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

Trevor Yap
trevor.yap@ntu.edu.sg
Nanyang Technological University
Singapore

Shivam Bhasin
sbhasin@ntu.edu.sg
Nanyang Technological University
Singapore

Stjepan Picek
picek.stjepan@gmail.com
Radboud University
Nijmegen, The Netherlands

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. Nevertheless, despite their effectiveness, DNNs employed in SCA 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 on which areas of the traces are 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 the knowledge of 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 as a feature selection tool even on datasets protected by first-order masking. Furthermore, it also allows a more efficient attack than other feature selections on the first-order masking dataset called ASCADf.

CCS CONCEPTS

• **Computing methodologies** → **Artificial intelligence**; • **Security and privacy** → **Side-channel analysis and countermeasures**.

KEYWORDS

Side-channel, Neural Network, Deep Learning, Profiling attack, Explanability, Feature Importance, Feature Selection

1 INTRODUCTION

Side-channel analysis (SCA) 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 target (or test) devices. The adversary can manipulate or knows the inputs and key of the prototype device. On the other hand, the adversary has no control or knowledge of the key in the target device but can obtain traces (e.g., power or EM measurements) through an oscilloscope and record the corresponding plaintext/ciphertext used. The adversary's goal is to obtain the secret key of the target device. Classical profiling techniques such as the template attack (TA) build their profiling models based on multivariate Gaussian distribution to obtain the key [8]. In recent years, the use of Deep Neural Networks (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 [7].

DNNs are capable of finding the necessary sample points required for key recovery, even in the presence of hiding countermeasures like desynchronization and masking countermeasure where the secret information is split into multiple shares [7]. DNN can implicitly combine these shares to retrieve the secret key of the target device. Since traces contain relevant and redundant 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 DNN requires 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. The goal of our novel algorithm is twofold. First, we aim to identify the features pertinent to the DNN in acquiring the secret key and comprehending the decision-making process of the DNN. 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 this 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 could require tremendous effort due to the large number of hyperparameters involved. While efforts have been made to automate searching for those hyperparameters [24, 29], significant time and processing power are needed. Therefore, template attack (TA) remains a popular choice among evaluators. While TA has no hyperparameters to tune[1], it has been demonstrated that extracting relevant points, also known as Points of Interest (PoIs), can lead to significantly more effective attacks [12, 21]. Therefore, introducing this 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.

Conference'17, July 2017, Washington, DC, USA
2023. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

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

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

(1) We present a new 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 the feature importance for DNN in retrieving the correct key. Thus, this 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 (CPA) 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, KGO attains these PoIs without any knowledge of the mask, unlike first-order CPA. Therefore, using KGO allows the evaluators to protect areas of the cryptographic implementation in which classical techniques fail.
(3) We demonstrate the capability of KGO as a feature selection tool for TAs 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 stable when used as a feature selection tool. Furthermore, KGO provides superior 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 the effectiveness of KGO as a feature selection tool.
(4) Lastly, 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 results presented focus on unprotected or first-order 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 works. The source code for our experiments can be accessed at https://anonymous.4open.science/r/OccPoIs-60DE/.

Paper Organization. The structure of the paper is as follows. First, we provide the notation and the necessary background on profiling attack, explainability techniques, and feature selection methods used in this paper 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. 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. Finally, we discuss the limitation of KGO in Section 9 and conclude our work in Section 10.

2 BACKGROUND

2.1 Notation and Terminology

We denote sets with calligraphic letters $\mathcal{X}$. The corresponding capital letter X defines 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 $\boldsymbol{x}[i]$ and $\boldsymbol{x}_i$ 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 $\boldsymbol{t}[i]$ sample points or features interchangeably. Let C 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, $Z = C(PT, k^*)$ with Z taking values in $\mathcal{Z} = \{s_1, s_2, \ldots, s_{|\mathcal{Z}|}\}$.

2.2 Profiling Attacks

Profiling attacks consist of two stages: the profiling and attack phases. In the profiling phase, the adversary builds a distinguisher $\mathcal{F}$ that takes in 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, $s_{N_a}(k) = \sum_{i=1}^{N_a} log(\boldsymbol{y}_i[z_{i,k}])$. Here, $z_{i,k} = C(p_i, k)$ denotes the hypothetical sensitive value based on the key k with the public variable p_i that corresponds to the trace $\boldsymbol{t}_i$. Next, we sort the keys of the log-likelihood scores 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 the key $G_{|\mathcal{K}|-1}$ is 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).

2.3 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 shall focus on attribution-based techniques.

In [15], 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*} 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 next present the three methods used in [15]. Here, we did not consider the use of gradient*input because it has been shown that under certain circumstances, it is equivalent to LRP [3].

2.3.1 Saliency Map. The Saliency Map was first applied to side-channel traces in [17]. It was then extended into an attribution method by [15]. The Saliency Map is implemented such that each sample point's relevance is computed by

$$r_j^c = \left\| \frac{\partial f_c(\boldsymbol{t})}{\partial \boldsymbol{t}_i} \right\|_\infty,$$

where $f_\theta(\boldsymbol{t}) = [f_1(\boldsymbol{t}), \ldots, f_C(\boldsymbol{t})]$ is the output of the DNN. It represents how the DNN's prediction is impacted by a small modification in the sample points of the trace.

2.3.2 Layer-wise Relevance Propagation. Another method used to calculate the relevance $\boldsymbol{r}^{C_{k*}}$ is the Layer-wise Relevance Propagation (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:

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

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).

