Bias-variance Decomposition in Machine Learning-based Side-channel Analysis

Daan van der Valk[1], Stjepan Picek[1]

[1] Delft University of Technology, Delft, The Netherlands

Abstract. Machine learning techniques represent a powerful option in profiling side-channel analysis. Still, there are many settings where their performance is far from expected. In such occasions, it is very important to understand the difficulty of the problem and the behavior of the machine learning algorithm. To that end, one needs to investigate not only the performance of machine learning but also to provide insights into its explainability. One tool enabling us to do this is the bias–variance decomposition where we are able to decompose the predictive error into bias, variance, and noise. With this technique, we can analyze various scenarios and recognize what are the sources of problem difficulty and how additional measurements/features or more complex machine learning models can alleviate the problem. While such results are promising, there are still drawbacks since often it is not easy to connect the performance of side-channel attack and performance of a machine learning classifier as given by the bias-variance decomposition. In this paper, we propose a new tool for analyzing the performance of machine learning-based side-channel attacks – the Guessing Entropy Bias–Variance Decomposition. With it, we are able to better understand the performance of various machine learning techniques and understand how a change in a setting influences the performance of an attack. To validate our claims, we give extensive experimental results for a number of different settings.

Keywords: Side-channel analysis, Machine learning, Deep learning, Bias-variance decomposition, Loss function

1 Introduction

Profiling side-channel analysis (SCA) represents the most powerful side-channel attack as we assume an attacker in possession of a clone device which he uses to obtain knowledge about the device while running. Later, the attacker uses that knowledge to mount the attack on the actual target device. In the last decade, machine learning in profiling side-channel analysis transitioned from an exotic approach to a well-established technique for powerful analysis. The first results with machine learning were promising but template attack, as the most powerful one from the information theoretic point of view [CRR02], was the first choice for most of the SCA community. As researchers started to explore more diverse attack scenarios, they recognized settings were machine learning could perform as good as template attack or even surpass its performance. Recently, deep learning started attracting more attention in numerous research domains as well as in SCA. The results with deep learning were very interesting: such techniques could outperform all other methods in numerous scenarios where especially important (and relevant) cases are implementations protected with countermeasures. Yet, despite all these positive results, there is still much to be done and especially, understood. For instance, deep learning techniques have many hyper-parameters one needs to tune, which is a computationally intensive process with no guarantee that the best hyper-parameters are obtained. Comparing this with the simplicity of a template attack that has no hyper-parameters to tune makes the situation even more complex. Indeed, we are aware of powerful SCA techniques, but often we do not know how to employ them in the best possible way. Naturally, this is not something encountered only in SCA: other domains suffer from the same problems and also rely on extensive experimental procedures in order to obtain as good as possible results.

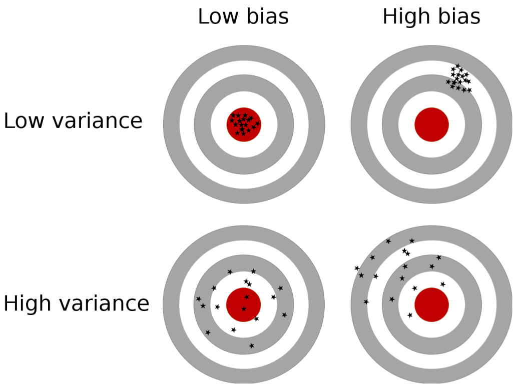


Figure 1: Bias and variance

A natural question is whether there are techniques that would allow us better insight into the performance of machine learning. While such insights are far from providing a complete picture, they can give some useful results and improve the process, which in turn can yield a more powerful attack. Additionally, we can increase the problem understanding and its difficulty for a certain machine learning algorithm.

One well-known and (relatively) successful technique for understanding the performance of machine learning is the bias-variance decomposition. There, we are able to decompose the predictive error of a machine learning algorithm into 3 parts: bias, variance, and noise. A common graphical depiction of bias and variance interplay is given in Figure 1. Up to now, in the SCA community, the bias-variance decomposition attracted only moderate attention. As far as we are aware, it was considered as a metric in profiling attacks [LBM15] or as a tool for leakage profiling [LVMS18]. While both examples gave promising results, we believe that the bias-variance decomposition can help much more in SCA.

In this paper, we investigate the bias-variance decomposition for machine learning techniques as applied in side-channel analysis. There, our results enable us to gain insight and provide answers for questions like 1) What is the influence of the number of features on the performance of the attack? 2) What is the influence of the number of training samples on the performance of the attack? 3) What is the influence of the machine learning model complexity? 4) Can we connect bias-variance decomposition with common SCA metrics like guessing entropy? While the first two questions are partially considered in related work, the second two, to the best of our knowledge, have not been considered before.

1.1 Contributions

The main contributions of this paper are:

1. We provide experimental results for the bias-variance decomposition with respect to the number of features, the number of training samples, and machine learning model complexity.
2. We show the connection between the mean squared error function and guessing entropy.
3. We introduce the new analysis technique that we call the guessing entropy bias-variance decomposition. With it, we are able to understand significantly more details

about the performance of machine learning in SCA and how various experimental settings influence the performance of the attack.

We conduct a detailed analysis and consider 4 publicly available datasets, 2 leakage models, 3 machine learning techniques, and 3 experimental settings (model complexity, the number of features, and the number of training examples). Finally, in order to facilitate reproducible results, our code is open source [fs19].

1.2 Related Work

In the last decade or so, machine learning techniques transitioned a path from exotic side-channel techniques to a standard for conducting profiling side-channel attacks. There, common techniques to use are Random Forest [LPB+15, HPGM16], Support Vector Machines [HZ12, PHJ+17], multilayer perceptron [HPGM17, PHJ+19], and convolutional neural networks [MPP16, CDP17, KPH+18a]. Note that the last two techniques belong to the deep learning domain and they gained a significant attention in the last few years. Those works usually concentrate on the empirical evaluation of machine learning techniques; some also compare with template attacks. At the same time, efforts directed towards explaining the machine learning performance or providing a more general understanding of it are much more sparse [PHAR18].

Lerman, Bontempi, and Markowitch introduced the bias-variance decomposition in SCA as they explored it as a metric in profiling attacks [LBM15]. They considered template attack and stochastic attack in several scenarios like the number of profiling traces, the number of informative points in traces, etc. Lerman et al. studied the curse of dimensionality in SCA and they applied the bias-variance decomposition in order to understand when template attack can outperform machine learning and vice versa [LPMS18]. They considered Random Forest and explored the influence of irrelevant features. Finally, Lerman et al. considered bias-variance decomposition as a tool for leakage profiling in order to improve the performance of the evaluation process [LVMS18]. They investigated 5 different settings in experiments with unprotected devices and 4 different settings in experiments with protected devices. They concluded their technique can be useful in practice when the noise is not too high.

2 Background

In this section, we start by briefly discussing profiling side-channel analysis. Next, we provide details on machine learning techniques we use in our experiments. Finally, we formally introduce the bias-variance decomposition.

2.1 Profiling Side-channel Analysis

We consider a scenario where a powerful attacker has a clone device that is of the same type as the device to be attacked. Then, he can use that clone device to obtain knowledge he later uses to attack the target device. Let calligraphic letters ($\mathcal{X}$) denote sets, capital letters (X) denote random variables over $\mathcal{X}$, and the corresponding lowercase letters (x) denote their realizations. The symbol $E_z[]$ denotes the expected value over z and symbol P_z the probability over z. From a profiling device, the attacker is able to obtain a set of N profiling traces T_p in order to characterize the leakage. By combining the profiling traces and a known secret key k_p^*, the attacker can calculate the leakage model $Y(T_p, k_p^*)$. Since the attacker has N profiling traces, he is able to obtain N pairs (x_i, y_i) to build the attack model f. Here, y_i denotes the label corresponding to trace x_i; we omit the index i when possible. Additionally, each trace x consists of n points of interests (features).

Once this phase is finished, the attacker measures additional M traces T_a from the device under attack and uses the function f in order to obtain the unknown secret key k_a^*. More specifically, for each attack trace, the attacker uses $f(x)$ to either predict the most likely class $\hat{y}$, or predicts a probability vector for all classes $\hat{\mathbf{p}}(x) = [\hat{p}_0, \ldots, \hat{p}_{|\mathcal{Y}|-1}]$ with $|\mathcal{Y}|$ the number of classes in the leakage model and $\sum_{j=0}^{|\mathcal{Y}|-1} \hat{p}_j = 1$. The outputted classes or class probabilities are used to obtain information about k_a^*. Although it is usually assumed that the attacker has an unlimited number of traces available during the profiling phase, this is of course not true and the attacker is bounded due to practical limitations.

When profiling the leakage, one must select the appropriate leakage model, which will determine the number of possible outputs. The Hamming weight (HW) and Hamming distance (HD) leakage models are often used as they have a reduced number of classes, which results in reduced training complexity. Unfortunately, the classes obtained from those models are imbalanced, which can cause problems for profiling SCAs [PHJ+18]. A common alternative is to profile directly on the intermediate value which requires a bigger profiling set but does not suffer from imbalance. When considering AES, HW results in $|\mathcal{Y}| = 9$ classes and intermediate value results in $|\mathcal{Y}| = 256$ classes as the attack is computed byte-wise.

To evaluate the performance of the attack, a common option is to use Guessing entropy (GE) [SMY09]. The guessing entropy metric gives the average number of key candidates the attacker needs to test to reveal the secret key after conducting a side-channel analysis. More formally, given T_a traces in the attacking phase, an attack outputs a key guessing vector $\mathbf{g} = [g_0, g_1, \ldots, g_{|\mathcal{K}|-1}]$ in decreasing order of probability where $|\mathcal{K}|$ denotes the size of the keyspace. To calculate the key guessing vector $\mathbf{g}$ over a number of samples, we use the following:

$$g_i = \sum_{j=1}^{T_a} \log(\hat{p}_{ij}), \tag{1}$$

where $\hat{p}_{ij}$ is the estimated probability for key candidate $i = \{1, \ldots, |\mathcal{K}|\}$ using sample j. Guessing entropy is the average position of k_a^* in $\mathbf{g}$ over a number of experiments. As we will investigate in this paper, guessing entropy is influenced by the profiling and attack sets that the attacker uses:

$$GE(T_p, T_a) = E_{T_p, T_a}[i] \text{ with } i \text{ such that } g_i = k_a^*. \tag{2}$$

2.2 Bias-Variance Decomposition

In bias-variance decomposition, it is possible to decompose the predictive error of a machine learning algorithm into 3 parts: bias, variance, and noise. The error occurring due to the bias is the difference between the expected prediction of the model and the correct value which we are trying to predict. The error occurring due to the variance is the variability of a model prediction for a given data point. Finally, the noise (or, more precisely irreducible noise) is the error we cannot reduce by any machine learning model. Consequently, the noise factor is often disregarded and the decomposition considers only bias and variance.

Having high bias means that the machine learning algorithm is not able to capture the relationships in data, and that the model underfits. Having high variance means that the machine learning algorithm learns to model the noise in the training data and will not generalize to new, unseen data. In other words, having a high variance means the algorithm overfits. The model will underfit if it is not complex enough and it will overfit if it is too complex, so we ideally want to find such a model that is complex enough to capture the relations in data but not so complex to overfit. We depict this trade-off in Figure 2. Naturally, finding it is complex and often not possible (or possible only under certain conditions).

