Bachelor's Thesis Computing Science

Comparing the performance of MLPs in side channel attacks with different dataset / architecture customizations

Jasper Hage
s1040782

March 27, 2023

First supervisor/assessor:
Dr Stjepan Picek

Second assessor:
Prof. Lejla Batina

Radboud University

Abstract

In this paper, we compare three different techniques that have been applied to multilayer perceptrons to improve side channel attacks against AES. For that we have chosen to compare ensembles, mixup data augmentation and focal loss ratio against a baseline. Our results showed that ensembles had the best overall results when more than 25,000 profiling traces were used. However, if time is an important factor that should be taken into account then focal loss ratio is a better option for the HW leakage model and the baseline or FLR better options for the ID leakage model with all three taking less than 20% of the time it took to train the ensemble models.

Contents

Chapter 1

Introduction

When talking about the security of encryption algorithms, such as the Advanced Encryption Standard (AES), the focus often lies on flaws that are part of the theory itself. Another aspect that must be considered in the security, however, is the implementation of the algorithm and the hardware that it is running on. If the implementation or the device itself is not properly protected then it may be susceptible to side channel attacks (SCA) if an adversary can get access to the device or a copy thereof.

The increase in the number of devices that deal with confidential information, such as Internet of Things (IoT) devices [15], leads to an ever-growing attack vector if implementations are not properly secured. As most of these devices are commonly available to the public it allows an attacker to easily obtain their own copy of a target device. One of these side channels is the power consumption of the hardware, it has been shown that information that is leaked by measuring the power consumption of microprocessors can be used to retrieve the key, or parts of that key, that was used during encryption [14].

Over the last couple of years, deep learning has taken a prominent role in this research area. Researchers have designed relatively small multi-layer perceptrons (MLPs) and convolutional neural networks (CNNs) that were able to successfully retrieve partial or full keys that were used during encryption. We expand upon this work by analyzing several different state-of-the-art techniques for MLPs that have been proposed by researchers to improve the performance of SCA on power traces.

The goal of this research paper is to provide a thorough comparison of the different techniques that have been proposed. We will give an insight into what the best-performing technique is at this point in time. Furthermore, we will investigate how these techniques perform when an attacker has access to fewer profiling traces. To compare their performance we will use the Guessing Entropy (GE) and Success Rate (SR) metrics that are commonly used when evaluating the performance of SCA. All comparisons will be done

for both the Identity (ID) leakage model as well as the Hamming Weight (HW) leakage model. The outcome of this paper can be used in future research as a baseline to compare new novel techniques to.

In the next chapter, we will provide an overview of the different techniques that we will be using and the background knowledge required to read the rest of the paper. After that there is a chapter in which we will take a look at other research that has been done in this area. This will be followed by a chapter in which we will go over the research that we have done and the results that we have acquired. Finally, we give the conclusion to our research and explain what can be done in the future by other researchers.

Chapter 2

Preliminaries

In this section, we will provide an overview of the techniques that will be used in this paper. First we will introduce the dataset that we will be using throughout the paper and what (deep learning-based) side channel attacks are. After that we will go over the different techniques that we have used during our research.

2.1 Dataset

The dataset that was used for this paper comes from the ANSSI SCA Database, referred to as ASCAD. ASCAD is a set of datasets that contains the power consumption data of a microprocessor doing AES encryptions. At the time of writing this paper, ASCAD contains three different datasets but in our research we have focused on the `ATM_AES_v1_variable_key` dataset.

This dataset contains the power consumption of an ATMega8515 microcontroller unit that is doing AES encryptions. The dataset consists of 200,000 traces that can be used to train the model, hereafter referred to as the profiling traces, and an additional 100,000 traces that can be used to check the performance of the created model, after this referred to as the attack traces. In the raw dataset each trace consists of 250,000 individual power measurements. However, the researchers have also provided a smaller, extracted dataset where each trace only consists of 1400 points of interest around the spot where the information for the key leaks. In this paper, we have used the extracted `ascad-variable.h5` dataset and any future references to ASCAD in this paper will be referring to this particular dataset unless otherwise stated.

2.2 Side channel attacks

In this paper we will focus on side channel attacks, which are attacks that focus on flaws that are not part of the algorithm that we are attacking but

rather of the hardware or the system that it is running on. The system may reveal critical information over these side channels that can be used to retrieve information that should be secret otherwise.

There are a lot of side channels on modern-day hardware, we have provided examples of a couple of side channel attacks below but this list is not exhaustive:

- **Timing attacks:** these attacks focus on the time that our algorithm takes. If different branches take different amounts of time then an attacker can detect that to determine which branch was taken.
- **Optical attacks:** for these attacks an attacker uses optical signals to extract information from the system. This can be a simple recording over someone's shoulder or more advanced by utilizing the hard disk or keyboard LEDs on the computer to extract information.
- **Power consumption attacks:** when a machine computes information it uses different amounts of power depending on what it is computing. If we can measure these differences then we can extract critical information from that.

2.3 Deep learning-based SCA

In this paper we will be focusing on power consumption attacks, with the power consumption data from the ASCAD dataset. To be able to extract part of the key that was used during encryption we will be using deep learning, in particular supervised machine learning. Supervised machine learning takes as an input labeled data, in our case the input data are the power traces and the label is the key that was used for each trace, and is then trained to learn the data so that when given solely the input data that it can correctly predict the label for the input[17].

2.3.1 Profiling attacks

The side channel attacks that we will be doing in this paper are profiling attacks. For a profiling attack an attacker needs to have access to a (copy of) the target device that they want to attack in order to gather useful information during the profiling phase of an attack. They can control this device completely, allowing them to extract exactly that information that they need to gather for their future attack. For this paper, the profiling phase has been done by the researchers that have provided the ASCAD databases.

There are also non-profiling attacks, where an attacker does not have access to a (copy of) the target device. The attacker in this case only has

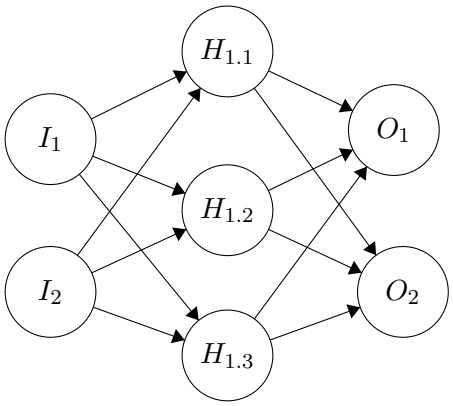


Figure 2.1: MLP consisting of an input layer with 2 nodes, a single hidden layer with 3 nodes and an output layer with 2 nodes. Each I_i and $H_{x.y}$ node has a bias and each arrow has a weight.

access to the leakage information that was gathered but not to information such as the key that was used to gather it [14].

2.3.2 Multilayer perceptrons

The models that we are learning in this paper are multilayer perceptrons (MLPs). These models consist of three or more layers; the input layer, one or more hidden layers and finally the output layer. Each of these layers consists of one or more nodes and all layers are fully connected to the layer that comes after it. Each connection between two nodes has a weight and a bias, which are the values that are actually being modified when we are training a model. There is a weight for each individual connection between two nodes but the bias is the same for all outputs of a single node. A very simple example MLP can be seen in Figure 2.1.

In this paper we have exclusively focused on MLP models. We have chosen to focus on MLPs as they can be trained more quickly when compared to for example convolutional neural networks (CNNs) while still having good performance.

After applying the weights and bias to an input value it is passed through an activation function, which will determine what the output of this node will be. For most of our models this is the rectified linear unit activation function (relu) for the input and hidden layers. This function maps every negative input to 0 and will not alter already positive inputs. As a function it looks as follows: $\texttt{relu}(x) = \texttt{max}(x, 0)$.

All of our labels are one-hot encoded, which means that we have a vector of length x if we have x different output labels. This vector has zeros for all of the components, except for a single component which has the value 1. This 1 indicates that this output belongs to that label.