2.3.3 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 [35]. [15] 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

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

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 the 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, [17] shows that when DNN is overfitting, the PoIs are not distinguishable. It was shown that to distinguish the PoIs, the authors had 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

This section discusses related works in interpretability and explainability of DNN in SCA. We then discuss works on feature selection techniques.

Prior Works on Interpretability and Explainability. We first provide definitions of interpretability and explainability to differentiate them [2]. Interpretability is used to describe transparent models where humans can easily interpret their decision. For example, decision trees provide interpretation based on the rules in which it splits the data [6]. 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 [6, 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 [24, 29]. Yet, there exist some works on this topic. In [19], Perin et al. provided a metric based on the Information Bottleneck theory to visualize the information that the DNN is learning for each epoch. This technique is further improved to visualize how shares are processed for each layer during training [20]. The authors of [27] tried to understand the layers of DNN by training the same DNN architecture between two different datasets. They used a technique called Singular Vector Canonical Correlation Analysis (SVCCA) to see how correlated the weights of the same layers are. Wu et al. applied the ablation technique to explore how hiding countermeasures are processed within the DNN [31]. This is achieved by randomly removing weights or channels of a particular layer in the DNN. They concluded the early layers are used to process Gaussian noise while harder countermeasures like desynchronization are being processed in the deeper layers. Instead of exploring interpretability in terms of a discriminative model, [33] 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.

Out of all the significant techniques for improving the explainability of deep neural networks, a crucial approach is determining feature importance. This involves analyzing the traces to identify which specific features are most significant to a trained DNN. 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 [34]. However, the visualization of the sample points here did not consider the secret key in their analysis and only applies to CNN architecture. Next, Wouters et al. [28] uses gradient*input to understand the impact of filters 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) [32]. This model provides rules to identify windows of PoIs required to recover a secret key. However, while TTDCNN provides valuable insights into the network's behavior, it

does not provide feature importance for every point. Additionally, the results obtained from this approach are model-specific and only applicable to the TTDCNN architecture and cannot be generalized to other types of DNNs. To understand the worst-case scenario of an adversary, it is essential to comprehend what a general DNN has learned rather than just within one specific family of models. Therefore, we would like to 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. [17], where they used a Saliency Map (also known as Gradient Visualization) to visualize the importance of each sample point. Hettwer et al. [15] 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 [35]. 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.

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 [36]. They concluded that Pearson Correlation is the best in general. However, they did not consider using machine learning (ML) techniques for feature selection. [21] explores using ML techniques for feature selection. The authors proposed using wrapper selection methods 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 to pick out the relevant subset of features. On the other hand, the hybrid selection methods utilize both wrapper methods and classical filter selection methods like SOSD/SOST to determine these important features. The authors showed that with enough tuning, ML techniques for the feature selection could be better than classical filter selection. Picek et al. also explored the use of feature selection through information gain for profiling attacks using ML models [22]. The works mentioned above conduct their experiments on unprotected implementations.

As 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 [23]. 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 [25].

Instead of working with the original sample points, feature extraction techniques transform and reduce the dimensionality of the traces into relevant embedding that consists of important information for a better attack. Feature extraction techniques such as Principal Component Analysis (PCA) [16], Linear Discriminant Analysis [26], and the use of triplet network [30] have succeeded in improving the TA's results and efficiency. 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 (KGO)

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?* In this section, we present our proposed method called Key Guessing Occlusion (KGO). KGO is a greedy heuristic algorithm that provides a post-hoc explanation of the trained DNN. It does this by giving the smallest set of features/sample points required by the DNN for retrieving the secret key. Furthermore, it is model-agnostic and can be applied to any DNN, regardless of its architecture.

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 values: $\boldsymbol{t}[spt] = \boldsymbol{t}_{original}[j]$.
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**

The methodology of KGO 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. In each iteration of the while loop, 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 0 in all traces (Lines 8 to 15) and runs the attack phase using the trained DNN f_θ with the perturbed traces to obtain GE (Line 16). It 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 to the DNN in retrieving 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 iteration of the while loop, 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 being 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. But the same sample point *spt* may no longer be necessary for the DNN to recover 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 $GE = 0$. 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. 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.

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 the *GE* corresponding to OccPoI spt_{OccPoI}.
(4) Set the original value of $\boldsymbol{t}[spt_{OccPoI}]$ back to the trace.
(5) Repeat step 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 $GE = 0$ when occluded.

5 EXPERIMENTAL AND TRAINING SETTING

We utilize widely employed public datasets: Chipwhispherer (CW), ASCAD (i.e., ASCADf/ASCADr), and AES_HD. We direct readers to Appendix A.2 for more comprehensive details on these datasets.

For most datasets, we use random search to find the DNN architecture. For the AES_HD dataset, we consider the architecture proposed by Zaid et al. [34]. We use a learning rate of either $0.0001, 0.001$, or 0.005, while the optimizers are either Adam or Root Mean Squared Propagation (RMSprop) . Next, we apply either the He Uniform , Glorot Uniform , or Random Uniform for the weight initialization and use either ReLU or SeLU for the activation function. For regularizer, we consider either no regularizer, l_2 norm , or dropout . The training hyperparameters are presented in Table 1. We train our DNNs using the categorical cross-entropy loss . For more details of the DNNs' architecture that we considered, we refer readers to Appendix A.3.

Table 1: Hyperparameters used when training DNNs for each dataset.

	CW	ASCADf	ASCADr	AES_HD	ASCADf_desync50	ASCADf_desync100
Batch Size	128	50	200	256	500	300
Learning Rate	0.0001	0.005	0.005	0.005	0.001	0.001
Epochs	20	50	100	20	100	100
Weight Initialization	Glorot Uniform	He Uniform	He Uniform	He Uniform	Random Uniform	Glorot Uniform
Optimizer	RMSprop	Adam	Adam	Adam	RMSprop	RMSprop
Regularizer	None	None	None	None	l_2	Dropout
Regularizer Strength	-	-	-	-	0.0001	-

Table 2: 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

Since our goal is 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 the trained DNN can obtain $GE = 0$ during the attack phase.