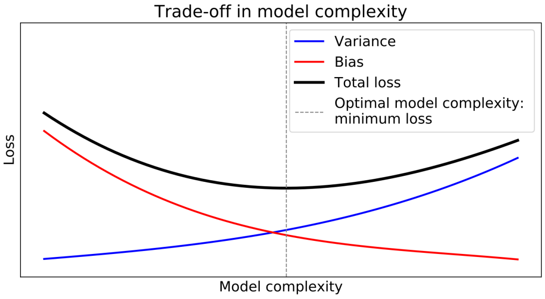


Figure 2: The trade-off between bias and variance

There are several models on how to conduct bias-variance decomposition but we follow the one from Domingos [Dom00]. First, we introduce the notation we follow. Let y denote the true value of a predicted label for a certain test example x. Then, the loss function $L(y, \hat{y})$ measures the cost of predicting $\hat{y}$ when the true label equals y. The aim is to minimize the loss, i.e., to find the models that minimize the average loss over all examples X. In this paper, we consider two loss functions: (mean) squared loss and $0-1$ loss. The squared loss is defined as:

$$L(y, \hat{y}) = (y - \hat{y})^2, \tag{3}$$

and $0-1$ loss as:

$$L(y, \hat{y}) = \begin{cases} 0, & \text{if} \quad \hat{y} = y \\ 1, & \text{otherwise.} \end{cases} \tag{4}$$

Next, we define the optimal prediction y^* for an example x as the prediction $E_y[L(y, y^*)]$ that minimizes the expected loss over all possible values y weighted by their probabilities given x. Consequently, we see that a machine learning model aims to learn the mapping $f(x) = y^*$ as good as possible. Additionally, we must account for the fact that the same machine learning algorithm will produce different models for different training sets. There, the easiest solution is to simply average over all training sets D. Finally, we end up with the expected loss $E_{D,y}[L(y, \hat{y})]$ that we want to minimize. Next, we define the main prediction for a loss function L and training sets D as $y_m^{L,D} = argmin_{y'} E_D[L(\hat{y}, y^{'})]$. The main prediction is the value $y^{'}$ whose average loss relative to all predictions from Y is minimum. Finally, we are ready to define the bias of a learner on x as $B(x) = L(y^*, y_m)$, i.e., the loss due to the main prediction relative to the optimal prediction. The variance of a learner on example x is $V(x) = E_D[L(y_m, \hat{y})]$, i.e., the average loss due to the predictions relative to the main prediction. The noise on x is $N(x) = E_y[L(y, y^*)]$ where we see it does not depend on the learner.

The decomposition of $E_{D,y}[L(y, \hat{y})]$ for a loss function L equals:

$$\begin{aligned} E_{D,y}[L(y, \hat{y})] &= c_1 E_y[L(y, y^*)] + L(y^*, y_m) + c_2 E_D[L(y_m, \hat{y})] \\ &= c_1 N(x) + B(x) + c_2 V(x). \end{aligned} \tag{5}$$

Here, c_1 and c_2 denote the multiplicative factors that differ depending on the loss function. In the case of squared loss, then $c_1 = c_2 = 1$. In the case of $0-1$ loss, $c_1 = P_D(\hat{y} = y^*) - P_D(\hat{y} \neq y^*) P_t(\hat{y} = y | y^* \neq y)$, and $c_2 = 1$ if $y_m = y^*$ and $-P_D(\hat{y} = y^* | \hat{y} \neq y_m)$ otherwise. Effectively, we can distinguish between the unbiased variance V_u and biased variance V_b. When the main prediction is correct ($y_m = y$), all predictions deviating from the main prediction contribute to the loss in V_u. Alternatively, if the main prediction is wrong ($y_m \neq y$), only the predictions indicating the right class count towards reducing the loss via the biased variance V_b.

2.3 Machine Learning Methods

In our experiments, we experiment with one machine learning algorithm that uses the $0-1$ loss function (Random Forest) and two algorithms that use the mean squared error loss function (multilayer perceptron and convolutional neural network).

Random Forest The Random Forest (RF) algorithm is an ensemble decision tree learner [Bre01]. Here, ensemble classifier means it consists of a number of simpler classifier techniques (decision trees). Decision trees choose their splitting attributes from a random subset of k attributes at each internal node. The best split is taken among these randomly chosen attributes and the trees are built without pruning.RF is a stochastic algorithm because of its two sources of randomness: bootstrap sampling and attribute selection at node splitting.

Multilayer Perceptron The multilayer perceptron (MLP) algorithm is a feed-forward neural network that maps sets of inputs onto sets of appropriate outputs. MLP consists of multiple layers (at least three) of nodes in a directed graph, where each layer is fully connected to the next one and training of the network is done with the backpropagation algorithm. The first layer is always the input layer (inputs are features) and the last layer is the output layer (outputs are classes). Any layer in between the input and output layers is referred to as a *hidden layer*.

Convolutional Neural Networks Convolutional neural networks (CNNs) represent a type of neural networks which were first designed for 2-dimensional convolutions as it was inspired by the biological processes of animals' visual cortex [LB+95]. They are primarily used for image classification but lately, they have proven to be powerful classifiers for time series data such as music and speech [ODZ+16]. From the operational perspective, CNNs are similar to ordinary neural networks (e.g., multilayer perceptron): they consist of a number of layers where each layer is made up of neurons. CNNs use three main types of layers: convolutional layers, pooling layers, and fully-connected layers. a convolutional neural network is a sequence of layers, and every layer of a network transforms one volume of activation functions to another through a differentiable function. First, input holds the raw features. Next, convolution layer computes the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. ReLU layer will apply an element-wise activation function, such as the $max(0, x)$ thresholding at zero. Max Pooling performs a down-sampling operation along the spatial dimensions. Finally, the fully-connected layer computes either the hidden activations or the class scores. Batch normalization is used to normalize the input layer by adjusting and scaling the activations after applying standard scaling using running mean and standard deviation.

3 Experimental Setting

In this section, we start by briefly discussing the datasets we use. Afterward, we discuss data pre-processing, bootstrap algorithm, and hyper-parameter tuning.

3.1 Datasets

In our experiments, we use 4 datasets corresponding to characteristic cases: measurements with low noise, measurements with high noise, measurements protected with a random delay countermeasure, and measurements protected with a masking countermeasure. Note that all datasets are publicly available and with results reported in related work. All experiments consider the AES algorithm and either the HW/HD leakage model or the intermediate value model. In HW/HD model, there are in total 9 possible classes and in the intermediate model, there are 256 classes. We denote plaintext with PT and ciphertext with CT.

DPAcontest v4 DPAcontest v4 contains masked AES software implementation measurements [TEL14]. Since the masking leaks the first-order information [MGH14], we can consider the mask to be known and turn the implementation into an unprotected scenario. The most leaking operation in this implementation is the processing of the S-box operation. We attack the first round and our leakage model equals:

$$Y(k^*) = \texttt{Sbox}[PT_1 \oplus k^*] \oplus \underbrace{M}_{\text{known mask}}, \tag{6}$$

where P_1 is the first plaintext byte. The measured signal to noise ratio (SNR) equals 5.8577. The measurements have 4 000 features around the S-box part of the algorithm execution and in total there are 100 000 traces available.

Unprotected AES-128 on FPGA (AES_HD) AES-128 core was written in VHDL in a round based architecture, which takes 11 clock cycles for each encryption. The core is wrapped around by a UART module to enable external communication. It is designed to allow accelerated measurements to avoid any DC shift due to environmental variation over prolonged measurements. The total area footprint of the design contains 1 850 LUT and 742 flip-flops. The design was implemented on Xilinx Virtex-5 FPGA of a SASEBO GII evaluation board. Side-channel traces were measured using a high sensitivity near-field EM probe, placed over a decoupling capacitor on the power line. Measurements were sampled on the Teledyne LeCroy Waverunner 610zi oscilloscope. We attack the register writing in the last round [TEL14]:

$$Y(k^*) = \underbrace{\texttt{Sbox}^{-1}[CT_i \oplus k^*]}_{\text{previous register value}} \oplus \underbrace{CT_j}_{\text{ciphertext byte}}, \tag{7}$$

where CT_i and CT_j are two ciphertext bytes, and the relation between i and j is given through the inverse ShiftRows operation of AES. We use $i = 12$ resulting in $j = 8$ as it is one of the easiest bytes to attack. The model-based SNR has a maximum value of 0.0096. Each trace has 1 250 features, and in total, there are 1 000 000 traces.This dataset is available at https://github.com/AESHD/AES_HD_Dataset.

Random Delay Countermeasure (AES_RD) The target smartcard is an 8-bit Atmel AVR microcontroller where the protection uses random delay countermeasure as described by Coron and Kizhvatov [CK09]. Adding random delays to the normal operation of a cryptographic algorithm has as an effect on the misalignment of important features, which in turns makes the attack more difficult to conduct. As a result, the overall SNR is reduced. We attack the first AES key byte targeting the first S-box operation:

$$Y(k^*) = \texttt{Sbox}[PT_1 \oplus k^*]. \tag{8}$$

The SNR has a maximum value of 0.0556. Each trace has 3 500 features and in total there are 50 000 traces.Recently, this countermeasure was shown to be prone to deep learning based side-channel [CDP17]. However, since its quite often used countermeasure in the commercial products, while not modifying the leakage order (like masking), we use it as a target case study. This dataset is available at https://github.com/ikizhvatov/randomdelays-traces.

ASCAD The target platform is an 8-bit AVR microcontroller (ATmega8515) running a masked AES-128 implementation and measurements are made using electromagnetic emanation [PSB+18]. The dataset follows the MNIST database and provides 60 000 traces, where originally 50 000 traces were used for profiling/training and 10 000 for testing. We use the raw traces and use the pre-selected window of 700 relevant samples per trace corresponding to masked S-box for the third plaintext byte:

$$Y(k^*) = \texttt{Sbox}[PT_3 \oplus k^*]. \tag{9}$$

The SNR for the ASCAD dataset is ≈ 0.8 under the assumption we know the mask while it is almost 0 with the unknown mask. This dataset is available at https://github.com/ANSSI-FR/ASCAD.

3.2 Data Pre-Processing

The datasets described in Section 3.1 have various numbers of features and data formats. In order to compare the classifiers, we have pre-processed the data to a unified format. First, all features were normalized by subtracting the mean $\bar{x}$ and dividing by the standard deviation σ:

$$x' = \frac{x - \bar{x}}{\sigma} \tag{10}$$

After scaling, we have new mean $\bar{x}' = 0$ and standard deviation $\sigma' = 1$.

As explained in Section 2, classifiers suffer from the curse of dimensionality [Bel03]. In particular, for the Random Forest and MLP classifiers, a smaller number of features is beneficial in order to improve the classification score. For each dataset, we selected 200 features for the HW model and 200 features for the intermediate value model, based on the Pearson correlation.

Pearson correlation coefficient measures linear dependence between two variables, x and y, in the range $[-1, 1]$, where 1 is the total positive linear correlation, 0 is no linear correlation, and -1 is the total negative linear correlation. Pearson correlation for a sample of the entire population is defined by [JWHT01]:

$$Pearson(x, y) = \frac{\sum_{i=1}^{N}((x_i - \bar{x})(y_i - \bar{y}))}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{N}(y_i - \bar{y})^2}}. \tag{11}$$

Note, in order to use the convolutional shift of CNNs, we do not require feature selection and consequently, all features are used as input for the CNNs.

