DLSCA: a Tool for Deep Learning Side Channel Analysis

Martin Brisfors, Sebastian Forsmark
EECS, Royal Institute of Technology (KTH)
Stockholm, Sweden
Email: brisfors@kth.se, sforsm@kth.se

***Abstract*—Research on Side Channel Analysis (SCA) is very active and progressing at a fast pace. The idea of using Machine Learning (ML), and more recently Deep Learning (DL), to help SCA data is explored extensively. One issue facing security researchers interested in contributing to this cause is the difficulties getting started. While replicating previous works with open source code is not difficult, taking the next steps from there can be daunting. The presented open-source DLSCA tool is created to aid with research on DL-based SCA and to help newcomers to DL to get started. It is hoped to contribute to investigating the strengths and limitations of ML-based SCA.**

***Keywords*-Machine Learning, Side Channel Attack, Software Tool**

I. Introduction

The recent idea [1] of applying Deep Learning (DL) techniques to Side Channel Analysis (SCA) is very currently actively researched. The barrier of entry for deep learning is one of the hurdles preventing more people familiar with side channel analysis from researching it.

In an attempt to make the subject more approachable we created the presented tool for Deep Learning Side Channel Analysis (DLSCA). It is a collection of scripts and tools bound together by a graphical user interface (GUI). This is a layer of abstraction further away from the fundamentals of deep learning that makes it significantly easier to get started. Its purpose is to be a stepping stone in learning more. The open source nature of the code means that more advanced functionality can easily be implemented.

In this paper we will be using DLSCA to show the process of training and testing a Multiple Layer Perceptron (MLP) model trained on Electromagnetic (EM) side channel traces. The results of this experiment are presented in section IV. By showing the whole process we believe its ease of use is made apparent.

The effects of introducing an open source tool, like DLSCA, for getting started with applying DL to SCA could be beneficial to the scientific community. It helps demystify the subject to new researchers. It also makes it easier to convey parameters and results for future research. These are factors that could contribute to advancing future SCA research at an accelerated rate. That would contribute to finding the strengths and weaknesses of applying DL techniques to SCA which could help advance both fields.

II. Side Channel Analysis & Deep Learning

A brief introduction to the history and theory of SCA and DL will be provided here. For further reading see [2] and [3].

A. Side Channel Analysis

The idea of using meta information about a running algorithm to learn more about the internal workings of the algorithm in question was pioneered by Paul Kocher. He first thought of it in the context of timing how long it took to perform private key operations as part of the Diffie Hellman key exchange [4]. By using time measurements as an information leakage point, obfuscated pieces of information could be inferred. As the field developed there have been several new avenues of information leakage such as electromagnetic radiation [5] and the power consumption [6] of the hardware.

B. Machine Learning

Machine Learning (ML) refers to the use of algorithms that take input data and optimize parameters in order to produce desirable responses to other input data. The fundamental ideas of ML date back to the 50s when computer scientists like Alan Turing theorized about artificial intelligence and Arthur Samuel coined the term Machine Learning [7]. However, more practical and modern applications of ML date back to the 90s [8].

DL as a subfield of ML typically refers to the use of Artificial Neural Nets (ANNs). These can among other things be used to classify data samples based on their features. Using a one-hot encoding for categorically correct answers means the networks output can be normalized and interpreted as a prediction array over all possible categories. For every category $c_i \in C$ we determine a truth vector t such that

$$t = \begin{cases} t_i = 1 & \iff c_i \; is \; the \; true \; class \\ t_i = 0 & otherwise \end{cases}$$

A rudimentary way of structuring an ANN is by creating a Multiple Layer Perceptron (MLP). As the name implies, an MLP network is divided into multiple discrete layers - the input layer, hidden layers, and the output layer. Each layer contains one more more so called neurons, which have connections to neurons in previous and next layers of

the network. For each such connection the network has an assigned weight variable. The value of each neuron (n_i) is given as a function of the sum of the value of all the previous layer's neurons (n_{i-1}) it is connected to, each multiplied by their respective weight (w) variable.

$$n_{i,j} = f(\sum_j n_{i-1,j} * w_j + b_i)$$

An optional bias term (b) is added to this sum before applying the so called activation function (f). Activation functions serve an important role in creating non-linearity in the multivariable function represented by the MLP network. Commonly used activation functions include Rectified Linear Unit (ReLU), sigmoid (σ) and softmax.