6 KGO'S EXPLAINABILITY OF DNN

6.1 Understanding the Number of OccPoIs

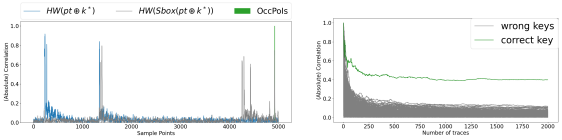
The OccPoIs are sample points the DNN considers necessary for obtaining the secret key. Let ω represent the number of OccPoIs that KGO deems relevant for key recovery as acquired by Algorithm 1. Table 2 presents the number of sample points ω and the total number of sample points for different datasets. As observed from Table 2, the DNN requires 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 minimum number of sample points required. For example, the CW and AES_HD datasets are unprotected and require only one sample point for key recovery. Thus, KGO demonstrates that DNNs can effectively recover secret keys with minimal sample points.

6.2 Validating Leakage within OccPoIs

Next, we want to validate what leakages these OccPoIs contain and hopefully glimpse into how the DNN could learn a function to retrieve the secret through these sample points by KGO. To validate the leakages that the OccPoIs contain, we apply CPA to these sample points using only the attack traces. We further investigate if OccPoIs contain leakages that are missed by a first-order CPA.

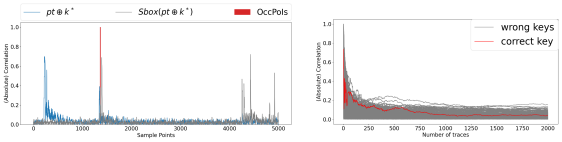
6.2.1 *Unprotected Setting.*

Chipwispherer (CW). First, we shall explore the leakage of OccPoIs for CW with the HW leakage model. Figure 1a 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 1b). This shows that DNN could pinpoint sample points with a high correlation to the hypothetical sensitive variable Z to recover the key.



(a) Correlation with different leakage model vs OccPoIs **(b) CPA on the sample point** 4922 **using leakage model** $HW(Sbox(pt \oplus k))$ **for all keys** k **with respect to the number of traces.**

Figure 1: CPA in CW (HW).



(a) Correlation with different leakage model vs OccPoIs. **(b) CPA on the sample point** 1365 **using leakage model** $Sbox(pt \oplus k)$ **for all keys** k **with respect to the number of traces.**

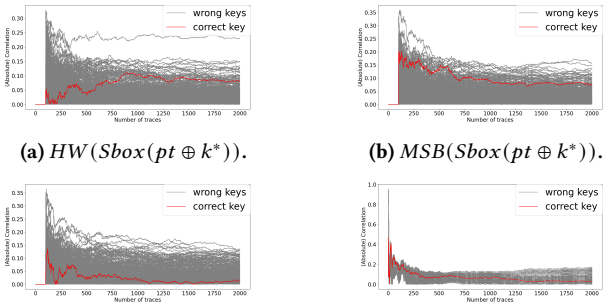
Figure 2: CPA in CW (ID).

Next, we examine the leakage of OccPoI for CW trained with the ID leakage model and found that the OccPoI located at sample point 1365 has a very low correlation, unlike the previous case. Note that 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 Hamming Weight (HW), Most Significant Bit (MSB), or Least Significant Bit (LSB). We are also considering the leakage $pt \oplus k^*$, as Figure 2a suggests that some sample points in the area consist of that leakage. However, from Figure 3, 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 2). This suggests that the sample point 1365 is indeed leaking some secret information, but we cannot find any such information with the leakage model 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. [2] Since we have already explored the OccPoIs for both HW and ID leakage model for the CW dataset, and most works on DNN consider only the ID leakage model [24, 34], we shall focus on the ID leakage model for the rest of the experiments.

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

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



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

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

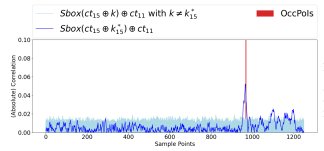
Figure 3: CPA on the sample point 1365 **using different leakage models with respect to the number of traces for the CW dataset (ID).**

all the other keys (see Figure 4a). This validates that the OccPoIs obtained through KGO are leaking concerning CPA, even on datasets with low SNR running on FPGA.

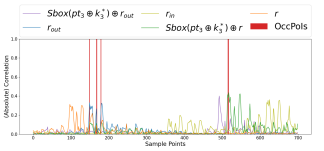
6.2.2 First-order Masked Setting. Next, we apply KGO to masking datasets. For comparison, we run correlation with the same leakage model used in [11] for the ASCAD datasets.

ASCADf. First, we observe the OccPoIs obtained through KGO on the ASCADf dataset. There are 5 OccPoIs that the DNN regard as important for key recovery, namely sample points 149, 168, 179, 515, and 516. We observe from Figure 4b 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 4c). All of the 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 4d). 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 know whether the DNN used both leakages or only one of these leakages at these sample points, and this remains an open question. We highlight that the correlation results are obtained with the knowledge of mask, while KGO does not need mask values to acquire these OccPoIs.