3.3 Bootstrap

Bootstrapping is a sampling method based on drawing with replacement [Efr79]. It is widely used to improve distribution approximations and it allows the estimation of measures of accuracy, such as bias and variance. If the original dataset X consists of N samples (e.g., $X = (x_1, \ldots, x_N)$), a bootstrap sample would be N^* samples of X randomly drawn with replacement: $X^* = (x_1^*, \ldots, x_{N^*}^*)$; by convention, $N^* = N$. For a large enough N, this means the probability of bootstrapping to the exact same dataset is extremely small: in practice some traces will appear multiple times, and others not at all, in X^*. When drawing multiple bootstrapped sets, we refer to such sets as $X^{1*} = (x_1^{1*}, \ldots, x_{N^*}^{1*})$, $X^{2*} = (x_1^{2*}, \ldots, x_{N^*}^{2*})$, etc.

We use bootstrapping to create different training sets for classifiers. Compared to Efron's method, we made the following adjustments to the algorithm:

- Classifying techniques need at least one example from each class (e.g., intermediate value or the Hamming weight). As the classifier is trained using one bootstrap sample, we must ensure all classes are represented in each bootstrap sample. Formally, with $y(x_j)$ indicating the true class label of sample x_j, we require all classes are represented in the set $\{y(x_1^{i*}),\ldots,y(x_{N^*}^{i*})\}$ for each bootstrap sample X^{i*}. We redraw until this condition is satisfied.

- In the experiments where we varied the number of training samples, we do not enforce that the bootstrapped sets have the same size as the original sets. In this case, we denote N^* as the parameter we vary, which can be smaller or greater than N. In all other experiments, we have $N^* = N$ so the training set size for each classifier is equal to the original training set size.

The procedure is described in Algorithm 1.

Algorithm 1: Generating l bootstrapped sets of size N^* from dataset X .

```
for $i = 1$ to $l$ do
  repeat
    for $j = 1$ to $N^*$ do
      $x_j^{i*} \leftarrow$ random sample from $X$
    end for
  until $\{y(x_1^{i*}),\ldots,y(x_{N^*}^{i*})\}$ contains all classes
  $X^{i*} = (x_1^{i*},\ldots,x_{N^*}^{i*})$
end for
return $X^{1*},\ldots,X^{l*}$
```

To get an accuracy depiction of the estimated parameters (i.e., loss components), a high number of bootstraps is required. For each setting with Random Forest classifiers, we took $l = 100$ bootstraps. For neural networks, this number of experiments is unfeasible: for each time a parameter is changed, l networks need to be trained. Therefore, we ran experiments with MLPs and CNNs using $l = 10$ bootstraps.

3.4 MLP and CNN Architectures and Hyper-parameters

As described in Sections 2.3 and 2.3, multilayer perceptron (MLP) and convolutional neural network (CNN) are feed-forward neural network paradigms. Despite their popularity in recent years, there is no clear-cut methodology to select a specific architecture for a certain classification problem. Therefore, we employ 2 network structures (one MLP and one CNN) that were shown to be effective in literature [PSB+18,KPH+18b].

We designed an MLP network based on [PSB+18], in which the hidden layers consist of 200 neurons each, and use the rectified linear unit (ReLU) activation. The output layer consists of either 9 or 256 neurons, representing the classes in the HW and intermediate value, respectively, and is normalized using the softmax activation. We investigated the influence of complexity by varying the number of hidden layers from 0 to 5. For the experiments varying the training data size in terms of features and traces, we used a fixed size of 2 hidden layers.

For the CNN, we base our architecture on [KPH+18b], in which a *VGG-like* design is proposed. This style is based on multiple convolutional blocks, with convolutional layers with an increasing number of filters (8, 16, 32, 64 (twice), 128 (twice)). The filter size is 3, with strides 1. After the convolutional layer(s) in a convolutional block, we employ Batch Normalization. To allow for a large number of experiments, we modified this design by adding max pooling layers after each convolutional layer. This decreases the intermediate dimensions, and thus the total number of parameters that must be optimized in the network. After the last convolutional block, the output is flattened and fed into a fully connected layer of 512 neurons, before reaching the output layer. To avoid overfitting, we employ dropout before the fully connected layer and the output layer. We apply ReLu activation for all convolutional layers and fully connected layer, while the output layer is again normalized using softmax. In our experiments on model complexity, we start without any convolution, then add one convolutional block at the time. For all other experiments, we use the model involving 2 convolutional blocks.

When training the neural networks, we attempt to minimize the mean squared error as loss. We employ a batch size of 64, and train for 50 epochs using the Adam optimizer with a learning rate of 0.0001.

4 Bias-Variance Decomposition Results

In this section, first we show how mean squared loss and guessing entropy are connected. Next, we give results for the bias-variance decomposition for model complexity, number of features, and number of measurements. Due to the lack of space, we show only characteristic cases where we concentrate on model complexity as a scenario not considered before in related works. Differing from the usual depiction of the bias-variance decomposition, we additionally also show the average accuracy. Accuracy is defined as the ratio of traces in the attack set that were classified correctly.

4.1 Mean Squared Loss and Guessing Entropy

While the $0-1$ loss function is addressed in related works, see e.g., [LBM15, LPMS18], to the best of our knowledge, the mean squared loss (MSE) was not up to now investigated in the context of SCA and bias-variance decomposition. We give additional information on the decomposition of error for MSE and then elaborate on why MSE is relevant in the SCA context due to its connection to guessing entropy. The squared loss function equals

$$L(y, \hat{y}) = (y - \hat{y})^2$$

and it can be averaged over N examples to arrive to the mean squared loss (alternatively, we can see this as the expected value of the set of examples). By substituting the $E_{D,y}[L(y,\hat{y})]$ from Eq. 5 with the squared loss function, we arrive to the expression:

$$E_{D,y}[(y - \hat{y})^2] = c_1 N(x) + B(x) + c_2 V(x). \tag{12}$$

As already stated, coefficients c_1 and c_2 equal 1 for squared loss function. The optimal prediction y^* is $E_y[y]$ and the main prediction y_m is $E_D[\hat{y}]$. Finally, Eq. (12) becomes:

$$E_{D,y}[(y - \hat{y})^2] = E_y[(y - E_y[y])^2] + (E_y[y] - E_D[\hat{y}])^2 + E_D[(E_D[\hat{y}] - \hat{y})^2]. \tag{13}$$

Note that the bias term $(E_y[y] - E_D[\hat{y}])^2$ is squared so it is also often called bias squared. Still, we follow Domingos' notation and refer it as bias [Dom00].

Next, we derive the connection between the MSE and guessing entropy. The mean squared error loss function is commonly used for training feed-forward neural networks[1]. There, the loss is defined as follows:

$$MSE = \frac{1}{N} \sum_{i=0}^{N-1} (\mathbf{p}_i - \hat{\mathbf{p}}_i)^2 \tag{14}$$

where $\mathbf{p}_i$ is the hot-encoding of the correct class label y, and $\hat{\mathbf{p}}_i$ the predicted probabilities by some classifier, for a trace i. For the intermediate value model, there are 256 classes,

[1] A more common loss function is categorical cross-entropy since it gives less emphasis to incorrect predictions and training can be more efficient due to better weight adjustments during backpropagation. Still, we consider it as a very interesting choice since 1) there is a connection between guessing entropy and MSE, 2) with MSE, there is an easy decomposition into bias and variance, and 3) the results with MSE will often not differ much from those obtained with the cross-entropy loss function.

so $\mathbf{p}_i$, $\hat{\mathbf{p}}_i$ are 256-dimensional vectors with their entries non-negative and having a sum equal to 1. For example, if a trace i has the true class 0 and the classifier assigns equal probabilities to classes 0 and 1 (and zero probability for all other classes), we have:

$$\mathbf{p}_i = \begin{bmatrix} p_{i,0} \\ p_{i,1} \\ p_{i,2} \\ \vdots \\ p_{i,255} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \qquad \hat{\mathbf{p}}_i = \begin{bmatrix} \hat{p}_{i,0} \\ \hat{p}_{i,1} \\ \hat{p}_{i,2} \\ \vdots \\ \hat{p}_{i,255} \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

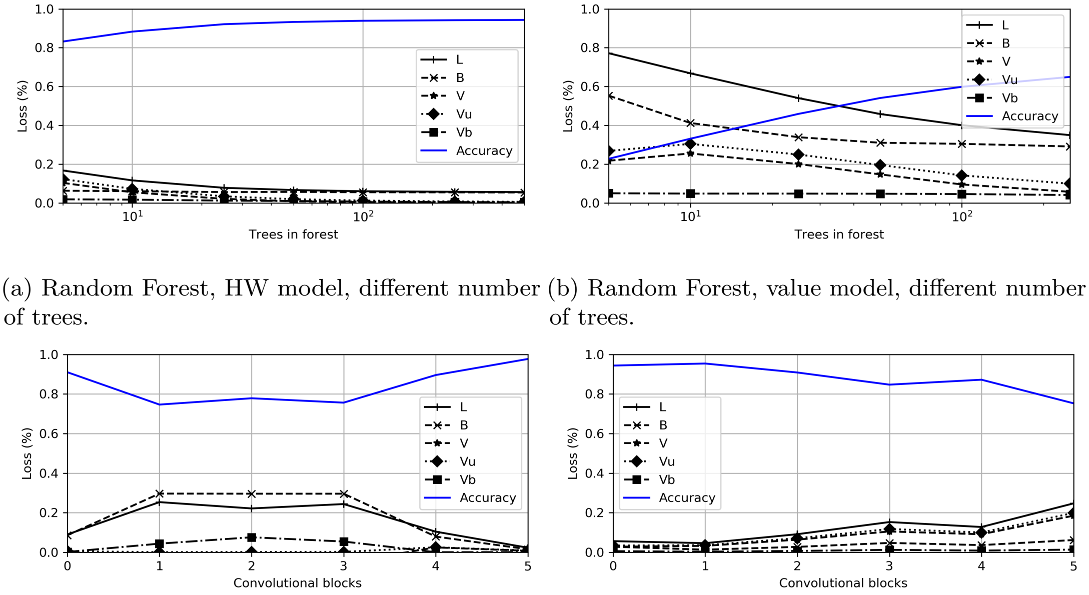
Notice that the class label represents the leakage of i according to the leakage model. For trace i with plaintext PT_i, one particular class label $\hat{y}$ corresponds to a key guess: by inverting the leakage model, the attack finds key guess $k = Y^{-1}(PT, \hat{y})$. Note Y^{-1} is bijective: when the plaintext is fixed, only one key guess refers to one leakage class. Thus, we can decompose the mean squared error into a term for each key guess:

$$\begin{aligned}
MSE &= \frac{1}{N}\sum_{i=0}^{N-1} \left(\mathbf{p}_i - \hat{\mathbf{p}}_i\right)^2 && (15)\\
&= \frac{1}{N}\sum_{i=0}^{N-1} \begin{bmatrix} p_{i,0} - \hat{p}_{i,0} \\ \vdots \\ p_{i,255} - \hat{p}_{i,255} \end{bmatrix}^2 && (16)\\
&= \frac{1}{N}\sum_{i=0}^{N-1}\sum_{j=0}^{255} \left(p_{i,j} - \hat{p}_{i,j}\right)^2 && (17)\\
&= \frac{1}{N}\sum_{j=0}^{255}\sum_{i=0}^{N-1} \left(p_{i,j} - \hat{p}_{i,j}\right)^2 && (18)\\
&= \frac{1}{N}\sum_{k=0}^{255}\left(\sum_{i=0}^{N-1} \left(p_{i,Y^{-1}(PT_i,j)} - \hat{p}_{i,Y^{-1}(PT_i,j)}\right)^2\right) && (19)\\
&= \sum_{k=0}^{255} MSE(k), && (20)\\
& && (21)
\end{aligned}$$

with $MSE(k) = \frac{1}{N}\sum_{i=0}^{N-1} \left(p_{i,Y^{-1}(PT_i,j)} - \hat{p}_{i,Y^{-1}(PT_i,j)}\right)^2$ being the attribution to the MSE of one particular key guess. Notice that for the correct key $k*$ the corresponding label $p_{i,Y^{-1}(PT_i,j)} = 1$ for all i, and for all other keys $Y_{i,Y^{-1}(PT_i,j)} = 0$ for all i. Consequently, when a classifier is trained to minimize mean squared error, this means it also minimizes the average key rank of the correct key. In this sense, MSE optimizes for guessing entropy.

