How Diversity Affects Deep-Learning Side-Channel Attacks

Huanyu Wang, Martin Brisfors, Sebastian Forsmark, Elena Dubrova

Royal Institute of Technology (KTH), Stockholm, Sweden
{huanyu,brisfors,sforsm,dubrova}@kth.se

Abstract. Deep learning side-channel attacks are an emerging threat to the security of implementations of cryptographic algorithms. The attacker first trains a model on a large set of side-channel traces captured from a chip with a known key. The trained model is then used to recover the unknown key from a few traces captured from a victim chip. The first successful attacks have been demonstrated recently. However, they typically train and test on power traces captured from the same device. In this paper, we show that it is important to train and test on traces captured from different boards and using diverse implementations of the cryptographic algorithm under attack. Otherwise, it is easy to overestimate the classification accuracy. For example, if we train and test an MLP model on power traces captured from the same board, we can recover all key byte values with 96% accuracy from a single trace. However, the single-trace attack accuracy drops to 2.45% if we test on traces captured from a board different from the one we used for training, even if both boards carry identical chips.

Keywords: Side-channel attack, power analysis, deep learning, MLP, CNN, AES.

1 Introduction

Side-channel attacks were introduced by Paul Kocher in his seminal paper on *timing analysis* [16] where he has shown that non-constant running time of a cipher can leak information about its key. Kocher has also pioneered *power analysis* [15] which exploits the fact that circuits typically consume differing amounts of power based on their input data. The power consumption remains one of the most successfully exploited side-channels today. Hardware and software implementations of many important cryptographic algorithms, including the Advanced Encryption Standard (AES) [7], have been broken by power analysis [6, 21, 23].

With advances in Machine Learning (ML) side-channel attacks got a powerful ally. ML techniques are good at finding correlations in raw data, making side-channel analysis considerably more efficient. ML enables an attacker to bypass many existing side-channel countermeasures and break protected implementations. Given huge investments in ML, we are likely see an explosion of ML-based security breaches unless appropriate measures are taken. Therefore, it is important to understand capabilities and limitations of ML side-channel attacks.

Previous Work. Different types ML side-channel attacks have been demonstrated in recent years. Unsupervised learning techniques such as *clustering* (e.g. K-means) and

dimensionality reduction (e.g. principal component analysis) were used to perform either key recovery [12, 30] or pre-processing of the side-channel traces [1]. Supervised learning techniques such as *support vector machines*, *self-organizing maps*, *random forests* and different types of artificial *neural networks* were successful in recovering the key not only from unprotected [2, 11, 13, 18], but also from protected [8, 11, 19, 31] implementations of cryptographic algorithms.

Multiple Layer Perceptron (MLP) and Convolutional Neural Networks (CNNs)-based side channel attacks seem to be particularly powerful. CNNs can overcome trace misalignment and jitter-based countermeasures [4] and break masked AES implementations [24]. MLP and CNN-based attacks significantly outperform template attacks when applied on noisy traces [26]. A reason for this might be that template attacks approximate the data distribution by a multivariate Gaussian distribution whose parameters (i.e. the mean vector and the covariance matrix) are estimated during the profiling phase [1]. This implies that higher-order statistical moments of the leakage distribution are not exploited in the attack, making it sub-optimal or ineffective [20]. Contrarily, MLP- and CNN-based attacks make no assumption on the data distribution and build classifiers directly from a raw data set.

Our Contribution. The majority of previously reported MLP- and CNN-based attacks [4, 14, 20, 25, 26, 29] train and test their networks using traces captured from the same device. These attacks do not take into account the impact of inter-device process variation, which can be significant in advanced technologies as previous work on template attacks shows [5, 10, 22, 27]. In this paper, we go one step further and test MLP and CNN models on traces captured from a printed circuit board different from the one we used for training (see Fig. 1 and 2). We show that ignoring board's diversity can easily lead to an overestimation of classification accuracy. (see Fig. 1 and 2)

We also investigate how cryptographic algorithm implementation diversity affects classification accuracy. It is unlikely that the attacker will know exactly which implementation of the algorithm is executed on the victim chip. We have not seen this problem being addressed before.

The research on deep learning side-channel attacks just started and many questions are open, including how many classifiers are required to recover a full key. According to [24], for AES, "the amount of times that a neural network must be trained is equivalent to the number of bytes in the key". We investigate if MLP and CNN models trained on a fixed key byte position can recover key bytes in other positions.

Paper Outline. The paper is organized as follows. Section 2 gives the background on deep learning, MLP, CNN and AES. Section 3 introduces deep-learning side-channel attacks. Section 4 provides details of the presented attacks on AES-128. Section 5 describes the experimental results. Section 6 concludes the paper and discusses open problems.

2 Background

In this section, we describe how deep learning is used for data classification, give an introduction to MLP and CNN networks and review the AES-128 algorithm. For a broader introduction to deep learning the reader is referred to [9].

