layer to the next. This enables researchers to better understand how models extract and utilize information from leakage traces.

Perin et al. [23] investigated how shares are processed during training across different layers. Their approach involves training a neural network on the output features of each layer to predict the probability of the shares, which is then used to compute the perceived information of the shares. This provides insight into the information contained in each intermediate feature (also referred to as feature representation). However, this technique considers a layer-wise explainability but does not offer explainability at specific points within the feature representations. Layer-wise explainability can indicate that a layer’s representation carries information about a sensitive variable, but since it condense the entire representation into a single value, it cannot reveal which specific points carry that information. As such, explainability on each points in the feature representation could provide interesting insight.

Therefore, in this work, we propose an alternative method to the previous method that uses classical side-channel technique, the stochastic leakage model [26, 13] to observe how leakage propagates from one layer to the next at each points. While it does not replace the work by [23], this work provides an additional tool for evaluators to understand what leakages the DNNs are using for successful key recovery.

**Contribution.** In this work, we provide an alternative method to gain a glimpse of how leakage flows from one layer to another within a well-trained neural network. This work presents the following key contributions:

- We propose a framework called GlimpseDNN that utilizes the stochastic leakage model that was developed by Guilley et al. [13]. The framework is model-agnostic (i.e. regardless of model’s architecture). Therefore, it can be

executed on any well-trained DNN that successfully recovers the secret key. The GlimpseDNN framework trains a stochastic leakage model of a chosen sensitive variable to characterise the leakage found in the intermediate features of each layer. Subsequently, we apply Pearson Correlation using this leakage model on the intermediate feature of each layer to give insight into the leakage present in the feature representation. Furthermore, the learned weights of the stochastic leakage model offer interpretability, allowing evaluators to identify which bits of the sensitive variable are extracted by the DNN through the layers.

- We confirm the effectiveness of the framework on the dataset simulated traces and the ASCADv1 dataset (ASCADf & ASCADr). We show that the leakage found in the raw traces continues to flows through the DNN and exploited for key recovery in both simulated and realistic setting.
- Subsequently, we show the capability of GlimpseDNN on datasets that are protected by the desynchronization, ASCADf\_desync50 and ASCADf\_desync100. In fact, we show that when leakages failed to be detected in the original traces, the GlimpseDNN framework manages to find leakages by considering the deeper layers of the well-performing DNN. We observe that the well-performing DNN can actually extract leakage even from the first layer. This provides the evaluators with insightful information on what leakages are being exploited by the DNN.

The source code can be accessed publicly at the following link.<sup>3</sup>

**Paper Organization.** We first present related works on explainability of DNN within SCA in Section 2. Then, we will provide the necessary background on SCA and stochastic leakage models in Section 3. Section 4 presents the details of the framework. Subsequently, the experimental results are presented in Section 5. Finally, we conclude the paper in Section 6 and outline potential directions for future work.

## 2 Related Works

We begin by distinguishing between interpretability and explainability [2]. Interpretability refers to models whose decision-making processes are inherently transparent and can be readily understood by humans. For instance, decision trees are considered interpretable because their predictions can be directly traced through explicit decision rules that govern how the data is partitioned [7]. In contrast, explainability involves the use of post hoc techniques to make sense of black-box models such as deep neural networks (DNNs), whose internal decision processes are not naturally accessible or understandable to humans [7, 12].

The interpretability and explainability of DNNs has been explored from various angles over the decades. One direction focuses on interpretable models, such as Truth Table Deep Convolutional Neural Networks, which can convert

<sup>3</sup> [https://github.com/trevor-yap/Glimpse\\_Leakage\\_Flow\\_DLSCA](https://github.com/trevor-yap/Glimpse_Leakage_Flow_DLSCA)

the models into rules or SAT equations for interpretation [27], or Kolmogorov-Arnold Networks (KANs) [29] that employ interpretable activation functions. Recently, [30] proposed another interpretable model that uses the Conditional Variational Autoencoder (cVAE) in which the weights of the stochastic leakage are trained together with the variational autoencoder. [6] further enhances this for optimal feature extraction.

Despite this, evaluators remain interested in understanding what black-box neural networks, such as Multilayer Perceptrons (MLPs) or Convolutional Neural Networks (CNNs), learn that allows them to successfully recover secret keys. To address this, explainability techniques have been developed to identify which features are crucial. For example, Zaid et al. introduced heatmaps to visualize convolutional layers and applied weight visualization to MLPs in order to understand the significance of each feature [31]. Subsequently, Masure et al. investigated gradient visualization across entire DNNs [21], which was further improved with attribution methods [14]. Hettwer et al. [14] also explored techniques such as LRP and 1-Occlusion. Additionally, different occlusion-based methods, including n-occlusion [25] and Key Guessing Occlusion (KGO) [28], were developed to incorporate the attack phase into the explainability process. Although these techniques identify the features that are important to the neural network, they do not explain what leakage are being used by the network to recover the key. We will explain this in more in-depth in Section 5.5.

Recent work has begun to explore this aspect of explainability in more detail. One approach uses Singular Vector Canonical Correlation Analysis (SVCCA) to provide explanations by training two architectures on different datasets and analyzing how their weights are correlated. Their findings show that a DNN trained on an SCA dataset exhibits greater correlation with a DNN trained for image recognition than with a DNN trained on a different SCA dataset with alternative countermeasures. On the other hand, [23] examined how shares are processed during training across different layers by extending the work that uses information bottleneck theory [22]. Their method trains a neural network on the output features of each layer to predict the probability of the shares, which is then used to calculate the perceived information of the shares. This reveals the information contained in each intermediate feature. More recently, Karayalcin et al. applied mechanistic explainability techniques to a well-performing DNN, utilizing Principal Component Analysis (PCA) on activations and logit analysis to reverse-engineer the emergence of features related to secret shares/masking during phase transitions [17]. However, these methods do not provide fine-grained explainability at specific points within each feature representation. Our work is not intended to replace these prior contributions; instead, it offers an alternative perspective on the problem of explaining DNNs used in SCA.

### 3 Background

#### 3.1 Notation & SCA

We represent a random variable using a capital letter  $X$ , and a random vector using a bold capital letter  $\mathbf{X}$ . Their respective realizations are denoted by the corresponding lowercase letters  $x$  and  $\mathbf{x}$ . Let  $x[i]$  stands for the  $i^{th}$  component of the vector  $\mathbf{x}$ . When dealing with an element  $x \in \mathbb{F}_{2^n}$ , we may equivalently write  $\mathbf{x} \in \mathbb{F}_2^n$  to indicate its vector representation. Moreover, we define  $(\mathbf{x}_S)_{S \in \mathbb{F}_2^n} = (\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{2^n})$ .

A cryptographic primitive is denoted as  $f(P, k)$  where  $P$  is a public random variable (e.g., plaintext or ciphertext) and  $k$  is a hypothetical subkey. In fact, we let  $k^*$  be the correct key. A side-channel trace is defined as a vector  $\mathbf{t} = (t[0], \dots, t[D-1]) \in \mathbb{R}^D$  with  $D$  as the number of sample points.

Within SCA, the most common countermeasure is call masking. Masking has been shown to provide protection against side-channel attacks up to a given security order [24]. It operates by splitting a sensitive variable  $Y$  into multiple shares. More formally, a cryptographic primitive is said to use a masking scheme of order  $d$  if  $Y$  is decomposed into  $d+1$  distinct shares,  $Y = \gamma(m_1, m_2, \dots, m_{d+1})$ , where the original value  $Y$  can be reconstructed via a combining operator  $\gamma$ . In this work, we solely consider Boolean masking, where  $\gamma$  corresponds to the bitwise XOR of all shares  $m_i$  for  $i \in \{1, \dots, d+1\}$ .

#### 3.2 Profiling Attack

Profiling attacks represent the most powerful threat scenario in side-channel analysis (SCA). It consists of a profiling and an attack phase. The adversary has two devices in this setup: a prototype (clone) device and a target device. While the adversary either controls or knows the key of the prototype device, the key of the target device remains unknown. During the profiling phase, the adversary builds a distinguisher  $\mathcal{F}$  using profiling traces collected from the prototype device. In the subsequent attack phase, for each attack trace  $\mathbf{t}^{(i)}$  obtained from the target device, a score  $y^{(i)} = \mathcal{F}(\mathbf{t}^{(i)})$  is computed.

For each candidate key  $k$ , the log-likelihood score is then calculated over a fixed number of attack traces  $N_a$   $s_{N_a}(k) = \sum_{i=1}^{N_a} \log(y^{(i)}[z_k^{(i)}])$  where  $z_k^{(i)} = f(p^{(i)}, k)$  denotes the intermediate sensitive variable with  $p^{(i)}$  being the public variable associated with trace  $\mathbf{t}^{(i)}$ . The log-likelihood scores are sorted into a guessing vector  $G = [G_0, G_1, \dots, G_{|\mathcal{K}|-1}]$ , where  $G_0$  corresponds to the most likely key candidate and  $G_{|\mathcal{K}|-1}$  the least likely one. The rank of the correct key  $k^*$  is defined by its position within this ordered list. The guessing entropy (GE) is then obtained as the average rank of the correct key over multiple independent experiments. An attack is deemed successful when  $GE = 0$ , and the minimum number of traces required to achieve this is denoted as  $NTGE$ . In deep learning-based SCA (DLSCA), DNN serves as the distinguisher  $\mathcal{F}$ . The most commonly used DNN architectures in SCA are Multilayer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs). In this work, if the DNN recovers the key

successfully, we would like to analyze and explain how it processes the leakage from one layer to the next.