4.2 Model Complexity

When discussing the model complexity, we consider the number of trees for Random Forest, the number of hidden layers for MLP, and the number of convolutional blocks for CNN. While there are other hyper-parameters we could investigate, we consider these to be the core ones and consequently, the most relevant ones to investigate.



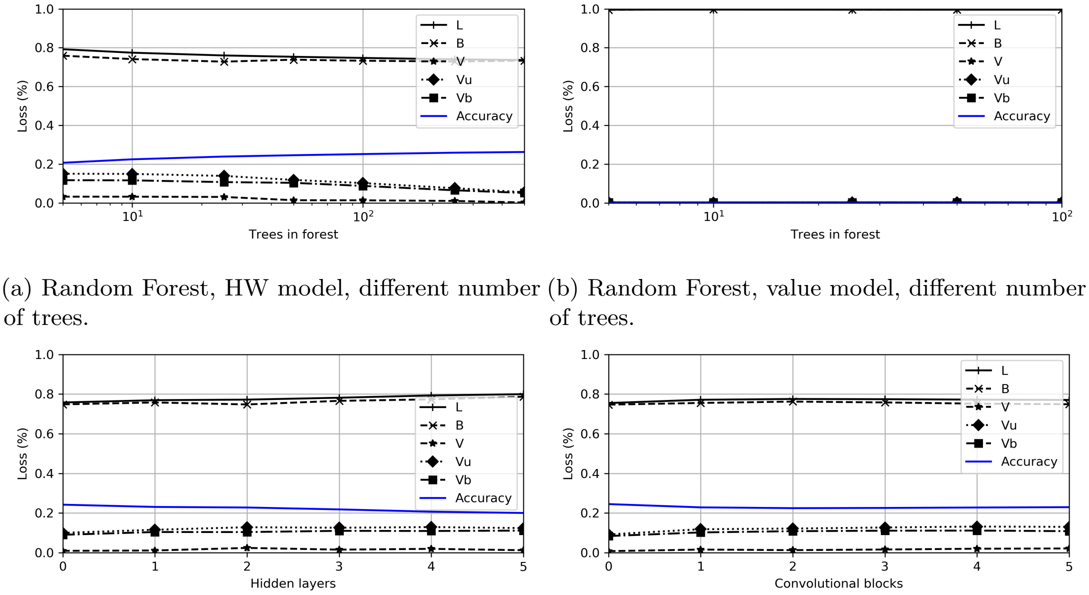
(a) Random Forest, HW model, different number of trees.

(b) Random Forest, value model, different number of trees.

(c) CNN, HW model, different number of convolutional blocks.

(d) CNN, value model, different number of convolutional blocks.

Figure 3: DPAv4 dataset results for model complexity.



(a) Random Forest, HW model, different number of trees.

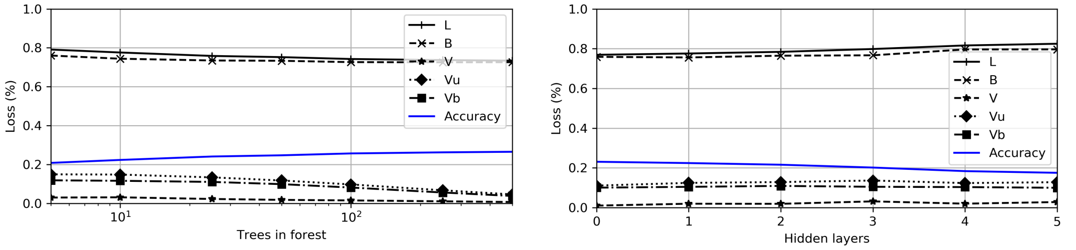
(b) Random Forest, value model, different number of trees.

(c) MLP, HW model, different number of hidden layers.

(d) CNN, HW model, different number of convolutional blocks.

Figure 4: AES_HD dataset results for model complexity.

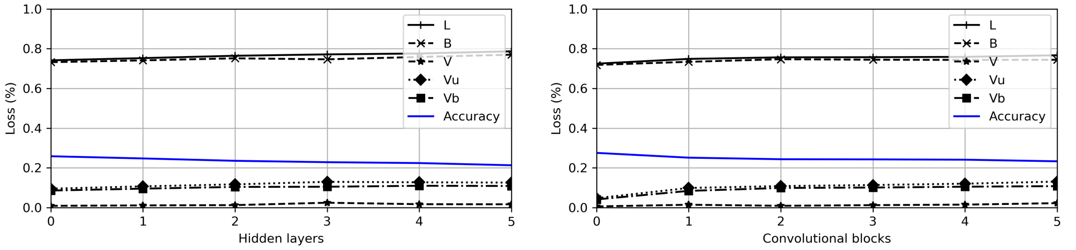
The first scenario we consider is DPAv4 as depicted in Figure 3. For Random Forest and the HW model, we see that even a small number of trees is enough for low loss (and high accuracy). Still, when increasing the number of trees, loss additionally reduces due to



(a) Random Forest, HW model, different number of trees.

(b) MLP, HW model, different number of hidden layers.

Figure 5: AES_RD dataset results for model complexity.



(a) MLP, HW model, different number of hidden layers.

(b) CNN, HW model, different number of convolutional blocks.

Figure 6: ASCAD dataset results for model complexity.

a reduction in variance (unbiased variance). For the value model, the decrease in bias is significant with the increase in the number of trees, which indicates that more trees are needed to capture the complexity of data and avoid underfitting. Besides bias, we can also notice a decrease in the unbiased variance. For CNN and HW model, we see first an increase in bias with increasing complexity. Note, even when there are no convolutional blocks, the CNN still includes a fully connected layer. Then, the bias becomes stable until we increase the number of convolutional blocks to more than 3 when bias starts to increase significantly. Notice the region between 1 and 3 convolutional layers where loss is lower than bias due to the increase in the biased variance. For the value model, loss increases with adding convolutional layers as we increase the unbiased variance, which is indicating that our model overfits and cannot adapt to new, previously unseen measurements.

For the AES_HD dataset (Figure 4), we see that with added trees for Random Forest, loss slowly decreases due to the increase in bias. This indicated the dataset is difficult for this classifier and we require a much larger number of trees to prevent underfitting. The scenario for Random Forest and value model is interesting as here we cannot discern much from the bias-variance decomposition. Indeed, bias is very high and our model does not learn anything about data so naturally, variance must be low. This is also a common result we obtain for most of the scenarios with the value model. Both MLP and CNN in HW model exhibit similar behavior: almost constant loss and bias regardless of adding model complexity. Biased and unbiased variance are almost the same and they slowly increase. This means our models overfit. When considering accuracy, we also see it does not change and upon closer inspection, we observe that these classifiers constructed models that classify all measurements into HW 4 class.

Next, in Figure 5, we depict results for the AES_RD dataset. Using more trees, Random Forest is able to improve performance by reducing bias. For MLP, we see increasing complexity has a counterproductive effect, increasing bias. This means that Random Forest benefits from added model complexity while for MLP, more hidden layers means it overfits.

Finally, in Figure 6, we depict results for the ASCAD dataset. We consider the HW model and MLP and CNN architectures. Their performance is very similar (differing in the loss value only slightly) and with added model complexity, variance increases due to overfitting.

In general, we see that adding complexity often manages to reduce loss by reducing bias. Variance is usually low, except when we use complex architectures and investigate easy datasets (DPAv4). This indicates that our considered models mostly suffered from underfitting. With these conclusions, we are able to obtain insights into the behavior of various classifiers and adapt them to reach better performance. Thus, the bias-variance decomposition is quite useful for adjusting the architecture of machine learning algorithms in SCA. Still, there is a number of scenarios where the bias-variance decomposition does not help: if a dataset is very difficult for a certain classifier, we see a very high bias (and low variance) but increasing the model complexity does not improve performance. Consequently, for such cases, we can conclude our models underfit (which is relevant), but unfortunately, we obtain no insight on whether more complex models help at all.

4.3 Number of Features

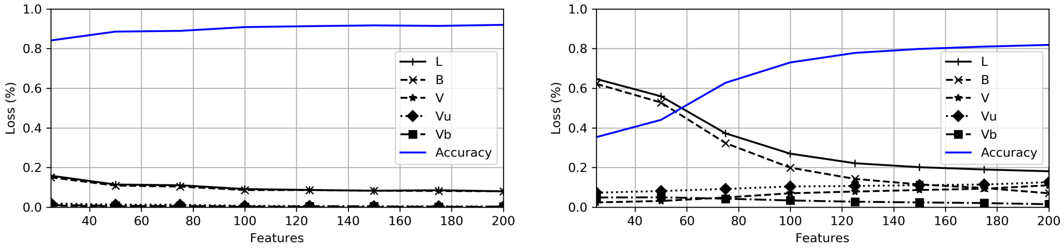
In Figure 7, we give results for DPAv4 and MLP when considering a different number of features. There, for the HW model, we see that adding features brings only a small improvement in performance due to a decrease in bias. Variance is almost at 0, which indicates this dataset does not overfit. In general, this is as good performance as one could expect. For the value model, we see that adding features improves the performance significantly as bias reduces. There is a point around 100 features when variance starts to increase, but that is negligible when compared to the bias decrease. As the value model is more complex than the HW model, more features help to capture the relations in data and reducing underfitting. Still, at 200 features bias and variance are quite close and one could expect that by adding more features we would soon arrive at a point where the model would start to overfit.

Figure 9 depicts results for AES_HD. Despite using two very different classifiers (that use different loss functions), the results are similar. Adding more features is not beneficial. This is because our model is not able to learn the relations in data and then, adding more features does not bring stronger models. Similar results are already seen in other scenarios and are common when considering datasets with countermeasures or a large amount of noise. Next, AES_RD exhibits almost identical behavior as seen in Figure 9. Unfortunately, on the basis of these results, it is difficult to say whether added features would help for AES_HD and AES_RD. Finally, for the ASCAD dataset as depicted in Figure 10 we see a small decrease in bias when adding features and using Random Forest. For MLP, no conclusion can be given besides the fact that there is a small increase in variance.

The benefit of added features is much less clear than the benefit of model complexity. This is not so surprising because it is to be expected that relevant information is included in the most important features. When the datasets are difficult to classify, then the performance is low as there is actually no learning happening so one naturally, one cannot expect benefit from introducing additional information in the form of added features.