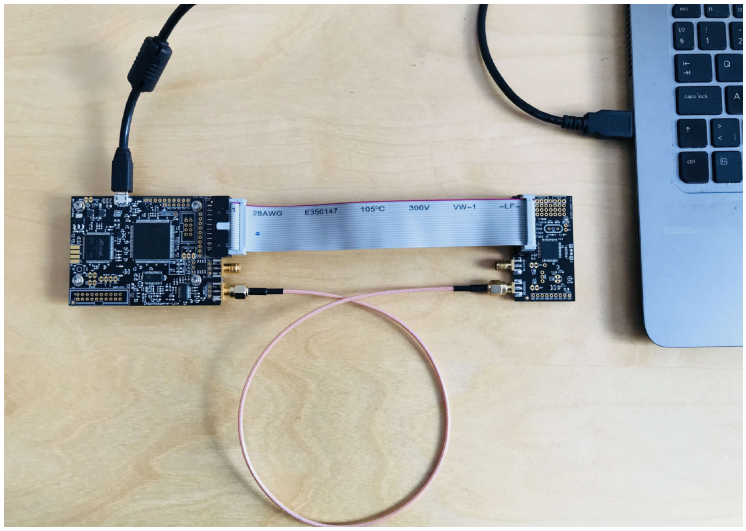

Fig. 1. The first board with ATxmega128D4 microcontroller (right) connected to the ChipWisperer (left).

2.1 Deep Learning for Data Classification

Deep learning is a branch of machine learning which uses deep neural networks as models.

The objective of data classification is to classify data $x \in \mathbb{R}^k$ based on their labels $l(x) \in C$, where k is the number of points in the data and $C = \{0, 1, \ldots, |C|-1\}$ is the set of classification classes. A neural network can be viewed as a mapping $N : \mathbb{R}^k \to \mathbb{R}^{|C|}$ which takes as input data x to classify, and produces as output a *score* vector $s = N(x) \in \mathbb{R}^{|C|}$.

Each label $l(x) \in C$ can be represented as a one-hot encoded *ground truth* vector $t \in \{0,1\}^{|C|}$ defined by:

$$t_i = \begin{cases} 1 \text{ if } i = l(x) \\ 0 \text{ overwise,} \end{cases}$$

where $i \in C$.

To quantify the classification error of the network, different types of *loss functions* are used, in this work we are using *categorical cross-entropy loss*:

$$CE = -\sum_{i \in C} t_i \log \left(\frac{e^{s_i}}{\sum_{j \in C} e^{s_j}} \right)$$

where t_i and s_i are the ground truth and the classifier score for each class $i \in C$. CE loss is often used for multi-class classification, in which each sample can belong to one of $|C|$ classes. In this case, the network outputs a probability over the $|C|$ classes for each sample.

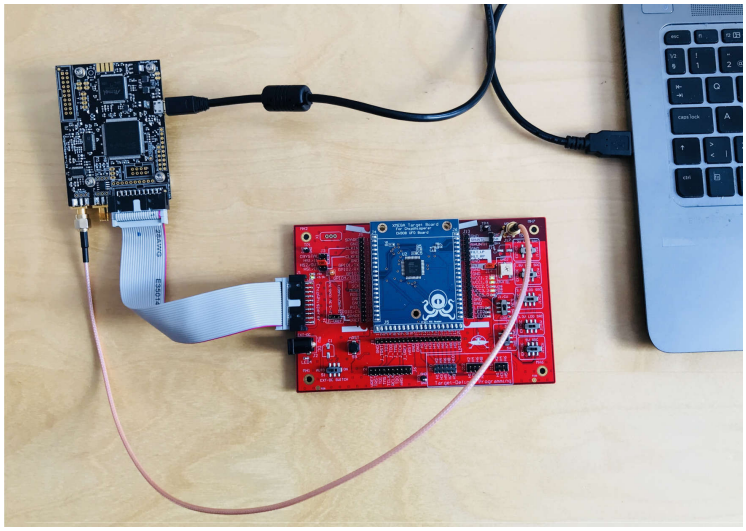

Fig. 2. The second board with ATxmega128D4 microcontroller (right) connected to the ChipWisperer (left).

To minimize the loss, the *gradient* of CE with respect the score s is computed and back-propagated through the network to tune its internal parameters according to the Stochastic Gradient Descent (SGD) algorithm [28] or one its advanced adaptations, e.g. RMSprop which we are using in this work. This is repeated for a chosen number of iterations called *epochs*.

Once the network is trained, to classify a data x whose label $l(x)$ is unknown, we compute

$$l = \underset{i \in |C|}{\text{argmax}}\ s_i$$

If $l = l(x)$, the classification is successful.

2.2 Multiple Layer Perceptron

Multiple Layer Perceptron (MLP) is one of the basic types of deep learning architectures.

MLP is a class of feed-forward neural networks composed of multiple layers with linear and non-linear activation functions. Every layer of an MLP consists of basic elements, called *neurons*. Every neuron is fully connected to all the neurons in the previous and the next layers.

Each neuron in the network has a *bias value*, b, and an *activation function*, f. The connections of the neuron to neurons in the previous layer are defined by the *connection weights*, $w_i \in \mathbb{R}$, $i \in \{1,\ldots,n\}$. These parameters are updated during training of the MLP. Popular activation functions are Rectified Linear Unit (ReLU), hyperbolic tangent (TANH), sigmoid, and softmax.

The output value of a neuron, y, is defined as follows:

$$y = f(\sum_{i=1}^{n} w_i x_i + b), \tag{1}$$

where x_i is the output value of the neuron i in the previous layer, $i \in \{1, 2, \ldots, n\}$.

