OccPoIs: Points of Interest based on Neural Network's Key Recovery in Side-channel Analysis through Occlusion

Trevor Yap[1][0000−0001−8651−574X], Stjepan Picek[3][0000−0001−7509−4337], and Shivam Bhasin[1,2][0000−0002−6903−5127]

[1] Temasek Laboratories, Nanyang Technological University, 50 Nanyang Ave, 639798, Singapore
[2] National Integrated Centre for Evaluation (NICE), Nanyang Technological University, Singapore
{trevor.yap, sbhasin}@ntu.edu.sg
[3] Digital Security Group, Radboud University, 6525 EC Nijmegen, The Netherlands
stjepan.picek@ru.nl

Abstract. Deep neural networks (DNNs) represent a powerful technique for assessing cryptographic security concerning side-channel analysis (SCA) due to their ability to aggregate leakages automatically, rendering attacks more efficient without preprocessing. Despite their effectiveness, DNNs are predominantly black-box algorithms, posing considerable interpretability challenges. In this paper, we propose a novel technique called Key Guessing Occlusion (KGO) that acquires a minimal set of sample points required by the DNN for key recovery, which we call OccPoIs. These OccPoIs provide information about the areas of the traces important to the DNN for retrieving the key, enabling evaluators to know where to refine their cryptographic implementation. After obtaining the OccPoIs, we first explore the leakages found in these OccPoIs to understand what the DNN is learning with first-order Correlation Power Analysis (CPA). We show that KGO obtains relevant sample points that have a high correlation with the given leakage model but also acquires sample points that first-order CPA fails to capture. Furthermore, unlike the first-order CPA in the masking setting, KGO obtains these OccPoIs without knowing the shares or mask. Next, we employ the template attack (TA) using the OccPoIs to investigate if KGO could be used as a feature selection tool. We show that using the OccPoIs with TA can recover the key for all the considered synchronized datasets and is consistent even on datasets protected by first-order masking. Finally, KGO also allows a more efficient attack than other feature selection techniques on the first-order masking dataset called ASCADf.

Keywords: Side-channel Analysis · Neural Network · Deep Learning · Profiling attack · Explainability · Feature Importance · Feature Selection.

1 Introduction

Side-channel analysis is a class of cryptanalytic attacks that aims to extract sensitive information from a system by observing its physical attributes. Profiling SCA represents the worst-case security assumptions where the adversary has access to two similar devices: the prototype (or clone) and the target (or test) device. The adversary can manipulate or know the inputs and keys of the prototype device but has no control or knowledge of the key in the target device. The adversary can obtain traces (e.g., power or EM measurements) through an oscilloscope and record the corresponding plaintext/ciphertext used for both devices. The adversary's goal is to obtain the secret key of the target device. In recent years, the use of DNNs in profiling attacks has gained much attention as it outperforms existing techniques without requiring the evaluator to conduct arduous preprocessing on the traces before mounting the attack [8]. These networks can find the necessary sample points required for key recovery, even in the presence of hiding countermeasures and masking countermeasures where the secret information is split into multiple shares [8]. DNN can implicitly combine these shares to retrieve the secret key of the target device. Since traces contain both relevant and irrelevant sample points, an inherent question arises: *What features/sample points does a trained DNN use to obtain the secret key?*

In this paper, we propose an algorithm that extracts the minimal number of relevant sample points that a DNN would require to find the secret key. Our proposed algorithm applies the technique commonly known as occlusion, which involves replacing each sample point with a baseline value [36]. The goal of our novel algorithm is twofold. First, we aim to identify the features relevant to the DNN's acquisition of the secret key and comprehend the DNN's decision-making process. Knowing which sample points are leaking is of utmost importance for evaluators so that the designers can understand and rectify their cryptographic implementation to ensure the system's security. Second, we introduce our algorithm as a feature selection tool to extract a smaller set of sample points. Although DNNs have found great success in retrieving the key, finding a successful model requires tremendous effort due to the numerous hyperparameters involved. Significant time and processing power are needed, despite efforts made to automate hyperparameters search [23,30]. Therefore, template attack remains a popular choice among evaluators. While TA has no hyperparameters to tune[4], it has been demonstrated that extracting relevant points, also known as Points of Interest (PoIs), can lead to significantly more effective attacks [12,20]. Therefore, introducing our algorithm as a feature selection tool can aid the evaluation of cryptographic implementations. In other words, we utilize the explainability of DNN in the form of feature extraction to improve classical attacks like TA.

Our Contributions. Our main contributions can be summarized as follows:

[4] Besides the choice of using the template attack or pooled template attack, the leakage model, and the number of features.

1. We present a novel algorithm called Key Guessing Occlusion (KGO) that utilizes occlusion to identify the smallest set of PoIs necessary for the successful key recovery by the DNN. We refer to this set of PoIs as OccPoIs. Using the KGO method, we can determine how important certain features are for a DNN to retrieve the correct key. Thus, our method provides valuable insights to explain the DNN's decision-making process.
2. We compare OccPoIs with leakage samples obtained from first-order correlation power analysis with known key and mask. Our results show that while the KGO method, in some cases, provides the PoIs with high correlation with the leakage model, there are instances where KGO captures sample points that the first-order CPA fails to detect (i.e., low correlation with the leakage model). Moreover, unlike first-order CPA, KGO attains these PoIs without knowledge of the mask. Finally, using KGO allows the evaluators to protect areas of the cryptographic implementation exploited by DNNs but not recognized by classical techniques.
3. We demonstrate the capability of KGO as a feature selection tool for TA and compare it with other "classical" feature selection methods as well as other DNN's explainability methods like Saliency Map, LRP, and 1-Occlusion. We show that utilizing the OccPoIs provided by KGO as a feature selection tool enables us to recover the secret key for all datasets tested, indicating that KGO is reliable when used as a feature selection tool. Furthermore, we found that KGO could obtain better results for the first-order masking dataset called ASCADf. In some cases, KGO obtains the minimum number of sample points required to obtain the secret key and thus identify the masking order. These results show KGO's effectiveness as a feature selection tool.
4. We explore the leakage profile of a dataset with first-order masking and desynchronization protection by applying KGO. We visualize the OccPoIs through the algorithm called 1-KGO to show that KGO obtains sample points that evaluators may overlook when using other attribution-based methods. As KGO is linked with guessing entropy, it allows visualization with a human-interpretable context that none of the attribution-based methods provides. This assures evaluators that the OccPoIs shown in the visualization are leaking secret information.

The presented results focus on unprotected or first-order protected (masking) due to the nature of available datasets. The proposed approaches can be easily applied to higher-order masking, but we leave this to future work. The source code for our experiments can be accessed at an anonymous repository.[5]

Paper Organization. The structure of the paper is as follows. First, we provide the notation and the necessary background on profiling attack and explainability techniques used in Section 2. Next, we present related works on explainability and feature selection in Section 3. Section 4 introduces the KGO algorithm to obtain the relevant sample points. Section 5 provides the experimental settings

[5] `https://anonymous.4open.science/r/OccPoIs-7E0A/`

and datasets used. We explore the leakages in OccPoIs in Section 6 and investigate KGO as a feature selection tool for TA in Section 7. Subsequently, Section 8 examines the use of KGO on desynchronized traces with a masking countermeasure. We discuss the limitation of KGO in Section 9 and conclude our work in Section 10. Appendix A provides information about the feature selection techniques.

2 Background

2.1 Profiling Attacks

Notation and Terminology. We denote sets with calligraphic letters $\mathcal{X}$. The corresponding capital letter X denotes a random variable, and the bold capital letter $\boldsymbol{X}$ denotes a random vector. We use the corresponding lowercase letters x and $\boldsymbol{x}$ to represent the realizations of X and $\boldsymbol{X}$, respectively. We use $\boldsymbol{x}[i]$ as the i^{th} entry of a vector $\boldsymbol{x}$. A side-channel trace is defined as a vector $\boldsymbol{t} \in \mathbb{R}^D$ where D is the number of sample points in a trace. Throughout this paper, we will call $\boldsymbol{t}[i]$ a sample point or feature interchangeably. To denote a specific trace in a dataset, we use the notation $\boldsymbol{t}^j$ to denote j^{th} trace in the set. Let $Crypt$ represent a cryptographic primitive with PT denoting some public variable (e.g., plaintext or ciphertext). We denote k as a realization of the key byte candidate, taking its value from the keyspace $\mathcal{K}$ and the correct key as k^*. The targeted sensitive variable is the output of the cryptographic primitive, $Z = Crypt(PT, k^*)$ with Z taking values in $\mathcal{Z} = \{s_1, s_2, \ldots, s_{|\mathcal{Z}|}\}$.