The way an MLP is trained usually involves optimizing the value of the weight and bias variables to fit the expected output of the network to any given input. Data used for training is divided into *minibatches*. For each minibatch the network calculates its output to the given input. The difference between the network output and the expected output, called a *label*, is used to quantify how inaccurate the model currently is. This is done via a so called *loss function*. Examples of loss functions are Mean Square Error (MSE) and Categorical Cross Entropy Loss [9].

The loss function is typically minimized through the use of a type of optimization algorithm called *gradient descent*. There are many adaptations of the gradient descent algorithm, such as RMSProp and ADAM [10], but the core idea remains the same. In order to minimize the loss function, which is a multivariable function, we consider the surface it spans. By calculating the gradient of the function using the chain rule we can find which changes in internal variables gets us closer to a minimum. These adjustments are called back propagation.

III. Development

DLSCA was inspired by and largely made possible by the landmark contributions of Benadjila et al. at ANSSI who in their paper [11] explored the use of DL-based SCA. Parts of DLSCA are adaptations of the public domain code published by ANSSI.

All code for DLSCA is written in python 3.6 and the ML scripts were developed for tensorflow 1.12.0 and keras 2.2.4. GUI was made using Qt5, through the pyqt library version 5.9.2. Testing with the most recent versions of all these packages has shown that the software runs as expected, but several warning messages about deprecated function names do appear. The reason tensorflow 2.0 was not used is because it had yet to be released when work started.

DLSCA runs its modules by creating subprocesses that calls on external scripts. It was designed that way in order to make it more modular. It was a natural consequence of trying to collect and link together disparate scripts into one hub. This modularity not only makes it easier to introduce custom scripts; it also makes the software more resilient to crashes. If one of the subprocess scripts crashes it does not impact the main program - it shows the error message from the crash in the terminal but DLSCA continues running. Compartmentalizing scripts this way creates less interdependence. A benefit of this is that it makes it easier to change scripts without breaking other parts of the software.

The scripts included with DLSCA were created with ChipWhisperer in mind. Power traces captured using ChipWhisperer can interface with all the included scripts.

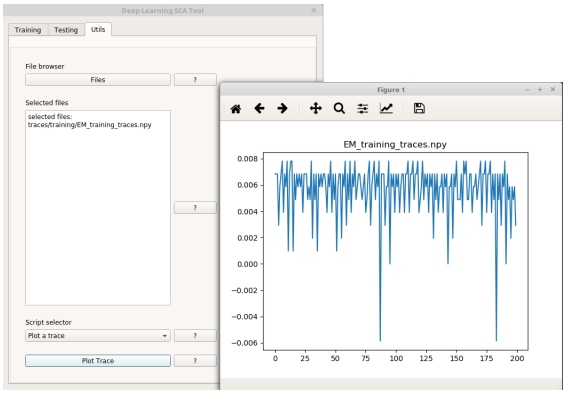

Figure 1. A randomly selected EM trace. Only the first 200 time points were sampled for this experiment.

IV. Features and Results

Rather than listing the features of DLSCA, we will demo the software by presenting the process and results of running an experiment. We collected EM side channel traces using ChipWhisperer and a simple EM probe. The experiment is aimed to answer the question "can an MLP model be trained on EM traces accurately using the same hyperparameters as a successful model for power analysis?". This experiment pertains to the use of an ATxmega128D4 microcontroller running AES-128 encryption in Electronic Code Book (ECB) mode. The process and the answer will be presented here. The scripts included with DLSCA are currently written with an assumption that a similar system is being tested, but most of the code was written with modification in mind for other applications.

After capturing the EM traces we can easily plot a randomly selected trace using the "plot a trace" utility script. A sample EM trace is shown in Fig. 1.