The output layer does not use the relu activation function but it instead uses the `softmax` activation function. This activation function makes sure that all of the outputs are a proper probability distribution. This means that all outputs together sum up to 1 exactly and that all of the outputs are in the interval $(0, 1)$. The output with the highest probability will then most likely be the correct output according to the model that we have trained.

Hyper-parameters are those values that are used when creating the model and which control what a model looks like and how the learning process functions. Below we have provided a list of a few hyper-parameters, we will explain the exact meaning of them in layer subsections:

- **Number of layers:** this controls how many layers our network will have
- **Number of nodes:** this controls the number of nodes that each layer will have
- **Learning rate:** a value that is used to control how big our weight and bias updates are
- **Activation function:** a function that controls the output of a node
- **Batch size:** the number of inputs that we give to our model before updating the weights and biases
- **Epochs:** controls how many times we will process the entire dataset
- **Optimizer:** algorithms that are used to minimize the loss of our model

2.3.3 Loss function

When a model is being trained, it needs to know how it is doing so it can readjust the weights and biases. For that a loss function is used, it assigns a value to the difference between the guess that our model has made and the real value that is given by our training data, that value is referred to as the loss [17]. For most of the models that were trained for this paper, categorical cross-entropy is used as the loss function.

2.3.4 Optimizer and learning rate

To minimize the loss of the used loss function we need an algorithm that is able to fine-tune the weights and biases of our model. To achieve this an optimizer with a given learning rate is used. The learning rate is what controls the size of the steps that we take when the values are updated; a smaller learning rate will lead to smaller steps but it may take more time for the model to fit the data. A learning rate that is too large on the other hand

might keep overshooting the best target values, meaning that the model will never converge [17]. For most of the models that we have trained in this paper, the `RMSprop` optimizer was used with a learning rate of 0.00001 [14].

2.3.5 SCA Metrics

In this paper we are interested in the Guessing Entropy (GE) of our trained models and the Success Rate (SR) of them. As mentioned in the previous section, we get a probability distribution as output for each of our inputs. For each input, we can order the outputs by their probability and store that in a vector **g**. **g** contains as many components as the number of output labels that our MLP has. The first component in **g** is the output that had the highest probability, then the second component is the output with the second highest probability and so on and so forth for each of the possible output labels.

The guessing entropy of a number of inputs is the average correct key position in vector **g** that was given as output. The success rate of a model is how often the correct key ended up on the first component in vector **g** for all of the inputs [12].

2.3.6 Leakage models

We have trained all models in this paper using two different leakage models: Hamming Weight (HW) and Identity (ID). These leakage models affect the number of output labels that we have. The identity leakage model for a single byte has 256 output classes, one for each possible combination of zeros and ones for eight bits. This number is greatly reduced if we use the Hamming weight leakage model, as we then only need nine different output labels. The Hamming weight of a byte is the number of ones that are in that byte, so ranging from zero up to and including eight.

An advantage of using the Hamming weight is that it may reduce the time that it takes for the model to fit the training data correctly, as there are now fewer output labels. However, this is also a downside of using this leakage model, because it only learns the number of ones in a byte it reveals way less information than the identity leakage model. It is also not an equal distribution for each number of ones; there is only a single way to have zero ones and the same for eight ones but there are way more combinations of one to seven ones in a single byte [14].

2.4 Techniques

In this section, we will go over the different techniques that will be used in our research.

2.4.1 Focal loss ratio

This loss function was designed with SCA in mind and tries to address some of the problems that other loss functions have, such as an imbalance in the number of output labels that we have for the HW leakage model.

2.4.2 Mixup data augmentation

When training a neural network we need to make sure that our model generalizes well. With generalization we mean that the model does not only perform well on the training data but that it also performs well on other unseen data. Deep learning in general performs better when it has more input samples to learn from [17]. If our dataset size is limited then we can use data augmentation on the original dataset to increase the amount of data by generating new samples from the original set of samples.

These new samples can be constructed using various techniques but in this paper we will be using mixup data augmentation [18] [1]. With this technique we can create a new trace and expected answer from two traces in the original dataset, by mixing their input and output values together. We do this with the following formulas:

$$\hat{x} = \lambda x_i + (1 - \lambda) x_j$$

$$\hat{y} = \lambda y_i + (1 - \lambda) y_j$$

In these formulas (x_i, y_i) and (x_j, y_j) are two random samples taken from the original dataset. x_i refers to our input data and y_i refers to the corresponding label of that input data. The lambda value is randomly drawn from the beta distribution for a given α value [18].

2.4.3 Ensembles

Generalization can not only be improved by using data augmentation. Another technique that we can use for that is ensembles. Instead of training one model with the same hyper-parameters, we train multiple models, each with their own combination of hyper-parameters. When we want to know the value of our input then we need to pass it through all of the models that we have learned. To get from multiple outputs to a single output we can sum up all of the different probabilities from all of the different outputs and then use the largest probability as the first guess for our ensemble model [11].

2.5 AISY framework

To create the models and visualizations in this report, we have used the AISY framework created by Guilherme Perin et al [12]. This framework

was created for side channel analysis and attacks that utilize deep learning. AISY has built-in support for all three of the techniques that we will be using in this paper but for data augmentation we manually processed it instead, which is explained in section 4.2.4.

2.5.1 Z-score normalization

AISY framework uses z-score normalization to normalize the profiling, validation and attack traces. It does this by first computing the mean value, $\mu_{\texttt{profiling}}$, and standard deviation, $\sigma_{\texttt{profiling}}$, of the profiling set which are then used in the following formula for all traces, t_i, in the profiling set, the validation set and the attack set:

$$t_i = \frac{t_i - \mu_{\texttt{profiling}}}{\sigma_{\texttt{profiling}}}$$

Z-score normalization makes sure that all values in the dataset have a mean of 0 and a standard deviation of 1. This makes sure that all of the traces are using the same scale which is helpful when updating the weights and biases. Furthermore, because the values are now using the same scale it can help to prevent our weights and biases from becoming too large or too small.

Chapter 3

Related Work

In this chapter of our paper, we will go over other research that has been done in this particular area.

Gabriel Zaid et al. [16] introduced Ranking Loss, a loss function designed with SCA in mind. This loss function penalizes guesses where the correct key has a lower ranking than another incorrect guess. In a paper by Maikel Kerkhof et al. [6] four different loss functions are compared for training models on ASCAD, including the Ranking Loss function from [16]. Their research showed that the cross-entropy ratio loss function, designed by Jiajia Zhang et al. [19], had the best performance out of the four loss functions they compared. Maikel Kerkhof et al. later introduced the Focal Loss Ratio loss function [5] that we used in this paper, which performed better than the four loss functions that they compared.

Feng Gao et al. [3] proposed ensemble learning in combination with SMOTE-based preprocessing to recover the key from the DPA Contest V4 power trace dataset. They compared the result of four different ensemble methods in combination with five different data balancing techniques against a MLP model. These combinations were all able to improve the performance compared to the original MLP model.

Anh Hoang et al. [4] proposed the usage of stacked ensembles to improve the performance of CNN models. Their approach consists of two separate training steps. In the first step multiple models are generated and in the second step a MLP model is trained on outputs and the maximum likelihood scores of the models trained in the first step. Their work showed significant improvements in the performance.

Eleonora Cagli et al. [2] used data augmentation to overcome trace misalignment and overfitting to the training dataset. For that they have used a shifting window over the original dataset and they simulated a clock jitter effect. Their results showed that data augmentation could effectively be used to improve the performance of CNNs.

By adding additional artificial noise to the training data Jaehun Kim et

al. [7] were able to increase the performance of their models. The artificial noise added to their training data helped to prevent overfitting.

In [10] by Naila Mukhtar et al. a neural network is used to generate new data samples to increase the size of the original dataset. The models that were trained on the dataset with additional generated samples performed better than the models trained on the original dataset exclusively.