An MLP network usually contains an input layer, an output layer and one or more dense layers.

1. *Input layer:* The number of neurons in the input layer is determined by the number of points k in the input data.
2. *Dense layers:* These are the fully-connected layers between the input and the output layers.
3. *Output layer:* The number of neurons in the output layer is determined by number of classes $|C|$ which needs to be identified in the input data set.

During the training of an MLP, the weights and bias parameters are updated at every iteration to minimize the loss function.

2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNN) can be considered as a variant of MLP which uses one or more convolution layer in addition to the input layer, dense layers, and the output layer. CNNs were originally proposed for object recognition in images [17]. They have the ability to develop an internal representation of a multi-dimensional data. This allows a trained network to recognize patterns in the data.

The *convolution layer* usually consists of a convolutional operation followed by an activation function such as ReLU, and a pooling layer. The pooling layer is used for dimensionality reduction. In its essence, the convolution layer implements several convolution filters with the same kernel size. The convolution operation is performed between the filter and the input data.

As in the MLP case, the weights and bias parameters are updated at every iteration during training to minimize the loss function.

2.4 AES-128

The AES [7] is a symmetric encryption algorithm standardized by NIST in FIPS 197 and included in ISO/IEC 18033-3. It takes a 128-bit block of plaintext and an n-bit key K, $n = \{128, 192, 256\}$, as input and computes a 128-bit block of ciphertext as output. We use the AES with the key size $n = 128$, AES-128.

The pseudo-code for AES-128 encryption algorithm is shown in Algorithm 1. AES performs encryption iteratively, in 10 rounds for the 128-bit key. Each round except the last repeats the four steps: non-linear substitution, transposition of rows, mixing of columns, and round key addition, but uses a different round key derived from the key K. The last round does not mix columns.

As any block cipher, AES can be used in several modes of operation, including:

Algorithm 1 Pseudo-code of the AES-128 algorithm.

```
// AES-128 Cipher
// in: 128 bits (plaintext)
// out: 128 bits (ciphertext)
// Nr: number of rounds, Nr = 10 for AES-128
// Nb: number of columns in state, Nb = 4
// w: expanded key K, Nb*(Nr+1) = 44 words, (1 word = Nb bytes)
state = in;
AddRoundKey(state, w[0, Nb-1]);
for round = 1 step 1 to Nr-1 do
    SubBytes(state); // Point of attack in round 1
    ShiftRows(state);
    MixColumns(state);
    AddRoundKey(state, w[round*Nb, (round+1)*Nb-1]);
end for
SubBytes(state);
ShiftRows(state);
AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1]);
out = state;
```

1. In the *Electronic Codebook* (ECB) mode, the message is divided into blocks and each block is encrypted separately.
2. In the *Cipher Block Chaining* (CBC) mode, each plaintext block is XORed with the previous ciphertext block before being encrypted. The first plaintext block is XORed with the Initialization Vector (IV).
3. In the *Counter* (CTR) mode, pseudo-random keystream blocks are generated by encrypting consecutive values of a counter. These blocks are then XORed with the plaintext to get the ciphertext. The counter can be any function which produces a sequence which has a guaranteed long period.
4. In the *Output Feedback* (OFB) mode, keystream blocks are generated based on the key, IV, and previous ciphertext blocks. The keystream blocks are then XORed with the plaintext to get the ciphertext.
5. In the *Cipher Feedback* (CFB), first the IV is encrypted and the result is XORed with a block of the plaintext to produce a block of ciphertext. The process is repeated for the next plaintext block using the previous ciphertext block instead of the encrypted IV.

3 Deep-Learning in Side-Channel Analysis

In this section, we describe how deep-learning is used in side-channel analysis.

The aim of side-channel analysis is to find the secret key stored on a victim device implementing some cryptographic algorithm. In this paper, we focus on AES [7] because it is the most popular cryptographic algorithm at present.

3.1 Settings

Deep learning can be used in side-channel analysis in two settings: profiling and non-profiling. *Profiling* attacks [4, 14, 20, 24–26, 29] first learn a leakage profile of the cryptographic algorithm under attack, and then attack. *Non-profiling* attacks [31] attack directly, as the traditional Differential Power Analysis [15] or Correlation Power Analysis (CPA) [3]. We focus on profiling attacks because we believe that they utilize the potential of deep learning to a larger extent.

Side-channel analysis can be done based on different side-channels, including power consumption, electromagnetic emission, or timing. In our experiments, power consumption is used.

3.2 Assumptions

Profiling deep-learning side-channel attacks assume that:

1. The attacker has a device, called the *profiling* device, which is similar to the device under attack and implements the same cryptographic algorithm.
2. The attacker has full control over the profiling device (can apply chosen plaintext, program chosen keys, and capture power traces).
3. The attacker has physical access to the victim device to capture at least one power trace during the execution of the cryptographic algorithm.

3.3 Attack Stages

A profiling deep-learning side-channel attack is done in two stages.

At the *profiling* stage, the selected deep-learning network is trained to learn a leakage profile of the cryptographic algorithm under attack for all possible values of the sensitive variable. For AES, the sensitive variable is typically a key byte. The training is done using a large number of traces captured from the profiling device which are labeled according to the selected power model.