First, labels for the training data had to be created. Running the unzipper utility tool on our ChipWhisperer traces automatically labels the data and can combine multiple capture sessions' data into one. The traces and labels are automatically stored in /traces/training. In Fig. 2, 3 and 4 you can see how this was done.

Next we created a training script for an MLP model using hyperparameters that have proven successful in past

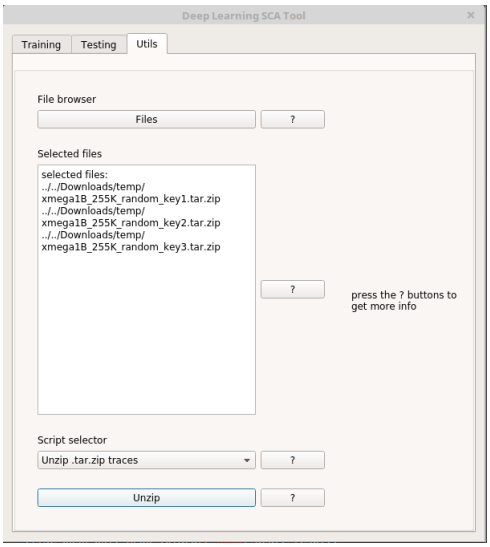

Figure 2. The unzipper is run by selecting either .npy files or .zip or .tar files containing them

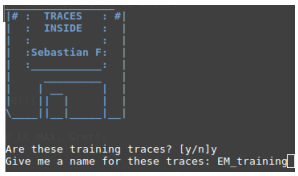

Figure 3. The unzipper scripts prompts the user for purpose and name for traces

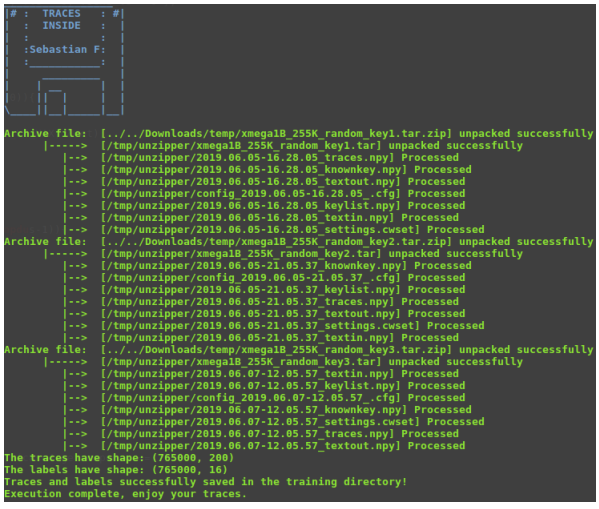

Figure 4. The unzipper ends with a summary of what it did

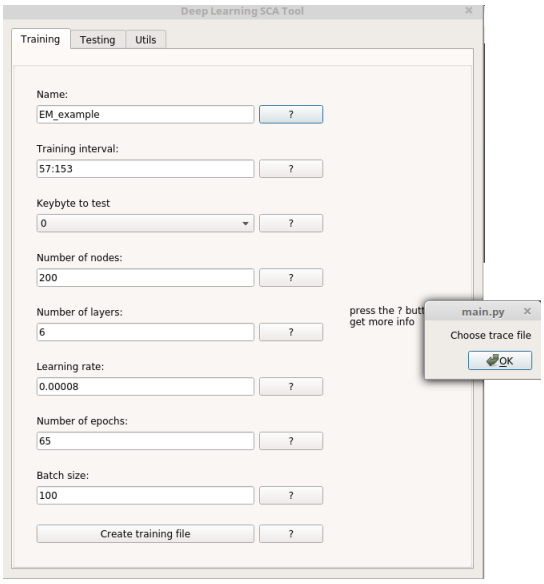

Figure 5. The following hyperparameters were chosen for training the model using a dataset of 765k traces and a validation split 0.3. These parameters have proven successful for training with power traces in past experiments.

experiments, as seen in Fig. 5. This script can be run from the base directory as is. In this case it was instead sent to a computational node on a supercomputer.