Profiling attacks consist of two stages: the profiling and the attack phase. In the profiling phase, the adversary builds a distinguisher $\mathcal{F}$ that takes a set of profiling traces from the prototype device and returns a conditional probability mass function $\Pr(Z|\boldsymbol{T} = \boldsymbol{t})$. In the attack phase, a probability score is returned from the distinguisher $\boldsymbol{y}^i = \mathcal{F}(\boldsymbol{t}^i)$ for each attack trace $\boldsymbol{t}^i$ acquired from the target device. Given a fixed number of attack traces N_a, the log-likelihood score is calculated for all key candidates k, $s_{N_a}(k) = \sum_{i=1}^{N_a} log(\boldsymbol{y}^i[z_k^i])$. Here, $z_k^i = Crypt(pt^i, k)$ denotes the hypothetical sensitive value based on the key k with the public variable pt^i that corresponds to the trace $\boldsymbol{t}^i$. Next, we sort the log-likelihood scores of the keys in decreasing order and place them into a guessing vector $\boldsymbol{G} = [G_0, G_1, \ldots, G_{|\mathcal{K}|-1}]$. The key corresponding to the score G_0 is the most likely candidate, and $G_{|\mathcal{K}|-1}$ is the score of the least likely candidate. The index of the guessing vector $\boldsymbol{G}$ is called the rank of the key. The guessing entropy GE is defined as the average rank of the correct key k^* for a fixed number of experiments. The attack is successful if $GE = 0$ (or some sufficiently small value). The two common profiling attacks are the TA and the deep learning-based SCA (DLSCA).

TA uses Bayes' Theorem to build its distinguisher by assuming the conditional probability $Pr(\boldsymbol{T}|Z = z)$ to be the multivariate Gaussian distribution [9]. On the other hand, DLSCA uses a DNN, $f_{\boldsymbol{\theta}}$, as the distinguisher where $\mathcal{F} = f_{\boldsymbol{\theta}}$ with trainable weights θ. The most commonly used DNNs in SCA are Multilayer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs).

2.2 Explainability Techniques for Feature Importance in DNNs

In this section, we review some techniques used to identify the importance of features based on DNN in a side-channel setting. In particular, we focus on attribution-based techniques, as they represent the common choice to examine the importance of features.

In [14], Hettwer et al. proposed an attribution heatmap to visualize the relevance of each sample point. This is done by calculating

$$\bar{\boldsymbol{r}} = \frac{1}{N_{attri}} \sum_{j=1}^{N_{attri}} \boldsymbol{r}^{C_{k*}}(\boldsymbol{t}^j, f_\theta), \tag{1}$$

where $C_{k*} \in |\mathcal{Z}|$ is the output class of correct key and N_{attri} is the number of traces used to obtain the relevance $\boldsymbol{r}^{C_{k*}}$ of the DNN f_θ. The relevance $\boldsymbol{r}^{C_{k*}}$ can be calculated in different ways. We present the three methods used in [14]. Here, we did not consider using gradient input because it has been shown that under certain circumstances, it is equivalent to Layer-wise Relevance Propagation (LRP) [3].

Saliency Map The Saliency Map was first applied to side-channel traces in [16]. It was then extended into an attribution method [14]. The Saliency Map is implemented so that each sample point's relevance is computed by

$$r_i^c = \left\| \frac{\partial f_c(\boldsymbol{t})}{\partial \boldsymbol{t}[i]} \right\|_\infty, \tag{2}$$

where $f_\theta(\boldsymbol{t}) = [f_1(\boldsymbol{t}), \ldots, f_{|\mathcal{Z}|}(\boldsymbol{t})]$ is the output of the DNN, and c is the hypothetical class considered. It represents how a small modification in the sample points of the trace impacts the DNN's prediction.

Layer-wise Relevance Propagation Another method used to calculate the relevance $\boldsymbol{r}^{C_{k*}}$ is the LRP developed by Bach et al. [4]. This method provides a relevant value to each neuron and layer of the DNN. The process starts from the last layer and computes the relevant value layer-wise backward through the following propagation rule:

$$r_i^{(l)} = \sum_j \frac{z_{ij}}{\sum_{i'} z_{i'j} + \epsilon \times sign(\sum_{i'} z_{i'j})} r_j^{(l+1)}, \tag{3}$$

where $z_{ij} = a_i^{(l)} w_{ij}^{(l,l+1)}$ with $a_i^{(l)}$ being the neuron i in layer j and $w_{ij}^{(l,l+1)}$ being the (i,j)-th weight between the layer l and $l+1$. The value $r_i^{(l)}$ denotes the relevance associated with the i^{th} neuron in layer l. The ϵ is applied to ensure numerical stability. Therefore, one can obtain the relevance value of each sample point in the trace and visualize it using Eq. (1).

1-Occlusion Occlusion sensitivity analysis is developed by Zeiler and Vergus to find the location of an image relevant to the DNN by systematically setting areas of the input with grey input [36]. Subsequently, the authors of [14] applied the 1-Occlusion approach on side-channel traces by setting exactly one sample point to zero per time. The authors calculated the attribution of a single sample point as

$$r_i^c = f_c(\boldsymbol{t}) - f_c(\boldsymbol{t}[i] = 0), \tag{4}$$

where c is the class and $\boldsymbol{t}[i] = v$ is the trace $\boldsymbol{t}$ whose i^{th} sample point is replaced with the value v. This is then applied to Eq. (1) for visualization.

Although explainability techniques applied to side-channel analysis show some success in pinpointing the sample points that are leaking, there are times when the visualization is not clear. For example, [16] shows that when DNN is overfitting, the PoIs are not distinguishable. It was shown that to distinguish the PoIs, it was necessary to retrain a neural network with early stopping. However, it is still important to understand the decisions made by DNNs and how they retrieve the secret key to protect against such attacks.

3 Related Works

Prior Works on Interpretability and Explainability. We first provide definitions of interpretability and explainability to differentiate them [2]. Interpretability describes transparent models where humans can easily interpret their decisions. For example, decision trees provide interpretation based on the rules in which it splits the data [7]. On the other hand, explainability encompasses techniques used to explain black-box models like DNN as their decision-making process is not interpretable by humans [7, 13]. Two main areas of explainability of DNNs are understanding the DNN's learning process during training and providing post-hoc explanations to understand what a trained DNN has learned.

The field of explainability and interpretability in SCA has received very little attention over the past few years as most of the works are focused on the difficult task of hyperparameter tuning [23, 30]. Yet, some works exist on this topic. In [18], Perin et al. provided a metric based on the Information Bottleneck theory to visualize the information the DNN is learning for each epoch. This technique is further improved to visualize how shares are processed for each layer during training [19]. The authors of [27] tried to explain DNN through the technique called Singular Vector Canonical Correlation Analysis (SVCCA) by training with the same architecture between two different datasets and see how correlated the weights of the same layers are. Wu et al. employed ablation to study DNN processing of hiding countermeasures [32]. By randomly removing weights or channels from specific layers, they found that early layers primarily process Gaussian noise, while deeper layers handle more complex countermeasures like desynchronization. Instead of exploring interpretability in terms of a discriminative model, [34] designed a generative model by combining with a stochastic attack using an autoencoder called Conditional Variational Autoencoder, which provides equations of the leakage in the trace through the autoencoder's weights.

One crucial approach to explaining DNN is identifying which features are important to the trained network. Zaid et al. provided a feature importance technique by visualizing only the convolutional layers in a heatmap. They further used weight visualization for MLP to understand which features are important [35]. However, the visualization of the sample points here did not consider the secret key in their analysis, and it only applies to CNN architecture. Wouters et al. [28] then used gradient input to understand the impact of the filter size on desynchronized traces. Yap et al. introduced a partially interpretable DNN by utilizing the interpretable model named Truth Table Deep Convolutional Neural Network (TTDCNN) [33]. This model provides rules to identify windows of PoIs required to recover a secret key. While TTDCNN provides valuable insights into the network's behavior, it does not provide feature importance for every point. The results from this approach are model-specific to TTDCNN and cannot be generalized to other DNNs. To understand an adversary's worst-case scenario, it is crucial to comprehend what a general DNN has learned beyond just one specific model family. Thus, we must explore techniques that do not depend on the DNN's architecture (i.e., are model-agnostic). Other feature importance techniques are also explored by Masure et al. [16], where the authors used a Saliency Map (also known as Gradient Visualization) to visualize the importance of each sample point. Hettwer et al. [14] further applied this technique using attribution methods to include the class of the correct key into consideration. They also considered other feature importance techniques like LRP [4] and 1-Occlusion [36]. However, these methods did not directly consider the attack process of retrieving the key. Therefore, this highlights a gap in our understanding of the sample points required by any DNN to retrieve the secret key. Recently, Schamberger et al. [25] introduced the concept of n-occlusion to examine how the window of occlusion impacts key recovery. Furthermore, they extended their approach to second-order occlusion by adding an extra window for datasets with first-order masking. Their goal differs from ours. They seek to understand how various occlusion windows impact performance and identify the most critical area. In contrast, we aim to determine the minimum set of sample points required for a trained DNN to recover the key using occlusion, believing that multiple sets can enhance recovery.