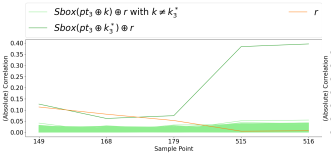
ASCADr. Lastly, KGO reveals that DNN could use sample points that consist of other leakages to retrieve the secret key. In ASCADr, we observe that 6 OccPoIs are extracted by KGO, i.e., 445, 699 , 880, 914, 988, and 1318. The sample point 445 consists of leakage on $pt_3 \oplus k^*_3 \oplus r_{in}$ while the sample points 699, 880, 914, and 1318 consist of the leakage r_{in} (see Figure 5a). 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 5b). Despite that, the DNN can still use this point to retrieve the key. This is similar to the scenario with the CW dataset trained on the ID leakage model. Furthermore, the sample point 988 does not correlate highly with $pt_3 \oplus k^*_3 \oplus r_{in}$ and r_{in}. Moreover, from Figure 5a, 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



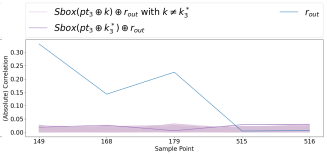
(a) CPA with respect to sample points for AES_HD.



(b) Correlation with different leakage model vs OccPoIs for ASCADf.

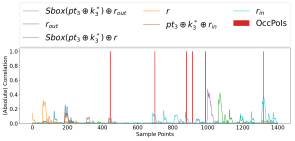


(c) CPA of $Sbox(pt_3 \oplus k_3^*) \oplus r$ and r on OccPoIs for ASCADf.

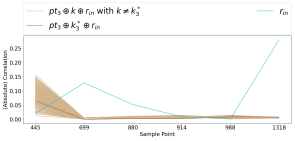


(d) CPA of $Sbox(pt_3 \oplus k_3^*) \oplus r_{out}$ and r_{out} on OccPoIs for ASCADf.

Figure 4: Explainability of DNN in AES_HD and ASCADf.



(a) Correlation with different leakage model vs OccPoIs.



(b) CPA of $pt_3 \oplus k_3^* \oplus r_{in}$ and r_{in} on OccPoIs.

Figure 5: CPA in ASCADr.

means that the DNN manages to extract complicated information from the sample point 988. Overall, we show that KGO reveals sample points that help an evaluator identify the points that may have been missed when evaluating with known leakage models using first-order CPA. Furthermore, the first-order CPA requires the knowledge of the mask to know where the PoIs are located but KGO works without any knowledge of the mask. Therefore, using KGO provides sample points that are exploitable by DNN. This allows evaluators and designers to better understand the leakage profile of their cryptographic implementation, especially since DNN could learn intricate information about the secret key.

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 the question if these could be used as features to improve classical SCA like TA. Therefore, we compare KGO with classical feature selection methods like SOSD, SOST, and Pearson Correlation (denoted as CPA throughout this section). We refer readers to Appendix A.1 for the exact details of these feature selections used. Since KGO is an explainability technique, we also compare it with other known explainability techniques that are used for general DNN 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

Table 3: NT_{GE} 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

feature selection methods simply feature selection methods unless it is necessary to differentiate between them.

Threat Model. We adopt a threat model in which an attacker has access to both profiling traces and attack traces. In addition, we assume that the attacker has retrieved the sample points for each feature selection technique independently. Finally, we select the relevant features and investigate key recovery using TA.[3]

Experimental Results. We select the top ω sample points from the other feature selection techniques indicated to compare with KGO. The number of sample points ω selected by KGO can be found in Table 2 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 dataset with masking order 1 like the ASCADf and ASCADr datasets. We denote the first-order CPA as CPA (first-order) and the multivariate second-order CPA as CPA (multi.).

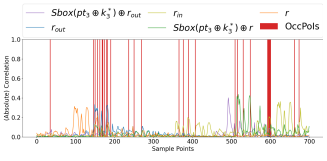
We present the number of traces TA requires to obtain $GE = 0$, also known as NT_{GE}, when applying the corresponding feature selection technique for each dataset in Table 3. We observe that the OccPoIs obtained through KGO can successfully recover the key for all datasets. While SOSD obtained the best NT_{GE} for the CW dataset and LRP obtained the best results for AES_HD, KGO obtains competitive results with second-order CPA for ASCADf. In fact, KGO obtains superior results for ASCADf (see Table 3). We observe that for both Saliency and 1-Occlusion, we obtain $GE = 0$ for unprotected datasets, but the GE did not converge at all for the first-order masking implementation - ASCADf and ASCADr. For LRP, it obtains $GE = 0$ for the unprotected datasets and ASCADr but fails to obtain a $GE = 0$ for ASCADf. In other words, KGO obtains stable results compared with all other 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.

Non-overfitting DNN.. It was reported by Masure et al. [17] that non-overfitting DNN provides better visualization for the Saliency Map when applying on first-order masking datasets like ASCADf and ASCADr. Therefore, we explore how this affects the feature selection process. We run the same DNN architecture and use the

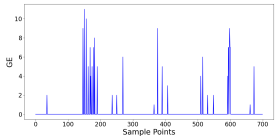
[3]For feature selection that requires a known key setting (i.e., CPA and KGO), we follow the assumption used in [36].

Table 4: The number of OccPoIs ω obtained by KGO for non-overfitting DNN.

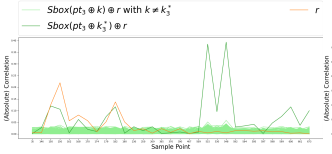
	ASCADf	ASCADr
Total sample points	700	1400
ω	31	36



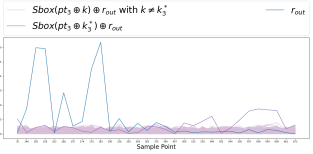
(a) CPA concerning sample points.



(b) GE when applying 1-KGO.