### 3.3 Pearson correlation & Correlation Power Analysis

We begin by revisiting the Pearson correlation, which forms the basis of Correlation Power Analysis (CPA). The Pearson correlation between two vectors  $\mathbf{x} = (x^{(0)}, \dots, x^{(N-1)})$  and  $\mathbf{y} = (y^{(0)}, \dots, y^{(N-1)})$  is defined as:

$$r(\mathbf{x}, \mathbf{y}) = \frac{\sum_{j=0}^{N-1} ((x^{(j)} - \bar{\mathbf{x}})(y^{(j)} - \bar{\mathbf{y}}))}{\sqrt{\sum_{j=0}^{N-1} (x^{(j)} - \bar{\mathbf{x}})^2} \sqrt{\sum_{j=1}^N (y^{(j)} - \bar{\mathbf{y}})^2}} \quad (1)$$

where  $\bar{\mathbf{x}}$  and  $\bar{\mathbf{y}}$  denotes the mean of  $\mathbf{x}$  and  $\mathbf{y}$  respectively. The Pearson correlation coefficient ranges from  $-1$  to  $1$  (inclusive), which measures the degree of linearity between  $\mathbf{x}$  and  $\mathbf{y}$ . A value  $r(\mathbf{x}, \mathbf{y}) = 1$  implies that  $\mathbf{x}$  and  $\mathbf{y}$  has a perfect positive linear relationship. Conversely, when  $r(\mathbf{x}, \mathbf{y}) = -1$ , it indicates a perfect negative linear relationship.

As for the CPA, the Pearson correlation between each sample point of all the traces and the hypothetical sensitive variable of a chosen leakage model  $\hat{Z}_k$  for each key  $k$  is first computed. Now, let the largest absolute correlation values among the sample points for key  $k$  be  $\rho_k$ . Then, we sort the absolute value  $|\rho_k|$  in descending order. The key corresponding to the largest score  $\rho_{k_{i_0}}$  is the most likely candidate, and the key corresponding to the lowest score is the least likely candidate. Typically, the absolute value of the correlation coefficient is plotted against the sample points to reveal potential leakages within the traces.

To evaluate how the DNN's internal representations relate to physical leakages, we measure the correlation between each point in the intermediate features and a trained stochastic leakage model using the Pearson correlation. Pearson correlation is a well-established statistical tool and is commonly used in side-channel analysis. We will show in the experiments that it can effectively capture leakage present in the internal representations. While alternative measures, such as mutual information [10], could also be considered, we leave their exploration to future work.

### 3.4 Stochastic leakage model

The stochastic attack, introduced as an alternative to template attacks [26, 11, 9], models side-channel leakage through linear regression on functions of the sensitive intermediate variables. In this framework, the leakage is approximated as a linear combination of basis functions derived from the bits of the target variable.

Recent work [13] shows that constructing the stochastic model using an orthonormal Fourier basis, rather than the non-orthogonal monomial basis used in classical approach [26, 11, 9], yields a leakage decomposition that can be estimated accurately and efficiently via a Fourier transform. Motivated by these

results, we adopt a stochastic leakage model based on the Fourier basis. The corresponding model, denoted by  $\hat{\Psi}_{F256}$  is as follows:

$$\hat{\Psi}_{F256}(\mathbf{Y})[j] = \sum_{\mathbf{S} \in \mathbb{F}_2^8} b_{\mathbf{S}} * (-1)^{\mathbf{Y} \cdot \mathbf{S}} = \sum_{\mathbf{S} \in \mathbb{F}_2^8} b_{\mathbf{S}} * \phi_{\mathbf{S}}(\mathbf{Y}) \quad (2)$$

where  $\mathbf{Y} = (Y[0], \dots, Y[7]) \in \mathbb{F}_2^8$  denotes the input vector, and  $\phi_{\mathbf{S}}(\mathbf{x}) = \frac{1}{2^4}(-1)^{\mathbf{x} \cdot \mathbf{S}}$  with  $\mathbf{x} \cdot \mathbf{S} = \sum_{i=0}^7 x[i] * S[i]$ .

During profiling stage, a set of profiling traces and their corresponding sensitive variables are given, i.e.  $\{(\mathbf{t}^{(0)}, \mathbf{y}^{(0)}), \dots, (\mathbf{t}^{(N_p-1)}, \mathbf{y}^{(N_p-1)})\}$ . Given that  $\mathcal{T} = (\mathbf{t}^{(0)}, \dots, \mathbf{t}^{(N_p-1)})^T$  to be the matrix that consists of all the  $N_p$  profiling traces collected. We consider  $\mathcal{T}[j] = (\mathbf{t}^{(0)}[j], \dots, \mathbf{t}^{(N_p-1)}[j])$  to be the  $j^{th}$  sample points for all the traces. The goal is to find a set of trainable parameters  $\mathbf{b}_j = (b_{j,\mathbf{S}})_{\mathbf{S} \in \mathbb{F}_2^8}$  for each sample point  $j \in \{0, \dots, D-1\}$  such that it minimizes the following Mean Square Error [5]:

$$\min_{\mathbf{b}} \|\mathcal{T}[j] - \mathbf{b}_j \cdot \mathcal{Y}^T\|_2 \quad (3)$$

where  $\mathcal{Y} = (\beta(\mathbf{y}^{(0)}), \dots, \beta(\mathbf{y}^{(N_p-1)})) \in \{-1, 1\}^{N_p \times 2^8}$  where  $\beta(\mathbf{Y}) = (\phi_{\mathbf{S}}(\mathbf{Y}))_{\mathbf{S} \in \mathbb{F}_2^8}$ . This can be solved via the ordinary least squares method [5]. We note that Equation 3 applies to a single sample point. Typically, this is generalized to all sample points. The time complexity of building the stochastic model is simply  $O(N_p)$ . We refer readers to [9] for a practical implementation of the ordinary least squares method applied across all sample points to obtain the stochastic leakage model.

## 4 GlimpseDNN Framework

Now, we have all the building blocks required for the GlimpseDNN framework. In this section, we will explain the proposed GlimpseDNN framework in a step-by-step manner. The overall structure of the GlimpseDNN Framework is illustrated in Figure 1. The GlimpseDNN consists of four steps.

1. As a first step, in order to explain anything, we need a trained DNN that successfully recovers the secret key as stated in Section 3.2. Therefore, we train the DNN with the profiling traces. Formally, we let the well-trained DNN be defined as

$$\mathcal{F}_{\Theta}(\mathbf{t}) = \mathcal{F}_{\ell} \circ \dots \circ \mathcal{F}_2 \circ \mathcal{F}_1(\mathbf{t}) \quad (4)$$

where  $\ell$  is the number of layers and  $\mathcal{F}_i : \mathbb{R}^{D_{i-1}} \rightarrow \mathbb{R}^{D_i}$  with  $D_i$  is the dimension of the output from the  $i^{th}$  layer for  $\mathcal{F}_{\Theta}$ . We set  $D_0$  to be the input dimension and define  $D_{\ell}$  as the number of output classes associated with the hypothetical sensitive variable used for key recovery. We highlight here that GlimpseDNN is model-agnostic. This means it can be applied to any DNN architecture from MLP, CNN to even Transformers. But in this work, we shall focus mainly on MLPs and CNNs which are the most common DNN used in side-channel.

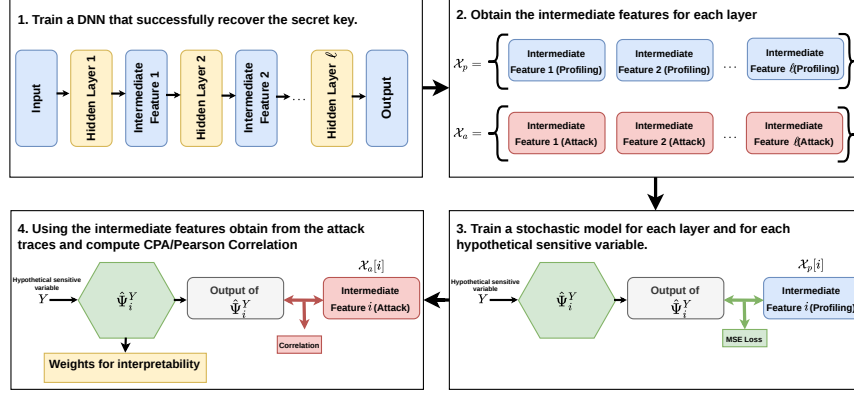


Fig. 1: GlimpseDNN Framework

2. In the second step, we extract the intermediate features from each layer of the trained DNN by applying both profiling and attack traces. We denote  $\mathcal{X}_p$  as the collection of intermediate features derived from the profiling traces, and  $\mathcal{X}_a$  as the intermediate features obtained from the attack traces. Formally, for each layer  $i \in \{1, \dots, \ell\}$ , we set

$$\mathcal{X}_p[i][j] = \mathcal{F}_i \circ \mathcal{F}_{i-1} \circ \dots \circ \mathcal{F}_1(t_p^j) \quad (5)$$

$$\mathcal{X}_a[i][j] = \mathcal{F}_i \circ \mathcal{F}_{i-1} \circ \dots \circ \mathcal{F}_1(t_a^j) \quad (6)$$

where  $\{t_p^j\}_{j=1}^{N_p}$  is the set of profiling traces and  $\{t_a^j\}_{j=1}^{N_a}$  is the set of attack traces.