Works on Feature Selection. Regarding the feature selection in SCA, early works proposed using the sum of squared differences (SOSD) and the sum of squared T-differences (SOST) to enhance the performance of TAs [9]. Zheng et al. further compared SOSD and SOST with other techniques like Pearson Correlation [37]. They concluded that the Pearson Correlation is the best in general. Bhasin et al. proposed another classical tool closely related to Pearson Correlation known as Normalized Inter-Class Variance (NICV) to detect leakages without any public variable [6]. However, it was stated that such a tool is not viable when primitives are protected by higher-order masking. Furthermore, these works did not consider using machine learning (ML) techniques for feature selection. [20] explored using ML techniques for feature selection. The authors proposed using wrapper and hybrid selection methods to find a subset of features for profiling attacks.

Wrapper selection methods employ classifier algorithms such as linear support vector machines while hybrid selection methods utilize both wrapper methods and classical filter selection methods to select the relevant feature subset. The authors showed that with enough tuning, ML techniques can outperform classical techniques. Picek et al. explored using ML models with information gain for feature selection [21]. The works mentioned above conduct their experiments on unprotected implementations. For masked implementation, Reparaz et al. presented a method for using mutual information to find tuples of sample points before employing the multivariate differential power analysis (DPA) for key recovery [22]. Rioja et al. considered using metaheuristics known as Estimation of Distribution Algorithms (EDAs) to help automate the selection of the PoIs in both unprotected and masking settings [24].

Instead of working with the original sample points, feature extraction techniques such as Principal Component Analysis (PCA) [15], Linear Discriminant Analysis [26], and triplet network [31] transform and reduce the dimensionality of the traces into relevant embeddings that consist of important leakage information for a better attack. However, feature extraction techniques do not consider the original traces (samples), and the evaluator may have difficulty knowing where the leakage is coming from. Therefore, we shall focus on feature selection methods.

4 Key Guessing Occlusion

As discussed in the introduction, relevant and irrelevant features exist in the traces, and a DNN explicitly chooses the features to retrieve the secret key. Then a natural question arises: *What is the smallest set of features required by the DNN for a successful key recovery?* We highlight that we are not trying to find the minimum number of points required to recover the secret key. This is trivial where the number of points equals $d + 1$ where d is the masking order. Instead, we are trying to find the minimum number of sample points *needed by the trained DNN* for a successful attack. In this section, we tackle this problem by presenting our proposed method called Key Guessing Occlusion - KGO. KGO is a greedy heuristic algorithm that provides a post-hoc explanation of the trained DNN. The KGO algorithm gives the smallest set of features/sample points required by the DNN to retrieve the secret key. This is done by occluding sample points recursively so that the remaining sample points are necessary to attain $GE = 0$. Furthermore, the proposed algorithm is model-agnostic and can be applied to any DNN, regardless of its architecture.

The complete KGO methodology is illustrated in Algorithm 1 The algorithm uses a while loop and a flag to generate the smallest list of relevant sample points that the trained DNN f_θ needs to recover the correct key. We visualize one iteration of the while loop in Figure 1. In each while loop iteration, the algorithm first randomly shuffles *queue* uniformly and initializes $index_{rd}$ as the empty set. Then, it iterates through the indices in *queue*. For each sample point *spt* in *queue*, the algorithm sets it to the baseline value in all traces (Lines 8

Algorithm 1 Key Guessing Occlusion (KGO)

Input:
Attack traces $\mathcal{D}_{attack}$,
Threshold, λ,
Trained DNN f_θ.
Output:
A set of OccPoIs $index_{OccPoI}$.

1: **procedure** KGO($\mathcal{D}_{attack}$, λ, f_θ)
2: $flag = False$
3: $queue = [0, \ldots, D-1]$
4: **while** $flag == False$ **do**
5: $flag = True$
6: $queue = shuffle(queue)$
7: $index_{rd} = \{\}$
8: **for** $i \in [0, \ldots, D-1]$ **do**
9: $spt = queue[i]$.
10: Initialize $\boldsymbol{t}_{original}$ with zeros of length $|D_{attack}|$.
11: **for all** $j \in \{0, \ldots, |D_{attack}|\}$ **do**
12: Obtain the j^{th} attack trace: $\boldsymbol{t} = D_{attack}[j]$
13: Keep the original trace value $\boldsymbol{t}_{original}[j] = \boldsymbol{t}[spt]$.
14: Replace sample point spt with 0: $\boldsymbol{t}[spt] = 0$.
15: **end for**
16: Run attack phase to obtain GE by using the updated traces D_{attack} on DNN f_θ.
17: **if** $GE \geq \lambda$ **then**
18: Add spt into $index_{rd}$.
19: **for all** $j \in \{0, \ldots, |D_{attack}|\}$ **do**
20: Obtain the j^{th} attack trace: $\boldsymbol{t} = D_{attack}[j]$
21: Replace sample point spt with its original value: $\boldsymbol{t}[spt] = \boldsymbol{t}_{original}[j]$.
22: $flag = False$.
23: **end for**
24: **end if**
25: **end for**
26: $queue = index_{rd}$.
27: **end while**
28: $index_{OccPoI} = index_{rd}$
29: **return** $index_{OccPoI}$.
30: **end procedure**

to 15) and runs the attack phase using the trained DNN f_θ with the perturbed traces to obtain GE (Line 16). Here, we set the baseline value to 0, although it can be set to other values like the sample-wise mean of the trace. As shown later in Section 7, the baseline value of 0 attains better results than the sample-wise mean. Therefore, we will use 0 as the baseline value unless otherwise stated. Next, the algorithm then checks if the resulting GE is greater than or equal to a threshold λ (Lines 17 to 24). If $GE \geq \lambda$, the original value of the sample point spt is restored in the traces because it is essential for the DNN to retrieve the correct key at the moment. If $GE < \lambda$, the sample point spt is not currently useful for the DNN in recovering the key, and it remains 0 throughout the algorithm. In this paper, we shall set $\lambda = 1$.

After one while loop iteration, we obtain a set of sample points $index_{rd}$. We notice that some of the sample points in $index_{rd}$ could still be further removed after one iteration. This could be due to the order in which the sample points are occluded. Irrelevant sample points positioned behind the sample point spt could add noise to the traces and result in $GE \geq \lambda$ for that iteration. However, the same sample point spt may no longer be necessary for the DNN to recover

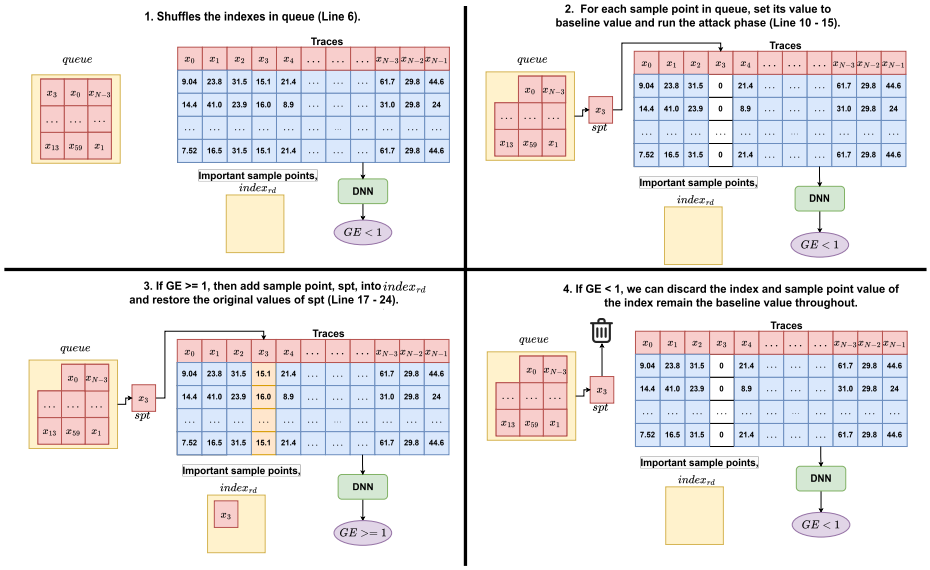


Fig. 1: Pictorial illustration of KGO algorithm (Lines 6 to 24 of Algorithm 1)

the key in the next iteration. Therefore, we fix *queue* to be the set of sample points $index_{rd}$ for the next round of occlusion (Line 26). The algorithm will exit the while loop when all the sample points in *queue* are required by the DNN to recover the secret key. In other words, none of the sample points in *queue*, when being occluded, will result in $GE < \lambda$. This will cause the $flag$ to be fixed as $True$ and exit the while loop. Since all the sample points will result in $GE \geq \lambda$ in the last round, the $index_{rd}$ will contain the same elements as *queue*. Therefore, we fix the set of relevant sample points $index_{rd}$ as $index_{OccPoIs}$ and return $index_{OccPoIs}$ as output (line 28). We shall call this set of relevant sample points obtained by the KGO algorithm, $index_{OccPoI}$, as the Occluded Points of Interest - OccPoIs.

