Attacking AES using Deep Learning

Amin Sarihi
New Mexico State university
Las Cruces, New Mexico
sarihi@nmsu.edu

Hunter Stuckey
New Mexico State university
Las Cruces, New Mexico
hss127@nmsu.edu

ABSTRACT

With the rise of machine learning, encryption algorithms are not as safe as before. AES, which is considered as the standard for encryption in the industry, has been compromised by deep learning. A secret key can be inferred by analyzing the side channel information obtained from hardware executing the encryption method. This imposes a real threat to integrity of information that is meant to be kept secret. The goal of this project is to extract AES's secret key, byte by byte form power traces obtained form Chipwhisperer using deep learning. We are targeting the first 4 key bytes in this report. Our results show that we can recover the first key byte with a probability of 99.7% and recover the other three with a probability of at least 94% .

KEYWORDS

AES, Side-channel Attack, Deep Learning,Multi-layer perceptron, Convolutional neural network

1 INTRODUCTION

AES is a sound cryptography algorithm, and it has been proven safe. It is impossible to test all the $2^{keysize}$ possible keys in practice. To overcome that, side-channel information can be used as a shortcut to predict the right key. Side-channels are information leaked from the hardware implementation. Examples of side channel information are power trace, timing information, and electromagnetic waves [7]. An attacker can analyze this information to extract information about the encryption algorithm, which in turn would take a serious toll on the data security. Statistical approaches such as differential power analysis [8] and correlation power analysis [3] have been successful in recovering the key before; however, deep-learning attacks need fewer power traces to infer the secret key. Moreover, the attacker doesn't deal with statistical information when using this method [7]. The goal of the project is to feed a set of unseen traces to our deep learning architecture and try to recover the first 4 bytes of the whole 16-byte key.

Figure 1 shows a sample of a power trace obtained from an XMEGA128 device. The shape of the spikes in the trace changes as the plaintext and the key changes, because each part of the trace is correlated with the output hamming weight of different stages[6]. Each power trace contains 5000 samples. To achieve the best result in the shortest time, we will feed a small window of total samples for each key byte. Then, by using two deep neural network architectures, we will analyze the data and compare the results. Our experiments show that DNN (Deep neural network) outperforms CNN by nearly 15% in terms of accuracy. The remainder of the report is organized as follows. Section 2 sheds light on AES,information leakage, and side-channel information. Our method is explained in section 3. The results are analyzed in section 4. Section 5 concludes the paper and future works are summarize in section 6.

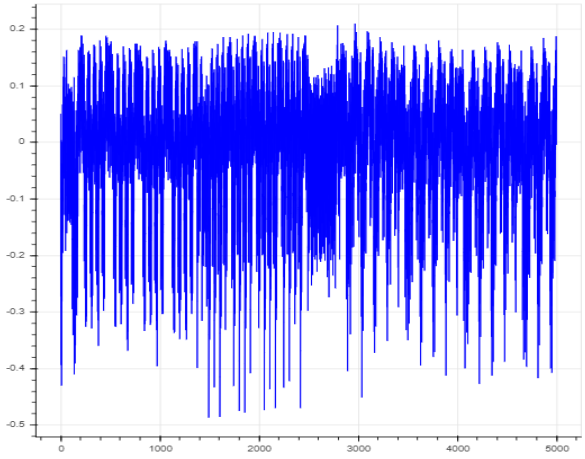

Figure 1: Power trace obtained from XMEGA128 running AES

2 BACKGROUND

2.1 AES

AES is a subset of the Rijndael block cipher proposed by two cryptographers, Vincent Rijmen and Joan Daemen, who submitted a proposal to NIST (National Institute of Standards and Technology) during AES selection process. Rijndael is a family of ciphers with different key and block sizes. For AES, NIST selected three members of the Rijndael family, each with a block size of 128 bits, but three different key lengths: 128, 192, and 256 bits [4]. In this project, we will use the 128-bit version. The first two stages of AES contain the most information for recovering the key. These two stages are called AddRoundKey and SubByte. In the AddRoundKey, the plaintext is X-ored with the secret key. Next, in the SubByte stage, the output of xor result is fed to Sbox, and the data is substituted. Full animated steps of AES can be found in [12]. Sbox is typically used to obscure the relationship between the key and the ciphertext. Information at this stage can be used to recover the key. Figure 2 shows the first two stages of the AES Algorithm. There are other leakage models such as plaintext-key-xor, round 1-2 state-diff-Sbox which are out of the scope of this report and are preserved for future work.