3. For each  $i^{th}$  intermediate feature, we train a stochastic leakage model that takes the hypothetical sensitive variable  $Y$  as input, which we denote by  $\mathcal{F}_i^Y$ . Suppose we let the  $i^{th}$  intermediate features be denoted as  $\mathbf{x}_{\text{int}} \in \mathbb{R}^{D_i}$ . The underlying assumption of GlimpseDNN is that each point of  $\mathbf{x}_{\text{int}}$  is modeled as

$$\mathbf{x}_{\text{int}}[j] = \begin{cases} \Psi_{i,j}(Y) + \epsilon & \text{if } j \in \{j_0, \dots, j_{m-1}\}, \\ \epsilon & \text{otherwise,} \end{cases} \quad (7)$$

where  $\{j_0, \dots, j_{m-1}\}$  are the indices at which sensitive information  $Y$  leaks, with  $\Psi_{i,j} : \mathbb{F}_2^n \rightarrow \mathbb{R}$  being the true deterministic leakage model for the  $j^{th}$  index of  $i^{th}$  intermediate feature and  $\epsilon$  represents additive Gaussian noise. The goal of training stochastic leakage model  $\mathcal{F}_i^Y$  is to approximate  $\Psi_{i,j}$  from  $\mathbf{x}_{\text{int}}$  effectively capturing the deterministic leakage of the sensitive variable  $Y$ . Therefore, the selection of the hypothetical sensitive variable is crucial in the GlimpseDNN framework.

4. In the last step, we compute the Pearson correlation between  $\mathcal{F}_i^Y$  and the intermediate feature  $\mathcal{X}_a[i]$ . If the hypothetical sensitive variable  $Y = Y_K$  depends on the key, e.g.,  $Y_K = \text{Sbox}(P \oplus K)$ , one can perform a CPA instead

Table 1: Hyperparameter search space for profiling attack.

| Hyperparameter         | Options                                        |
|------------------------|------------------------------------------------|
| <b>MLP</b>             |                                                |
| Number of Dense Layers | 1 to 10 in a step of 1                         |
| Neurons per layer      | 10, 20, 50, 100, 200, 300, 400, 500            |
| <b>CNN</b>             |                                                |
| Convolution layers     | 1 to 4 in step of 1                            |
| Convolution filters    | 4 to 16 in step of 4                           |
| Kernel size            | 26 to 52 in step of 2                          |
| Pooling type           | Average or Max                                 |
| Pooling size           | 2 to 10 in step of 2                           |
| Number of Dense Layers | 1 to 10 in a step of 1                         |
| Neurons per layer      | 10, 20, 50, 100, 200, 300, 400, 500            |
| <b>Others</b>          |                                                |
| Batch size             | 100 to 1100 in a step of 100                   |
| Activation function    | <i>ReLU</i> or <i>SeLU</i>                     |
| Optimizer              | Adam or RMSprop                                |
| Learning Rate          | 0.0005, 0.0001, $1e^{-4}$ , $5e^{-4}$          |
| Weight Initializer     | Random Uniform or Glorot Uniform or He Uniform |
| Epochs                 | 50 to 500 in a step of 50                      |

of merely computing the Pearson correlation. The resulting correlation values can then be plotted to identify which components of the intermediate feature exhibit dependence on the hypothetical sensitive variable.

Furthermore, since the stochastic leakage model admits a closed-form expression, it is possible to explicitly characterize the underlying leakage structure to which the intermediate feature is strongly related. This characterization can be obtained from the model weights  $\mathbf{S}$ , namely  $b_{\mathbf{S}}$  as shown in Equation 2.

We note that, for convolutional layers, all channels are flattened into a one-dimensional feature vector for Steps 3 and 4 of the GlimpseDNN framework.

## 5 Experiments

### 5.1 Experimental Settings

#### *Hyperparameters search for the trained MLP/CNN for key recovery.*

For all the datasets, we randomly sampled 100 models from Table 1 and selected the network with the best *NTGE*. Although similar results were observed for the Identity leakage model, key recovery was unsuccessful for datasets such as ASCADr and ASCADf\_desync50. Therefore, in this work, we focus on the Hamming weight leakage model for the profiling attack. The chosen neural network is then studied to examine how leakage information propagates across its layers. Instead of emphasizing state-of-the-art models, we apply GlimpseDNN on a randomly sampled well-trained model to show that it is model-agnostic and applicable across different architectures.

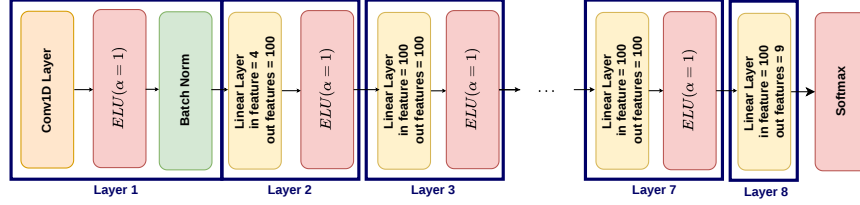


Fig. 2: Trained CNN architecture for key recovery of simulated traces dataset.

## 5.2 Simulated Traces

We generate a set of simulated traces for masking order 1, where the shares are defined as  $s_1 = Sbox(pt \oplus k^*) \oplus s_2$ , and  $s_2$  is sampled uniformly at random. Each simulated trace contains 20 sample points. Let  $t$  denote a trace. For each trace, we generate uniformly random byte values for all sample points, except that  $t[5] = HW(s_1)$  and  $t[10] = HW(s_2)$ . Using the hyperparameter search stated above, we obtain a CNN that successfully recovers the secret key with  $NTGE = 12$ . The overall architecture is illustrated in Figure 2. The convolutional layer consists of four output channels. For each kernel, the size and stride are both 42, with a padding of 16.

Next, we apply the Glimpse DNN framework to understand what are the leakages within each intermediate features. We observe that the CNN begins extracting leakage related to  $Sbox(pt \oplus k^*)$  starting from the second layer. This leakage progressively intensifies and ultimately becomes the dominant component of the intermediate feature representations in layer 8 onwards. This highlights that the DNN can extract the necessary information required for key recovery.

For completeness, we present the stochastic weights obtained from layer 9 to illustrate the type of insights provided by GlimpseDNN. Figure 4 shows the corresponding weights for this layer. The values of  $S$  are sorted in ascending order, following the approach in [13], which facilitates the interpretation of the most significant bit interactions.

To focus on the most relevant components, we select sample point 2 for both  $s_1$  and  $s_2$ , and sample point 3 for  $Sbox(pt \oplus k^*)$ , as these exhibit the highest correlations. Notably, for  $Sbox(pt \oplus k^*)$ , the stochastic leakage assigns the largest weights to the basis with  $HW = 8$ . This indicates that the DNN leverages all bits of  $Sbox(pt \oplus k^*)$  in the key recovery process. On the other hand, we observe that the DNN captures pairwise interactions between  $s_1$  and  $s_2$ , suggesting that joint leakage of these components are also exploited during feature extraction. While these observations are insightful, the broader utility of interpreting stochastic models remains an open question and is left for future work.