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



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

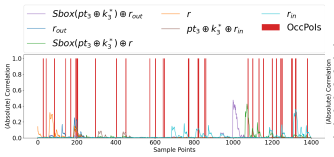
Figure 6: Explainability of non-overfitting DNN in ASCADf.

smallest epoch number required to obtain $GE = 0$. We show this on datasets with masked implementation, namely ASCADf and ASCADr, as Masure et al. reported that the issue only arises in this scenario.

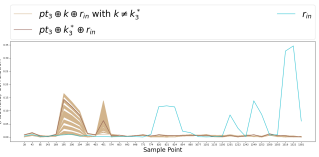
The leakages of these OccPoIs are similar to those explained in Section 6.2. The OccPoIs and their leakages are illustrated in Figures 6 and 7. The number of OccPoIs ω selected as essential for key recovery by KGO on DNN that are non-overfitting can be found in Table 4. We obtain 31 OccPoIs for the ASCADf dataset when using the non-overfitted DNN and observe that the OccPoIs found in the ASCADf dataset are situated around the $Sbox(pt_3 \oplus k_3^*) \oplus r$ and r or the $Sbox(pt_3 \oplus k_3^*) \oplus r_{out}$ and r_{out}. This means that the DNN can explicitly combine these shares. In addition, there exist some points that the DNN considers important that are not shown in first-order CPA, namely sample points 190, 250, 365, 390, and 407. These are still as relevant to the DNN, meaning the DNN learns more complex information from these points that we could not obtain from first-order CPA.

On the other hand, we acquire 36 OccPoIs for the ASCADr dataset when using a non-overfitting DNN. Similarly, we obtained OccPoIs with high correlation $Sbox(pt_3 \oplus k_3^*) \oplus r$ and r (see Figure 7c) or $Sbox(pt_3 \oplus k_3^*) \oplus r_{out}$ and r_{out} (see Figure 7b). There are also OccPoIs which contains the leakage of $pt_3 \oplus k_3^* \oplus r_{in}$ and r_{in}, especially the sample points 820, 821, 824, 854, and 860 (see Figure 7d). We also observe that some OccPoIs have a very low correlation with the leakage model tested. For example, the OccPoIs on 603, 642, and 648 are located in the area not picked up by first-order CPA (see Figure 7a). This means there are leakages in these sample points that the DNN found that the first-order CPA fails to capture.

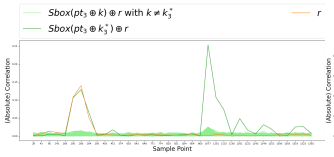
Next, we explore if the OccPoIs can still be exploited even when the DNN is non-overfitted. The NT_{GE} required by TA for using the different feature selection methods by extracting ω sample points



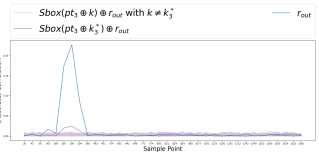
(a) CPA concerning sample points.



(b) CPA of $pt_3 \oplus k_3^* \oplus r_{in}$ and r_{in} on OccPoIs.



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



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

Figure 7: Explainability of non-overfitting DNN in ASCADr.

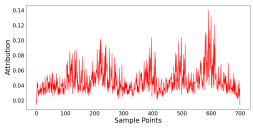
Table 5: NT_{GE} of the TA for non-overfitting DNN using various explainability techniques.

	ASCADf	ASCADr
SOSD	3311	$> 100k$ ($GE = 117$)
SOST	$> 10k$ ($GE = 2$)	7937
CPA (first-order)	$> 10k$ ($GE = 25$)	$> 100k$ ($GE = 50$)
CPA (multi.)	2707	**5427**
Saliency	$> 10k$ ($GE = 1$)	22631
LRP	1496	$> 100k$ ($GE = 118$)
1-Occlusion	2775	10763
KGO	**1197**	10807

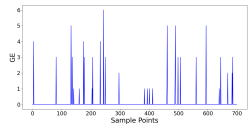
can be found in Table 5. Among the explainability techniques, KGO and 1-Occlusion attain the $GE = 0$ for both the ASCADf and ASCADr datasets. However, we recall that 1-Occlusion obtains $GE \neq 0$ when DNN are overfitted (see Table 3). On the other hand, Saliency Map did not achieve $GE = 0$ for ASCADf, and LRP did not manage to recover the key with $GE > 0$ for ASCADr. Therefore, these show that KGO is a better feature selection technique compared to other explainability techniques in choosing the relevant sample points regardless if DNN is overfitted to the profiling data.

8 TRACES WITH DESYNCHRONIZATION

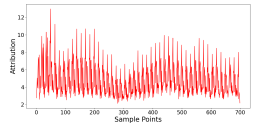
In this section, we explore the OccPoIs for desynchronized datasets and provide the visualization of these OccPoIs through the 1-KGO algorithm. Currently, to know which sample points are leaking for desynchronized traces, one could resynchronize them and apply CPA or Signal-to-Noise ratio (SNR) to observe any leakage. However, the process of 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 a large enough number of traces, an evaluator could obtain the PoIs without resynchronizing the traces [10]. However, when it comes to identifying the PoIs of traces from implementations that are protected by both desynchronization and masking, it is not possible without resynchronizing the trace [10]. Through the explainability methods, we hope to find out the positions where the



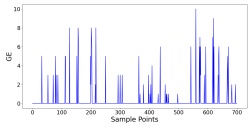
(a) Saliency Map for ASCADf_desync50.



(b) 1-KGO of DNN for ASCADf_desync50.



(c) Saliency Map for ASCADf_desync100.