2.2 Side-channel attacks

Side channel attacks are done based on the hardware leakage information rather than the algorithm itself [11]. The most trivial example would be trying to guess what number you're dialling on

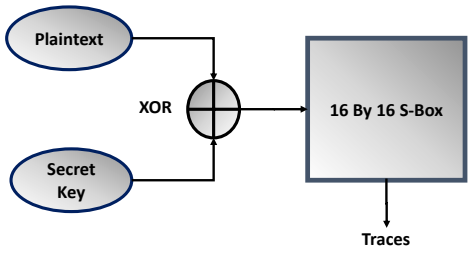

Figure 2: The traces are the the output of the S-Box stage

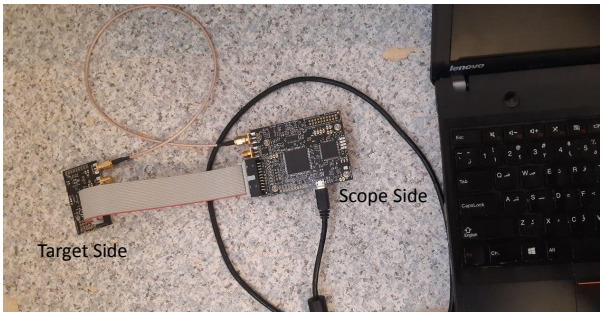

Figure 3: Chipwhisperer [10] setup for capturing power traces

your phone based on the tone sounds. In AES context, electromagnetic emissions and power consumption traces are used for analysis by researchers. With the advent of GPUs and open-source software libraries for machine learning such as Tensorflorw [1], training and testing process of DNN (Deep neural networks) has become fatser than before. Since brute forcing AES is computationally expensive, researchers have utilized side channels to recover the key. Our attack on AES is categorized as profiling side channel attacks which are in fact among the most powerful ones. They assume that the adversary acquires a copy of the final target to precisely tweak all the parameters to tune into best model to attack [9].

3 OUR METHOD

Our method is based on capturing AES power traces using Chipwhisperer. Our setup is shown in figure 3. The target board is an XMEGA128. We run AES on the target side and collect the traces by the scope side. Each trace contains a set of metadata as well, such as the used keys and plaintexts. The steps include capturing N=145000 traces with t=5000 samples each. During the capture stage, we fix the key-value and obtain different waveforms with different plaintexts. Further, we will label each trace with its first stage SBox output. Figure 4 shows ten samples of a power trace, which is the input to our DNN.

In AES128, the key and the plaintext are 128 bits, which is 16 Bytes. We are targeting the first 4 bytes of the key in this project. For the remaining 12, we need to train 12 other model which correspond to each key byte separately which is preserved for future work. Each power trace corresponds to a specific key and a random plaintext

```
1  6.054687500000000000e-02
2  -2.646484375000000000e-01
3  -1.494140625000000000e-01
4  -1.523437500000000000e-01
5  1.953125000000000000e-03
6  -3.925781250000000000e-01
7  -2.275390625000000000e-01
8  -2.080078125000000000e-01
9  -3.320312500000000000e-02
10 -4.257812500000000000e-01
```

Figure 4: 10 samples of a single trace

pair. Then, we'll use the trained model to test some unseen traces in which the plaintext is known, but the key is unknown.

The computer and the board communicate through a VirtualBox image run on Oracle VM VirtualBox. This image contains the necessary libraries and compilers to program the scope and the target. The required steps for this have been put in a jupyter notebook file in the repository [2]. After we save in NumPy arrays format, we can store them in separate files to use them in our python code. The codes and the dataset have been uploaded to our Github repository [2].