At the *attack* stage, the trained model is used to classify the traces captured from the victim device.

3.4 Power Models

A *power model* describes the power consumption of a device at a selected intermediate point, called *the attack point*, during the execution of the algorithm. If the intermediate result is x, then the power consumption is modelled as $f(x)$, where f is defined by the model.

Common power models for deep-learning side-channel attacks on software implementations of AES are the identity and the Hamming weight.

The *Identity* power model assumes that the power consumption is proportional to the value of the data processed at the attack point.

The *Hamming weight* power model assumes that the power consumption is proportional to the number of 1s in the data processed at the attack point.

A power model defines the number of outputs in a classifier. If the data is a byte, the identity model leads to the set of 256 classes $C = \{0, 1, \ldots, 255\}$ and the Hamming weight model to the set of 9 classes $C = \{0, 1, \ldots, 8\}$.

4 Attacking AES-128

In this section, we give details of the presented attacks on AES-128.

4.1 Attack Point and Power Model

We use the state after the first $SubBytes()$ step in round 1 as the attack point (see marked line in pseudocode 1). The function $SubBytes()$ applies the AES 8-input 8-output substitution box $SBox$ to state byte-by-byte. After the first $SubBytes()$ step, the state becomes

$$\text{state} = SBox[\text{in} \oplus \text{key}],$$

where "$\oplus$" is the bitwise XOR and key $= w[0, Nb-1]$.

The output of the S-box is a suitable attack point for software implementations of AES because its value needs to be loaded from a memory onto a data bus.

We use the identity power model.

4.2 Attack Steps

Profiling stage:

1. Use the profiling device to encrypt a large number of random plaintexts $P = \{p_1, \ldots, p_n\}$ for known random keys $K = \{k_1, \ldots, k_n\}$ and record the resulting power traces $T = \{t_1, \ldots, t_n\}$.
2. Partition the secret key into bytes and select one byte position j, for any $j \in \{1, \ldots, 16\}$.
3. Let $p_{i,j}$ and $k_{i,j}$ be the jth byte of the plaintext p_i and the key k_i, respectively, for $i \in \{1, \ldots, n\}$. Assign to each trace $t_i \in T$ a label $l_{i,j}$ equal to the value of the $SBox$ output byte computed for $p_{i,j}$ and $k_{i,j}$.
4. Use the labeled power traces to train a classifier to learn the leakage profile of AES for all possible byte values of the $SBox$ output. The training strategy is described in the next subsection.

Attack stage:

1. Use the victim device to encrypt a small number of random plaintexts $P = \{p_1, \ldots, p_m\}$ for the unknown key and record the resulting power traces $T = \{t_1, \ldots, t_m\}$.
2. Use the trained network to classify the traces in T.

4.3 Training MLP and CNN Models

In this section, we describe the how we trained MLP and CNN models.

During the first round of an AES, the $SBox$ operation is executed within the first 3000 data points. Using CPA, we manually identified the positions of leakage points for each key byte. This allowed us to reduce the number of points in the input data to $k = 150$ for MLP and $k = 50$ for CNN. Obviously, reducing the size of the input layer

Layer Type	Output Shape	Parameter #
Input (Dense)	(None, 200)	30200
Dense 1	(None, 200)	40200
Dense 2	(None, 200)	40200
Dense 3	(None, 200)	40200
Dense 4	(None, 200)	40200
Output (Dense)	(None, 256)	51456

Total Parameters: 242,456

Table 1. MLP architecture summary.

Layer Type	Output Shape	Parameter #
Input (Dense)	(None, 50, 1)	0
Conv1D 1	(None, 50, 64)	768
AveragePooling1 1	(None, 25, 64)	0
Conv1D 2	(None, 25, 128)	90240
AveragePooling1 2	(None, 12, 128)	0
Conv1D 3	(None, 12, 256)	360704
AveragePooling1 3	(None, 6, 256)	0
Conv1D 4	(None, 6, 512)	1442304
AveragePooling1 4	(None, 3, 512)	0
Conv1D 5	(None, 3, 512)	2884096
AveragePooling1 5	(None, 1, 512)	0
Flatten	(None, 512)	0
Dense 1	(None, 4096)	2101248
Dense 2	(None, 4096)	16781312
Output (Dense)	(None, 256)	1048832

Total Parameters: 24,709,504

Table 2. CNN architecture summary.

reduces the model complexity and training time. We trained MLP on the 1st key byte and CNN on the 3rd key byte. The position of the byte does not seem to matter.

Architectures of the best models which we use in the experiments in Section 5 (unless specified otherwise) are show in Tables 1 and 2. The CNN architecture is the same as the one in [26] except for the input layer size. The optimal input layer size depends on the implementation of the cryptographic algorithm under attack. 50K traces were used of training and 1K traces for validation.