4.4 Training Set Size

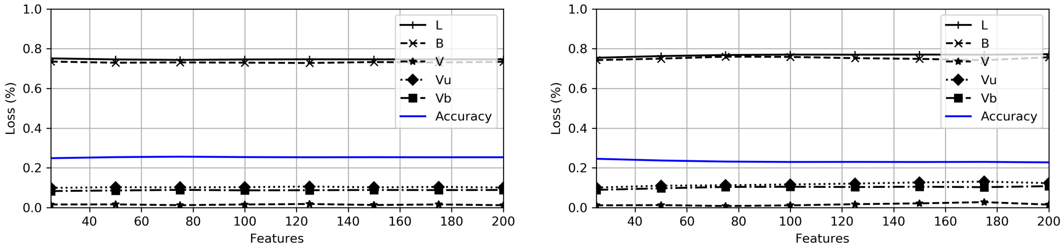
In Figure 11, we give results for DPAv4 when considering various training set sizes. For the HW model and CNN, we see that as the training set size increases, loss decreases due



(a) MLP, HW model, 2 hidden layers

(b) MLP, value model, 2 hidden layers

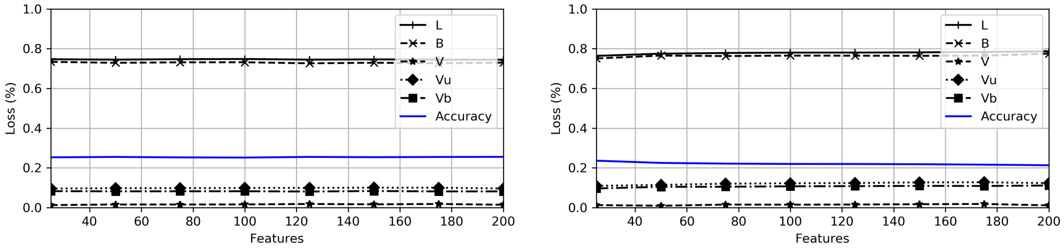
Figure 7: DPAv4 dataset results for different numbers of features.



(a) Random Forest, HW model, 100 trees.

(b) MLP, HW model, 2 hidden layers.

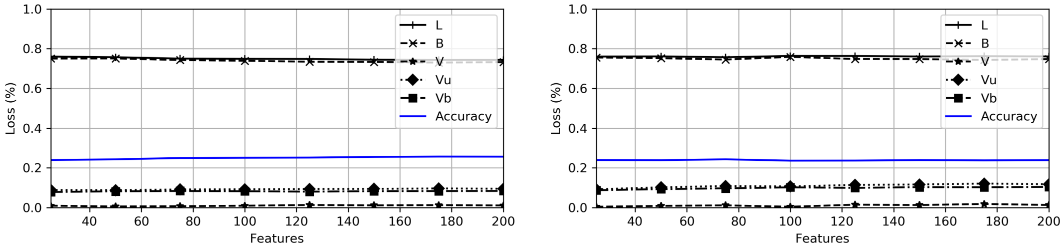
Figure 8: AES_HD dataset results for different numbers of features.



(a) Random Forest, HW model, 100 trees.

(b) MLP, HW model, 2 hidden layers.

Figure 9: AES_RD dataset results for different numbers of features.



(a) Random Forest, HW model, 100 trees.

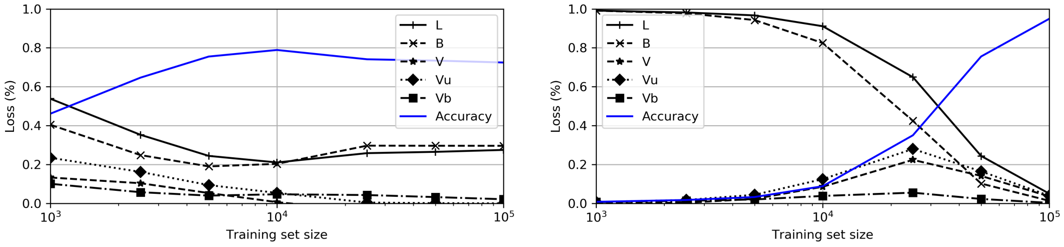
(b) MLP, HW model, 2 hidden layers.

Figure 10: ASCAD dataset results for different numbers of features.

to a decrease in bias. Then, around 10^4 training set size,we see that bias becomes larger than loss. This behavior happens due to an insufficient model complexity to model the data when there is a large number of measurements. Variance, on the other hand slowly decreases with the increase in the training set size. For the value model, we require much more data to build a strong model which is clear from the significant drop in bias when more than 10^4 measurements is used. With more measurements, also variance increases but it stays below bias, which indicates we require more complex model and more data for this scenario to improve even further.

All the other datasets, AES_HS in Figure 12, AES_RD in Figure 13, and ASCAD in Figure 14 show similar behavior. There, adding measurements does not decrease loss and it appears there is no benefit from more measurements. Naturally, the situation is not so simple as here we also need to consider what are the model complexities and how those can benefit from extra measurements.

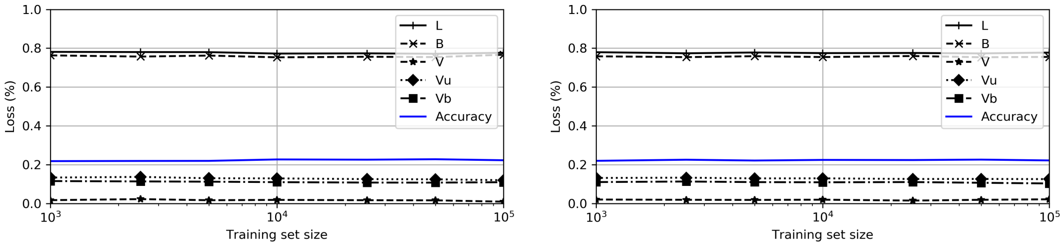
While it seems intuitive that the additional training measurements must help in the machine learning-based SCAs, we actually see here that it is often not possible to see the benefit of it if we do not also increase the model complexity. Consequently, here we observe a clear advantage of added measurements only for the easiest dataset (DPAv4) while for all other datasets we see a constant behavior with high bias, which means that all those models underfit.



(a) CNN, HW model, 2 convolutional blocks.

(b) CNN, value model, 2 convolutional blocks.

Figure 11: DPAv4 dataset results for different training set sizes.

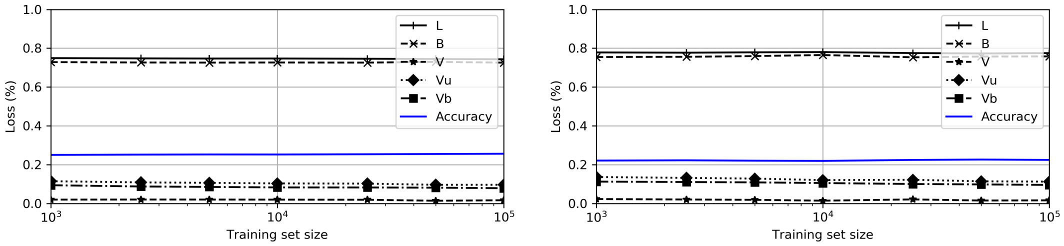


(a) MLP, HW model, 2 hidden layers.

(b) CNN, HW model, 2 convolutional blocks.

Figure 12: AES_HD dataset results for different training set sizes.

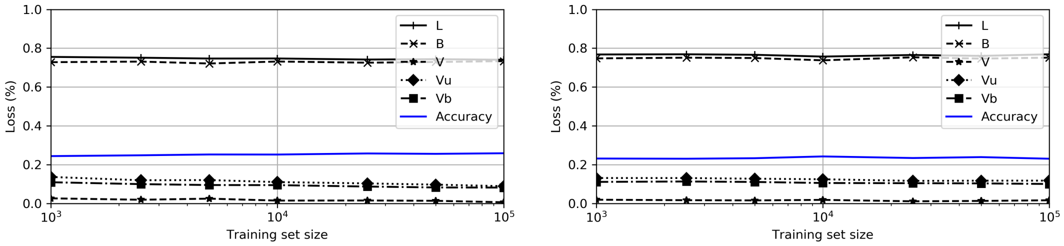
The bias-variance decomposition allows us insights into the behavior of various machine learning-based side-channel attacks. At the very minimum, we are able to discern whether the loss comes from bias or variance (neglecting the noise part). From there, we are able to estimate whether our models underfit or overfit, which then indicated how to improve the performance. In many cases, we can obtain even more information as we see the best trade-off between bias and variance. On the other hand, there are also many cases where the bias-variance decomposition is not helping. For instance, when the loss does



(a) Random Forest, HW model, 100 trees.

(b) CNN, HW model, 2 convolutional blocks.

Figure 13: AES_RD dataset results for different training set sizes.



(a) Random Forest, HW model, 100 trees.

(b) MLP, HW model, 2 hidden layers.

Figure 14: ASCAD dataset results for different training set sizes.

stays constant throughout the experiment. There, bias is usually high while variance is low, which means our model did not learn but we do not have any indication whether the change in parameters (e.g., number or features or measurements) help or not. Such scenarios are usually occurring either when the noise is high or there are countermeasures. Our results are also in accordance with findings in [LVMS18] despite the fact that there, the authors considered different datasets and machine learning algorithms. Additionally, in the value model, loss and bias are usually very high and we do not see much change in those values. On the other hand, in the HW model, the results are more revealing but we cannot directly connect the bias performance with the performance of side-channel attack (as for instance measured with guessing entropy).

5 Guessing Entropy Bias–Variance Decomposition

As discussed in previous section, while the bias-variance decomposition can give us valuable information about the performance of machine learning-based SCAs, there are also some drawbacks when using it, especially when considering specificity of a certain domain. Domingos' bias-variance decomposition has a pitfall in the side-channel domain: the metric is based on decomposition for individual measurements in a test set. This means only changes in the training data can be used to research a classifier's characteristics. It is closely related to the classification accuracy, only taking the assigned class label for each attack trace into account. In SCA, a single trace is typically not enough to find the correct key; therefore, the predicted class probabilities from the test set are combined. In some cases, this means the key can be determined even with a fairly low accuracy score. As Domingos' bias–variance decomposition is not useful in this case, we inspect the bias and variance of the classifiers' guessing entropy. This allows for analysis of a more practical setting: a scenario in which multiple traces are used to guess the key byte.

From the perspective of the attack, guessing entropy indicates the work to be done. In the ideal case, the guessing entropy is 0: the best-ranked key guess as attack output is correct key byte. As there are 256 possible key bytes, the worst possible GE would be 255. Notice that if a classifier consistently predicts the right class label for each trace, both the GE and the (Domingos') loss would be 0. If the correct label is consistently ranked second, the Domingos' bias would be maximal (i.e., 1) and classification accuracy would be 0%. However, when combining multiple outputted probabilities per class, the guessing entropy may drop after only a couple of traces.