3.1 Data Analysis

Since Every portion of the traces corresponds to a significant stage in AES, we have to find the right window to filter out the traces for each of the key bytes separately. We are using Pearson correlation coefficients to find out where leakage happens. The results of our findings can be found in table 1. Moreover, figure 7 shows where the most leakage is happening for the first byte. Although it's happening between 1480 and 1530, to further improve our accuracy, we expanded the window to 200 samples. It will hurt the accuracy of Byte 1-3 in case it is shrunk to 50 samples. The depicted signal variations are caused by the differences in the Sbox output hamming weight. Hamming weight is defined as the number of the ones in the Sbox output. Hence, the more ones we have, the greater the hamming weight is. The multi-colored region also has the highest correlation. Since the most correlated information lies within the samples 1400-2500 for the first 4 key bytes, we trimmed our dataset to 1100 samples long. We train the network with 101500 traces and 43500 traces are kept for validation to make sure the network in not overiftting. Next, we normalize the data to have a meaningful range of data and minimize training time.

3.2 MLP

The MLP architecture is depicted in figure 6. The details of each layer is given in table 2. Our MLP model was created using Keras library. The produced output is the probability of each Sbox labels.

Table 1: Best leakage points in the trace

Byte Number	1	2	3	4
Sample range	1405-1605	1655-1855	1955-2105	2155-2355
Highest correlation	0.83	0.75	0.77	0.78

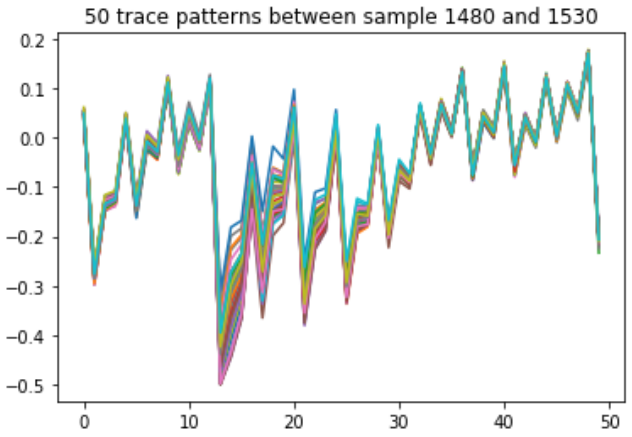

Figure 5: Traces vary according to their Sbox Hamming weight.

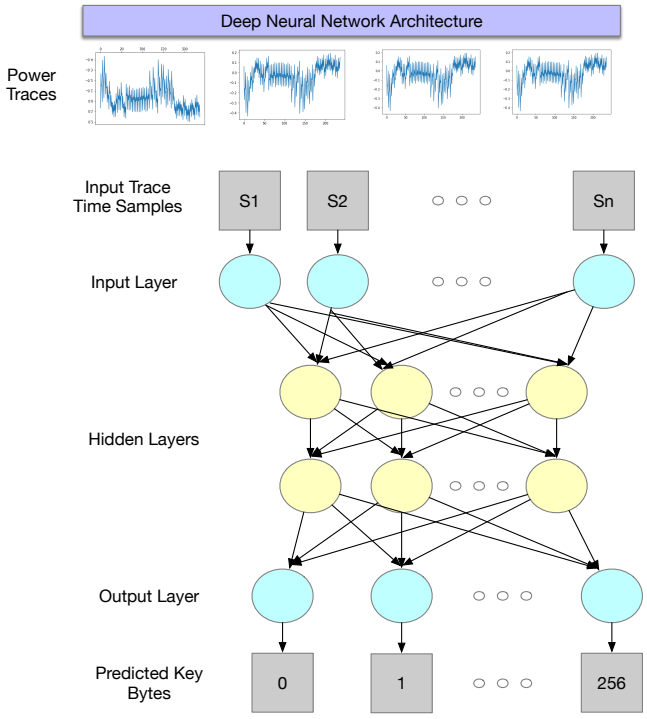

Figure 6: MLP architecture used in our method

Softmax layer produces a 256 element array containing the probability of each Sbox label. We will xor the most probable Sbox label with the corresponding plaintext to get the key.