Note that there might be more than one smallest set of relevant sample points that the DNN could use to recover the correct key. The sample point may not be necessary to the DNN at the moment when it was occluded, as the DNN could still use other sample points to retrieve the secret key. Therefore, the sample points not selected by KGO might still contain leakages. Instead, the proposed method uses the occlusion technique to reveal one set of sample points relevant to the DNN for retrieving the secret key. Furthermore, we have tried $queue = [0, \ldots, D-1]$ without shuffling. We see that KGO could favor sample points at the end of the trace. To ensure a uniform selection of sample points throughout the traces, we randomly shuffle the *queue* for each iteration (Line 6). Given the time complexity of running the DNN on the attack traces as M, the time complexity of the KGO algorithm is $O(D * |\mathcal{Z}|lg|\mathcal{Z}| + D * M)$.

To gauge the importance of each of the OccPoIs, we propose to use the following algorithm called 1-Key Guessing Occlusion (1-KGO):

1. For all the attack traces $\boldsymbol{t}$, we set the value $\boldsymbol{t}[spt] = 0$ for all sample points spt that are not OccPoIs.
2. Given a OccPoI spt_{OccPoI}, we set the value $\boldsymbol{t}[spt_{OccPoI}] = 0$ for all attack traces $\boldsymbol{t}$, and run attack phase to obtain the guessing entropy GE using the updated traces on the DNN f_θ.
3. Keep GE corresponding to OccPoI spt_{OccPoI}.
4. Set the original value of $\boldsymbol{t}[spt_{OccPoI}]$ back to the trace.
5. Repeat steps 2 to 4 for all OccPoIs.

The metric for measuring the contribution of each OccPoI in key recovery is GE provided by 1-KGO. The greater the GE value of an OccPoI, the larger its contribution toward retrieving the secret key through the DNN. We highlight that since all these sample points are OccPoIs, none of them will result in $GE < 1$ when occluded.

5 Experimental Setting

5.1 Datasets

We utilize widely employed public datasets: Chipwhispherer (CW), ASCAD (ASCADf and ASCADr), and AES_HD. We consider datasets running the Advanced Encryption Standard (AES) and focus on attacking a single byte of the secret key. Furthermore, we examine two common hypothetical leakage models used in DLSCA: the Identity (ID) and the Hamming Weight (HW) leakage models.

ChipWhisperer (CW). The ChipWhisperer dataset provides a standard comparison base for evaluating different algorithms [17]. The dataset we considered runs the unprotected AES-128 implementation on the ChipWhisperer CW308 Target. We denote this dataset as CW throughout this paper. This dataset targets the first byte in the first round of the AES substitution box, $Sbox(pt \oplus k^*)$, with a fixed key k^*. The dataset consists of 10000 traces. We use 8000 traces for profiling and 2000 traces for the attack. We use the full 2000 attack traces to run the KGO algorithm.

ASCAD. The ASCAD dataset is a first-order masked AES implementation on an 8-bit AVR microcontroller (ATMega8515) [5]. We target the third byte of the first round AES substitution box, which we denote as $Sbox(pt_3 \oplus k_3^*)$ where pt_3 is the third plaintext byte and k_3^* is the third byte of the first round key. The dataset contains two versions known as ASCADf and ASCADr. ASCADf consists of fixed key traces, while ASCADr contains random key traces for profiling and a fixed key for the attack phase. For ASCADf and ASCADr, we use 45000 traces for profiling. In the attack phase, we use 10000 attack traces for ASCADf and 100000 attack traces for ASCADr. Since running KGO is time-consuming, we consider 55000 attack traces for ASCADr when applying the KGO algorithm. This is also because the number of traces required by our trained DNN to attain $GE = 0$ is less than 55000.

AES_HD. The AES_HD is an unprotected AES hardware implementation dataset executed on an FPGA in a round-based architecture. We target the last round leakage $Sbox^{-1}(ct_{15} \oplus k^*_{15}) \oplus ct_{11}$ where ct_i is the i^{th} ciphertext byte and k^*_{15} is the 15^{th} byte of the last round secret key. We consider the extended version but only use 45000 traces for the profiling phase and 20000 traces out of the 50000 for the attack phase. For the KGO algorithm, we use 10000 attack traces to obtain the relevant points.

DNN Architecture and Training Setting. We use automated hyperparameter search tools like [29] and random search for the various datasets to find successful DNNs. Exceptionally, for the AES_HD dataset, we consider the architecture proposed by Zaid et al. [35]. We train our DNNs using the categorical cross-entropy loss. For more details of the DNNs' architecture that we considered, we refer readers to our weblink[6]. We ran our experiments on one Nvidia GeForce GTX 970 together with four Intel Core i5-4460 running at 3.2GHz with one thread each. Since we aim to understand which sample points are important to the DNN when retrieving the correct key, all the DNNs we analyze successfully retrieve the secret key. This means that we examine DNNs that are trained and can obtain $GE = 0$ during the attack phase.

6 KGO's Explainability of DNN

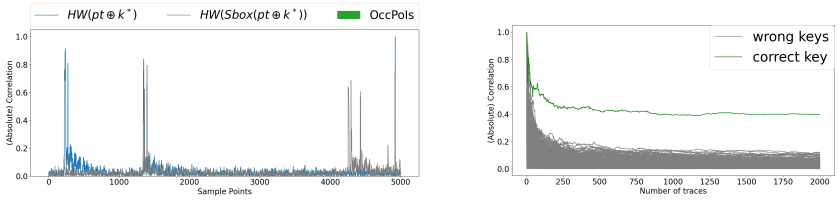
6.1 Understanding the Number of OccPoIs

The OccPoIs are sample points the DNN considers necessary for obtaining the secret key. Let ω represent the number of OccPoIs that KGO deems relevant for key recovery as acquired by Algorithm 1. Table 1 presents the number of sample points ω and the total number of sample points for different datasets. As observed from Table 1, DNNs require a very small number of sample points to recover the key successfully. In the best case, KGO demonstrates that the DNN could retrieve the key with the number of sample points equal to the order of the leakage. For example, the CW and AES_HD datasets are unprotected and require one sample point for key recovery.

Table 1: The number of OccPoIs ω obtained by KGO.

	CW	**ASCADf**	**ASCADr**	**AES_HD**
Total number of sample points	5000	700	1400	1250
ω	1	5	6	1

[6] https://github.com/yap231995/OccPoIs



(a) Correlation for different leakage models vs. OccPoIs.

(b) CPA on the sample point 4922 using leakage model $HW(Sbox(pt \oplus k))$ for all keys k concerning the number of traces.

Fig. 2: CPA in CW (HW).

6.2 Validating Leakage within OccPoIs

Next, we aim to validate what leakages the OccPoIs contain and (hopefully) glimpse into how a DNN could obtain the secret information through the sample points attained with KGO. To validate the leakages that the OccPoIs contain, we apply CPA to these sample points using only the attack traces. Since the results are similar (and due to limited space), we present results for CW and ASCADf only. We demonstrate that KGO is still effective even in low SNR settings with AES_HD and for the random key dataset (ASCADr). We show these in Supplementary Material B.