In this section, we evaluate the guessing entropy for different classifiers. We consider the same scenarios as described in the previous section, but now they include the size of the attack set. For each variable we investigate (number of traces, number of features, complexity), we generate guessing entropy between 0 and 5 000 attack traces. Then, we define the *guessing entropy bias* as the expected guessing entropy based on profiling set T_p and attack set T_a:

$$B_{GE}(T_p, T_a) = E[GE(T_p, T_a)]. \tag{22}$$

Next, we define *guessing entropy variance* as the expected deviation from the GE mean:

$$V_{GE}(T_p, T_a) = E\Big[\big|GE(T_p, T_a) - E\left[GE(T_p, T_a)\right]\big|\Big]. \tag{23}$$

Note that we follow Domingos in taking the standard deviation (instead of the standard deviation squared) to define variance. Now, we can define a simple decomposition of guessing entropy loss:

$$L = B(T_p, T_a) + V(T_p, T_a) - V(T_p, T_a) = B + V_b - V_u. \tag{24}$$

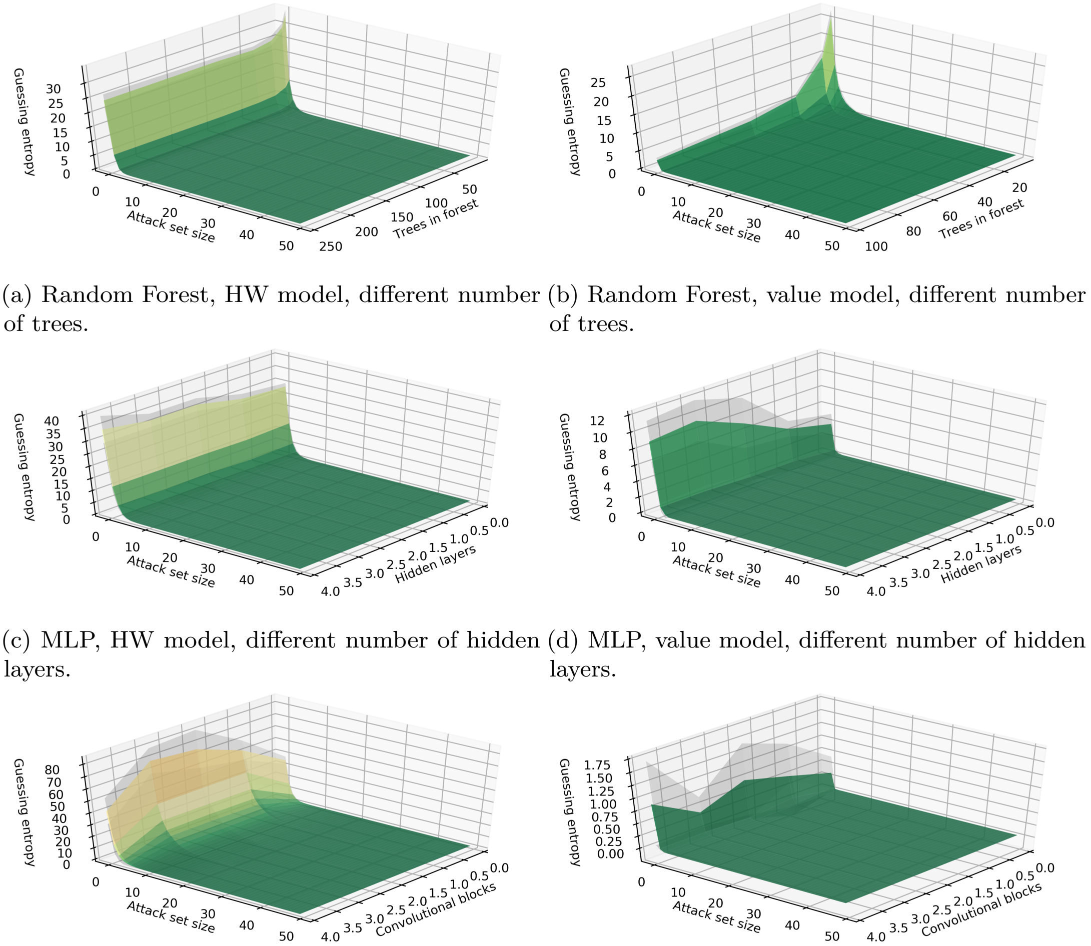
This decomposition is simple in the sense that the loss is equal to the bias, but allows us to analyze the guessing entropy taking into account both the average score and the deviations from it. As we will show in this section, this gives new insight in the performance of classifiers used in SCA. The 3D graphs in this section show the bias in colored surface plots, while the biased (V_b) and unbiased (V_u) variance are shown as grey surface plots added and subtracted from the bias, respectively.

We depict the results for the model complexity, different number of features and different number of training measurements. Note that for the model complexity, we depict all results, while for the other two scenarios, we select a few characteristic cases. We emphasize that differing from the bias-variance decomposition, guessing entropy bias-variance decomposition is able to offer insights for all considered scenarios.

5.1 Model Complexity

In Figure 15, we show the guessing entropy decomposition for different classifiers for the DPAv4 dataset. For the simplest dataset, we see no influence of model complexity to the guessing entropy: typically, only a handful (< 5) of traces suffice to find the correct key.

Figure 16 shows the influence of model complexity to guessing entropy when attacking the AES_HD dataset. We can immediately observe it is easier to attack using the HW model, compared to intermediate values. Additionally, increasing complexity is only beneficial for the Random Forest classifier. We see the average guessing entropy slowly approaches 0 when increasing the number of trees. Also, using more trees, we see a strong decline of variance. This can be explained by the fundamental randomness of the classifier: for each tree a random subset of features is kept. Thus, when using a small number of trees, the Random Forest's performance is very volatile based on the quality of selected features. For the MLP and CNN classifiers, we see a negative effect of increasing complexity. In particular, the MLP's smallest network (no hidden layers) performs best, while adding hidden layers quickly leads to overfitting. The CNN also suffers from overfitting when



(a) Random Forest, HW model, different number of trees.

(b) Random Forest, value model, different number of trees.

(c) MLP, HW model, different number of hidden layers.

(d) MLP, value model, different number of hidden layers.

(e) CNN, HW model, different number of convolutional blocks.

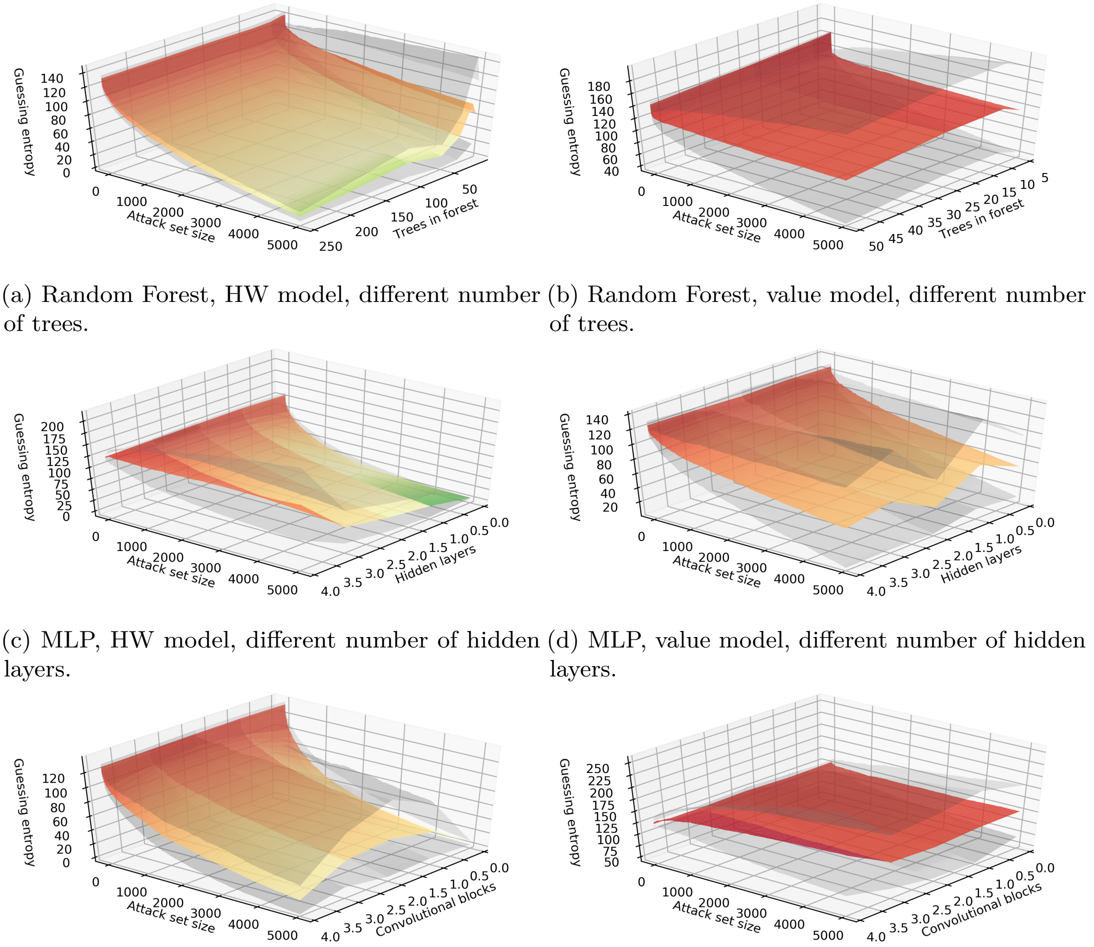
(f) CNN, value model, different number of convolutional blocks.

Figure 15: Guessing entropy decomposition, DPAv4 dataset results for model complexity.

increasing complexity, but then starts performing better after having more than 3 blocks. We observe a high variance, as well as a worse average performance, when having 1, 2, or 3 convolutional blocks.

For the AES_RD dataset, we see in Figure 17 that again it's easier to attack using the HW model, compared to intermediate values. As convolutional layers are invariant to shifts in input data, the CNN classifier has a clear advantage for this dataset. Indeed, we see that the only successful attack utilizes a CNN. In this case, a higher complexity is clearly beneficial, as a small number of convolutional blocks is not sufficient to reach a guessing entropy of 0 – but still outperforms the MLPs and Random Forests. For the other classifiers, we see no substantial influence of complexity on the guessing entropy bias and variance.

Finally, Figure 18 shows the results for the ASCAD dataset. Similar as for the AES_HD dataset, we observe that the Random Forest classifier benefits from higher complexity, while MLP and CNN benefit from a low complexity. Those classifiers perform best when having no hidden layers or no convolutional blocks, respectively. When increasing complexity, these models immediately start to overfit and require more traces to find the secret key. For more than 2 hidden layers in MLP, or more than 3 convolutional blocks in CNN, these



(a) Random Forest, HW model, different number of trees.

(b) Random Forest, value model, different number of trees.

(c) MLP, HW model, different number of hidden layers.

(d) MLP, value model, different number of hidden layers.

(e) CNN, HW model, different number of convolutional blocks.

(f) CNN, value model, different number of convolutional blocks.

Figure 16: Guessing entropy decomposition, AES_HD dataset results for model complexity.