Zhimin Luo et al. [8] did a more in-depth test of mixup data augmentation on the fixed-key ASCAD dataset. Their results showed that mixup data augmentation can be used to successfully generate new data samples even if fewer profiling traces from the original dataset are accessible.

In a paper by Guilherme Perin et al. [13] different feature selection scenarios are discussed and evaluated. Each scenario has an adversary with different knowledge of the implementation or secret shares that were used when the algorithm is running. The extracted ASCAD dataset that we used falls under the second scenario that they proposed where an adversary has access to both the implementation details and the random secret shares. With this information, the window of 1400 samples can be selected around the point of interest that we were using. The first scenario considers an even more powerful adversary, allowing for an even smaller window. The third scenario leaves the selection of the window to the model itself, for our ASCAD dataset that would mean that we would first need to train a model to select a window from the 250,000 power measurements instead of having access to a dataset with 1400 power measurements.

Furthermore, they use hyperparameter search to train MLP and CNN models using the three different scenarios that they introduced. With this methodology they were able to train MLP networks that reached a guessing entropy of 1 which required 328 or 129 traces for the HW and ID leakage models respectively for our particular dataset.

A more protected dataset, ASCADv2, has been proposed by Loïc Masure et al. in "*Side Channel Analysis against the ANSSI's protected AES implementation on ARM*" [9]. This dataset is similar to the ASCAD dataset that was used in this paper but it has two countermeasures to prevent SCA like the ones we have talked about in this paper.

Chapter 4

Research

In this chapter we will first explain the research that we have done and following that, the results that we have acquired from our research. In the final section of this chapter we will discuss our results.

4.1 Experimental setup

For our experiments we have trained a variety of multi-layer perceptron models on the ASCAD dataset that we introduced in the preliminaries section. All of these models were trained on a GTX 1070 TI.

For each of the techniques we trained networks with a range of profiling traces from the original dataset. There are 200,000 profiling traces in the original dataset but we also trained models with 150,000, 100,000, 50,000, 25,000, 10,000, 5,000 and 1,000 profiling traces.

4.1.1 Baseline

We did not only want to compare the different techniques to each other but also to a baseline so that we can show that the different techniques do or do not provide an advantage over a more traditional multi-layer perceptron. For the baseline MLP we used the work done by Emannuel Prouff et al [14]. Their work was done on the original ASCAD fixed key dataset but during testing it was also able to reach a good GE and SR on the ASCAD dataset that we used. Both the loss function and mixup use this exact model as their starting point, after which their techniques are applied to it. Ensembles uses a random search space and as such does not use this model as the starting point.

In their work they showed that a network consisting of 6 fully connected layers gave the best result. The first 5 layers of the network each had 200 nodes and the relu activation function. The final output layer had one node for each possible output and the softmax activation function. The optimizer

Hyperparameter	Min	Max	Step
Neurons	100	1000	100
Layers	2	8	1
Learning rate	0.0001	0.001	0.0001
Batch size	100	1000	100

Table 4.1: Range of hyperparameters used for the random ensembles

Hyperparameter	Values
Loss function	Category cross-entropy
Activation function	Relu, Selu, elu, tanh
Optimizer	Adam
Epochs	50

Table 4.2: Values of hyperparameters used for the random ensembles

that was used during the training phase was the RMSProp optimizer and categorial cross-entropy was used as the loss function.

4.1.2 Loss function

The setup for Focal loss ratio, developed by Maikal Kerkhof et al. [5] is the same as for the baseline except for the loss function itself. In their research they showed that choosing an n of 3 had a negligible impact on the training time so that is the value that they chose for the number of iterations. Furthermore, their research showed that the default values for α and γ, which are 0.25 and 2.0 respectively, are good choices for the dataset that we are using.

4.1.3 Ensembles

In the research paper by Guilherme Perin et al. [11], they showed that rather than training one network for more epochs, that training multiple networks each with less epochs than the single network and combining their output led to better performance. For the dataset that we used in this paper they showed that an ensemble of 50 random models trained for 50 epochs performed better than both 10 models or a single model.

For our experiment we trained 50 networks for 50 epochs with random hyperparameters for each network. We used the same hyperparameter search space as what was used in the original paper, which can be seen in tables 4.1 and 4.2.

4.1.4 Data augmentation

Karim Abdellatif et al. [1] explored the use of mixup data augmentation for side channel attacks. By tripling the original dataset size using mixup data augmentation they showed that the number of traces that were required to recover the key decreased drastically.

We recreated their results for our dataset by generating 400,000 additional traces from the 200,000 traces that our original dataset had. As the α we used 0.2, similar to what was used in the paper. For our results with fewer profiling traces, for example, with 10,000 profiling traces, we first picked the first 10,000 profiling traces from the original dataset and then we generated an additional 20,000 from those original profiling traces.

To avoid over-fitting to the profiling traces we used drop-out layers after each hidden layer, like in the original paper. The drop-out value was set to 0.2, so 20% of our values would be discarded after each hidden layer.

4.2 Results

In this section we will go over the results that we have acquired first and then after that compare those results to each other.

4.2.1 Baseline

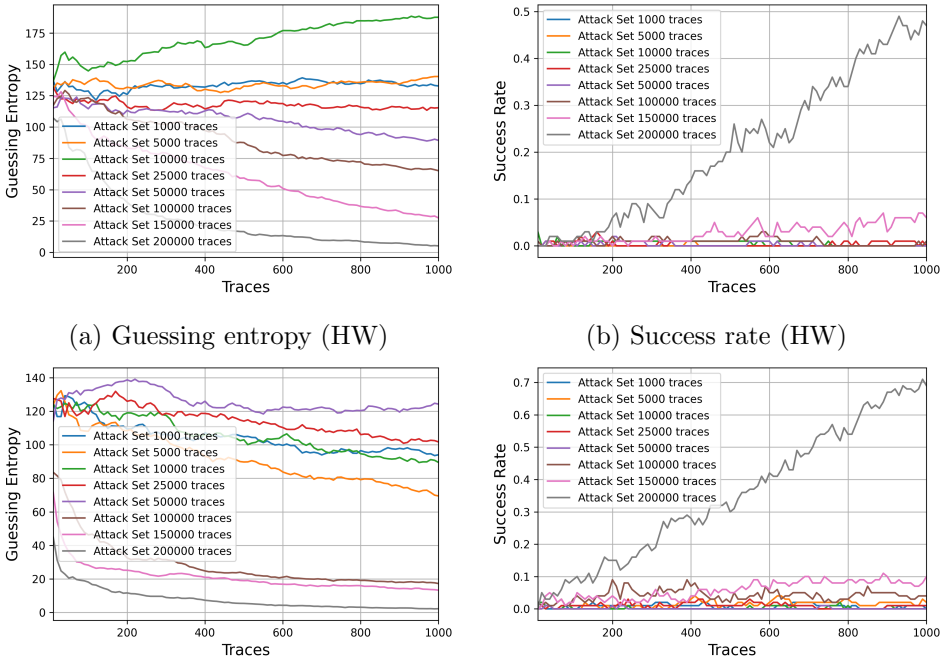
In Figure 4.1 we have plotted the guessing entropy and success rate of our trained models on the attack dataset for both the HW and ID leakage models. Our model that was trained on the full dataset performed the best out of all of the base models for both the HW and ID leakage models. The ID leakage model generally performed slightly better than our HW leakage models but neither of them was able to reach a guessing entropy of 1 or a success rate over 0.7.

HW leakage model comparison

The baseline model that we created performed the worse compared to the three techniques that we analyzed in this paper when trained on less than or exactly with 100,000 profiling traces using the HW leakage model.

ID leakage model comparison

The base model reached a lower guessing entropy than the mixup technique for all profiling trace set sizes. When the dataset was limited to 150,000 profiling traces or less it was also able to reach a lower GE than the models that were trained with FLR.



(a) Guessing entropy (HW)

(b) Success rate (HW)

(c) Guessing entropy (ID)

(d) Success rate (ID)

Figure 4.1: Guessing entropy and success rate of training our base models with a varying amount of profiling traces.