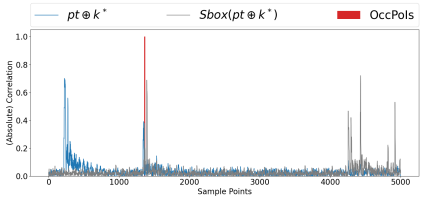
Unprotected Setting

ChipWhisperer (CW). First, we explore the leakage of OccPoIs for CW with the HW leakage model. Figure 2a shows that only one OccPoI is chosen by KGO. This OccPoI is the sample point 4922. It is highly correlated to the target hypothetical sensitive variable $HW(Sbox(PT \oplus k^*))$ (see Figure 2b). This shows that DNN could pinpoint sample points with a high correlation to the hypothetical sensitive variable Z to recover the key.

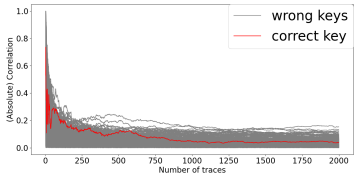
Next, we examine the leakage of OccPoI for CW trained with the ID leakage model and find that, unlike in the previous case, the OccPoI located at sample point 1365 has a very low correlation. The DNN can recover the key with just the sample point 1365, as KGO ensures that $GE = 0$.

We want to check if this OccPoI deemed by DNN as relevant is leaking in other leakage models like Most Significant Bit (MSB) or Least Significant Bit (LSB). We are also considering the leakage $pt \oplus k^*$, as Figure 3a suggests that some sample points in the area consist of that leakage. However, from Figure 4, we observe that none of the CPA attempts with the corresponding leakage models could recover the secret key.

We highlight that this OccPoI is situated near sample points that have a high correlation with the target hypothetical sensitive variable $Sbox(pt \oplus k^*)$ (see Figure 3). This suggests that the sample point 1365 is indeed leaking some secret information, but we cannot find such information with the leakage model

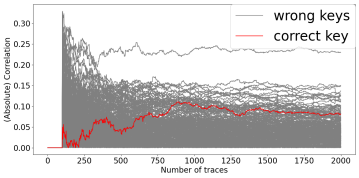


(a) Correlation for different leakage models vs. OccPoIs.

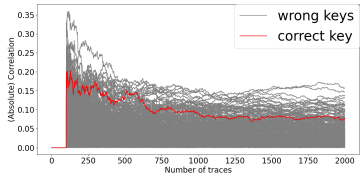


(b) CPA on the sample point 1365 using leakage model $Sbox(pt \oplus k)$ for all keys k concerning the number of traces.

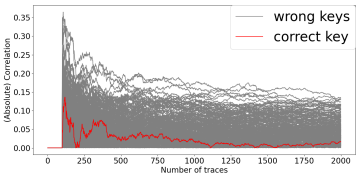
Fig. 3: CPA in CW (ID).



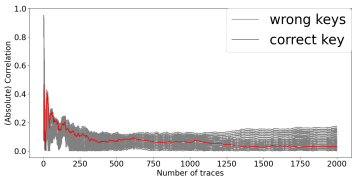
(a) $HW(Sbox(pt \oplus k^*))$.



(b) $MSB(Sbox(pt \oplus k^*))$.



(c) $LSB(Sbox(pt \oplus k^*))$.



(d) $pt \oplus k^*$.

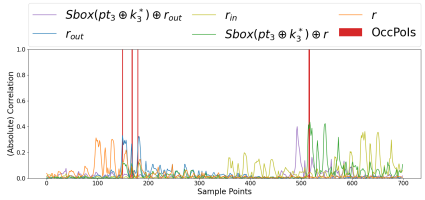
Fig. 4: CPA on the sample point 1365 using different leakage models concerning the number of traces for the CW dataset (ID).

used in CPA above. Yet, the DNN can use this sample point for key recovery. This shows that the DNN can extract complex information about the secret key from this sample point to recover the key that first-order CPA failed to capture.[7] Certain important sample points could be missed, especially when the traces consist of multiple different leakages and when an evaluator does not have a good leakage model. KGO discloses which sample points the DNN could use for key recovery. Since we have already explored the OccPoIs for both HW and ID leakage model for the CW dataset, and most works with DNNs consider only the ID leakage model [23,35], we shall focus on the ID leakage model for the rest of the experiments.

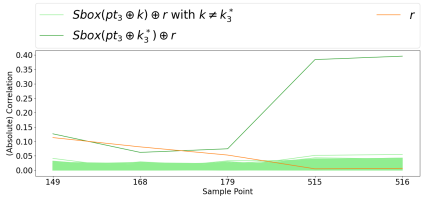
[7] Note that CPA is used under a known key setting to understand the OccPoIs provided by KGO and one should not consider KGO as a competing technique.

First-order Masked Setting Next, we apply KGO to masked datasets. For comparison, we run correlation with the same leakage model used in [11] for the ASCAD datasets.

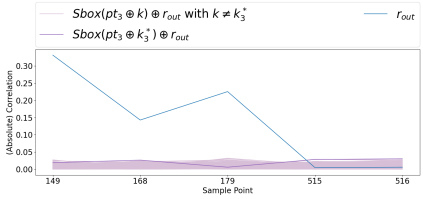
ASCADf. First, we observe the OccPoIs obtained through KGO for the ASCADf dataset. There are 5 OccPoIs that the DNN considers important for key recovery. Namely, sample points $149, 168, 179, 515$, and 516. Figure 5a shows that the OccPoIs are situated where the shares are primarily leaking. These points consists of leakage of the shares $Sbox(pt_3 \oplus k_3^*) \oplus r$ and r (see Figure 5b). All sample points have a high correlation with $Sbox(pt_3 \oplus k_3^*) \oplus r$ while sample points $149, 168$, and 179 have a high correlation with r. We also note that these points contain leakages from $Sbox(pt_3 \oplus k_3^*) \oplus r_{out}$ and r_{out} (see Figure 5c). Sample points 168 and 516 have the highest correlation with $Sbox(pt_3 \oplus k_3^*) \oplus r_{out}$ compared to the other keys, while sample points $1491, 68$, and 179 have a high correlation with r_{out}. However, we cannot determine whether the DNN used both leakages or only one of these leakages at these sample points. This remains an open question. Still, these OccPoIs are required by the DNN to recover the key when the rest of the sample points are set to 0. This highlights that these OccPoIs do indeed contain leakages used by the DNN to recover the key. Note that the correlation results are obtained with the knowledge of mask, while KGO does not need mask values to acquire these OccPoIs.



(a) Correlation for different leakage models vs. OccPoIs for ASCADf.



(b) CPA of $Sbox(pt_3 \oplus k_3^*) \oplus r$ and r on OccPoIs for ASCADf.



(c) CPA of $Sbox(pt_3 \oplus k_3^*) \oplus r_{out}$ and r_{out} on OccPoIs for ASCADf.

Fig. 5: Explainability of DNN for AES_HD and ASCADf.

Table 2: $NTGE$ of the TA when using various feature selection techniques.

	CW	ASCADf	ASCADr	AES_HD
SOSD	**3**	$> 10k$ ($GE = 2$)	$> 100k$ ($GE = 103$)	2623
SOST	9	$> 10k$ ($GE = 68$)	$> 100k$ ($GE = 51$)	2718
CPA (first-order)	11	$> 10k$ ($GE = 67$)	$> 100k$ ($GE = 162$)	2809
CPA (multi.)	-	367	**7184**	-
Saliency	9	$> 10k$ ($GE = 248$)	$> 100k$ ($GE = 210$)	3515
LRP	8	$> 10k$ ($GE = 100$)	50061	**2265**
1-Occlusion	18	$> 10k$ ($GE = 162$)	$> 100k$ ($GE = 7$)	3381
KGO	10	**313**	42991	6176

7 Exploitation of OccPoIs with the Template Attack

Since the number of OccPoIs is much smaller than the total number of sample points and DNN could recover the secret key from these sample points, we ask whether these could be used as features to improve classical SCAs like TA. We compare KGO with classical feature selection methods like SOSD, SOST, and Pearson Correlation (denoted as CPA throughout this section). Since KGO is an explainability technique, we also compare it with other explainability techniques used with DNNs to extract relevant sample points. These explainability techniques are the attribution-based methods such as Saliency Map, LRP, and 1-Occlusion. For simplicity, we will call both these attribution-based explainability techniques and classical feature selection methods feature selection methods unless it is necessary to differentiate between them.

Experimental Results. We select the top ω sample points from the other feature selection techniques indicated to compare with KGO.[8] The number of sample points ω selected by KGO can be found in Table 1 in Section 6. Throughout the paper, we use the library called INNvestigate [1] to apply the Saliency Map and LRP. In our experiments, we apply first-order CPA for all datasets and normalized multivariate second-order CPA with the multiplication of sample points as the combining function for the protected datasets with masking order 1 like ASCADf and ASCADr. We denote the first-order CPA as CPA (first-order) and the multivariate second-order CPA as CPA (multi.).