The training yielded a history file which we plotted using the history plotting script, see Fig. 6. Since training seems to be successful we verified the input shape and model summary using the corresponding utility scripts, as seen in Fig. 7. Then we unzipped the testing traces, which were put in the attack folder automatically, using the unzipper script again.

Since the model seems to be correct, we tested it using the average rank test with the parameters seen in Fig. 8. When the test finishes running it produces a plot of the performance. This plot and the data used to generate it are both saved in the results folder automatically. The plot from the experiment is shown in Fig. 9

The average rank test is included with DLSCA. The model makes a prediction which yields a probability array. This array is sorted from what the model predicts is the most likely answer to the least likely answer. We then find at what index in this sorted array the correct answer exists. This is called the rank of the prediction. A rank of 0 indicates the correct answer was predicted. The test keeps a cumulative sum of the logarithm of subsequent predictions and uses this for the subsequent rank calculation. This method of testing is highly dependent on the order in which data is processed. Due to sensitivity to the order in which testing data is seen, this test runs for a number of iterations set in the DLSCA test parameters. The mean of the results is calculated and plotted. If more sophisticated statistical analysis is warranted, the

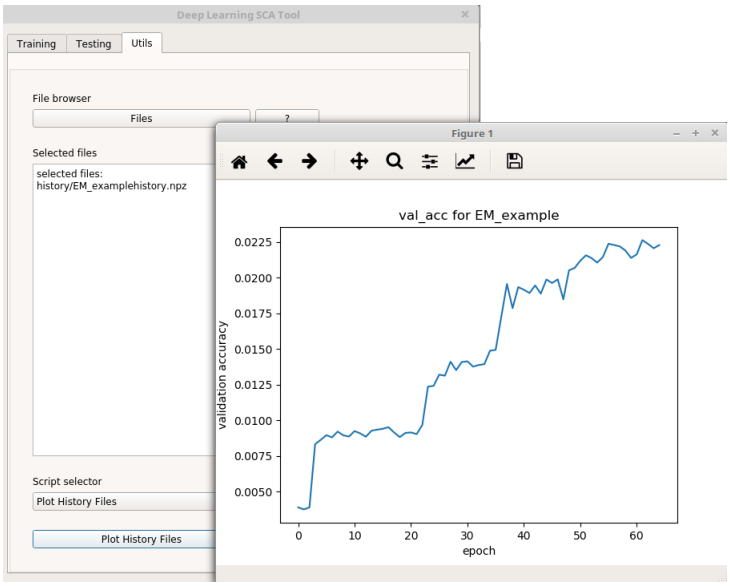

Figure 6. A plot of the history is shown before being automatically saved in the /history folder