4.2.2 Loss function

Our results with the model trained using the focal loss ratio (FLR) loss function can be seen in Figure 4.2. Both the HW and ID leakage models were able to learn a model that resulted in a guessing entropy close to 1 when it was trained on the full 200,000 traces. The model trained with the HW leakage model also achieves this guessing entropy when trained on only 150,000 profiling traces, which was not the case when the ID leakage model was used.

The success rate of the HW leakage models was considerably higher for both the 200,000 and 150,000 profiling traces but thereafter quickly dropped to around 0 for both leakage models.

HW leakage model comparison

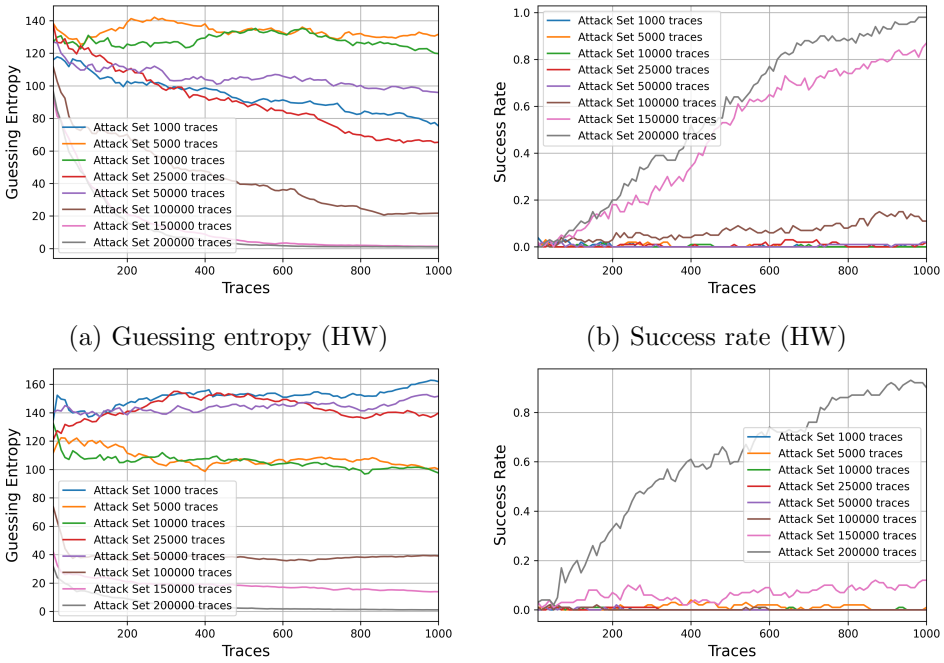
FLR with the HW leakage model performed slightly better when compared to the ensembles when trained on 150,000 or more profiling traces. This technique also reached a success rate of 0.98 when trained on the full data set, the highest out of all techniques using the HW leakage model. The networks that were trained with this custom loss function were the second

technique to reach a guessing entropy of 2 or lower when trained on 150,000 or more profiling traces, only ensembles was able to reach a GE under 2 with less traces than FLR.

We are unsure as to why the model that was trained on 50,000 profiling traces was not able to reach a GE and SR in line with the models before and after it. It does not seem to be caused by overfitting or underfitting as there is no significant difference visible for the accuracy or the loss for this particular model.

ID leakage model comparison

The FLR models have the lowest success rate out of all techniques until 150,000 profiling traces after which both the guessing entropy and success rate are better than the baseline and mixup techniques.



(a) Guessing entropy (HW)

(b) Success rate (HW)

(c) Guessing entropy (ID)

(d) Success rate (ID)

Figure 4.2: Guessing entropy and success rate of training our models with a varying amount of profiling traces with the focal loss ratio loss function.

4.2.3 Ensembles

The figures that we have created for ensembles slightly differ from the other figures that we have shown in this paper so far. At the time of writing, the aisy framework that we used is not able to combine ensemble models with

a varying amount of profiling traces, so the graphs shown in Figure 4.3 are only from the experiments with 200,000 profiling traces. The graphs for the remaining experiments with fewer profiling traces can be seen in Appendix A.

In these graphs there are 51 lines plotted: one for the value of the best model for the attack set, one for the best model for the validation set, one for the ensemble of the 50 models combined and then the 48 remaining models that were trained to create that ensemble in gray.

For both the HW and ID leakage models, the ensemble technique was able to reach a guessing entropy between 1 and 2 for 200,000 and 150,000 profiling traces. For the HW leakage model it also achieved this GE for 100,000 profiling traces. The success rate for the ensembles trained with 200,000 and 150,000 profiling traces with the ID leakage model are considerably higher than those two ensembles trained with the HW leakage model. The opposite is the case for the success rate of the ensembles trained on 100,000 profiling traces, for this ensemble the success rate of the HW leakage model is higher than the success rate of the ID leakage model. With less than 100,000 profiling traces, neither of the leakage models managed to converge to a good guessing entropy or success rate.

HW leakage model comparison

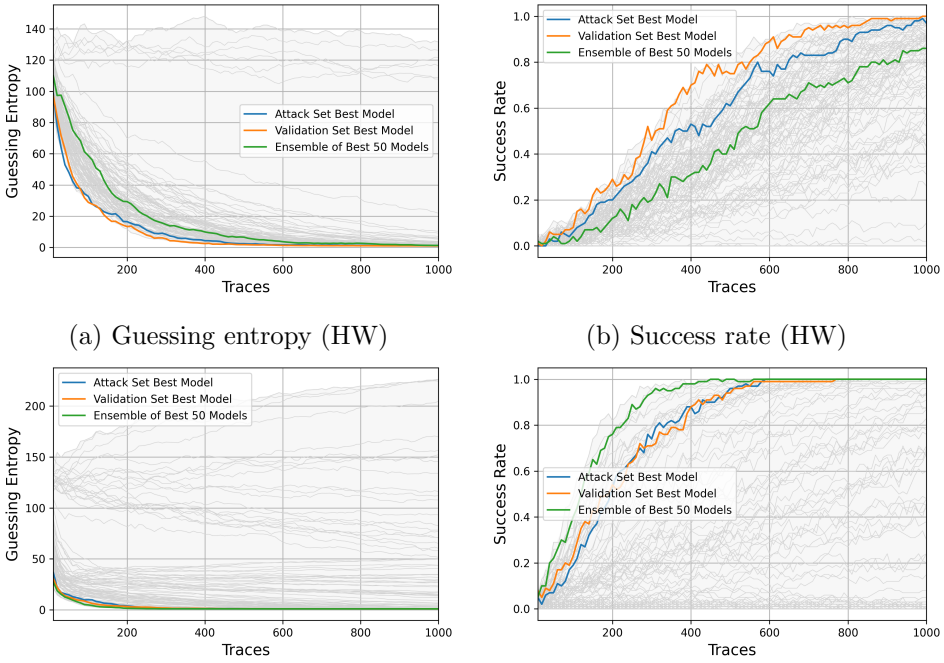
Ensembles with the HW leakage model performed the worse when trained on only 1,000 profiling traces but quickly improved the guessing entropy and success rate when the number of profiling traces increased. It reaches a better guessing entropy and success rate than all other models when between 10,000 and 100,000 profiling traces are used except for FLR with 25,000 profiling traces. It is the first to reach a guessing entropy of 2 or lower.

ID leakage model comparison

The ensembles reached the best guessing entropy and success rate out of all techniques when trained on 25,000 or more profiling traces with the ID leakage model. It was the only model to reach a perfect guessing entropy and success rate when the full dataset was used.

4.2.4 Data augmentation

The aisy framework that we used in this paper provides built-in methods to do data augmentation which we used at first. These final results do not use those methods as they caused the training to take close to 30 hours instead of just under 3 hours. Most of the time was used up by copying data over to the GPU for each batch, which we circumvented by instead calculating all of the augmented data at the start and then moving over that data in one go.