We present the number of traces TA requires to obtain $GE = 0$, also known as $NTGE$, when applying the corresponding feature selection technique for each dataset in Table 2. We observe that the OccPoIs obtained through KGO can successfully recover the key for all datasets. While SOSD obtained the best $NTGE$ for the CW dataset and LRP obtained the best results for AES_HD, KGO obtained competitive results with second-order CPA for ASCADf. We observe that for both Saliency and 1-Occlusion, we obtain $GE = 0$ for unprotected datasets, but GE did not converge at all for the first-order masking implementation - ASCADf and ASCADr. With LRP, we reach $GE = 0$ for the unprotected

[8] For feature selection that requires a known key setting (i.e., CPA and KGO), we follow the assumption from [37].

dataset and ASCADr but fail to break the target for ASCADf. In other words, KGO obtains stable results compared with all other tested explainability methods, especially when used on datasets protected with first-order masking. This demonstrates that the OccPoIs obtained through KGO are applicable as a feature selection tool for TA on synchronized traces, especially for implementation with first-order masking.

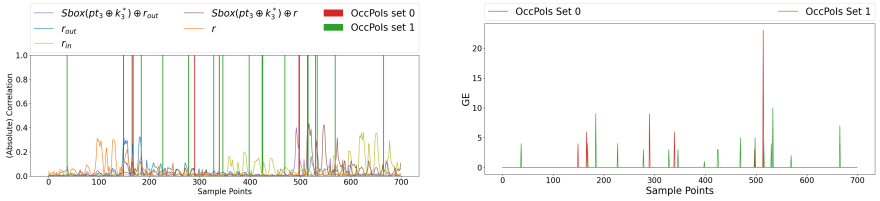
Averaging over Multiple DNNs. To verify the consistency of our results, we run a random search with the hyperparameters search space reported in our weblink[9]to find 10 DNNs that could recover the secret key successfully. Then, we run the KGO algorithm on these DNNs. We apply TA with the corresponding feature selection techniques for each model using the same number of features that KGO attained. We provide results for the ASCADf dataset as it obtains the best result. We have tried finding more DNNs with ASCADr with the same hyperparameters space. Unfortunately, we could not find more DNNs that managed to recover the secret key. Table 3 illustrates the number of successful attempts, the average GE, and the average $NTGE$ obtained. The GE is averaged over 10 attempts (each attempt corresponds to one DNN), while the $NTGE$ is averaged over the total number of successful attempts to recover the key with TA. We found that out of 10 models, 9 models have their OccPoIs leading to secret key recovery using TA. This shows that DNNs can pinpoint leakages within the traces to recover the secret and, in some instances, use a more complex leakage model. The average $NTGE$ for 9 models is 2198, which is relatively close to CPA (multi.) of 1873.9. We emphasize that although CPA (multi.) recovers the secret key in all 10 cases and obtains the best $NTGE$, it is in a known mask setting while KGO is in an unknown mask setting.

Table 3: TA performance for multiple DNNs that successfully recover the secret key using various explainability techniques on ASCADf.

	No. successful attempts	**Average** GE	**Average** $NTGE$
SOSD	1	5.00	4293.00
SOST	0	20.05	-
CPA (first-order)	0	8.94	-
CPA (multi.)	10	0	1873.90
Saliency	0	65.78	-
LRP	0	38.34	-
1-Occlusion	0	44	-
KGO	9	5.00	2198.56

Zero-valued Occlusion vs. Sample-wise Mean-valued Occlusion. Here, we explore the setting when we occlude the datasets with the sample-wise mean value instead of zero. Table 4 depicts the number of successful attempts, the average

[9] `https://github.com/yap231995/OccPoIs/blob/main/Architecure_and_Hyperparameter_SearchSpace_considered.pdf`



(a) Correlation concerning sample points for ASCADf.

(b) 1-KGO on all the sets of OccPoIs attained for ASCADf.

Fig. 6: Explainability of DNN for extended KGO on ASCADf.

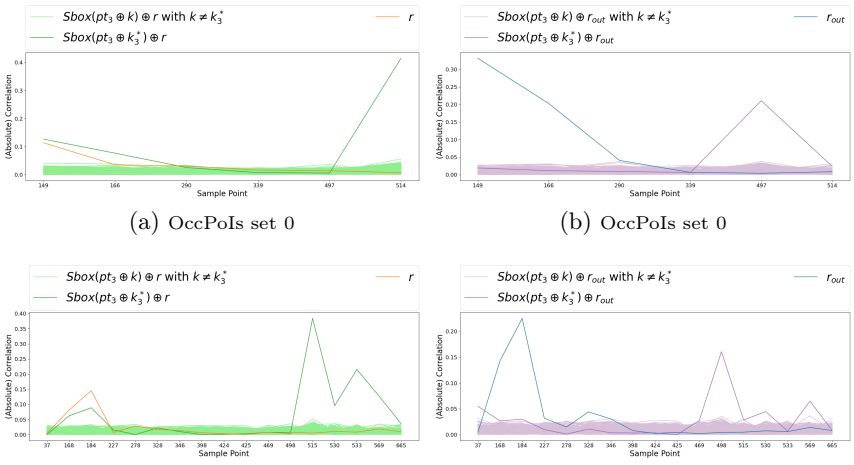
GE, and the average $NTGE$ obtained when the sample-wise mean is used as occlusion value within KGO. We see that 8 out of 10 TA attempts on the OccPoIs provided by KGO with sample-wise mean-value successfully recover the key, one less than with the zero-valued occlusion (see Table 3). Furthermore, the average GE is 10.5, which is higher than those found when the occlusion value is zero (when $GE = 5$). However, we highlight that the average $NTGE$ is lower than those found with CPA (multi.). It is not obvious why this is the case; we leave it for future work.

Table 4: TA performance for multiple DNNs that successfully recover key using various explainability techniques on ASCADf (KGO with mean-value).

	No. successful attempts	**Average** GE	**Average** $NTGE$
Saliency	0	88.44	-
LRP	0	43.44	-
1-Occlusion	0	37.83	-
KGO with sample-wise mean value	8	10.50	1075.25

Extending KGO by Applying it Multiple Times. To give a more detailed picture of what features a DNN uses to recover the key, we apply an extended version of KGO in the following manner. First, we apply KGO on all the sample points (i.e., $queue = [0, \ldots, D-1]$ in Algorithm 1) and obtain the first set of OccPoIs, $index_{OccPoI}$. We keep these in $OccPoI_{all} = index_{OccPoI}$. If the trained DNN attains $GE < 1$ on sample points $[0, \ldots, D-1] \setminus OccPoI_{all}$, we apply KGO again to obtain another set of OccPoIs, $index_{OccPoI}$. Then, we add the OccPoIs to $OccPoI_{all}$ (i.e., $OccPoI_{all} = OccPoI_{all} \cup index_{OccPoI}$. We repeat this step until the trained DNN attains $GE \geq 1$ on sample points $[0, \ldots, D-1] \setminus OccPoI_{all}$.

Since KGO with zero occlusion value allows us to have a higher chance of obtaining OccPoIs such that we recover the key successfully, we run the extended KGO with zero occlusion value. We ran extended KGO on ASCADf and obtained two disjoint sets of OccPoIs. We obtained $\omega = 6$ for the first set of OccPoIs and $\omega = 17$ for the second set of OccPoIs. In total, we recover 23 OccPoIs. Figure 6 provides the location of these OccPoIs on top of the correlation of the leakages



(a) OccPoIs set 0 (b) OccPoIs set 0

(c) OccPoIs set 1 (d) OccPoIs set 1

Fig. 7: CPA on the OccPoIs acquired from applying extended KGO.

and the 1-KGO on all the sets of OccPoIs. We also provide a zoom-in of the leakages constituting OccPoIs in Figure 7.

We observe similar results as above: the DNN uses OccPoIs where leakages can be justified with first-order CPA but also OccPoIs for which we do not know what the leakage models are. However, applying the extended KGO provides a more complete picture of the leakage location that the DNN exploits. Next, we apply TA on $OccPoIs_{all}$ and both sets of OccPoIs individually. The results are provided in Table 5. We observe that the best performances are reached using OccPoIs obtained from KGO. We also notice that when we combine both sets of OccPoIs (i.e., $OccPoIs_{all}$), the $NTGE$ increases. This suggests that the DNN also learns a function that combines different OccPoIs to recover the key more efficiently than TA.