(d) 1-KGO of DNN for ASCADf_desync100

Figure 8: Explainability of DNN for ASCADf with desynchronization.

leakages are. Although Masure et al. explored the use of a Saliency Map in an unprotected dataset and showed that the desynchronization is observable in the unprotected case [17], to the best of our knowledge, there are no works that explore the use of explainability techniques to visualize leakages within each sample points for datasets with desynchronization and masking protection. We investigate the ASCADf and ASCADr datasets with a desynchronization level of 50 and 100. We will present the results specifically for ASCADf (i.e., ASCADf_desync50 and ASCADf_desync100) as both datasets provide similar observations.

To observe how much contribution each OccPoIs 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 are similar to those by Saliency Map. However, we observe that some sample points with a relatively low relevance value in Saliency Map are attained by KGO. For example, the sample point 4 is picked up by KGO in ASCADf_desync50 (the first sample point on Figure 8b) while with Saliency Map, it gives a value relative lower than all the rest of the sample points. An evaluator may interpret this sample point as not useful for the DNN in recovering the secret key. Similarly, in ASCADf_desync100, the sample points around 300 could be missed if an evaluator interprets those points as irrelevant because the Saliency Map presents these sample points with the lowest relevance values. Therefore, this gives the evaluator a false sense of security when using Saliency Map. In fact, all other attribution-based methods could provide the user with the wrong interpretation [14]. This is because attribution-based methods provide feature importance without any context in a human-interpretable way. On the other hand, KGO provides confidence that the OccPoIs are necessary for the DNN to recover the secret key by introducing the attack phase into consideration.

Table 6: Time taken to run KGO.

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

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 in terms of time. Given the time complexity of running the DNN on the attack traces as M, the time complexity of the KGO algorithm is $O(D * |\mathcal{Z}|lg|\mathcal{Z}| + D * M)$. The total time to run the KGO algorithm is presented in Table 6. We ran this according to the experimental setup presented in Section 5 on four Nvidia GeForce GTX 970 together with four Intel Core i5-4460 running at 3.2GHz with only one thread each.

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 is small in comparison 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 technique, which gives different importance to all the sample points. Furthermore, it does not mean the sample points are not leaking if KGO does not consider them. In fact, 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 area requires 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, which is executed by occluding the sample point one by one. The KGO algorithm helps us to obtain a set of relevant sample points for DNN to retrieve the secret key known as OccPoIs. Through these OccPoIs, we can observe what kind of leakage these sample points contain for the DNN to recover the key. Some of these sample points are highly correlated to the hypothetical leakage model. We also observe that there exist sample points that the DNN can use to recover the secret key which 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 superior results for ASCADf compared to other methods. We also 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 KGO directly encompasses the running of GE into the algorithm, it provides confidence to the evaluators in areas leaking secret information, unlike other attribution-based methods.

Since KGO is a heuristic algorithm to obtain the minimum set of relevant sample points for key recovery, one could try to find a faster algorithm to find more sets of OccPoIs from the DNN. A possible approach could be using the occluding window of sample points first before switching to a single sample point. 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. Lastly, one could investigate the complex leakage model DNN uses to retrieve the key that first-order CPA fails to obtain in order to protect against DLSCA.

REFERENCES

[1] Maximilian Alber, Sebastian Lapuschkin, Philipp Seegerer, Miriam Hägele, Kristof T. Schütt, Grégoire Montavon, Wojciech Samek, Klaus-Robert Müller, Sven Dähne, and Pieter-Jan Kindermans. 2019. iNNvestigate Neural Networks! *Journal of Machine Learning Research* 20, 93 (2019), 1–8. http://jmlr.org/papers/v20/18-540.html

[2] Amazon. 2021. Model Explainability with AWS Artificial Intelligence and Machine Learning Solutions. https://docs.aws.amazon.com/whitepapers/latest/model-explainability-aws-ai-ml/interpretability-versus-explainability.html

[3] Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross. 2018. Towards better understanding of gradient-based attribution methods for Deep Neural Networks. arXiv:1711.06104 [cs.LG]

[4] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. 2015. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one* 10, 7 (2015), e0130140.

[5] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. 2020. Deep learning for side-channel analysis and introduction to ASCAD database. *J. Cryptogr. Eng.* 10, 2 (2020), 163–188. https://doi.org/10.1007/s13389-019-00220-8

[6] Nadia Burkart and Marco F. Huber. 2021. A Survey on the Explainability of Supervised Machine Learning. *Journal of Artificial Intelligence Research* 70 (jan 2021), 245–317. https://doi.org/10.1613/jair.1.12228

[7] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. 2017. Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures. In *Cryptographic Hardware and Embedded Systems – CHES 2017*, Wieland Fischer and Naofumi Homma (Eds.). Springer International Publishing, Cham, 45–68.

[8] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. 2003. Template Attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2002*, Burton S. Kaliski, çetin K. Koç, and Christof Paar (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 13–28.

[9] Omar Choudary and Markus G Kuhn. 2014. 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*. Springer, 253–270.

[10] Nicolas Debande, Youssef Souissi, Maxime Nassar, Sylvain Guilley, Thanh-Ha Le, and Jean-Luc Danger. 2011. "Re-synchronization by moments": An efficient solution to align Side-Channel traces. In *2011 IEEE International Workshop on Information Forensics and Security*. 1–6. https://doi.org/10.1109/WIFS.2011.6123143

[11] Maximilian Egger, Thomas Schamberger, Lars Tebelmann, Florian Lippert, and Georg Sigl. 2022. A Second Look at the ASCAD Databases. In *Constructive Side-Channel Analysis and Secure Design*, Josep Balasch and Colin O'Flynn (Eds.). Springer International Publishing, Cham, 75–99.