(a) Guessing entropy (HW)

(b) Success rate (HW)

(c) Guessing entropy (ID)

(d) Success rate (ID)

Figure 4.3: Guessing entropy and success rate of training our ensemble models with 200,000 profiling traces.

We have plotted the results of our models in Figure 4.4. The number of traces in these figures is triple that of what is used in the other figures but $\frac{2}{3}$ of these traces are generated using mixup data augmentation.

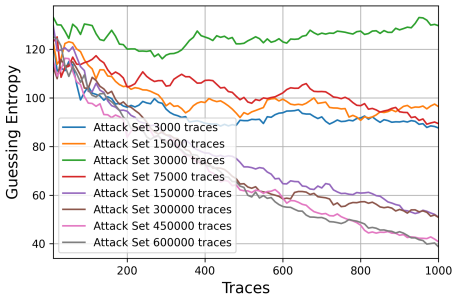
When trained using the HW leakage model, none of the models managed to reach a good guessing entropy or success rate. With the ID leakage model only the model trained on the full dataset that was tripled managed to reach a guessing entropy of 2.88 and a success rate of 0.54.

HW leakage model comparison

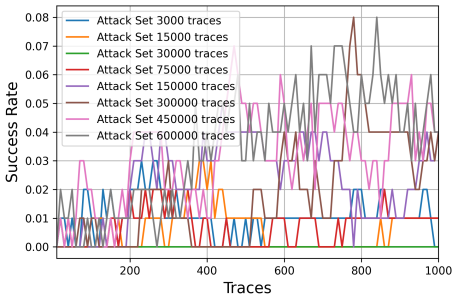
The mixup technique performed better than the baseline model when it was trained on 100,000 or fewer profiling traces, but there was little improvement compared to the other techniques in the guessing entropy or success rate after it was trained on 50,000 or more traces, causing the baseline model to reach a better guessing entropy and success rate when 150,000 or more profiling traces are used.

ID leakage model comparison

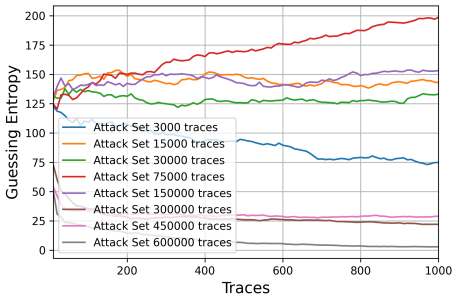
The success rate of the mixup technique is 0.01 higher than that of the baseline for the 1,000, 50,000 and 100,000 profiling traces but for all other models, the success rate of the baseline models is the same or higher than that of the mixup technique making it the least performing technique.



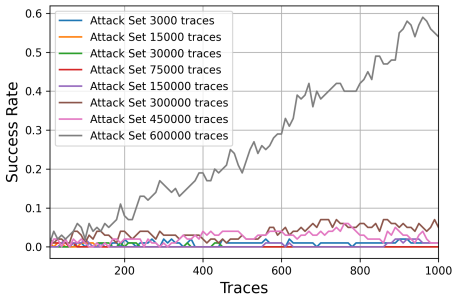
(a) Guessing entropy (HW)



(b) Success rate (HW)



(c) Guessing entropy (ID)



(d) Success rate (ID)

Figure 4.4: Guessing entropy and success rate of training our models with a varying amount of profiling traces with the mixup data augmentation method.

4.2.5 Further comparison

To compare the different techniques we will not only be looking at the guessing entropy and the success rate of the models like we have done in the previous sections but we will also take into account the time that it took to train them. The combined time for each technique and leakage model can be seen in Table 4.3, this is the sum of the training time for all eight models that we trained for each technique and leakage model.

In figure 4.5 we have created four graphs, two for the guessing entropy and success rate of the HW leakage model with the different techniques and two similar graphs for the ID leakage model. The graphs were created for the models that had access to 1,000 traces to retrieve the key byte. In Appendix

Technique	Leakage model	Time
Base	HW	1h2m
	ID	55m
FLR	HW	1h10m
	ID	1h11m
Ensembles	HW	6h6m
	ID	6h39m
Mixup	HW	3h09m
	ID	3h25m

Table 4.3: Times of the different techniques and leakage models

B we have given the tables with the raw values for each graph created here.

FLR performing better than the baseline can be explained by the fact that it is able to more efficiently penalize incorrect values while limiting by how much the weights and biases are updated when the predictions made by the model are getting closer to the correct predictions.

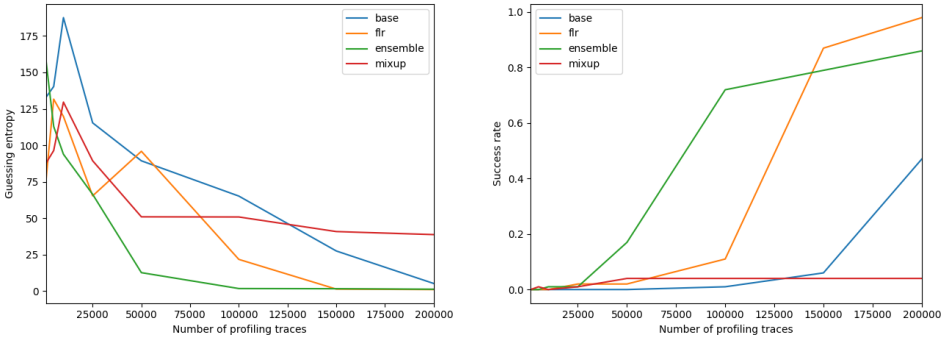
Compared to the baseline, ensembles overall had better GE and SR values. This can be explained by the fact that the hyperparameters that the baseline and the other two techniques used were not optimized for this particular dataset. The different individual random models that are trained for ensembles to work can overcome this limitation as they are not bound to the hyperparameters of the base model.

Mixup overall performed worse than the baseline model. This appears to be caused by the models overfitting to the training dataset regardless of the dropout layers that have been added to these models.

HW leakage model

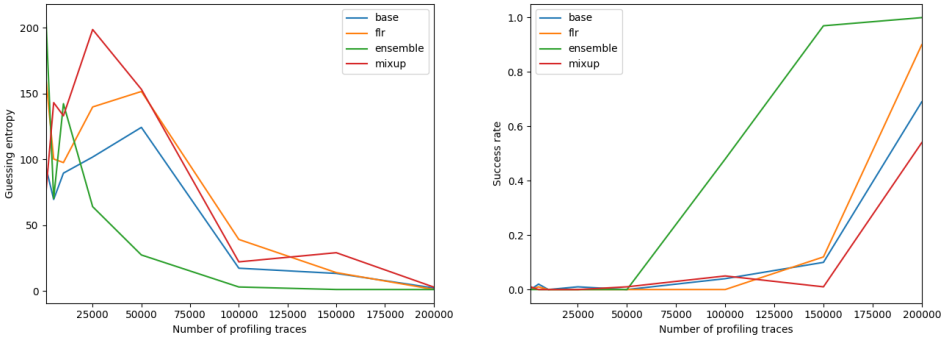
Besides ensembles all three methods had a peak when using 5,000 or 10,000 profiling traces. In certain cases this is caused by overfitting to the training data set, as indicated by a drop in validation accuracy while the training accuracy is still improving for these models. In other cases there does not appear to be a clear cause of the increase in the guessing entropy.

As can be seen in Table 4.3, the baseline models were the quickest to train, closely followed by the FLR models. The mixup technique takes more than three times as long to train the baseline models and the ensembles increase this further to almost six times the baseline time.