Table 5: TA $NTGE$ when using extended KGO on the DNN with various explainability techniques.

	$OccPoIs_{all}$	**OccPoIs set 0**	**OccPoIs set 1**
ω	23	6	17
SOSD	987	$> 10k$ $(GE = 7)$	$> 10k$ $(GE = 3)$
SOST	4057	$> 10k$ $(GE = 113)$	$> 10k$ $(GE = 1)$
CPA (first-order)	$> 10k$ $(GE = 24)$	$> 10k$ $(GE = 33)$	$> 10k$ $(GE = 12)$
CPA (multi.)	2327	423	1486
Saliency	$> 10k$ $(GE = 12)$	$> 10k$ $(GE = 241)$	$> 10k$ $(GE = 79)$
LRP	$> 10k$ $(GE = 12)$	$> 10k$ $(GE = 166)$	$> 10k$ $(GE = 56)$
1-Occlusion	$> 10k$ $(GE = 12)$	$> 10k$ $(GE = 205)$	$> 10k$ $(GE = 37)$
KGO	987	332	836

8 Traces with Desynchronization

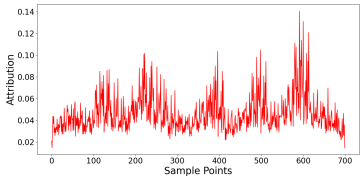
In this section, we explore OccPoIs in desynchronized datasets and visualize them using the 1-KGO algorithm. To find the locations that are leaking for desynchronized traces, one could resynchronize them and apply CPA or Signal-to-Noise ratio (SNR) to observe any leakage. However, resynchronizing the traces is tedious. Therefore, it is desirable to have a tool that can observe which sample points are leaking without any additional analysis. For an unprotected case with sufficient traces, an evaluator can obtain the PoIs without resynchronizing the traces [10]. However, for implementations that are protected by both desynchronization and masking, it is necessary to resynchronize the trace to identify the PoIs [10]. Instead, we hope to find the positions where the leakages are through the explainability methods. Although Masure et al. explored the use of the Saliency Map in an unprotected dataset and showed that the desynchronization is observable [16], to the best of our knowledge, no works have explored the use of explainability techniques to visualize leakages in datasets protected by both desynchronization and masking.

We investigate the ASCADf and ASCADr datasets with desynchronization levels of 50 and 100. We present the results for ASCADf (i.e., ASCADf_desync50 and ASCADf_desync100), as both datasets provide similar observations. To observe how much contribution each OccPoI has toward retrieving the secret key, we apply 1-KGO. Both 1-KGO and Saliency Map results are provided in Figure 8 for ASCADf_desync50 and ASCADf_desync100. We observe that the number of OccPoIs is still relatively small compared to the total number of samples. There are only 32 OccPoIs for ASCADf_desync50 and 60 OccPoIs for ASCADf_desync100 out of the 700 sample points. Most sample points acquired by KGO and Saliency Map are similar. However, some sample points with a relatively low relevance value in the Saliency Map are considered crucial by KGO. For instance, the sample point 4 is picked up by KGO for ASCADf_desync50 (the first sample point in Figure 8b), and the sample points around 300 in ASCADf_desync100, would be overlooked if relying solely on Saliency Maps. Therefore, this highlights the potential for false security when using the Saliency Map. This discrepancy arises likely because attribution-based methods provide feature importance that lacks context specific to the key recovery task. In contrast, KGO offers confidence in identifying necessary OccPoIs by incorporating the attack phase.

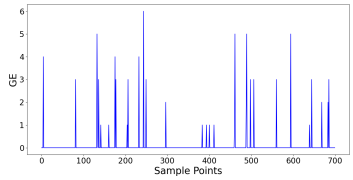
9 Limitations

Although the KGO algorithm helps find the minimal set of features that the DNN requires to retrieve the key, there is still a trade-off regarding time. The total time to run the KGO algorithm is presented in Table 6.

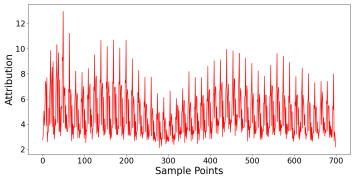
If we consider KGO as a feature selection technique instead of *just* an explainability technique, the drawback of this technique is the inability to choose the number of sample points. Even so, the number of OccPoIs acquired by KGO



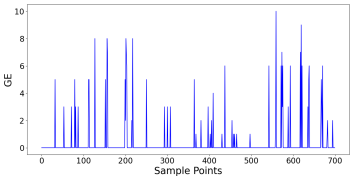
(a) Saliency Map for ASCADf_desync50.



(b) 1-KGO of DNN for ASCADf_desync50.



(c) Saliency Map for ASCADf_desync100.



(d) 1-KGO of DNN for ASCADf_desync100.

Fig. 8: Explainability of DNN for ASCADf with desynchronization.

Table 6: Time taken to run KGO.

	CW ASCADf	ASCADr	AES_HD
Time Taken (hrs)	26.3	21.3	227.7	44.9

is small compared to the total number of sample points in the traces. This means that using the OccPoIs still increases the efficiency of TA. Another drawback is that KGO only gives importance to a very small set of points, unlike other explainability techniques, which give different importance to all the sample points. Furthermore, it does not mean the sample points are not leaking if KGO does not consider them. There might be more than one minimal set in which the DNN manages to recover the secret key. Despite that, knowing at least one set of points needed by the DNN to obtain the secret key through the KGO algorithm allows evaluators to know which areas require further protection to increase the security of the cryptographic implementation.

10 Conclusion and Future Work

In this paper, we propose a novel explainability technique called KGO. KGO obtains a set of relevant sample points (OccPoIs) that are necessary for DNN to retrieve the secret key. We can observe what kind of leakage these OccPoIs contain so the DNN can recover the key. Some of these OccPoIs are highly correlated to the hypothetical leakage model, while others cannot be detected

by CPA. Next, we show that KGO could be used as a feature selection tool for TA on synchronized traces. Moreover, our approach obtains competitive results for ASCADf. Overall, we have shown that the TA performance is consistent when using KGO as a feature selection tool. Lastly, we demonstrated that KGO could be used on desynchronized traces to visualize the leakages even when the implementation is protected by first-order masking. In addition, since the attack phase is integrated within KGO, it assures evaluators that KGO's identified areas are genuinely leaking sensitive information, which is a key advantage over other attribution-based methods. For future work, one direction could be to speed up the algorithm to find more sets of OccPoIs. One could also study how to incorporate the attack phase into the attribution-based methods to provide importance for every sample point that gives human-interpretable context.

Acknowledgments

We acknowledge the generous support of Seagate Technology towards this study. This work received funding in the framework of the NWA Cybersecurity Call with project name PROACT with project number NWA.1215.18.014, which is (partly) financed by the Netherlands Organisation for Scientific Research (NWO). Additionally, this work was supported in part by the Netherlands Organization for Scientific Research NWO project DISTANT (CS.019).

References

1. Alber, M., Lapuschkin, S., Seegerer, P., Hägele, M., Schütt, K.T., Montavon, G., Samek, W., Müller, K.R., Dähne, S., Kindermans, P.J.: iNNvestigate Neural Networks! Journal of Machine Learning Research **20**(93), 1–8 (2019), `http://jmlr.org/papers/v20/18-540.html`
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. Ancona, M., Ceolini, E., Öztireli, C., Gross, M.: Towards better understanding of gradient-based attribution methods for deep neural networks. arXiv preprint arXiv:1711.06104 (2017)
4. Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.R., Samek, W.: On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. PloS one **10**(7), e0130140 (2015)
5. 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)
6. Bhasin, S., Danger, J.L., Guilley, S., Najm, Z.: Nicv: Normalized inter-class variance for detection of side-channel leakage. In: 2014 International Symposium on Electromagnetic Compatibility, Tokyo. pp. 310–313 (2014)
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. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2017. pp. 45–68. Springer International Publishing, Cham (2017)
9. Choudary, O., Kuhn, M.G.: Efficient template attacks. In: Smart Card Research and Advanced Applications: 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers 12. pp. 253–270. Springer (2014)
10. Debande, N., Souissi, Y., Nassar, M., Guilley, S., Le, T.H., Danger, J.L.: "Re-synchronization by moments": An efficient solution to align Side-Channel traces. In: 2011 IEEE International Workshop on Information Forensics and Security. pp. 1–6 (2011)
11. 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. pp. 75–99. Springer International Publishing, Cham (2022)
12. Gierlichs, B., Lemke-Rust, K., Paar, C.: Templates vs. stochastic methods. In: Goubin, L., Matsui, M. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2006. pp. 15–29. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
13. 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)
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. pp. 645–666. Springer International Publishing, Cham (2020)
15. Lerman, L., Poussier, R., Bontempi, G., Markowitch, O., Standaert, F.X.: Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In: International Workshop on Constructive Side-Channel Analysis and Secure Design. pp. 20–33. Springer (2015)
16. 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. pp. 145–167. Springer International Publishing, Cham (2019)
17. O'flynn, C., Chen, Z.: Chipwhisperer: An open-source platform for hardware embedded security research. In: Constructive Side-Channel Analysis and Secure Design: 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers 5. pp. 243–260. Springer (2014)
18. 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., De Santis, F. (eds.) Constructive Side-Channel Analysis and Secure Design. pp. 53–81. Springer International Publishing, Cham (2021)
19. Perin, G., Wu, L., Picek, S.: I know what your layers did: Layer-wise explainability of deep learning side-channel analysis. Cryptology ePrint Archive, Paper 2022/1087 (2022), https://eprint.iacr.org/2022/1087, https://eprint.iacr.org/2022/1087
20. Picek, S., Heuser, A., Jovic, A., Batina, L.: A Systematic Evaluation of Profiling Through Focused Feature Selection. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **27**(12), 2802–2815 (2019)