[12] Benedikt Gierlichs, Kerstin Lemke-Rust, and Christof Paar. 2006. Templates vs. Stochastic Methods. In *Cryptographic Hardware and Embedded Systems - CHES 2006*, Louis Goubin and Mitsuru Matsui (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 15–29.

[13] Leilani H. Gilpin, David Bau, Ben Z. Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. 2019. Explaining Explanations: An Overview of Interpretability of Machine Learning. arXiv:1806.00069 [cs.AI]

[14] Yash Goyal, Amir Feder, Uri Shalit, and Been Kim. 2020. Explaining Classifiers with Causal Concept Effect (CaCE). arXiv:1907.07165 [cs.LG]

[15] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. 2019. Deep Neural Network Attribution Methods for Leakage Analysis and Symmetric Key Recovery. Cryptology ePrint Archive, Report 2019/143. https://eprint.iacr.org/2019/143.

[16] Liran Lerman, Romain Poussier, Gianluca Bontempi, Olivier Markowitch, and François-Xavier Standaert. 2015. 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*. Springer, 20–33.

[17] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. 2019. Gradient Visualization for General Characterization in Profiling Attacks. In *Constructive Side-Channel Analysis and Secure Design*, Ilia Polian and Marc Stöttinger (Eds.). Springer International Publishing, Cham, 145–167.

[18] Colin O'Flynn and Zhizhang David Chen. 2014. ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*.

[19] Guilherme Perin, Ileana Buhan, and Stjepan Picek. 2020. Learning when to stop: a mutual information approach to fight overfitting in profiled side-channel analysis. Cryptology ePrint Archive, Report 2020/058. https://eprint.iacr.org/2020/058.

[20] Guilherme Perin, Lichao Wu, and Stjepan Picek. 2022. I Know What Your Layers Did: Layer-wise Explainability of Deep Learning Side-channel Analysis. Cryptology ePrint Archive, Paper 2022/1087. https://eprint.iacr.org/2022/1087 https://eprint.iacr.org/2022/1087.

[21] Stjepan Picek, Annelie Heuser, Alan Jovic, and Lejla Batina. 2019. A Systematic Evaluation of Profiling Through Focused Feature Selection. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27, 12 (2019), 2802–2815. https://doi.org/10.1109/TVLSI.2019.2937365

[22] Stjepan Picek, Annelie Heuser, Alan Jovic, Simone A. Ludwig, Sylvain Guilley, Domagoj Jakobovic, and Nele Mentens. 2017. Side-channel analysis and machine learning: A practical perspective. In *2017 International Joint Conference on Neural Networks (IJCNN)*. 4095–4102. https://doi.org/10.1109/IJCNN.2017.7966373

[23] Oscar Reparaz, Benedikt Gierlichs, and Ingrid Verbauwhede. 2012. Selecting Time Samples for Multivariate DPA Attacks. In *Cryptographic Hardware and Embedded Systems – CHES 2012*, Emmanuel Prouff and Patrick Schaumont (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 155–174.

[24] Jorai Rijsdijk, Lichao Wu, Guilherme Perin, and Stjepan Picek. 2021. Reinforcement Learning for Hyperparameter Tuning in Deep Learning-based Side-channel Analysis. Cryptology ePrint Archive, Report 2021/071. https://ia.cr/2021/071.

[25] Unai Rioja, Lejla Batina, Jose Luis Flores, and Igor Armendariz. 2021. Auto-tune POIs: Estimation of distribution algorithms for efficient side-channel analysis. arXiv:2012.13225 [cs.CR]

[26] François-Xavier Standaert and Cedric Archambeau. 2008. Using Subspace-Based Template Attacks to Compare and Combine Power and Electromagnetic Information Leakages. In *Cryptographic Hardware and Embedded Systems – CHES 2008*, Elisabeth Oswald and Pankaj Rohatgi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 411–425.

[27] Daan van der Valk, Stjepan Picek, and Shivam Bhasin. 2021. Kilroy Was Here: The First Step Towards Explainability of Neural Networks in Profiled Side-Channel Analysis. In *Constructive Side-Channel Analysis and Secure Design*, Guido Marco Bertoni and Francesco Regazzoni (Eds.). Springer International Publishing, Cham, 175–199.

[28] Lennert Wouters, Victor Arribas, Benedikt Gierlichs, and Bart Preneel. 2020. Revisiting a Methodology for Efficient CNN Architectures in Profiling Attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020, 3 (Jun. 2020), 147–168. https://doi.org/10.13154/tches.v2020.i3.147-168

[29] Lichao Wu, Guilherme Perin, and Stjepan Picek. 2020. I Choose You: Automated Hyperparameter Tuning for Deep Learning-based Side-channel Analysis. *IACR Cryptol. ePrint Arch.* 2020 (2020), 1293.

[30] Lichao Wu, Guilherme Perin, and Stjepan Picek. 2022. The Best of Two Worlds: Deep Learning-assisted Template Attack. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2022, 3 (Jun. 2022), 413–437. https://doi.org/10.46586/tches.v2022.i3.413-437

[31] Lichao Wu, Yoo-Seung Won, Dirmanto Jap, Guilherme Perin, Shivam Bhasin, and Stjepan Picek. 2021. Explain Some Noise: Ablation Analysis for Deep Learning-based Physical Side-channel Analysis. Cryptology ePrint Archive, Report 2021/717. https://eprint.iacr.org/2021/717.