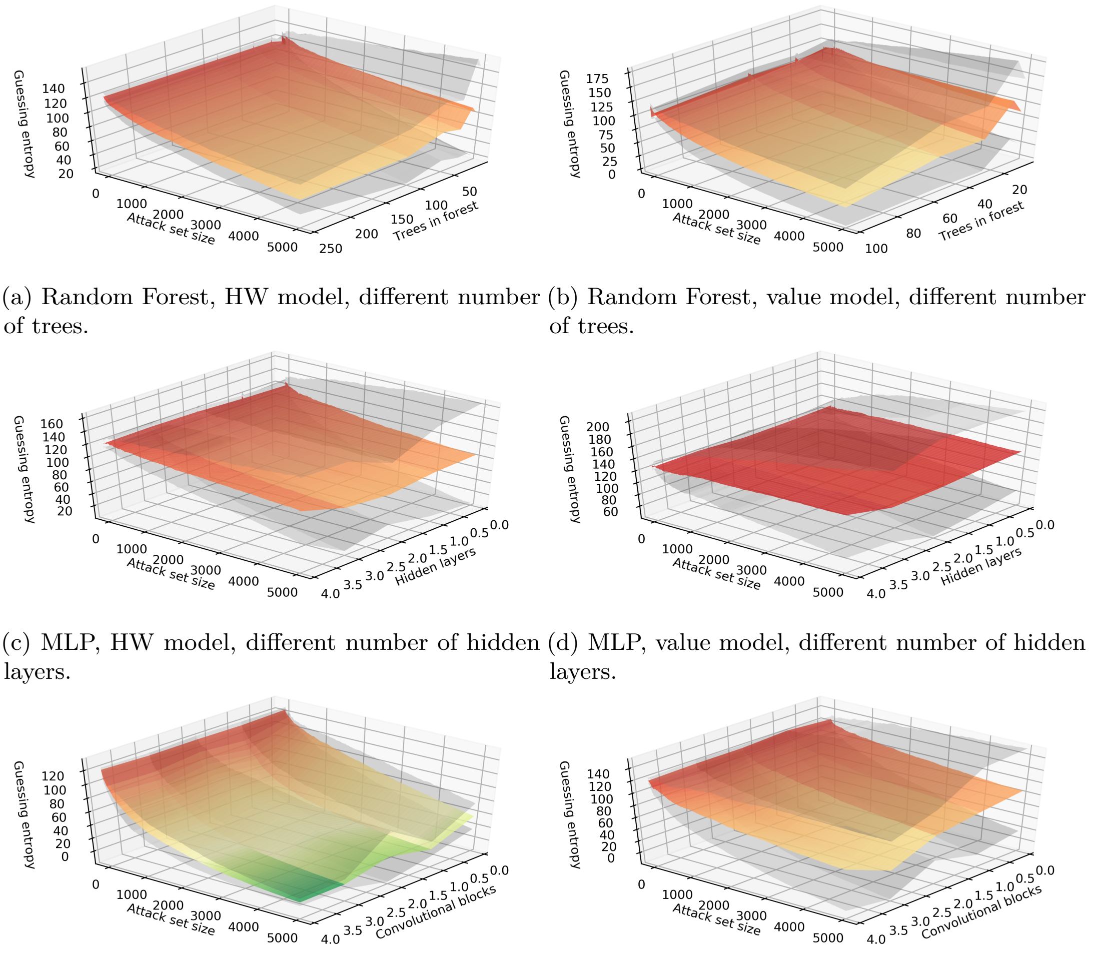
networks are unable to reach a GE of 0, even using 5 000 attack traces. In the intermediate value model, only MLP consistently attacks the data successfully. Both the Random Forest and CNN have a high bias and variance in this scenario.

5.2 Training Set Size

Figure 19 shows the relation between training set size for a CNN attack the AES_RD dataset. It is clear that the CNN needs a large dataset to conduct a successful attack.

Figure 20 shows the influence of the number of training traces used to attack the ASCAD dataset. In this scenario, having more traces clearly benefits the attacks in the HW model. All classifiers are able to determine the correct key byte when given 25 000 or more traces for training.

In the more difficult scenario of attacking the intermediate value, we see a positive correlation between training set size and guessing entropy for the neural networks. The MLP classifier is able to find the key byte when using the maximum number of traces, while the CNN achieves a guessing entropy of 30. For the Random Forest classifier, adding more traces does not seem to help: even for the largest training set, it performs only



(a) Random Forest, HW model, different number of trees.

(b) Random Forest, value model, different number of trees.

(c) MLP, HW model, different number of hidden layers.

(d) MLP, value model, different number of hidden layers.

(e) CNN, HW model, different number of convolutional blocks.

(f) CNN, value model, different number of convolutional blocks.

Figure 17: Guessing entropy decomposition, AES_RD dataset results for model complexity.

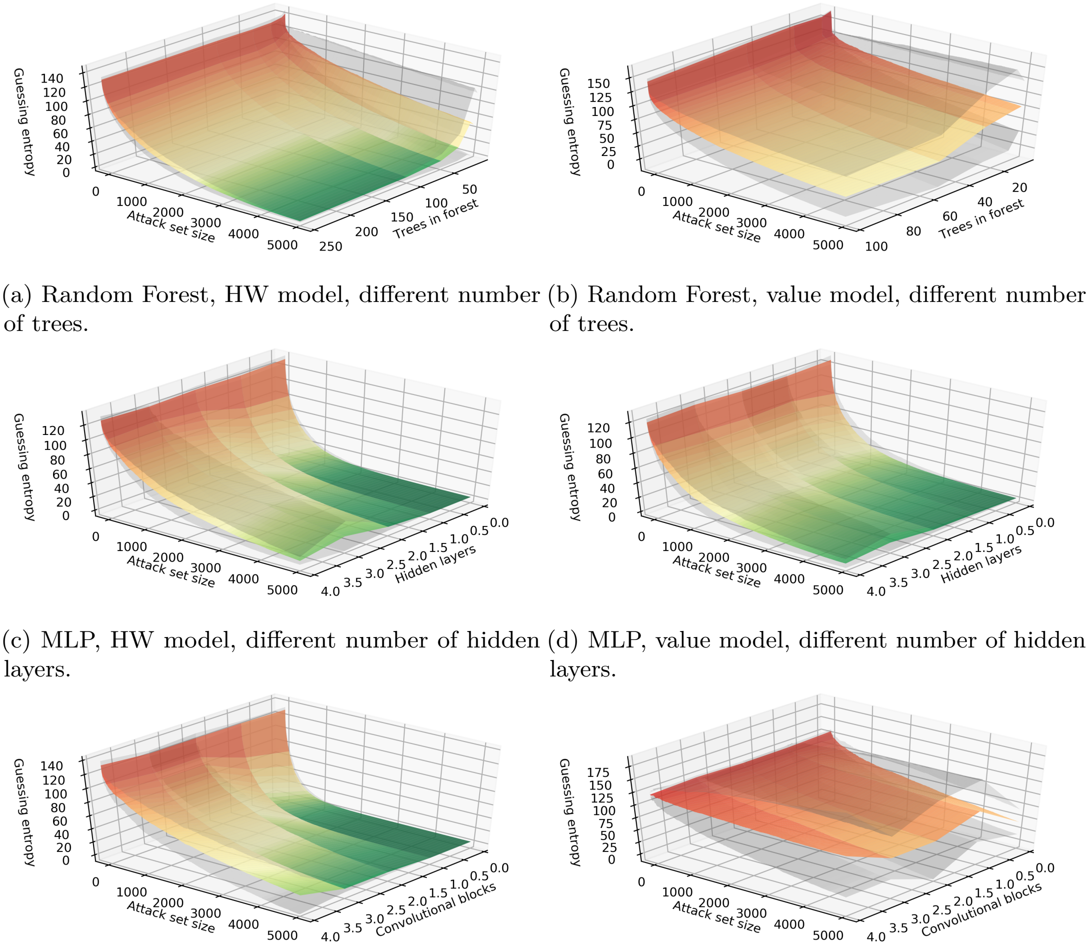
slightly better than random guessing (GE of 127.5).

Note that we omit results for other scenarios: in these settings, there was no significant relation between the training set size and the classifier's guessing entropy: either the classifier was able to attack successfully or not.

5.3 Number of Features

Figure 21 shows the influence of the number of features used to attack the AES_HD dataset in the HW model, for both the Random Forest and MLP classifier. We observe a small influence on the guessing entropy: selecting less features seem to benefit the GE slightly.

The effect of the number of features to attack ASCAD in the HW model is shown in Figure 22. Clearly, adding more features is highly beneficial in this scenario: when using the maximum number of features (200), the MLP attack is successful. The Random Forest also profits from more features, reaching a GE of 5 using 200 features and the full attack set. For the other datasets, we see no relation between number of features and the guessing entropy for the Random Forest and MLP classifiers. In DPAv4, the smallest



(a) Random Forest, HW model, different number of trees.

(b) Random Forest, value model, different number of trees.

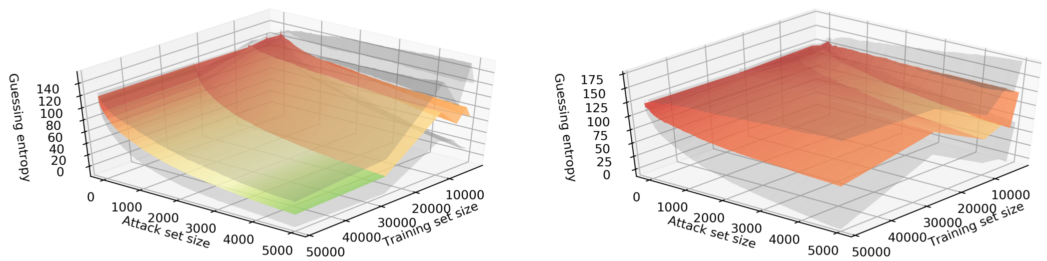
(c) MLP, HW model, different number of hidden layers.

(d) MLP, value model, different number of hidden layers.

(e) CNN, HW model, different number of convolutional blocks.

(f) CNN, value model, different number of convolutional blocks.

Figure 18: Guessing entropy decomposition, ASCAD dataset results for model complexity.

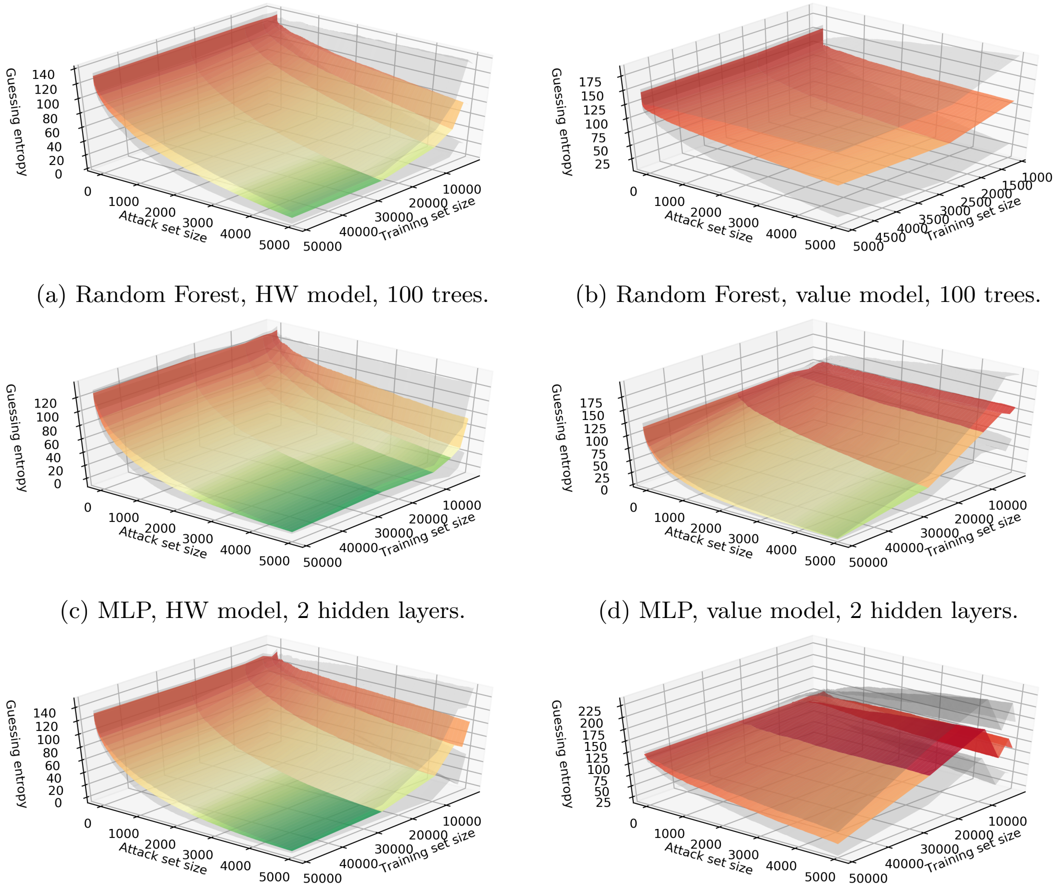


(a) CNN, HW model, 2 convolutional block.