This localization is what distinguishes GlimpseDNN from prior layer-wise approaches such as that of Perin et al. [23], who train a DNN on the entire output of a layer to estimate a single perceived information score per sensitive variable. Such a score can establish that a layer’s representation is informative

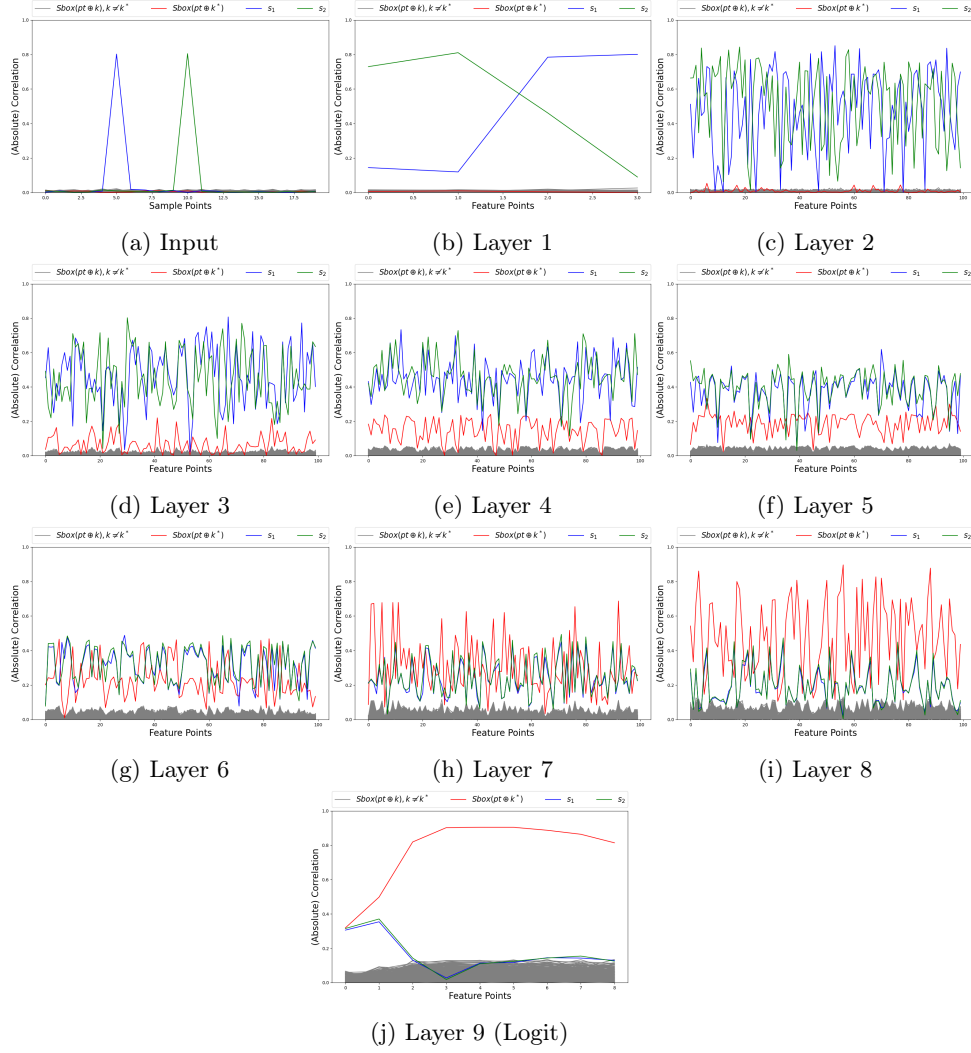


Fig. 3: GlimpseDNN on simulated traces.

about a share or the target variable, but because it considers the entire representation as a whole, it cannot indicate which coordinates carry that information or what function of the variable they encode. But GlimpseDNN's provide these information. In the simulated-trace experiment, for instance, sample point 2 of layer 9 is found to correlate with both mask shares  $s_1$  and  $s_2$ , identifying this coordinate as the location at which the shares are combined, while the recovered stochastic weights show a dominant pairwise interaction term between  $s_1$  and  $s_2$  rather than independent contributions from each, indicating the specific functional form the network has learned. A layer-wise perceived information score

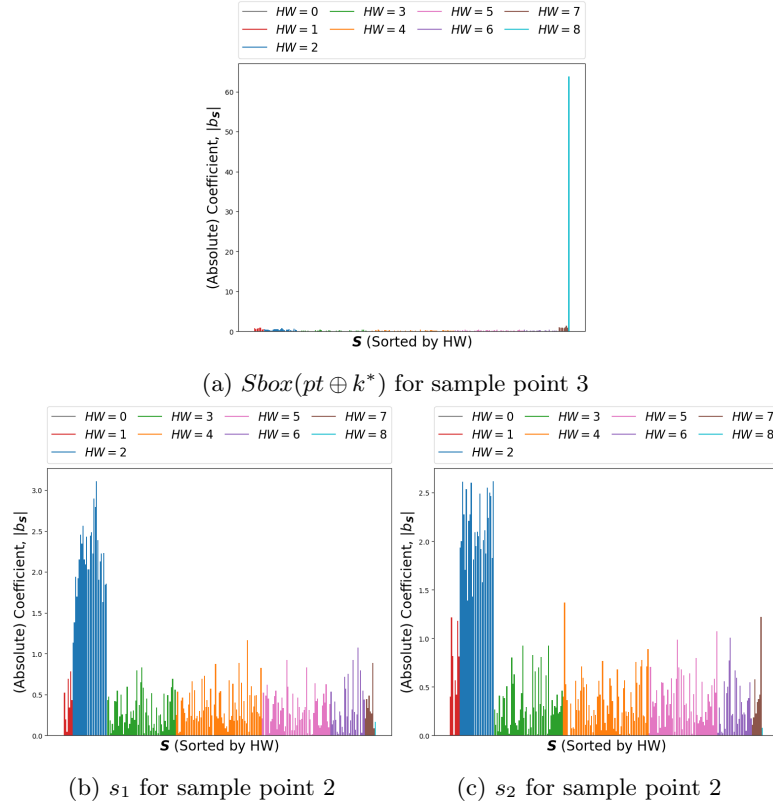


Fig. 4: Stochastic Weights for Layer 9

could report that layer 9 carries information about  $s_1$ ,  $s_2$ , and  $Sbox(pt \oplus k^*)$ ; it could not identify where within the layer this information resides or how the network combines it.

Overall, these results show that GlimpseDNN is able to capture the leakage being processed by the DNN in the simulated setting. In particular, the framework successfully tracks how information related to the sensitive variables is encoded and transformed across intermediate representations, even when the leakage is synthetically generated under controlled noise and modeling assumptions.

### 5.3 Real Measurement

The ASCADv1 dataset contains traces from a first-order masked AES implementation on an 8-bit AVR microcontroller (ATMega8515) [4], targeting the third byte of the first-round AES Sbox. It is provided in two versions: ASCADf and ASCADr. In ASCADf, both profiling and attack traces correspond to a fixed key, whereas in ASCADr, the profiling traces are generated with random keys

Table 2: Masked intermediate values

|                   |                                             |
|-------------------|---------------------------------------------|
| $m_1 := r,$       | $m_2 := Sbox(pt \oplus k^*) \oplus r$       |
| $m_3 := r_{out},$ | $m_4 := Sbox(pt \oplus k^*) \oplus r_{out}$ |
| $m_5 := r_{in},$  | $m_6 := Sbox(pt \oplus k^*) \oplus r_{in}$  |

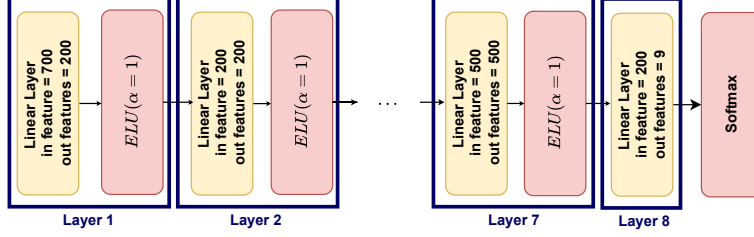


Fig. 5: Trained MLP architecture for key recovery of ASCADf dataset.

and the attack traces correspond to a fixed key target device. Each version we use 45,000 profiling traces and 10,000 attack traces. The traces in ASCADf contain 700 sample points, while ASCADr traces contain 1,400 sample points. We denote the intermediate value as  $Sbox(pt \oplus k^*)$ , where  $pt$  is the third byte of the plaintext and  $k^*$  is the third byte of the first-round key. Various masks contribute to the leakage, as summarized in Table 2. Since the learned stochastic model weights exhibit a similar structure to those observed in the simulated traces, we omit their presentation here. Therefore, we will only present the result from Pearson Correlation/CPA.

**GlimpseDNN on ASCADf** For ASCADf, we train an MLP to successfully recover the secret key with a NTGE = 1768. The MLP architecture used for ASCADf is shown in Figure 5. It comprises eight layers in total: the first seven layers each consist of a linear transformation followed by an *ELU* activation function, while the final layer is a linear output layer with a softmax activation function.

Next, we apply our GlimpseDNN method to interpret the trained MLP. We observe that as we progress through the layers, the correlations associated with the different masks  $m_1$  to  $m_6$  gradually decrease, while the correlation with  $Sbox(pt \oplus k^*)$  increases (red line). Furthermore, as the network depth increases, the model gradually learns to distinguish the correct  $Sbox(pt \oplus k^*)$  from those corresponding to incorrect keys (compare red line with grey lines). This trend is consistent with the findings reported in [23], where the perceived information between the network’s intermediate features and the masks diminishes, whereas the perceived information with  $Sbox(pt \oplus k^*)$  increases across deeper layers.