(a) Guessing entropy (HW)

(b) Success rate (HW)



(c) Guessing entropy (ID)

(d) Success rate (ID)

Figure 4.5: Guessing entropy and success rate of the different techniques and a varying number of profiling traces. The graphs were created for the models that had access to 1,000 traces to retrieve the key byte

ID leakage model

Similarly to the HW leakage models, most of the models that we trained had a peak in the guessing entropy when a particular range of profiling traces was used. While this peak for the HW leakage model is most noticeable when between 5,000 and 10,000 profiling traces are used, for the ID leakage model these peaks have moved towards the 25,000 to 50,000 profiling traces. Overfitting does seem to be causing these peaks in most cases because there is a larger difference between the training and validation accuracy for these models.

In Table 4.3 the times it took to train all of the models for each technique are displayed. The baseline models were the quickest, followed by the models trained using the FLR loss function. Mixup took over three times as long and the ensembles took almost two times as long as the mixup models to train.

Chapter 5

Conclusions

In this paper we have compared three different techniques for MLP networks against a baseline and each other. Out of these techniques, the ensembles reached the best overall guessing entropy for both the HW and ID leakage models. FLR was able to reach a better success rate and guessing entropy for the HW leakage model when 150,000 or more traces were used but ensembles had a superior GE and SR for the ID leakage model. The mixup data augmentation models that were trained on 100,000 profiling traces or less using the HW leakage model reached a better GE and SR than the baseline model but thereafter had worse GE and SR than the other techniques.

Overall, ensembles are the best option out of the techniques that we compared when more than 25,000 profiling traces are used. FLR only reaches a better GE and SR when more than 150,000 profiling traces are used for the HW leakage model. If time is an important factor that should be taken into account then FLR is a better option for the HW leakage model and the baseline or FLR for the ID leakage model depending on the number of profiling traces.

For this paper, we decided to use one base model which was shared as a starting point by three of the techniques covered. This is a limiting factor as different techniques might have performed better with different or more fine-tuned hyperparameters but that was out of scope for this paper. Furthermore, in this paper we only looked at three different techniques and a baseline, that could be expanded upon to provide a larger overview of techniques that may improve the performance of SCA.

For future work it would be interesting to do a similar comparison for different CNN model techniques, such as the ones that were discussed in the related work section. Similarly, this research could be repeated for different datasets, to see if the results of this paper can also be applied to those, like the ASCADv2 dataset by Loïc Masure et al [9].

Bibliography

[1] Karim M. Abdellatif. Mixup data augmentation for deep learning side-channel attacks. Cryptology ePrint Archive, Paper 2021/328, 2021. https://eprint.iacr.org/2021/328.

[2] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, pages 45–68, Cham, 2017. Springer International Publishing.

[3] Feng Gao, Baolei Mao, Lingjuan Wu, Zongmin Wang, Dejun Mu, and Wei Hu. Leveraging ensemble learning for side channel analysis on masked aes. In *2021 7th International Conference on Computer and Communications (ICCC)*, pages 267–271, 2021.

[4] Anh Hoang, Neil Hanley, Ayesha Khalid, Dur e Shahwar Kundi, and Maire O'Neill. Stacked ensemble model for enhancing the dl based sca. In *Proceedings of the 19th International Conference on Security and Cryptography - Volume 1: SECRYPT,*, pages 59–68. INSTICC, SciTePress, 2022.

[5] Maikel Kerkhof, Lichao Wu, Guilherme Perin, and Stjepan Picek. Focus is key to success: A focal loss function for deep learning-based side-channel analysis. Cryptology ePrint Archive, Paper 2021/1408, 2021. https://eprint.iacr.org/2021/1408.

[6] Maikel Kerkhof, Lichao Wu, Guilherme Perin, and Stjepan Picek. No (good) loss no gain: Systematic evaluation of loss functions in deep learning-based side-channel analysis. Cryptology ePrint Archive, Paper 2021/1091, 2021. https://eprint.iacr.org/2021/1091.

[7] 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. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(3):148–179, May 2019.

[8] Zhimin Luo, Mengce Zheng, Ping Wang, Minhui Jin, Jiajia Zhang, and Honggang Hu. Towards strengthening deep learning-based side channel attacks with mixup. Cryptology ePrint Archive, Paper 2021/312, 2021. https://eprint.iacr.org/2021/312.

[9] Loïc Masure and Rémi Strullu. Side channel analysis against the anssi's protected aes implementation on arm. Cryptology ePrint Archive, Paper 2021/592, 2021. https://eprint.iacr.org/2021/592.

[10] Naila Mukhtar, Lejla Batina, Stjepan Picek, and Yinan Kong. Fake it till you make it: Data augmentation using generative adversarial networks for all the crypto you need on small devices. Cryptology ePrint Archive, Paper 2021/991, 2021. https://eprint.iacr.org/2021/991.

[11] Guilherme Perin, Łukasz Chmielewski, and Stjepan Picek. Strength in numbers: Improving generalization with ensembles in machine learning-based profiled side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(4):337–364, Aug. 2020.

[12] Guilherme Perin, Lichao Wu, and Stjepan Picek. Aisy - deep learning-based framework for side-channel analysis. Cryptology ePrint Archive, Report 2021/357, 2021. https://eprint.iacr.org/2021/357.

[13] Guilherme Perin, Lichao Wu, and Stjepan Picek. Exploring feature selection scenarios for deep learning-based side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):828–861, Aug. 2022.

[14] Emmanuel Prouff, Remi Strullu, Ryad Benadjila, Eleonora Cagli, and Cecile Dumas. Study of deep learning techniques for side-channel analysis and introduction to ascad database. Cryptology ePrint Archive, Paper 2018/053, 2018. https://eprint.iacr.org/2018/053.

[15] Lionel Sujay Vailshery. Number of internet of things (iot) connected devices worldwide from 2019 to 2021, with forecasts from 2022 to 2030. 2022. https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/.

[16] Gabriel Zaid, Lilian Bossuet, François Dassance, Amaury Habrard, and Alexandre Venelli. Ranking loss: Maximizing the success rate in deep learning side-channel analysis. Cryptology ePrint Archive, Paper 2020/872, 2020. https://eprint.iacr.org/2020/872.

[17] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021. https://d2l.ai.

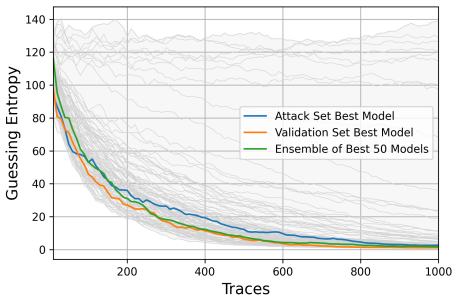
[18] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization, 2017. https://arxiv.org/abs/1710.09412.

[19] Jiajia Zhang, Mengce Zheng, Jiehui Nan, Honggang Hu, and Nenghai Yu. A novel evaluation metric for deep learning-based side channel analysis and its extended application to imbalanced data. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):73–96, Jun. 2020.

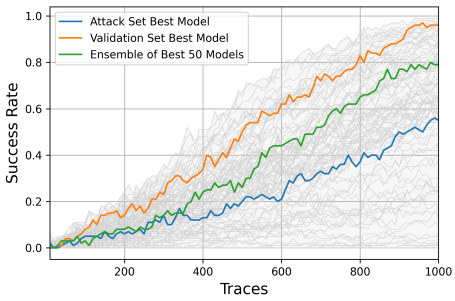
Appendix A

Ensemble figures

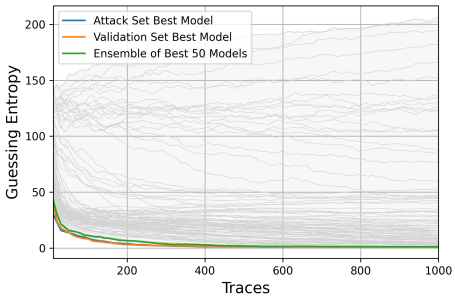
In this appendix, all of the ensemble figures that were not displayed in the research chapter 4.2.3 are given. These graphs are discussed in that section as well.