(b) CNN, value model, 2 convolutional block.

Figure 19: Guessing entropy decomposition, AES_RD dataset results for different training set sizes.

number of features (25) is already enough to retrieve the key byte. Differing from that, both classifiers are unable to break AES_RD, independent of the number of features, because of the shift in correlated value due to the random delay countermeasure.



(a) Random Forest, HW model, 100 trees.

(b) Random Forest, value model, 100 trees.

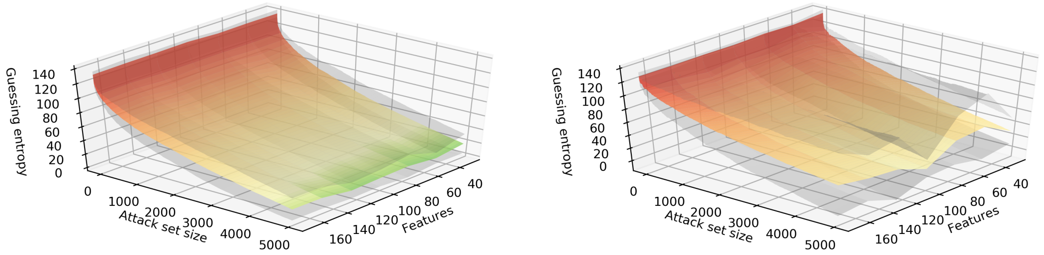
(c) MLP, HW model, 2 hidden layers.

(d) MLP, value model, 2 hidden layers.

(e) CNN, HW model, 2 convolutional blocks

(f) CNN, value model, 2 convolutional blocks

Figure 20: Guessing entropy decomposition, ASCAD dataset results for different training set sizes.

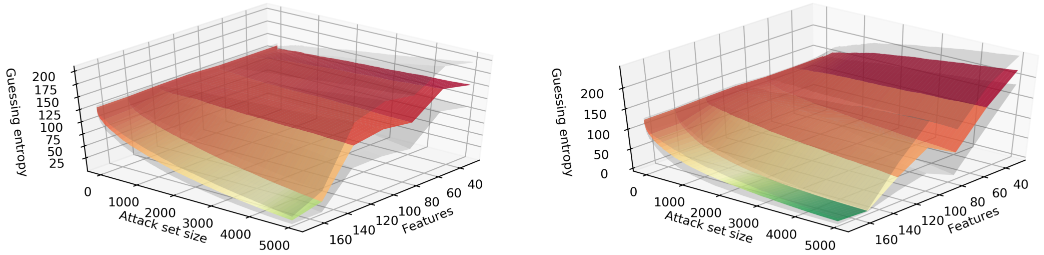


(a) Random Forest, HW model, 100 trees.

(b) MLP, HW model, 2 hidden layers.

Figure 21: Guessing entropy decomposition, AES_HD dataset results for different numbers of features.

Comparing the Guessing Entropy and Domingos Bias–Variance decompositions depends on the scenario. For simple problems (e.g., DPAv4 dataset), both decompositions show the bias and variance in a meaningful way. This is true for any case where the accuracy score is reasonably high. When accuracy is low, Domingos has a clear disadvantage: if accuracy is low throughout different scenarios, the graphs are flat and show no influence of



(a) Random Forest, HW model, 100 trees.

(b) MLP, HW model, 2 hidden layers.

Figure 22: Guessing entropy decomposition, ASCAD dataset results for different numbers of features.

the parameter (complexity, size of training set, number of features). In this case, guessing entropy decomposition is still able to distinguish these scenarios when there is a difference in outputted class probabilities, leading to a better key guess. For example, in Figure 14, we observe little change in Domingo's loss for the ASCAD dataset attacked with different training set sizes. Figure 20 shows there is a clear relation between the training set size and the performance of the attack. Consequently, we argue guessing entropy decomposition is a better method for evaluating classifiers' characteristics for difficult scenarios in SCA.

6 Conclusions and Future Work

In this paper, we consider the bias-variance decomposition and how it can be used in profiling SCAs. First, we show how mean squared error is connected with guessing entropy. This is very important since MSE is often used loss function in neural networks. Next, we conduct a detailed analysis where we consider different training set sizes, numbers of features, and model complexities. The results we obtained suggest that the bias-variance decomposition is able to provide significant insights, especially when the noise is not high and/or there are no countermeasures. In other, more difficult settings, the bias-variance decomposition gives only very general information whether the model underfits or overfits but not how to solve that problem, nor how many traces would be required to succeed in retrieving the secret key. In order to obtain more information on the behavior of machine learning-based SCAs and to connect the bias-variance decomposition with side-channel metrics, we proposed a new analysis tool, guessing entropy bias-variance decomposition. Finally, We experimentally show how this tool can be used to interpret SCAs and discern potential problems in machine learning attacks.

In our experimental setting, the number of repetitions for the bootstrap algorithm plays an important role since if there are not enough repetitions, the variance will not be correctly estimated. At the same time, using deep learning techniques can be computationally demanding even if we need to run then only once. In the setting where we need multiple repetitions, it is not easy to decide the trade-off between the investment in the complexity of the machine learning models and the number of experimental repetitions. In our future work, we plan to investigate the influence of the bootstrap parameters on the reliability of the results. Here, we are especially interested in investigating whether datasets with countermeasures need more repetitions.

References

[Bel03] Richard Ernest Bellman. *Dynamic Programming*. Dover Publications, Incorporated, 2003.

[Bre01] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures - Profiling Attacks Without Pre-processing. In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 45–68, 2017.

[CK09] Jean-Sébastien Coron and Ilya Kizhvatov. An Efficient Method for Random Delay Generation in Embedded Software. In *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, pages 156–170, 2009.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template Attacks. In *CHES*, volume 2523 of *LNCS*, pages 13–28. Springer, August 2002. San Francisco Bay (Redwood City), USA.

[Dom00] Pedro Domingos. A unified bias-variance decomposition and its applications. In *In Proc. 17th International Conf. on Machine Learning*, pages 231–238, 2000.

[Efr79] B. Efron. Bootstrap methods: Another look at the jackknife. *Ann. Statist.*, 7(1):1–26, 01 1979.

[fs19] Anonymized for submisison. https://github.com/f4etrtt0c/biasvariancedecomposition. Github, 2019. https://github.com/f4eTrTT0c/BiasVarianceDecomposition.

[HPGM16] Annelie Heuser, Stjepan Picek, Sylvain Guilley, and Nele Mentens. Side-channel analysis of lightweight ciphers: Does lightweight equal easy? In *Radio Frequency Identification and IoT Security - 12th International Workshop, RFIDSec 2016, Hong Kong, China, November 30 - December 2, 2016, Revised Selected Papers*, pages 91–104, 2016.

[HPGM17] Annelie Heuser, Stjepan Picek, Sylvain Guilley, and Nele Mentens. Lightweight ciphers and their side-channel resilience. *IEEE Transactions on Computers*, PP(99):1–1, 2017.

[HZ12] Annelie Heuser and Michael Zohner. Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines. In Werner Schindler and Sorin A. Huss, editors, *COSADE*, volume 7275 of *LNCS*, pages 249–264. Springer, 2012.

[JWHT01] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibsihrani. *An Introduction to Statistical Learning*. Springer Texts in Statistics. Springer New York Heidelbert Dordrecht London, 2001.

[KPH$^+$18a] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. Make some noise: Unleashing the power of convolutional neural networks for profiled side-channel analysis. Cryptology ePrint Archive, Report 2018/1023, 2018. https://eprint.iacr.org/2018/1023.

[KPH+18b] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. Make some noise: Unleashing the power of convolutional neural networks for profiled side-channel analysis. Cryptology ePrint Archive, Report 2018/1023, 2018. https://eprint.iacr.org/2018/1023.

[LB+95] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10), 1995.

[LBM15] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. The bias–variance decomposition in profiled attacks. *Journal of Cryptographic Engineering*, 5(4):255–267, Nov 2015.

[LPB+15] Liran Lerman, Romain Poussier, Gianluca Bontempi, Olivier Markowitch, and François-Xavier Standaert. Template Attacks vs. Machine Learning Revisited (and the Curse of Dimensionality in Side-Channel Analysis). In *COSADE 2015, Berlin, Germany, 2015. Revised Selected Papers*, pages 20–33, 2015.

[LPMS18] Liran Lerman, Romain Poussier, Olivier Markowitch, and François-Xavier Standaert. Template attacks versus machine learning revisited and the curse of dimensionality in side-channel analysis: extended version. *Journal of Cryptographic Engineering*, 8(4):301–313, Nov 2018.

[LVMS18] L. Lerman, N. Veshchikov, O. Markowitch, and F. Standaert. Start simple and then refine: Bias-variance decomposition as a diagnosis tool for leakage profiling. *IEEE Transactions on Computers*, 67(2):268–283, Feb 2018.

[MGH14] Amir Moradi, Sylvain Guilley, and Annelie Heuser. Detecting Hidden Leakages. In Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay, editors, *ACNS*, volume 8479. Springer, June 10-13 2014. 12th International Conference on Applied Cryptography and Network Security, Lausanne, Switzerland.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings*, pages 3–26, 2016.

[ODZ+16] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[PHAR18] Stjepan Picek, Annelie Heuser, Cesare Alippi, and Francesco Regazzoni. When theory meets practice: A framework for robust profiled side-channel analysis. Cryptology ePrint Archive, Report 2018/1123, 2018. https://eprint.iacr.org/2018/1123.

[PHJ+17] Stjepan Picek, Annelie Heuser, Alan Jovic, Simone A. Ludwig, Sylvain Guilley, Domagoj Jakobovic, and Nele Mentens. Side-channel analysis and machine learning: A practical perspective. In *2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017*, pages 4095–4102, 2017.

[PHJ+18] Stjepan Picek, Annelie Heuser, Alan Jovic, Shivam Bhasin, and Francesco Regazzoni. The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(1):209–237, Nov. 2018.

[PHJ+19] Stjepan Picek, Annelie Heuser, Alan Jovic, Shivam Bhasin, and Francesco Regazzoni. The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(1):209–237, 2019.

[PSB+18] Emmanuel Prouff, Remi Strullu, Ryad Benadjila, Eleonora Cagli, and Cécile Dumas. Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. *IACR Cryptology ePrint Archive*, 2018:53, 2018.

[SMY09] François-Xavier Standaert, Tal Malkin, and Moti Yung. A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In *EUROCRYPT*, volume 5479 of *LNCS*, pages 443–461. Springer, April 26-30 2009. Cologne, Germany.

[TEL14] TELECOM ParisTech SEN research group. DPA Contest (4th edition), 2013–2014. http://www.DPAcontest.org/v4/.