The MLP architecture is an evolved version of the MLP architecture in [26]. We searched for the best parameters to use for learning rate, number of epochs, validation split and number of layers as well as tested using dropout regularization, learning rate decay and changing the input. Further optimization could be done; indeed parallel to performing our tests a marginally better model was trained, but the model presented here was the best at the time. The RMSprop optimizer was used with a learning rate of 0.00008 and no learning rate decay. The model was trained for 2000 epochs on 175K

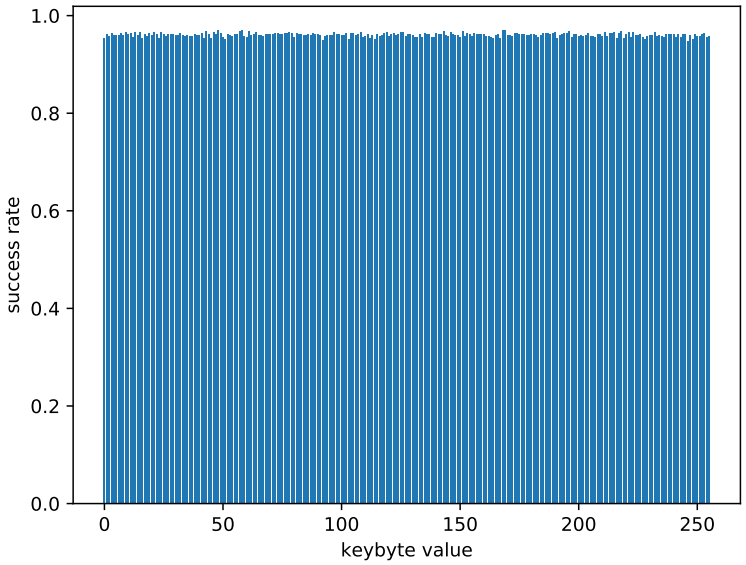

Fig. 3. Probability of recovering the 1st key byte from a single trace of XMEGA1 AES by MLP trained on XMEGA1 AES for different byte values.

traces with 75K traces set aside for validation. No dropout was used. Finally, we settled on 4 dense hidden layers plus an input and an output layer, as shown in Table 2.

5 Experimental Results

We used the ChipWhisperer board to capture traces from the profiling and victim devices. Two identical 8-bit Atmel microcontrollers ATxmega128D4 placed on different types of printed circuit boards (see Fig. 1 and 2) were used in the experiments. In this section, we refer to the devices shown in Fig. 1 and 2 as XMEGA1 and XMEGA2, respectively.

5.1 Inter-Board Diversity

The attacker may not be able to acquire exactly the same board as the victim chip for training. The aim of this experiment was to evaluate how the inter-board diversity affects MLP and CNN classification accuracy.

We used XMEGA1 and XMEGA2 programmed to the same version of AES to encrypt 600K random plaintexts and random keys and recorded their power traces. Then, we checked how often the MLP and CNN models trained on XMEGA1 can recover the 1st key byte from a single power trace in both cases.

The MLP model was able to recover the 1st key byte from a single power trace from XMEGA1 in 96.1% of cases. However, for XMEGA2, the success rate is only 2.45%.

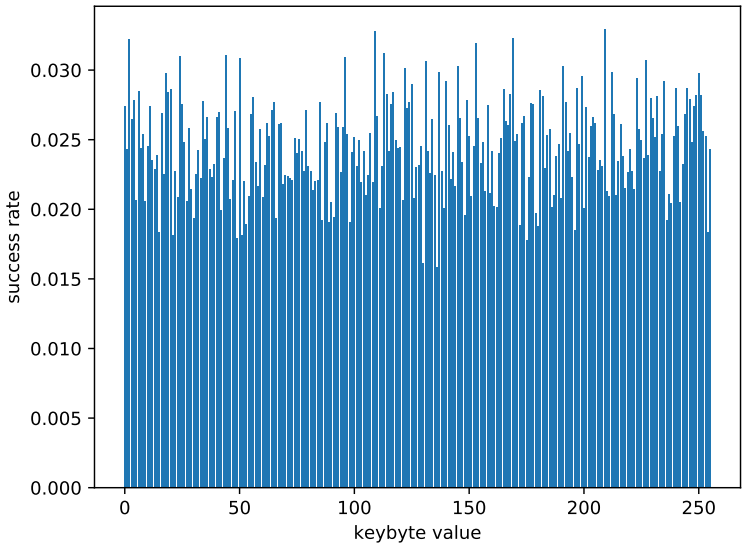

Fig. 4. Probability of recovering the 1st key byte from a single trace of XMEGA2 AES by MLP trained on XMEGA1 AES for different byte values. Note that the range of Y-axis is $[0;0.03]$, while in Fig. 3 it is $[0;1]$.

Figures 3 and 4 show the distribution of probabilities of recovering different key byte values of the 1st key byte for both cases. We can conclude that it is easy to overestimate the classification accuracy if the training and testing are done on the same board.

The CNN model was able to recover the 1st key byte from a single power trace from XMEGA1 and XMEGA2 in 1.9% and 1.4% of cases, respectively. CNNs seem not to be suitable for single-trace attacks.

5.2 Implementation Diversity

AES can be implemented in many different versions and several modes of block cipher operation. The attacker may not know exactly which AES implementation is run by the victim chip. The aim of this experiment was to investigate how different modes of operation affect classification accuracy.

We used XMEGA1 programmed to 5 different modes of operation: ECB, CBC, CFB, CTR, and OFB, to encrypt 100 random plaintexts with a fixed key and recorded the power traces. Then, we evaluated if the MLP and CNN models trained on XMEGA1 AES-ECB can recover the 1st key byte. As expected, both the MLP and CNN models were able to recover the key byte correctly only in the ECB mode.