Table 2: MLP configuration

Name	Neurons	Activation Function
Input Layer	200	-
Hidden Layer 1	450	ReLu
Hidden Layer 2	350	ReLu
Output Layer	450	Softmax

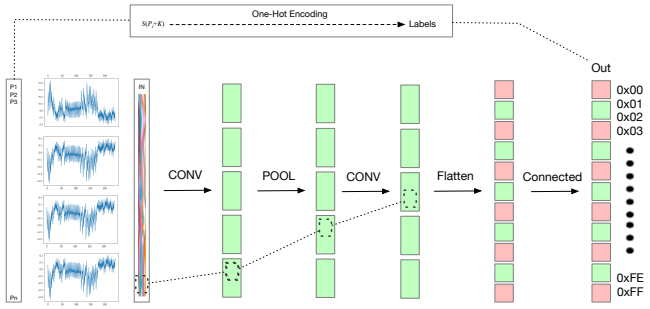

Figure 7: CNN architecture used in our method

3.3 CNN

The CNN was created via the Keras API and tested using a desktop computer. It was based off of a previously done study, with an extra convolutional layer was added after the max pooling layer. Figure 7 represents the inputs/outputs of the layers, and the overall architecture of the model.

We used a 5-layer CNN for this experiment, consisting of 2 convolutional layers, 1 max pooling layer, a flattening layer, and then a SoftMax layer. The CNN was sent data from a range surrounding the leakage points. For the first byte was 1405-1605, but for the later bytes, the window was shortened, and the time points vary more. The model took approximately 20 minutes per trial to train. This was just enough time to test different configurations and the like.

The data fed into the CNN was not time-synchronized, unlike previously done studies, the time windows were picked based off of trial and error. This method could be improved upon in the future greatly by figuring out certain 'trigger' events combined with the Pearson correlation that would determine the best spot for analysis surrounding the leakages in the power traces. If either the time windows fed into the CNN for analysis are too small or the number of layers increases, then the likelihood of overfitting increases as well.

4 RESULTS

Validation loss and accuracy of the first Byte using MLP are depicted in figure 8. Byte 0 reaches 99.9% validation and loss accuracy after 60 epochs. As a result, loss reaches nearly 0. Unlike Byte 0, bytes 1, 2, and 3 do not reach 99% accuracy. Their validation acracy is 95%, 95.5%, 95% respectively. Their loss value tend to hike from 0.25 to 0.50 starting form epoch number 10 according to figure 8. Although the model validation loss grows gradually, validation accuracy increases by about 1 percent as well at the same time. At first glance, it seem that the model is overfitting; however, it is not. The fact is that the probability of the right key guess is dropping, for example, from 0.9 to 0.7 and that causes the validation loss to grow; however, the key prediction gets more accurate by 1% for bytes 1 through 3. In order to make sure that above analysis is right, we measured the testing accuracy . Results are given in table 4. As can be seen, accuracy, Precision, Recall, and F1-score are keeping up with the validtaion accuracy.

As for the CNN architecture, mild success was achieved for recovering the first four bytes of the AES encryption key from the aTmega processor. The first byte was predicted very accurately, while the remaining bytes were predicted with accuracy of around