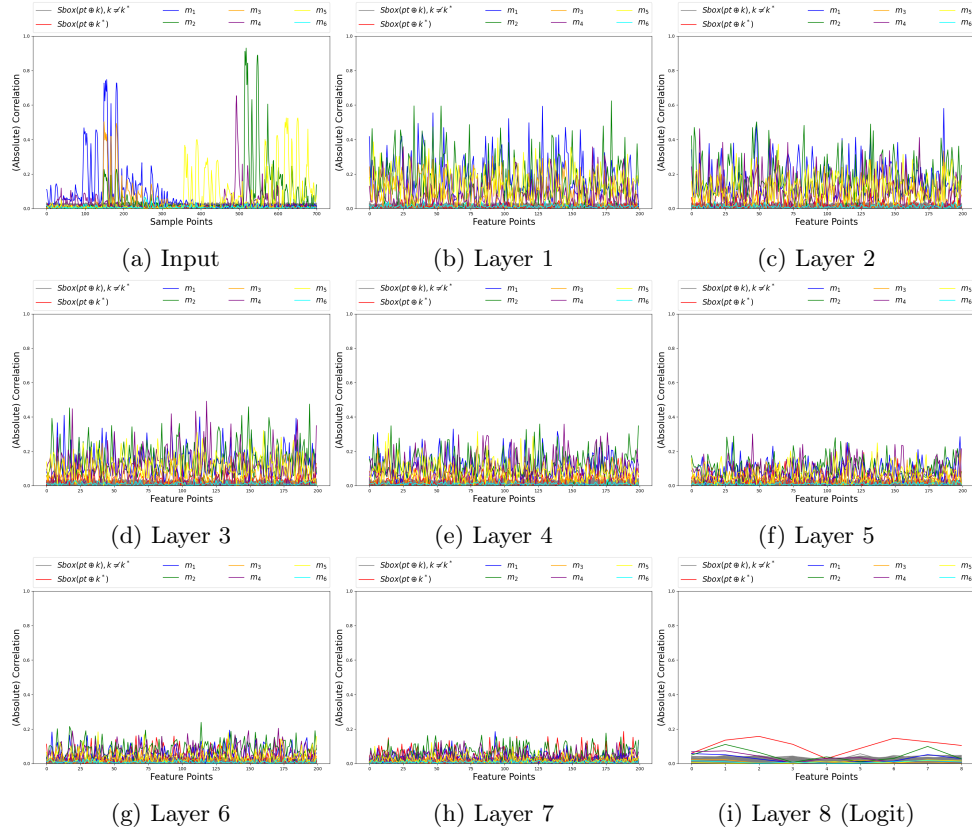


Fig. 6: GlimpseDNN on ASCADf

*Receptive field.* In addition, we evaluate GlimpseDNN on a state-of-the-art CNN proposed in [31]. The model consists of a convolutional layer with four kernels of size 1, followed by an average pooling layer with kernel size 2 and stride 2. The output is then flattened and passed through two dense layers with 10 neurons each.

Figure 7 presents the results of applying GlimpseDNN to this CNN model. The observations are consistent with those discussed above. Notably, since the first convolutional layer contains four kernels of size 1, the leakage is preserved across channels. After flattening, this we observed a repeated pattern appearing four times similar to those in the input. This observation highlights that, in CNNs, receptive fields [3] can help identify which parts of the input traces correspond to the leakage revealed by GlimpseDNN.

**GlimpseDNN on ASCADr** For the ASCADr dataset, we train a CNN that successfully recovers the secret key with  $NTGE = 5631$ . Figure 8 shows the architecture of the CNN used for ASCADr. It consists of three convolutional

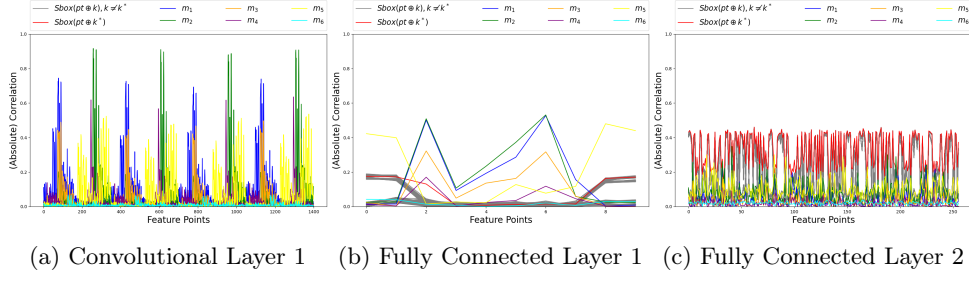


Fig. 7: GlimpseDNN on ASCAdf using state-of-art model

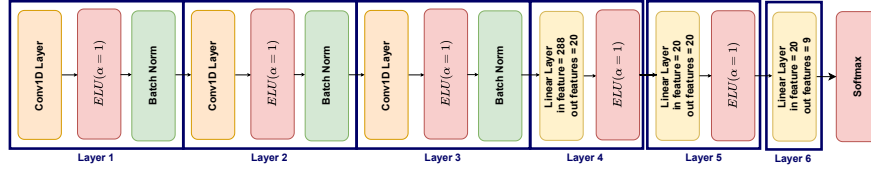


Fig. 8: Trained CNN architecture for key recovery of ASCAdr dataset.

blocks, each comprising a 1D convolution layer followed by an ELU activation and batch normalization. These are followed by two fully connected layers, each composed of a linear transformation and an ELU activation function. The final layer is a linear output layer with a softmax activation. The CNN has 8 output channels with the kernel sizes and stride of 52 and a padding of 4.

As before, we apply GlimpseDNN to examine how leakage propagates from one layer to the next within the trained CNN. However, in Layer 3, several intermediate feature values are constant (also known as dead neurons), preventing correlation computation for these features. Therefore, we remove these intermediate features before applying the Pearson correlation. Figure 9 shows the results of the GlimpseDNN for each layer of the neural network. We observe that the first layer processes the traces, distributing the leakage across multiple distinct regions of the intermediate feature instead of concentrating it in a single area. Subsequently, Layers 2 to 5 further compress the traces, enhancing the leakages of the masks  $m_1, m_2, m_3, m_4$ , and  $m_5$  as highlighted by their high correlation values. But the correlation of  $Sbox(pt \oplus k^*)$  did not come as dominant. As such, here the trend reported in in [23] followed is partial, as only the leakage of the mask are compressed. This indicates that the trained CNN is able to distinguish the correct key from incorrect keys either by combining the shares found in layer 6 or the stochastic leakage model fails to model the  $Sbox(pt \oplus k^*)$  perfectly. Nevertheless, we observe that GlimpseDNN is able to provide insights into how leakage is represented and transformed throughout the network which previously lacks.

For ASCAdf and ASCAdr, we shows that GlimpseDNN can capture the underlying leakage in the intermediate feature of the DNN to have a glimpse of

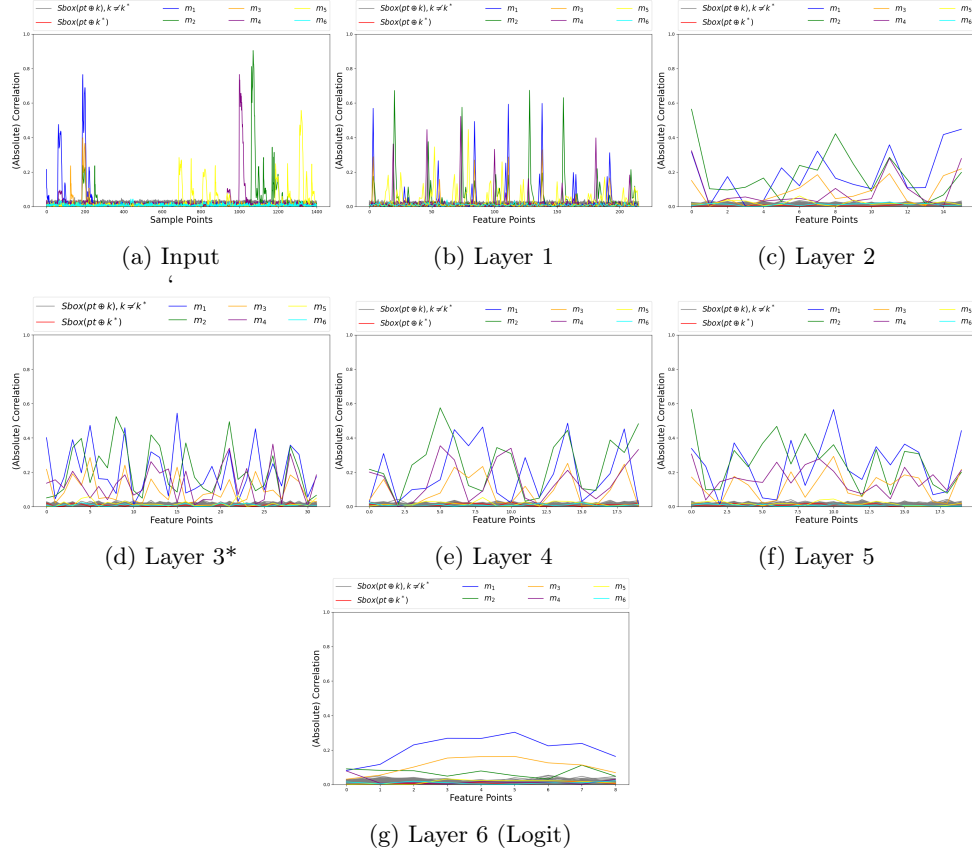


Fig. 9: GlimpseDNN on ASCADr

what the DNN is extracting for key recovery since the correlation of the shares are always prominent in the intermediate features across the layers. This indicates that GlimpseDNN remains effective not only simulated setting, but also in on real or realistic traces where the underlying leakage is much more complex.