For the MLP, the results of the 1st key byte ranking are shown in Fig. 5. When the rank of a byte reaches zero, the byte is known.

For the CNN, the results are shown in Fig. 7.

We would like to stress that we do not recommend using variable modes of operation as a countermeasure against side-channel attacks. The number of modes is small and

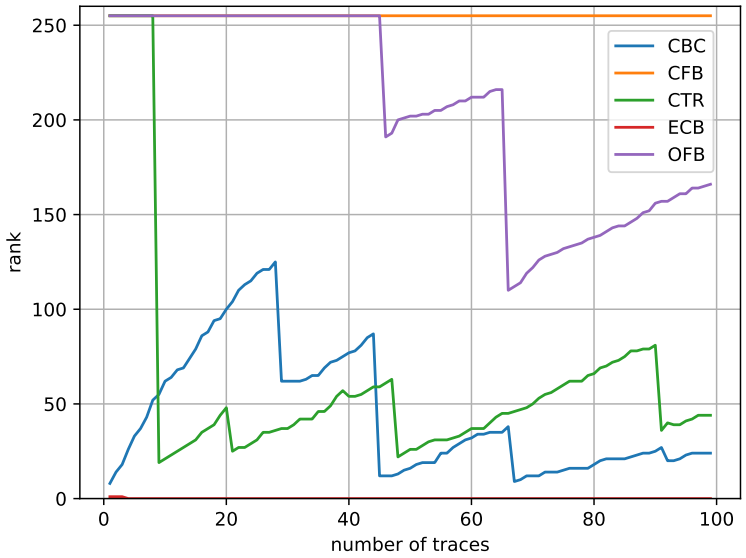

Fig. 5. Results of the 1st key byte ranking for n traces of XMEGA1 AES in different modes of operation by MLP trained on XMEGA1 AES-ECB, for $1 \leq n \leq 100$.

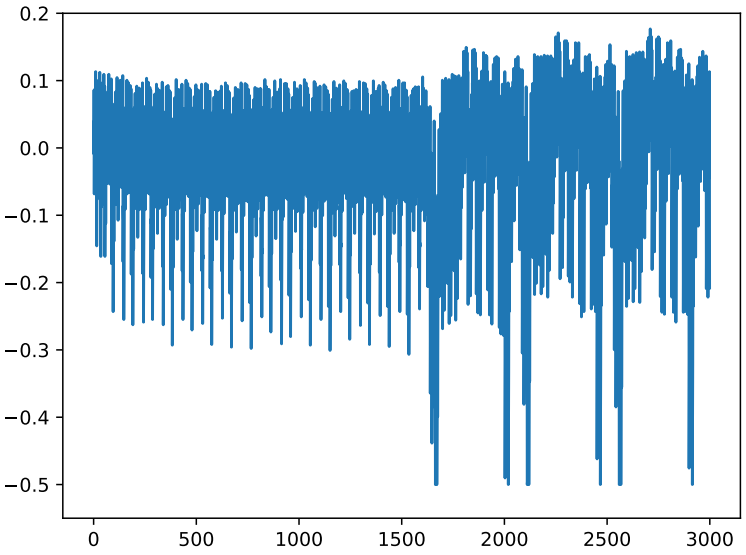

Fig. 6. The first 3000 data points of a power trace of XMEGA1 during the execution of AES-ECB.

power traces for different modes look different (see Fig. 6 and 8 comparing power traces of ECB to CTR modes). So, an experienced attacker might be able to analyze a power

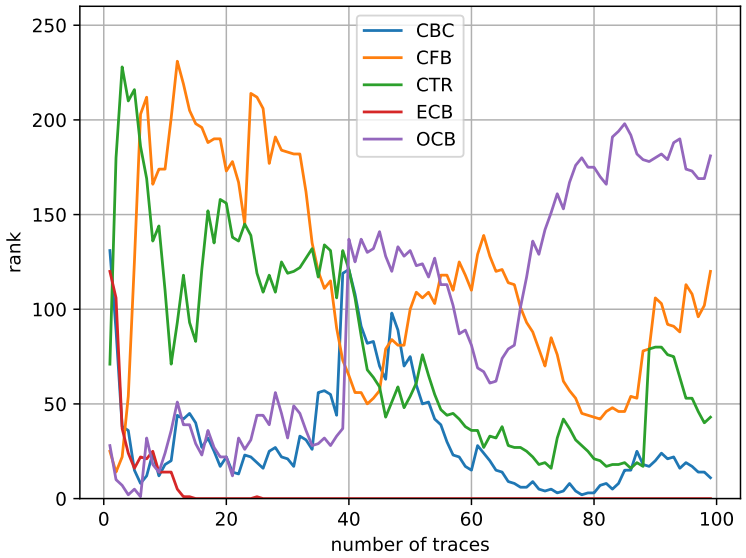

Fig. 7. Results of the 1st key byte ranking for n traces of XMEGA1 AES in different modes of operation by CNN trained on XMEGA1 AES-ECB, for $1 \leq n \leq 100$.

trace from the victim device, deduce which mode of operation is used, and adjust the attack strategy accordingly. For example, in the CBC mode, the first plaintext block is XORed with an IV before being encrypted. So, if the IV is public, the attacker can re-map the labels of ECB into the labels of CBC. Alternatively, the attacker can re-train the network for the targeted mode.