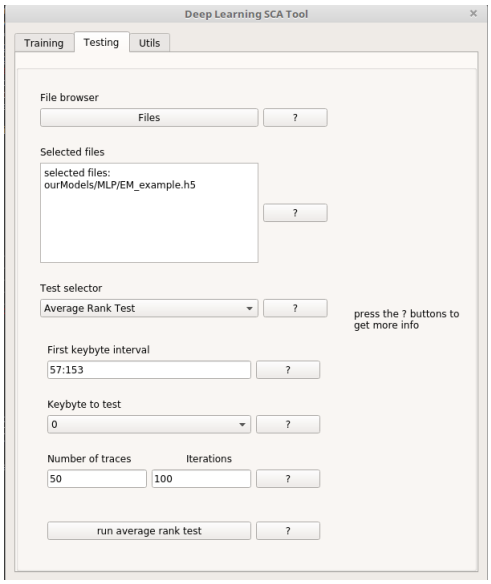

Figure 8. Settings used for running the average rank test

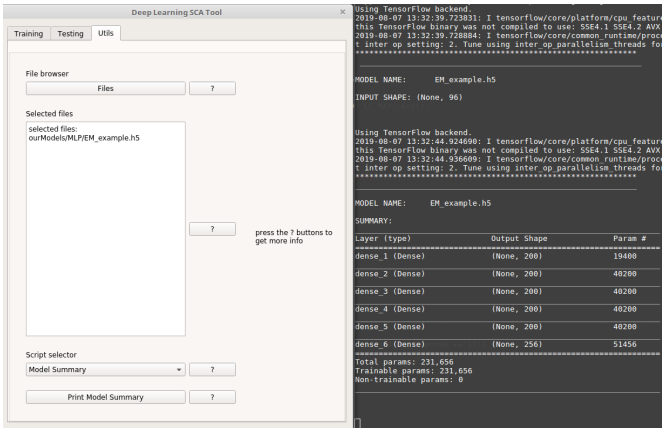

Figure 7. Results of running scripts for input shape and model summary

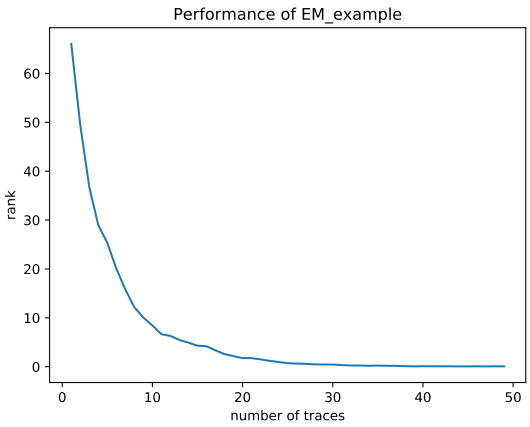

Figure 9. Results of average rank test using the model to attack the same device as trained on with new traces.

raw data from the tests, before the mean is calculated, is saved automatically by the test.

The other two tests, which were not performed in this case, work as follows.

The whole key test runs an average rank for every keybyte position. In order to recover the entire encryption key we must converge to a rank of 0 for each keybyte position. This is done by searching for a sequence of multiple 0s in a row in the rank progression. Since all keybytes need to be at rank 0, the whole key cannot be said to be recovered until the poorest performing keybyte has been recovered. Thus, whichever keybyte position takes the longest to converge to rank 0 determines the amount of traces needed to recover the whole key. The reason results from this test were not presented for the EM experiment is because we captured only the part of each trace corresponding to the first keybyte calculations.

The first trace success test works best with very large amount of testing data, preferably with randomized encryption key to ensure the results are key independent. The reason these results are not presented is that we did not capture enough random key EM traces to warrant the test. The test itself has the model make predictions on all the test data. It then checks if the correct key was predicted. The success rate is calculated as a fraction of how often something was correctly predicted divided by how often it should have been predicted.

As for what the results of the experiment show. As we can see in Fig. 6 we have yet to hit a plateau in the history plot. It is typical for these history plots to taper off. We can

also see in Fig. 9 that the performance of the model is not very good. Yet it still does converge to rank 0 eventually. Our conclusion from this very limited test would be that it is possible that due to the noisy and inexact nature of capturing traces with an EM probe the way we did it is more difficult to learn from the traces. While this proves that it is possible to train a functional model for EM traces using the same hyperparameters as a model for power traces, it does not seem to yield particularly good results.

V. Related Work

In [11] Benadjila et al. present their findings on the strengths and weaknesses of MLP networks and Convolutional Neural Nets (CNNs). The in depth study of the effects of hyperparametrization and the release of their source code to the public makes their research an ideal starting point for newcomers. Studying their code and their results is a great way to learn more.

The ChipWhisperer project [12] is an open source project with both hardware and software to help perform SCA on a multitude of different devices. The use of ChipWhisperer has become very popular due to its capability to reliably replicate results. It offers many advantages over traditional oscilloscope measurements such as guaranteeing synchronization of traces. The popularity of ChipWhisperer helps standardize parts of SCA making it easier to share results.

Introducing new hardware targets for ChipWhisperer is another great way to contribute to making SCA studies more accessible. Once recent such project comes once again from Ryad Benadjila. In [13] Benadjila et al. detail the specifications of their new ChipWhisperer target LEIA. It can be used for capturing side channel data from for instance USIM cards.

In their presentation at blackhat 2018 [1] Perin et al. inspired many researchers with their findings about DL-based SCA. Their results showed the potential of DL since they were able to recover the key from a masked AES implementation.