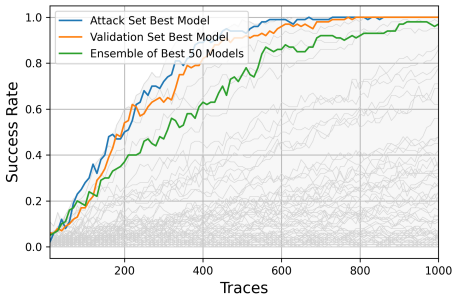
(a) Guessing entropy (HW)



(b) Success rate (HW)

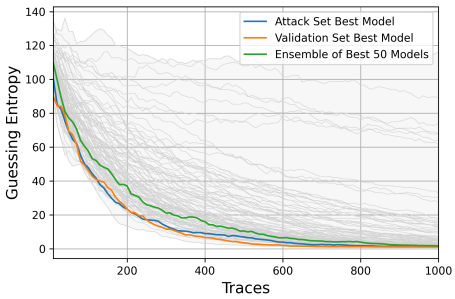


(c) Guessing entropy (ID)

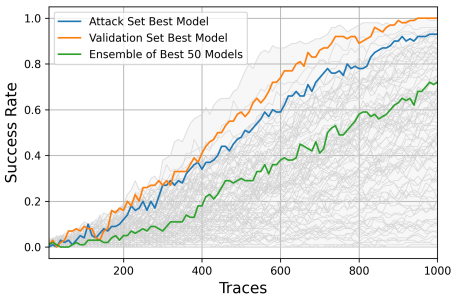


(d) Success rate (ID)

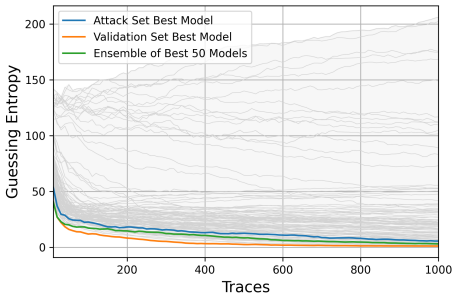
Figure A.1: Guessing entropy and success rate of training our ensemble models with 150,000 profiling traces.



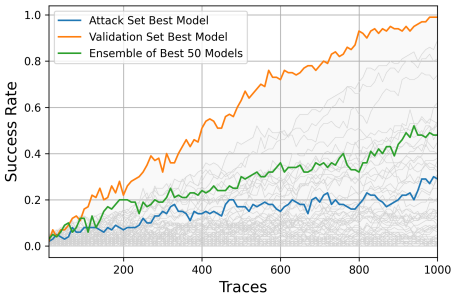
(a) Guessing entropy (HW)



(b) Success rate (HW)

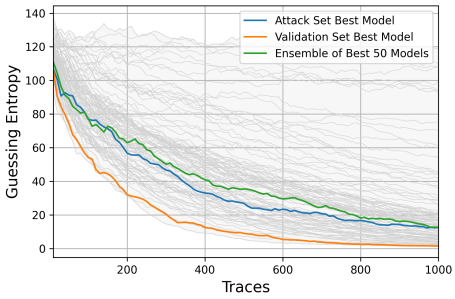


(c) Guessing entropy (ID)

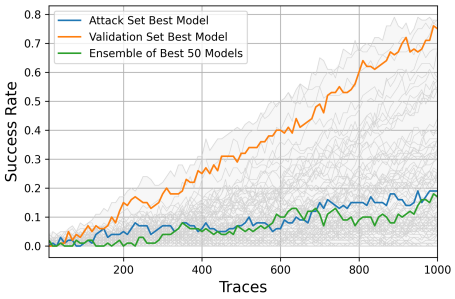


(d) Success rate (ID)

Figure A.2: Guessing entropy and success rate of training our ensemble models with 100,000 profiling traces.

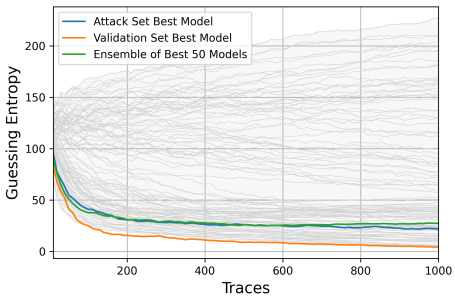


(a) Guessing entropy (HW)

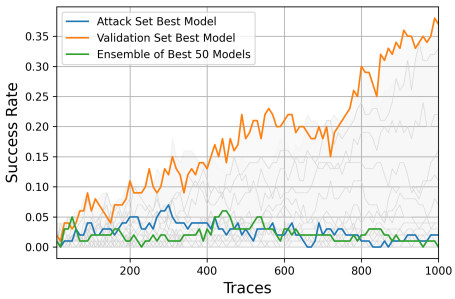


(b) Success rate (HW)

Figure A.3: Guessing entropy and success rate of training our ensemble models with 50,000 profiling traces.

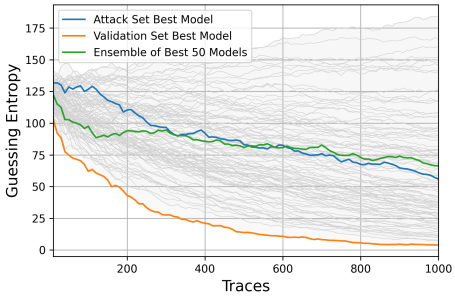


(c) Guessing entropy (ID)

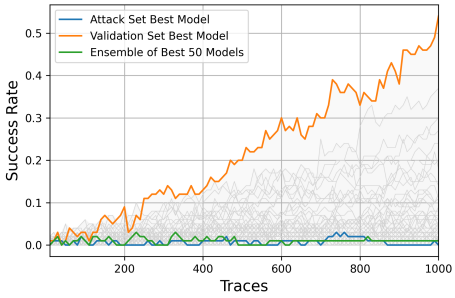


(d) Success rate (ID)

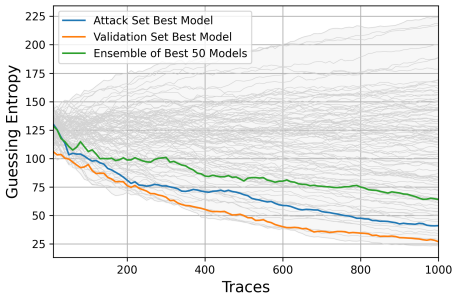
Figure A.3: Guessing entropy and success rate of training our ensemble models with 50,000 profiling traces.



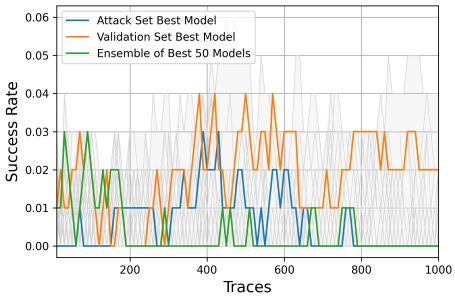
(a) Guessing entropy (HW)



(b) Success rate (HW)

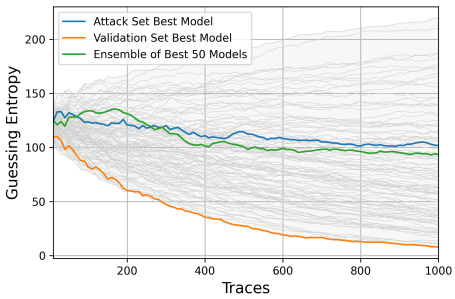


(c) Guessing entropy (ID)

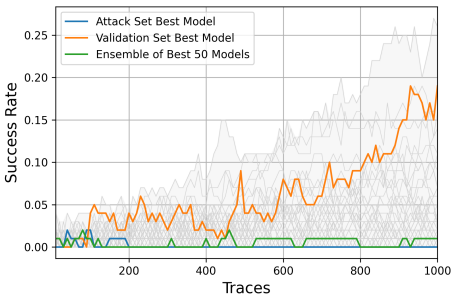


(d) Success rate (ID)

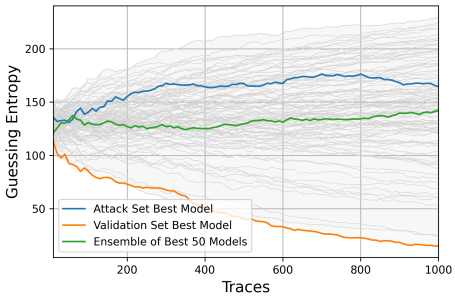
Figure A.4: Guessing entropy and success rate of training our ensemble models with 25,000 profiling traces.