5.3 Full Key Recovery

The aim of this experiment was to determine if a network trained on one key byte position can recover key bytes in other positions. We evaluated how often the MLP and CNN models trained on XMEGA1 power traces, on a fixed key byte, can recover all key bytes from XMEGA1 and XMEGA2.

It is important to mention that a CPA attack on XMEGA1 or XMEGA2 implementations of AES takes about 50 traces (for the default ECB mode). If an attacker can capture that many traces from a victim device, the attacker can simply use CPA to recover all key bytes directly. Therefore, we limit the number of traces given to MLP and CNN models to 50.

The CNN model was not able to recover all key bytes from 50 power traces from neither XMEGA2, nor XMEGA1. We increased the number of traces to 1K to see if the CNN will be able to recover the full key from XMEGA2 using more traces. Indeed, with 398 traces on average, the CNN model can recover the full key from XMEGA2 and with 159 traces on average from XMEGA1. However, given that CPA can recover the full key from 50 traces, using CNN for attacks on unprotected AES does not seem

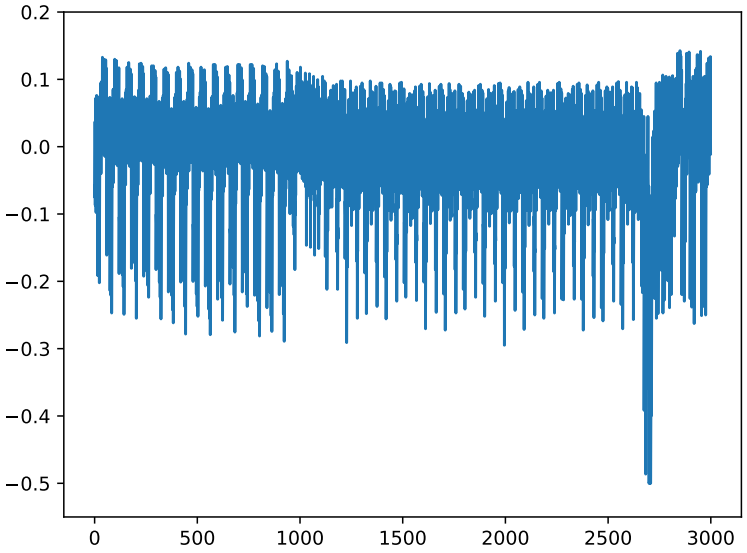

Fig. 8. The first 3000 data points of a power trace of XMEGA1 during the execution of AES-CTR.

useful. Note, though, that the CNNs can break protected implementations of AES, while CPA cannot. So, the full power of CNN becomes apparent on more difficult problems. We included CNNs in our experiments for completeness.

For the MLP model, the success rate was also 0 for both XMEGA1 and XMEGA2. To further investigate this problem, we trained a new MLP model using a union of 16 sets of power traces $T_i = \{t_{1,i}, \ldots, t_{n,i}\}$ from XMEGA1 in which each trace $t_{j,i} \in T_i$ contains 95 points corresponding to the time interval when *SBox* evaluates the ith byte, $i \in \{1, \ldots, 16\}$, $j \in \{1, \ldots, n\}$, $n = 70$K. This MLP model was always able to recover all key bytes from 5 power traces from XMEGA1 (using 2 traces on average), and was never able to recover all key bytes from 50 power traces from XMEGA2.

These results re-confirm the results of our experiment in subsection 5.1 - it is easy to overestimate the classification accuracy if training and testing are performed on the same board.

6 Conclusion

In this paper, we show that it is important to train and test neural network models on traces captured from different boards and using diverse implementations of the cryptographic algorithm under attack. Otherwise, it is easy to overestimate the classification accuracy of the trained network.

Many interesting open problems remain, including understanding why some key byte values are easier to learn than others, comparing 256- and 9-classifiers, and investigating if classification accuracy can be improved by introducing diversity at the training

stage. Certainly the most important open problem is how deep-learning side-channel attacks can be mitigated. Assuring a unique per device implementation diversity seems to be a way to go.

Acknowledgement

This work was supported in part by the research grant No 2018-04482 from the Swedish Research Council.