21. Picek, S., Heuser, A., Jovic, A., Ludwig, S.A., Guilley, S., Jakobovic, D., Mentens, N.: Side-channel analysis and machine learning: A practical perspective. In: 2017 International Joint Conference on Neural Networks (IJCNN). pp. 4095–4102 (2017)
22. Reparaz, O., Gierlichs, B., Verbauwhede, I.: Selecting time samples for multivariate dpa attacks. In: Prouff, E., Schaumont, P. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2012. pp. 155–174. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
23. Rijsdijk, J., Wu, L., Perin, G., Picek, S.: Reinforcement Learning for Hyperparameter Tuning in Deep Learning-based Side-channel Analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems **2021**(3), 677–707 (Jul 2021)
24. Rioja, U., Batina, L., Flores, J.L., Armendariz, I.: Auto-tune pois: Estimation of distribution algorithms for efficient side-channel analysis. Computer Networks **198**, 108405 (2021), https://www.sciencedirect.com/science/article/pii/S1389128621003789
25. Schamberger, T., Egger, M., Tebelmann, L.: Hide and seek: Using occlusion techniques for side-channel leakage attribution in cnns. 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., Su, C., Teh, J.S., Udovenko, A., Wang, C., Zhang, L., Zhauniarovich, Y. (eds.) Applied Cryptography and Network Security Workshops. pp. 139–158. Springer Nature Switzerland, Cham (2023)
26. Standaert, F.X., Archambeau, C.: Using Subspace-Based Template Attacks to Compare and Combine Power and Electromagnetic Information Leakages. In: Oswald, E., Rohatgi, P. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2008. pp. 411–425. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
27. 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. pp. 175–199. Springer International Publishing, Cham (2021)
28. 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)
29. Wu, L., Perin, G., Picek, S.: I Choose You: Automated Hyperparameter Tuning for Deep Learning-based Side-channel Analysis. IACR Cryptol. ePrint Arch. **2020**, 1293 (2020)
30. Wu, L., Perin, G., Picek, S.: I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis. IEEE Transactions on Emerging Topics in Computing pp. 1–12 (2022)
31. Wu, L., Perin, G., Picek, S.: The Best of Two Worlds: Deep Learning-assisted Template Attack. IACR Transactions on Cryptographic Hardware and Embedded Systems **2022**(3), 413–437 (Jun 2022)
32. 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)
33. 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)
34. 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)

35. 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)
36. Zeiler, M.D., Fergus, R.: Visualizing and Understanding Convolutional Networks (2013)
37. Zheng, Y., Zhou, Y., Yu, Z., Hu, C., Zhang, H.: How to Compare Selections of Points of Interest for Side-Channel Distinguishers in Practice? In: Hui, L.C.K., Qing, S.H., Shi, E., Yiu, S.M. (eds.) Information and Communications Security. pp. 200–214. Springer International Publishing, Cham (2015)

A Feature Selection

In this section, we recall classical feature selection techniques used in SCA. Let x be a single sample point in the trace.

SOSD. Gierlichs et al. [12] defined the sum of squared differences (SOSD) as

$$SOSD(x,y) = \sum_{\substack{i,j=1,\\ j>i}}^{|\mathcal{Z}|} (\bar{x}_{y_i} - \bar{x}_{y_j})^2$$

where $\bar{x}_{y_i}$ is the mean of the sample point under the class y_i.

SOST. Gierlichs et al. further introduced the normalized version called the sum of squared T-differences (SOST):

$$SOST(x,y) = \sum_{\substack{i,j=1,\\ j>i}}^{|\mathcal{Z}|} \left(\frac{\bar{x}_{y_i} - \bar{x}_{y_j}}{\sqrt{\frac{\sigma^2_{y_i}}{n_{y_i}} + \frac{\sigma^2_{y_j}}{n_{y_j}}}}\right)^2$$

where $\sigma^2_{y_i}$ is the variance of the sample point x under the class y_i and n_{y_i} is the total number of traces in class y_i.

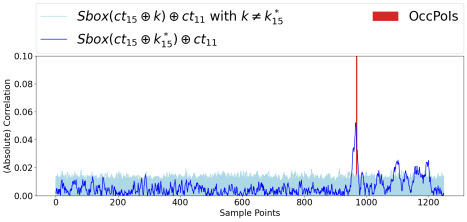
Pearson Correlation. Pearson Correlation is used in the classical non-profiling attack CPA and is calculated as

$$Pearson(x,y) = \frac{\sum_{i=1}^{N}(x^i - \bar{x})(y^i - \bar{y})}{\sqrt{\sum_{i=1}^{N}(x^i - \bar{x})^2}\sqrt{\sum_{i=1}^{N}(y^i - \bar{y})^2}},$$

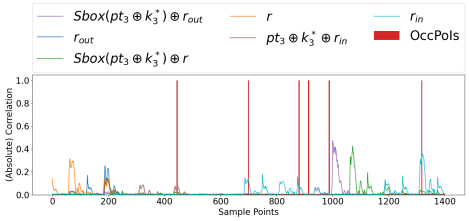
where N denotes the number of traces used.

B Validating Leakage within OccPoIs for AES_HD and ASCADr

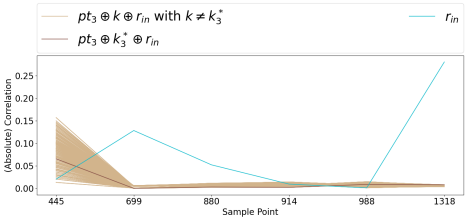
AES_HD. For AES_HD, despite the low SNR, KGO finds one OccPoI at sample point 969. This sample point is highly correlated to the hypothetical leakage model $Sbox^{-1}(ct_{15} \oplus k^*_{15}) \oplus ct_{11}$ among all the other keys (see Figure 9a). KGO leak concerning CPA validates vulnerability even on low-SNR FPGA datasets.



(a) CPA concerning sample points for AES_HD.



(b) Correlation of different leakage models vs. OccPoIs for ASCADr.



(c) CPA of $pt_3 \oplus k^*_3 \oplus r_{in}$ and r_{in} on OccPoIs for ASCADr.

Fig. 9: CPA for AES_HD and ASCADr.

ASCADr. Next, in ASCADr, we observe 6 OccPoIs extracted by KGO: 445, 699, 880, 914, 988, and 1318. The sample point 445 consists of leakage on $pt_3 \oplus k^*_3 \oplus r_{in}$ while the sample points 699, 880, 914, and 1318 consist of the leakage r_{in} (see Figure 9b). However, we note that CPA cannot distinguish the leakages of $pt_3 \oplus k^*_3 \oplus r_{in}$ among other keys (see sample point 445 in Figure 9c). Despite that, the DNN uses this point to retrieve the key. Furthermore, the sample point 988 does not correlate highly with $pt_3 \oplus k^*_3 \oplus r_{in}$ and r_{in}. From Figure 9b, sample point 988 is not correlated highly with any of the abstract leakage models tested, yet the DNN requires this sample point to obtain the secret key. This suggests that the DNN could extract complex leakage information from the sample point 988, which could come from higher-order leakage or other abstract leakage models.