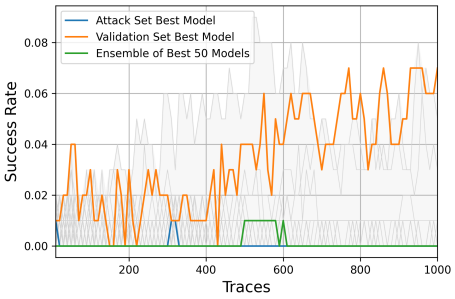
(a) Guessing entropy (HW)



(b) Success rate (HW)

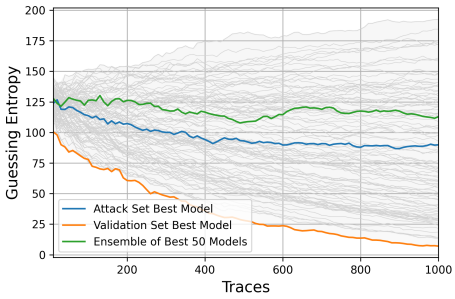


(c) Guessing entropy (ID)

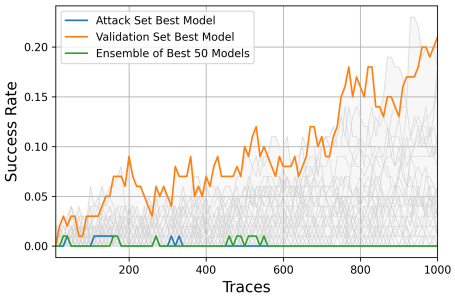


(d) Success rate (ID)

Figure A.5: Guessing entropy and success rate of training our ensemble models with 10,000 profiling traces.

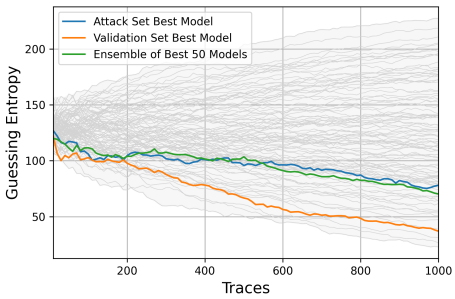


(a) Guessing entropy (HW)

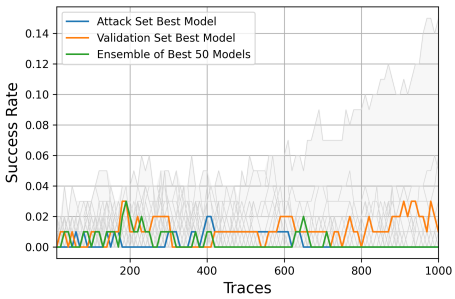


(b) Success rate (HW)

Figure A.6: Guessing entropy and success rate of training our ensemble models with 5,000 profiling traces.

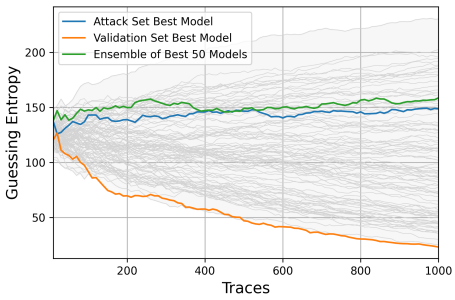


(c) Guessing entropy (ID)

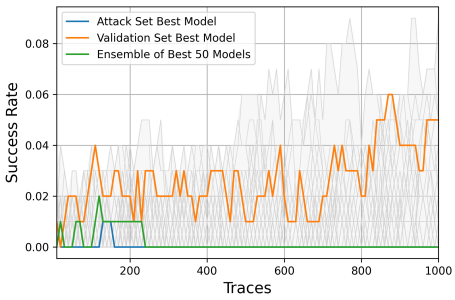


(d) Success rate (ID)

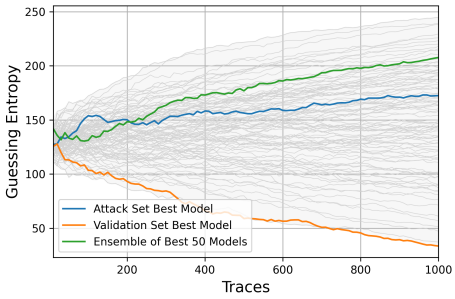
Figure A.6: Guessing entropy and success rate of training our ensemble models with 5,000 profiling traces.



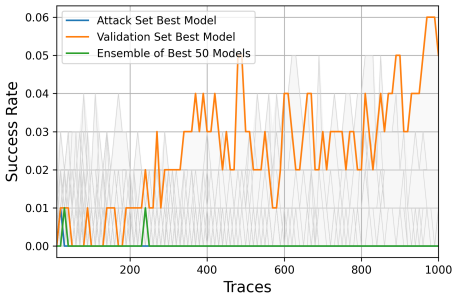
(a) Guessing entropy (HW)



(b) Success rate (HW)



(c) Guessing entropy (ID)



(d) Success rate (ID)

Figure A.7: Guessing entropy and success rate of training our ensemble models with 1,000 profiling traces.

Appendix B

GE and SR tables

In this appendix we have given four tables that contain the guessing entropies or success rates of our models that were trained with different techniques and a varying amount of profiling traces. These values were calculated for the models that had access to 1,000 profiling traces to retrieve the key byte.

	Number of profiling traces used							
Technique	1,000	5,000	10,000	25,000	50,000	100,000	150,000	200,000
Base	132.8	140.31	187.58	115.39	89.35	65.16	27.51	5.2
FLR	75.3	131.65	119.79	65.32	95.89	21.73	1.35	1.04
Ensembles	158.71	112.9	93.86	66.35	12.63	1.73	1.61	1.27
Mixup	87.6	96.37	129.63	89.38	50.93	50.83	40.81	38.73

Table B.1: The guessing entropy of our models using the HW leakage model with different number of profiling traces and different techniques used.

	Number of profiling traces used							
Technique	1,000	5,000	10,000	25,000	50,000	100,000	150,000	200,000
Base	0	0	0	0	0	0.01	0.06	0.47
FLR	0	0	0	0.02	0.02	0.11	0.87	0.98
Ensembles	0	0	0.01	0.01	0.17	0.72	0.79	0.86
Mixup	0	0.01	0	0.01	0.04	0.04	0.04	0.04

Table B.2: The success rate of our models using the HW leakage model with different number of profiling traces and different techniques used.

	Number of profiling traces used							
Technique	1,000	5,000	10,000	25,000	50,000	100,000	150,000	200,000
Base	94.05	69.49	89.48	101.72	124.27	17.22	13.26	2.21
FLR	161.81	100.24	97.51	139.79	151.56	39.09	13.88	1.14
Ensembles	207.69	70.27	142.28	63.99	27.29	2.95	1.03	1.0
Mixup	74.82	143.08	133.06	198.66	153.21	22.04	29.01	2.88

Table B.3: The guessing entropy of our models using the ID leakage model with different number of profiling traces and different techniques used.

	Number of profiling traces used							
Technique	1,000	5,000	10,000	25,000	50,000	100,000	150,000	200,000
Base	0	0.02	0	0.01	0	0.04	0.1	0.69
FLR	0	0.01	0	0	0	0	0.12	0.9
Ensembles	0	0	0	0	0	0.48	0.97	1
Mixup	0.01	0	0	0	0.01	0.05	0.01	0.54

Table B.4: The success rate of our models using the ID leakage model with different number of profiling traces and different techniques used.