References

1. Archambeau, C., Peeters, E., Standaert, F.X., Quisquater, J.J.: Template attacks in principal subspaces. In: L. Goubin, M. Matsui (eds.) Cryptographic Hardware and Embedded Systems - CHES 2006, pp. 1–14. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
2. Bartkewitz, T., Lemke-Rust, K.: Efficient template attacks based on probabilistic multi-class support vector machines. In: S. Mangard (ed.) Smart Card Research and Advanced Applications, pp. 263–276. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
3. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: M. Joye, J.J. Quisquater (eds.) Cryptographic Hardware and Embedded Systems - CHES 2004, pp. 16–29 (2004)
4. Cagli, E., Dumas, C., Prouff, E.: Convolutional neural networks with data augmentation against jitter-based countermeasures. In: W. Fischer, N. Homma (eds.) Cryptographic Hardware and Embedded Systems – CHES 2017, pp. 45–68. Springer International Publishing, Cham (2017)
5. Choudary, M.O., Kuhn, M.G.: Efficient, portable template attacks. IEEE Trans. Information Forensics and Security **13**(2), 490–501 (2018)
6. Clavier, C., Danger, J.L., Duc, G., Elaabid, M.A., Gérard, B., Guilley, S., Heuser, A., Kasper, M., Li, Y., Lomné, V., et al.: Practical improvements of side-channel attacks on AES: feedback from the 2nd DPA contest. Journal of Cryptographic Engineering **4**(4), 259–274 (2014)
7. Daemen, J., Rijmen, V.: The Design of Rijndael. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2002)
8. Gilmore, R., Hanley, N., O'Neill, M.: Neural network based attack on a masked implementation of AES. In: 2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 106–111 (2015)
9. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016). http://www.deeplearningbook.org
10. Hanley, N., O'Neill, M., Tunstall, M., Marnane, W.P.: Empirical evaluation of multi-device profiling side-channel attacks. In: 2014 IEEE Workshop on Signal Processing Systems (SiPS), pp. 1–6 (2014). DOI 10.1109/SiPS.2014.6986091
11. Heuser, A., Zohner, M.: Intelligent machine homicide. In: W. Schindler, S.A. Huss (eds.) Constructive Side-Channel Analysis and Secure Design, pp. 249–264. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
12. Heyszl, J., Ibing, A., Mangard, S., Santis, F.D., Sigl, G.: Clustering algorithms for non-profiled single-execution attacks on exponentiations. IACR Cryptology ePrint Archive, 2013:438
13. Hospodar, G., Gierlichs, B., De Mulder, E., Verbauwhede, I., Vandewalle, J.: Machine learning in side-channel analysis: a first study. Journal of Cryptographic Engineering **1**(4), 293 (2011)

14. Kim, J., Picek, S., Heuser, A., Bhasin, S., Hanjalic, A.: Make some noise: Unleashing the power of convolutional neural networks for profiled side-channel analysis. IACR Cryptology ePrint Archive, 2018:1023 (2018)
15. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. pp. 388–397. Springer-Verlag (1999)
16. Kocher, P.C.: 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, pp. 104–113 (1996)
17. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. In: Proceedings of the IEEE, pp. 2278–2324 (1998)
18. Lerman, L., Bontempi, G., Markowitch, O.: Power analysis attack: An approach based on machine learning **3**, ied Cryptography (2014)
19. Lerman, L., Bontempi, G., Markowitch, O.: A machine learning approach against a masked AES. Journal of Cryptographic Engineering **5**(2), 123–139 (2015). DOI 10.1007/s13389-014-0089-3. URL https://doi.org/10.1007/s13389-014-0089-3
20. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: 6th International Conference on Security, Privacy, and Applied Cryptography Engineering, pp. 3–26 (2016)
21. Mangard, S., Pramstaller, N., Oswald, E.: Successfully attacking masked AES hardware implementations. In: International Workshop on Cryptographic Hardware and Embedded Systems, pp. 157–171. Springer (2005)
22. Montminy, D.P., Baldwin, R.O., Temple, M.A., Laspe, E.D.: Improving cross-device attacks using zero-mean unit-variance normalization. Journal of Cryptographic Engineering **3**(2), 99–110 (2013). DOI 10.1007/s13389-012-0038-y. URL https://doi.org/10.1007/s13389-012-0038-y
23. Ors, S.B., Gurkaynak, F., Oswald, E., Preneel, B.: Power-analysis attack on an ASIC AES implementation. In: Proc. Int. Conf. on Information Technology: Coding and Computing, vol. 2, pp. 546–552 Vol.2 (2004)
24. Perin, G., Ege, B., van Woudenberg, J.: Lowering the bar: Deep learning for side-channel analysis (white paper) (2018). BlackHat'2018
25. Pfeifer, C., Haddad, P.: Spread: a new layer for profiled deep-learning side-channel attacks. IACR Cryptology ePrint Archive, 2018:880 (2018)
26. Prouff, E., Strullu, R., Benadjila, R., Cagli, E., Canovas, C.: Study of deep learning techniques for side-channel analysis and introduction to ascad database. IACR Cryptology ePrint Archive, 2018:053 (2018)
27. Renauld, M., Standaert, F.X., Veyrat-Charvillon, N., Kamel, D., Flandre, D.: A formal study of power variability issues and side-channel attacks for nanoscale devices. In: K.G. Paterson (ed.) Advances in Cryptology – EUROCRYPT 2011, pp. 109–128. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
28. Robbins, H., Monro, S.: A stochastic approximation method. Ann. Math. Statist. **22**, 400–407 (1951)
29. Samiotis, I.P.: Side-channel attacks using convolutional neural networks. MSc Thesis, TUDelft (2018)
30. Souissi, Y., Nassar, M., Guilley, S., Danger, J.L., Flament, F.: First principal components analysis: A new side channel distinguisher. In: K.H. Rhee, D. Nyang (eds.) Information Security and Cryptology - ICISC 2010, pp. 407–419. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
31. Timon, B.: Non-profiled deep learning-based side-channel attacks. IACR Cryptology ePrint Archive, 2018:196 (2018)