#### 5.4 Desynchronization datasets

Next, we consider artificially desynchronized variants of the ASCADf dataset, where desynchronization is applied within windows of 50 and 100 sample points, referred to as ASCADf\_desync50 and ASCADf\_desync100, respectively. We also construct corresponding variants for ASCADr, namely ASCADr\_desync50 and ASCADr\_desync100. However, under our hyperparameter search, we do not obtain any successful DNN models on the ASCADr desynchronized variants. Therefore, we focus our analysis on ASCADf\_desync50 and ASCADf\_desync100. It has been pointed out by [16] that these two ASCADf variants present a more

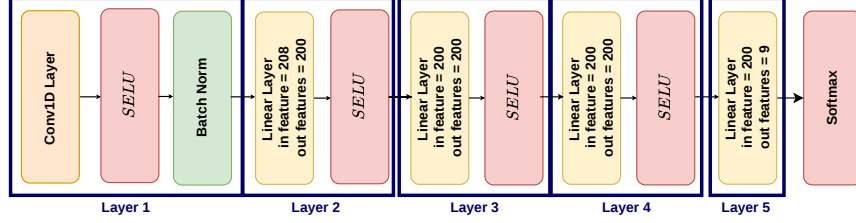


Fig.10: Trained CNN architecture for key recovery of ASCADf\_desync50 dataset.

challenging setting for DLSCA compared to the baseline dataset. We show that the advantages of the GlimpseDNN framework become particularly evident under these settings.

**ASCADf\_desync50** A CNN is trained to successfully recover the secret key for the ASCADf\_desync50 dataset. This CNN can recover the secret key with NTGE = 3070. Figure 10 provides the CNN architecture used for ASCADf\_desync50. It consists of convolutional blocks where each comprises a 1D convolution layer followed by an *SELU* activation and batch normalization. Then the features are flattened and processed with three fully connected layers, each composed of a linear transformation and an *SELU* activation function with the final layer is a linear output layer with a softmax activation. The 1D convolution layer has 16 output channels, and kernels with size of 52 and padding of 12. The strides of the kernel is same as its size.

We interpret the trained CNN using our GlimpseDNN framework to examine how the leakages are processed across layers. This is depicted in Figure 11. We observe that, due to desynchronization, the Pearson correlation using the stochastic leakage model fails to inform the evaluator which leakage are leaking (see Figure 11a). However, after the first layer of the trained CNN, the correlations for  $m_1, m_2, m_3, m_4$  and  $m_5$  increase significantly. The leakages observed in the intermediate features are consistent with those in ASCADf before any desynchronization was introduced (compare with Figure 6a in Section 5.3). This shows that within the first layer, the neural network is already dealing with the desynchronization countermeasure.

The subsequent layers, layer 2 to 3, seem to be processing the leakage and amplifying the masks further. Then, for layer 4, the leakage of the masks starts to decrease while the  $Sbox(pt \oplus k^*)$  leakage starts to increase. For the last layer, Layer 5, the leakage of  $Sbox(pt \oplus k^*)$  is slightly amplified for the distinguisher to fully distinguish the correct key. Compared to simulated traces, attacks on real measurements are significantly more difficult, as the leakage of  $Sbox(pt \oplus k^*)$  is no longer dominant as it was in the simulated setting.

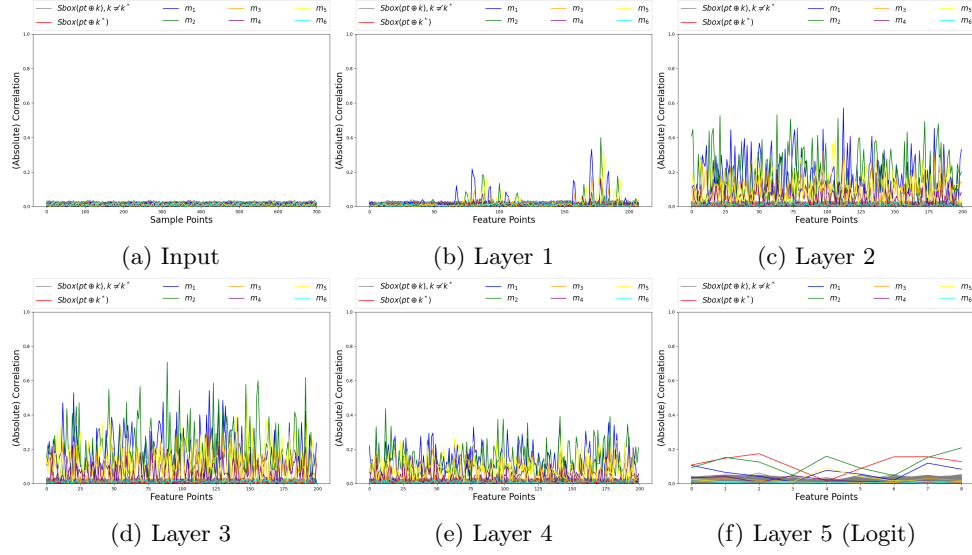


Fig. 11: GlimpseDNN on ASCADf\_desync50

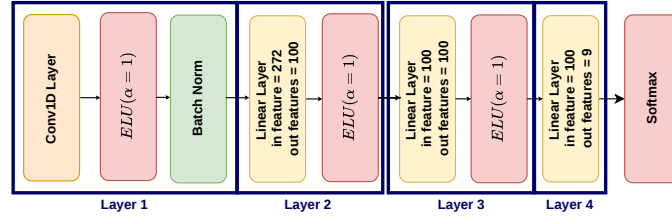


Fig. 12: Trained CNN architecture for key recovery of ASCADf\_desync100 dataset.

**ASCADf\_desync100** A CNN is trained to successfully recover the secret key for the ASCADf\_desync100 dataset. The proposed CNN architecture is composed of a single convolutional block followed by fully connected layers. The convolutional block consists of a 1D convolutional layer with 16 filters, a kernel size of 42, stride 42, and padding 8, followed by an ELU activation function and batch normalization. The overall CNN architecture is illustrated in Figure 12.

The extracted features are then flattened and passed through two fully connected layers, each followed by an ELU activation function. Finally, a linear output layer with 9 neurons is used, followed by a softmax activation to produce the probability distribution over the key classes. This architecture managed to recover the secret key with  $NTGE = 5477$  under higher desynchronization conditions.

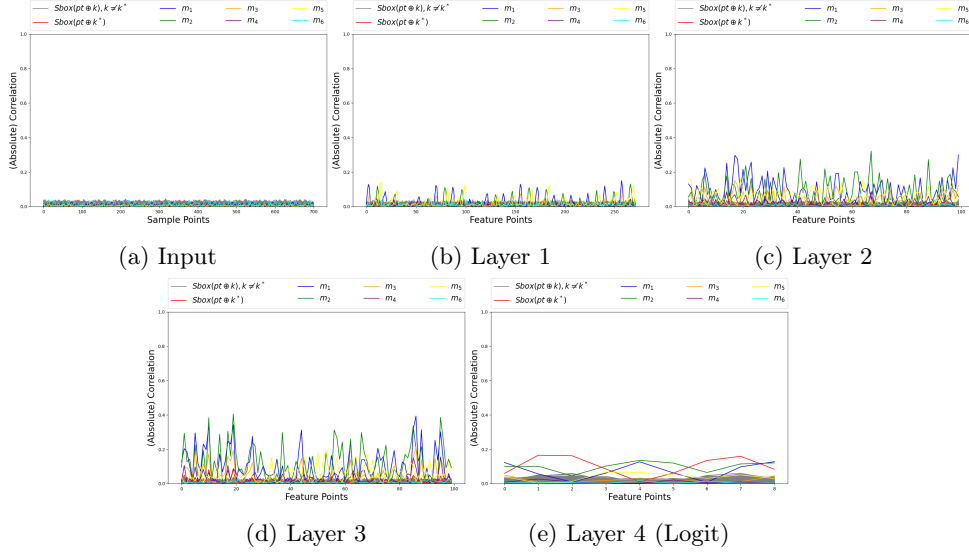


Fig. 13: GlimpseDNN on ASCADf\_desync100

We apply the GlimpseDNN framework to a well-trained CNN. As illustrated in Figure 13a, the stochastic leakage model alone is unable to recover any leakage directly from the raw input traces. However, because the CNN itself is capable of successful key recovery, applying GlimpseDNN allows us to observe how the hidden leakages are progressively extracted across the layers. After the first layer, the leakages associated with shares  $m_1$  to  $m_5$  become noticeably more prominent in the intermediate features, suggesting that the network effectively denoises the signal. From layer 2 to layer 3, these leakages become highly pronounced. Notably, starting at layer 3, the leakage corresponding to the target sensitive variable,  $Sbox(pt \oplus k^*)$ , begins to distinguish itself from the incorrect key hypotheses. Finally, by layer 4, the DNN successfully combines the shares and extracts the  $Sbox(pt \oplus k^*)$  leakage to achieve key recovery. Here, we do not see the trend of compression of the mask but there is an increase in the leakage of the target sensitive variable.