[32] Trevor Yap, Adrien Benamira, Shivam Bhasin, and Thomas Peyrin. 2022. Peek into the Black-Box: Interpretable Neural Network using SAT Equations in Side-Channel Analysis. Cryptology ePrint Archive, Paper 2022/1247. https://eprint.iacr.org/2022/1247 https://eprint.iacr.org/2022/1247.

[33] Gabriel Zaid, Lilian Bossuet, Mathieu Carbone, Amaury Habrard, and Alexandre Venelli. 2022. Conditional Variational AutoEncoder based on Stochastic Attack. Cryptology ePrint Archive, Report 2022/232. https://ia.cr/2022/232.

[34] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. 2019. Methodology for Efficient CNN Architectures in Profiling Attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020, 1 (Nov. 2019), 1–36. https://doi.org/10.13154/tches.v2020.i1.1-36

[35] Matthew D Zeiler and Rob Fergus. 2013. Visualizing and Understanding Convolutional Networks. https://doi.org/10.48550/ARXIV.1311.2901

[36] Yingxian Zheng, Yongbin Zhou, Zhenmei Yu, Chengyu Hu, and Hailong Zhang. 2015. How to Compare Selections of Points of Interest for Side-Channel Distinguishers in Practice?. In *Information and Communications Security*, Lucas C. K. Hui, S. H. Qing, Elaine Shi, and S. M. Yiu (Eds.). Springer International Publishing, Cham, 200–214.

A APPENDIX

A.1 Feature Selection

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

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

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

with $\bar{x}_{y_i}$ being 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) as

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

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. The Pearson Correlation 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.

In [36], the authors considered Pearson Correlation as a feature selection method and compared it with other classical feature selection techniques. They observed that, in general, using Pearson Correlation as a feature selection is the best. Throughout this paper, when we use Pearson Correlation as a feature selection, we shall call it CPA.

A.2 Datasets

In this paper, we consider datasets running the Advanced Encryption Standard (AES). We focus on attacking a single byte of the secret key. Furthermore, we examine two common hypothetical leakage models used in SCA: the Identity (ID) and the Hamming Weight (HW) leakage model.

Chipwhisperer (CW). The Chipwhisperer dataset provides a standard comparison base for the evaluation of different algorithms [18]. The dataset we considered runs the unprotected AES-128 implementation on the Chipwhisperer CW308 Target. We shall 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 while there is a fixed key for the attack phase. For ASCADf and ASCADr, we use 45000 traces for profiling. In the attack phase, we used 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 used 10000 attack traces to obtain the relevant points.

A.3 Details on DNN Architectures

Throughout this section, we denote $classes = 9$ for the HW leakage model and $classes = 256$ for the ID leakage model, and "padding = same" is padding evenly left and right such that the output has the same dimension as the input. We use the same DNN architecture for the CW dataset for both HW and ID leakage models. The architecture is explained in Table 7.

Table 7: DNN architecture used for the CW dataset for both HW and ID leakage models.

Layers	Hyperparameters
Conv1D_1	Number of filters/channels out = 8, kernel size = 11, stride = 1, padding = same, activation = ReLU
Average Pooling	kernel size = 2, stride = 2
Linear Regression 1	features out = = 128, activation = ReLU
Linear Regression 2	features out = = 128,activation = ReLU
Linear Regression 3	features out = *classes*, activation = Softmax

Table 8: DNN architecture used for the ASCADf and ASCADr datasets.

Layers	Hyperparameters
Conv1D_1	Number of filters/channels out = 128, kernel size = 25, stride = 1, padding = same, activation = SeLU
Batch Normalization	
Average Pooling	kernel size = 25, stride = 25
Linear Regression 1	features out = 20, activation = SeLU
Linear Regression 2	features out = 15, activation = SeLU
Linear Regression 3	features out = *classes*, activation = Softmax

We used the same architecture for ASCADf and ASCADr (Table 8). For the AES_HD dataset, we use the same DNN architecture as in Zaid et al. [34]. Details are in Table 9.

Table 9: DNN architecture used for the AES_HD dataset.

Layers	Hyperparameters
Conv1D_1	Number of filters/channels out = 2, kernel size = 1, stride = 1, padding = same, activation = SeLU
Average Pooling	kernel size = 4, stride = 4
Linear Regression 1	features out = 15, activation = SeLU
Linear Regression 2	features out = 10, activation = SeLU
Linear Regression 3	features out = 4, activation = SeLU
Linear Regression 4	features out = *classes*, activation = Softmax

As for DNNs trained on the desynchronized datasets, we show the architectures in Tables 10 and 11 for ASCADf_desync50 and ASCADf_desync100, respectively.

Table 10: DNN architecture used for the ASCADf_desync50 dataset.

Layers	Hyperparameters
Conv1D_1	Number of filters/channels out = 8, kernel size = 34, stride = 17, padding = same, activation = SeLU
Max Pooling	kernel size = 2, stride = 2
Batch Normalization	
Linear Regression 1	features out = 400, activation = SeLU
Linear Regression 2	features out = 400, activation = SeLU
Linear Regression 3	features out = 400, activation = SeLU
Linear Regression 4	features out = 400, activation = SeLU
Linear Regression 5	features out = *classes*, activation = Softmax

Table 11: DNN architecture used for the ASCADf_desync100 dataset.

Layers	Hyperparameters
Conv1D_1	Number of filters/channels out = 12, kernel size = 30, stride = 15, padding = same, activation = SeLU
Max Pooling	kernel size = 2, stride = 2
Batch Normalization	
Linear Regression 1	features out = 300, activation = SeLU
Dropout rate = 0.05	
Linear Regression 2	features out = 300, activation = SeLU
Dropout rate = 0.05	
Linear Regression 3	features out = *classes*, activation = Softmax