Finally, in [14] Huanyu et al. talk about the effects of board diversity in DL-based SCA. The surprising ability to generalize across different Printed Circuit Board layout is especially noteworthy. Yet the difficulties presented by board diversity, such as the need for much more generalization in models, makes it difficult to train good models unless very similar hardware is used for training. [15] and [16] by the same authors go more in depth about the subject.

VI. Conclusion

DLSCA is an easy to use tool for getting started with ML based SCA. Its open source nature makes it useful for those interested in furthering current research. In its current state it has enough functionality to explore new questions and yield results in a presentable format.

Using DLSCA we did some very rudimentary testing of EM side channel data using an MLP neural net. The results show that it is possible to train a network to recover a single keybyte from EM traces using DL. This seems to indicate it is possible to recover the entire key if multiple models are used in parallel or if the training strategy is altered. However, the results also showed that hyperparameters that were good for training using power traces captured from the same device are likely to not be as good for EM traces.

The software can be downloaded from github.com/brisfors/DLSCA

A. Acknowledgements

We would like to express our deep gratitude to Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas from ANSSI for their significant contributions to the field of applying DL to SCA and particularily for making their findings and their code publicly available.

This work was supported in part by the research grants 2018-04482 from the Swedish Research Council and 2018-03964 from VINNOVA.

References

[1] G. Perin, B. Ege, and J. van Woudenberg, "Lowering the bar: Deep learning for side-channel analysis (white paper)," August 2018, blackHat'2018.

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[3] Swarup Bhunia Ph.D. and Mark Tehranipoor Ph.D., *Hardware Security: A Hands-on Learning Approach*. Morgan Kaufmann, 2018.

[4] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Proc. of the 16th Annual Int. Cryptology Conf. on Advances in Cryptology*, 1996, pp. 104–113.

[5] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The em side—channel(s)," in *InCryptographic Hardware and Embedded Systems -CHES*, 2003, pp. 29–45.

[6] Paul Kocher, Joshua Jaffe, and Benjamin Jun, "Differential power analysis." Springer-Verlag, 1999, pp. 388–397.

[7] A. L. Samuel, "Some studies in machine learning using the game of checkers," vol. 3, no. 3, pp. 210–229, July 1959-07.

[8] Mor-Yosef S., Samueloff A., Modan B., Navot D., and Schenker JG., "Ranking the risk factors for cesarean: logistic regression analysis of a nationwide study." *Obstetrics & Gynecology*, 1990.

[9] R. Gómez. (2019, April) Understanding categorical cross-entropy loss, binary cross-entropy loss, softmax loss, logistic loss, focal loss and all those confusing names. [Online]. Available: https://gombru.github.io/2018/05/23/cross_entropy_loss/

[10] De, Soham, Mukherjee, Anirbit, and Ullah, Enayat, "Convergence guarantees for RMSProp and ADAM in non-convex optimization and an empirical comparison to Nesterov acceleration," *arXiv:1807.06766 [cs, math, stat]*, Jul. 2018, arXiv: 1807.06766.

[11] R. Benadjila, E. Prouff, R. Strullu, E. Cagli, and C. Dumas, "Study of deep learning techniques for side-channel analysis and introduction to ASCAD database," *IACR Cryptology ePrint Archive*, p. 45, 2018.

[12] NewAE Technology Inc. (2019, April) ChipWhisperer®. [Online]. Available: https://newae.com/tools/chipwhisperer/

[13] R. Benadjila, M. Renard, D. Elbaze, and P. Trébuchet, "LEIA: the Lab Embedded ISO7816 Analyzer A Custom Smartcard Reader for the ChipWhisperer," *SSTIC2019*, p. 30, 2019.

[14] H. Wang, M. Brisfors, S. Forsmark, and E. Dubrova, "How diversity affects deep-learning side-channel attacks," Cryptology ePrint Archive, Report 2019/664, 2019, https://eprint.iacr.org/2019/664.

[15] M. Brisfors and S. Forsmark, "Deep-learning side-channel attacks on aes," p. 24, 2019.

[16] H. Wang, "Side-channel analysis of aes based on deep learning," Master's thesis, KTH, School of Electrical Engineering and Computer Science (EECS), 2019.