### 5.5 Discussion

Previously, when a DNN succeeds in an attack, the evaluator may not have insight into the specific leakage being exploited by the model. Prior work, such as gradient visualization [21], can highlight regions of the trace that contribute to key recovery, but it does not provide direct information about the underlying leakage extracted by the DNN. Similarly, other attribution-based techniques [14], Key Guessing Occlusion (KGO) [28], and  $n$ -occlusion [25] primarily assign importance scores to sample points, and therefore share the same limitation. We illustrate this by plotting the gradient visualizations and  $n$ -occlusion of the CNN

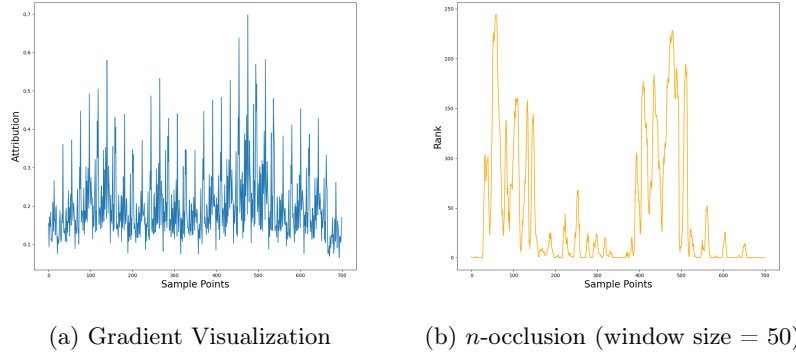


Fig. 14: Prior Explainability Techniques of the well-trained CNN from ASCADf\_desync100.

trained on ASCADf\_desync100 in Figure 14. For gradient visualizations, we can roughly observe two peaks, but  $n$ -occlusion the peaks are very prominent. These peaks may correspond to the two mask-dependent leakage points. However, these methods can only provide importance values at the level of sample points, without directly characterizing the nature of the leakage itself.

Unlike those prior works, the GlimpseDNN framework provides an additional tool for evaluators to better understand what leakage is being extracted. This is shown from the results on ASCADf\_desync50 and ASCADf\_desync100. In particular, these results reveals that leakage which were not recoverable using stochastic leakage modeling in the original trace domain can, in fact, be successfully captured in the deeper layers of a well-trained DNN through the GlimpseDNN framework.

Although there are no direct temporal relationship between intermediate representation of deeper layers and the original trace, the practical value of GlimpseDNN’s point-wise analysis at this depth is instead mechanistic rather than physical. Specifically, it characterizes how the network internally represents, transforms, and combines leakage information after countermeasures such as masking or desynchronization have already been processed, an aspect that remains largely unexplored in prior work.

We further note an interesting, though not universal, trend across our results: the leakage of the mask shares is first extracted and amplified before being compressed in later layers, while the leakage of the target sensitive variable correspondingly increases with depth. This is broadly related to the compression behavior observed by Perin et al. [23], where the perceived information of irrelevant features diminishes while that of the target sensitive variable increases across deeper layers. However, this trend does not hold uniformly across all our results. For ASCADr, the mask correlations remain enhanced through most of the network and the target variable correlation never becomes clearly dominant.

For ASCADf\_desync100, we observe a different deviation: the leakage of the masks is amplified but does not undergo the expected compression in later layers, even though the leakage of the target sensitive variable still increases and is ultimately extracted by the network.

Both GlimpseDNN and [23] answer complementary questions: perceived information measures *how much* information about a share or the target variable a layer contains, whereas GlimpseDNN determines *where* within the layer this information resides and *what function* of the variable is encoded there. Concretely, GlimpseDNN provides finer-grained explainability than [23], by revealing the flow of leakage in each intermediate feature point’s by modeling these points with stochastic leakage models. These stochastic leakage models offer exact closed-form expressions per point. Both approaches require access to the sensitive variable and the corresponding shares for explainability, but finer granularity of GlimpseDNN comes at the cost of stronger assumptions, namely a Gaussian noise model at each point and an assumption of linear dependence between variables.

It should be noted that the GlimpseDNN framework is not without limitations, it dependent on generalization power of stochastic leakage modeling and also relies on Pearson correlation, which captures only linear dependencies between intermediate features and the stochastic leakage model. This could be some reasons why the trend is different from those observed in [23]. Nonetheless, our experimental results indicate that Pearson correlation is useful for an evaluator to detect leakage in practice. A promising direction for future work is to explore non-linear dependency measures, such as mutual information, to capture more complex relationships on each specific point within representations.

Moreover, GlimpseDNN assumes the success of an attack under the DLSCA setting and also requires the evaluator’s domain knowledge to select which sensitive variable should be considered in the analysis. Currently, GlimpseDNN is primarily designed as an post-hoc explainability tool; accordingly, our analysis starts from models that successfully recover the key. We also examined models for which key recovery was unsuccessful, but did not observe any meaningful patterns. Therefore, our claims are restricted to models that achieve successful key recovery. Adapting GlimpseDNN to untrained networks, or in other words, to models not capable of recovering a key, is an interesting future direction.

Furthermore, while the stochastic leakage model provides weights that enable evaluators to analyze the learned representations, how these insights can be leveraged to improve the design of the underlying implementation remains an open problem. In [18], the authors demonstrated that specific bits of a sensitive variable can contribute disproportionately to the overall leakage of an FPGA when analyzed using a stochastic model. An interesting direction for future work is to investigate whether the stochastic leakage patterns identified in the deeper layers of the DNN can be translated back to the physical device, and whether such insights can inform more secure hardware or implementation-level countermeasures.

## 6 Conclusion

In this work, we introduced GlimpseDNN, a model-agnostic framework designed to provide layer-wise explainability of deep learning models in profiled side-channel analysis. While deep neural networks have demonstrated strong performance in recovering secret keys even under challenging countermeasures, their black-box nature limits our understanding of how leakage information is progressively extracted across layers. Existing explainability techniques typically highlight important sample points but do not characterize the nature of the underlying leakage.

GlimpseDNN addresses this limitation by integrating stochastic leakage modeling with intermediate feature analysis, enabling a structured view of how leakage evolves within a trained network. By training stochastic leakage models on intermediate representations and analyzing their correlation with learned features, we provide a method to identify what type of leakage is being extracted and exploited by the DNN. Furthermore, this also allows a better understanding of the bits interaction of the sensitive variable that the DNN extracted and exploited through the learned stochastic weights.

Our experiments on simulated traces, ASCADf, ASCADr, and their desynchronized variants demonstrate that GlimpseDNN can reveal meaningful leakage even in scenarios where the raw traces do not exhibit clear correlation of the leakage. In particular, we show that well-trained CNNs are capable of internally desynchronized traces and extract leakage of sensitive information. These results highlight that neural networks actively transform and extract hidden leakage representations for key recovery.

Future work will investigate whether mutual information can more effectively capture non-linear dependencies on each specific point within representations in GlimpseDNN, and how the weights of the stochastic leakage model extracted from deeper layers can be mapped back to physical implementations, with the goal of informing the design of more robust cryptographic hardware.

## References

1. Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P.: The EM Side-Channel(s). In: Jr., B.S.K., Koç, Ç.K., Paar, C. (eds.) *Cryptographic Hardware and Embedded Systems - CHES 2002*, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers. *Lecture Notes in Computer Science*, vol. 2523, pp. 29–45. Springer (2002). [https://doi.org/10.1007/3-540-36400-5\\_4](https://doi.org/10.1007/3-540-36400-5_4), [https://doi.org/10.1007/3-540-36400-5\\_4](https://doi.org/10.1007/3-540-36400-5_4)
2. Amazon: Model Explainability with AWS Artificial Intelligence and Machine Learning Solutions (sep 2021), <https://docs.aws.amazon.com/whitepapers/latest/model-explainability-aws-ai-ml/interpretability-versus-explainability.html>
3. Araujo, A., Norris, W., Sim, J.: Computing Receptive Fields of Convolutional Neural Networks. *Distill* (2019). <https://doi.org/10.23915/distill.00021>, <https://distill.pub/2019/computing-receptive-fields>

4. Benadjila, R., Prouff, E., Strullu, R., Cagli, E., Dumas, C.: Deep learning for side-channel analysis and introduction to ASCAD database. *J. Cryptogr. Eng.* **10**(2), 163–188 (2020). <https://doi.org/10.1007/s13389-019-00220-8>, <https://doi.org/10.1007/s13389-019-00220-8>
5. Bishop, C.M.: Pattern recognition and machine learning, 5th Edition. Information science and statistics, Springer (2007), <https://www.worldcat.org/oclc/71008143>
6. Boussam, S., Carbone, M., Gérard, B., Renault, G., Zaid, G.: Optimal Dimensionality Reduction using Conditional Variational AutoEncoder. *IACR Cryptol. ePrint Arch.* p. 1134 (2025), <https://eprint.iacr.org/2025/1134>
7. Burkart, N., Huber, M.F.: A Survey on the Explainability of Supervised Machine Learning. *Journal of Artificial Intelligence Research* **70**, 245–317 (jan 2021)
8. Cagli, E., Dumas, C., Prouff, E.: Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures - Profiling Attacks Without Pre-processing. 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), [https://doi.org/10.1007/978-3-319-66787-4\\_3](https://doi.org/10.1007/978-3-319-66787-4_3)
9. Choudary, M.O., Kuhn, M.G.: Efficient Stochastic Methods: Profiled Attacks Beyond 8 Bits. In: Joye, M., Moradi, A. (eds.) *Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers. Lecture Notes in Computer Science*, vol. 8968, pp. 85–103. Springer (2014). [https://doi.org/10.1007/978-3-319-16763-3\\_6](https://doi.org/10.1007/978-3-319-16763-3_6), [https://doi.org/10.1007/978-3-319-16763-3\\_6](https://doi.org/10.1007/978-3-319-16763-3_6)
10. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual Information Analysis: A Generic Side-Channel Distinguisher. In: *Cryptographic Hardware and Embedded Systems – CHES 2008*. pp. 426–442. Springer (2008)
11. Gierlichs, B., Lemke-Rust, K., Paar, C.: Templates vs. Stochastic Methods. In: Goubin, L., Matsui, M. (eds.) *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings. Lecture Notes in Computer Science*, vol. 4249, pp. 15–29. Springer (2006). [https://doi.org/10.1007/11894063\\_2](https://doi.org/10.1007/11894063_2), [https://doi.org/10.1007/11894063\\_2](https://doi.org/10.1007/11894063_2)
12. Gilpin, L.H., Bau, D., Yuan, B.Z., Bajwa, A., Specter, M., Kagal, L.: Explaining explanations: An overview of interpretability of machine learning. In: *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*. pp. 80–89. IEEE (2018)
13. Guilley, S., Heuser, A., Tang, M., Rioul, O.: Stochastic Side-Channel Leakage Analysis via Orthonormal Decomposition. In: Farshim, P., Simion, E. (eds.) *Innovative Security Solutions for Information Technology and Communications - 10th International Conference, SecITC 2017, Bucharest, Romania, June 8-9, 2017, Revised Selected Papers. Lecture Notes in Computer Science*, vol. 10543, pp. 12–27. Springer (2017). [https://doi.org/10.1007/978-3-319-69284-5\\_2](https://doi.org/10.1007/978-3-319-69284-5_2), [https://doi.org/10.1007/978-3-319-69284-5\\_2](https://doi.org/10.1007/978-3-319-69284-5_2)
14. 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), [https://doi.org/10.1007/978-3-030-38471-5\\_26](https://doi.org/10.1007/978-3-030-38471-5_26)
15. Heuser, A., Zohner, M.: Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines. In: Schindler, W., Huss, S.A. (eds.) *Constructive Side-Channel Analysis and Secure Design - Third International Workshop, COSADE 2012, Darmstadt, Germany, May 3-4, 2012. Proceedings. Lecture Notes in Computer Science*, vol. 7275, pp. 249–264. Springer (2012). [https://doi.org/10.1007/978-3-642-29912-4\\_18](https://doi.org/10.1007/978-3-642-29912-4_18), [https://doi.org/10.1007/978-3-642-29912-4\\_18](https://doi.org/10.1007/978-3-642-29912-4_18)
  16. Karayalçin, S., Krcek, M., Picek, S.: SoK: Deep Learning-based Physical Side-channel Analysis. *Cryptology ePrint Archive*, Paper 2025/1309 (2025), <https://eprint.iacr.org/2025/1309>
  17. Karayalçin, S., Krček, M., Picek, S.: Interpreting Emergent Features in Deep Learning-based Side-channel Analysis (2025), <https://arxiv.org/abs/2502.00384>
  18. Kasper, M., Schindler, W., Stöttinger, M.: A stochastic method for security evaluation of cryptographic FPGA implementations. In: Bian, J., Zhou, Q., Athanas, P., Ha, Y., Zhao, K. (eds.) *Proceedings of the International Conference on Field-Programmable Technology, FPT 2010, 8-10 December 2010, Tsinghua University, Beijing, China*. pp. 146–153. IEEE (2010). <https://doi.org/10.1109/FPT.2010.5681772>, <https://doi.org/10.1109/FPT.2010.5681772>
  19. Kocher, P.C.: Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In: Koblitz, N. (ed.) *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings. Lecture Notes in Computer Science*, vol. 1109, pp. 104–113. Springer (1996). [https://doi.org/10.1007/3-540-68697-5\\_9](https://doi.org/10.1007/3-540-68697-5_9), [https://doi.org/10.1007/3-540-68697-5\\_9](https://doi.org/10.1007/3-540-68697-5_9)
  20. 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), [https://doi.org/10.1007/3-540-48405-1\\_25](https://doi.org/10.1007/3-540-48405-1_25)
  21. 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), [https://doi.org/10.1007/978-3-030-16350-1\\_9](https://doi.org/10.1007/978-3-030-16350-1_9)
  22. 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), [https://doi.org/10.1007/978-3-030-89915-8\\_3](https://doi.org/10.1007/978-3-030-89915-8_3)
  23. Perin, G., Karayalçin, S., Wu, L., Picek, S.: I know what your layers did: Layer-wise explainability of deep learning side-channel analysis. *IACR Communications in Cryptology* **2**(4) (2026). <https://doi.org/10.62056/a0wahee-3>

24. Prouff, E., Rivain, M.: Masking against Side-Channel Attacks: A Formal Security Proof. In: Johansson, T., Nguyen, P.Q. (eds.) *Advances in Cryptology - EUROCRYPT 2013*, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7881, pp. 142–159. Springer (2013). [https://doi.org/10.1007/978-3-642-38348-9\\_9](https://doi.org/10.1007/978-3-642-38348-9_9), [https://doi.org/10.1007/978-3-642-38348-9\\_9](https://doi.org/10.1007/978-3-642-38348-9_9)
25. 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. In: Zhou, J., Batina, L., Li, Z., Lin, J., Losiouk, E., Majumdar, S., Mashima, D., Meng, W., Picek, S., Rahman, M.A., Shao, J., Shimaoka, M., Soremekun, E.O., Su, C., Teh, J.S., Udovenko, A., Wang, C., Zhang, L.Y., Zhauniarovich, Y. (eds.) *Applied Cryptography and Network Security Workshops - ACNS 2023 Satellite Workshops, ADSC, AIBlock, AIHWS, AIOts, CIMSS, Cloud S&P, SCI, SecMT, SiMLA*, Kyoto, Japan, June 19-22, 2023, Proceedings. Lecture Notes in Computer Science, vol. 13907, pp. 139–158. Springer (2023). [https://doi.org/10.1007/978-3-031-41181-6\\_8](https://doi.org/10.1007/978-3-031-41181-6_8), [https://doi.org/10.1007/978-3-031-41181-6\\_8](https://doi.org/10.1007/978-3-031-41181-6_8)
26. Schindler, W., Lemke, K., Paar, C.: A Stochastic Model for Differential Side Channel Cryptanalysis. In: Rao, J.R., Sunar, B. (eds.) *Cryptographic Hardware and Embedded Systems - CHES 2005*, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3659, pp. 30–46. Springer (2005). [https://doi.org/10.1007/11545262\\_3](https://doi.org/10.1007/11545262_3), [https://doi.org/10.1007/11545262\\_3](https://doi.org/10.1007/11545262_3)
27. Yap, T., Benamira, A., Bhasin, S., Peyrin, T.: Peek into the Black-Box: Interpretable Neural Network using SAT Equations in Side-Channel Analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2023**(2), 24–53 (2023). <https://doi.org/10.46586/TCHES.V2023.I2.24-53>, <https://doi.org/10.46586/tches.v2023.i2.24-53>
28. Yap, T., Picek, S., Bhasin, S.: Occpois: Points of interest based on neural network’s key recovery in side-channel analysis through occlusion. In: Mukhopadhyay, S., Stanica, P. (eds.) *Progress in Cryptology - INDOCRYPT 2024 - 25th International Conference on Cryptology in India*, Chennai, India, December 18-21, 2024, Proceedings, Part II. Lecture Notes in Computer Science, vol. 15496, pp. 3–28. Springer (2024). [https://doi.org/10.1007/978-3-031-80311-6\\_1](https://doi.org/10.1007/978-3-031-80311-6_1), [https://doi.org/10.1007/978-3-031-80311-6\\_1](https://doi.org/10.1007/978-3-031-80311-6_1)
29. Yoshida, K., Karayalcin, S., Picek, S.: Can KANs Do It? Toward Interpretable Deep Learning-based Side-channel Analysis. *IACR Cryptol. ePrint Arch.* p. 1570 (2024), <https://eprint.iacr.org/2024/1570>
30. Zaid, G., Bossuet, L., Carbone, M., Habrard, A., Venelli, A.: Conditional Variational AutoEncoder based on Stochastic Attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2023**(2), 310–357 (2023). <https://doi.org/10.46586/TCHES.V2023.I2.310-357>, <https://doi.org/10.46586/tches.v2023.i2.310-357>
31. Zaid, G., Bossuet, L., Habrard, A., Venelli, A.: Methodology for Efficient CNN Architectures in Profiling Attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2020**(1), 1–36 (2020). <https://doi.org/10.13154/TCHES.V2020.I1.1-36>, <https://doi.org/10.13154/tches.v2020.i1.1-36>