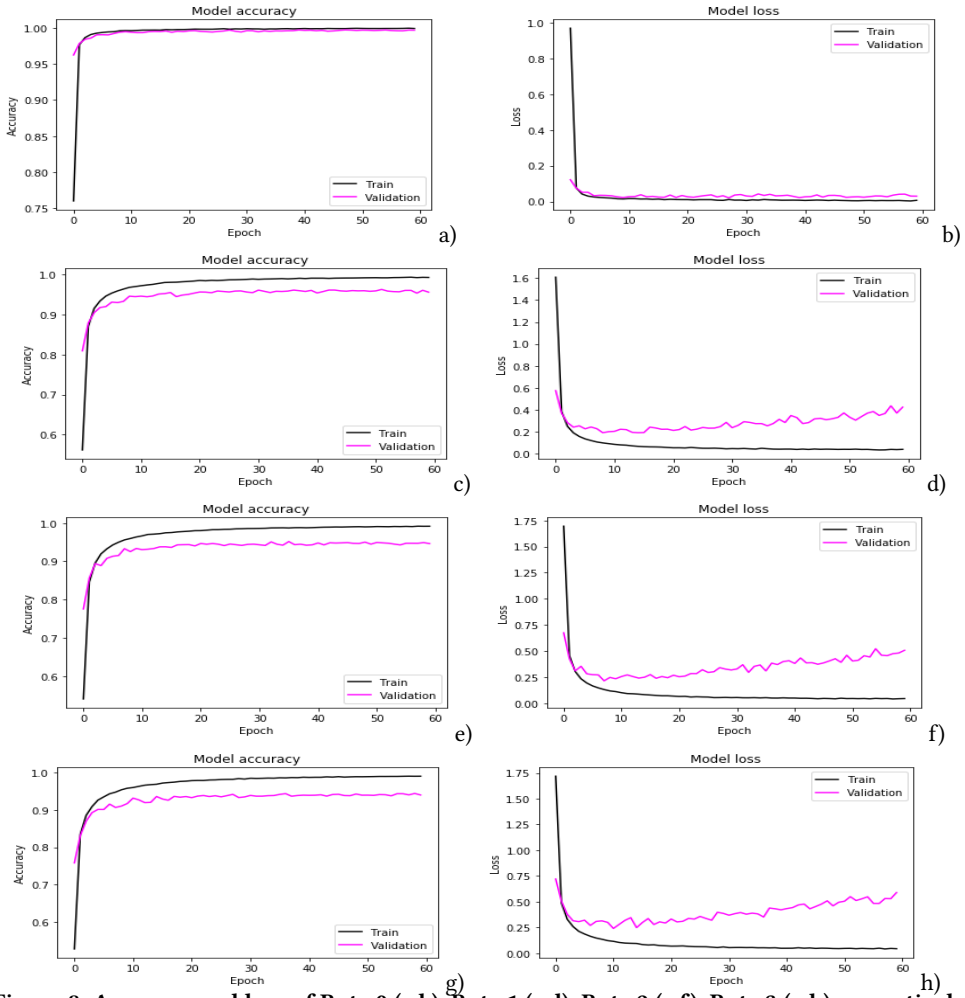

Figure 8: Accuracy and loss of Byte 0 (a,b), Byte 1 (c,d), Byte 2 (e,f), Byte 3 (g,h) respectively

the 80-percentile marks. The CNN models from training were able to correctly predict the key-bytes with a percentage resembling the validation accuracy from the training data. Figure 9 demonstrates the accuracy and the loss from the training on the first four bytes of data.

The accuracy and validation accuracy follow each other a bit more closely than the MLP models; however, the accuracy for the bytes beyond the first haven't reached higher than 83. There were also many more training epochs required to get that accuracy as well. The test/validation accuracy was much lower in the bytes1-3 most likely due to over-fitting and model incompatibility. Below is the table of results from testing the CNN models for key bytes 0-3.

There are various concerns of over-fitting the CNN model to the data. The validation accuracy does keep up with the training accuracy, and we are able to correctly predict the keys with given accuracy when running through testing, but it appears the data selection can be too narrow/specific. Being able to take a model and use it on data from another board would be one of the true tests of this AES side-channel attack. Over-fitting with the CNN model lowers the likelihood of it being able to create an effective attack on other boards with similar, but not the same, power leakage traces.

5 CONCLUSION

This work utilizes side-channel information and deep learning to recover AES key bytes. Side-channels are power traces from an aTmega chip running a software version of AES. The accuracy of the first byte using MLP is 99.7% and compared to [5], it was achieved

Table 3: MLP Accuracy for Bytes 0 through 3

Byte number	0	1	2	3
Accuracy	99.7%	95.7%	94.9%	94.1%
Precision	99.7%	95.9%	95.2%	94.4%
Recall	99.7%	95.7%	94.9%	94.1%
F1-score	99.7%	95.7%	94.9%	94.1%

Table 4: CNN Accuracy for Bytes 0 through 3

Byte number	0	1	2	3
Accuracy	96.5%	80.6%	78.6%	76.9%
Precision	96.6%	81.2%	79.4%	77.5%
Recall	96.5%	80.6%	78.6%	76.9%
F1-score	96.4%	80.6%	78.5%	76.8%

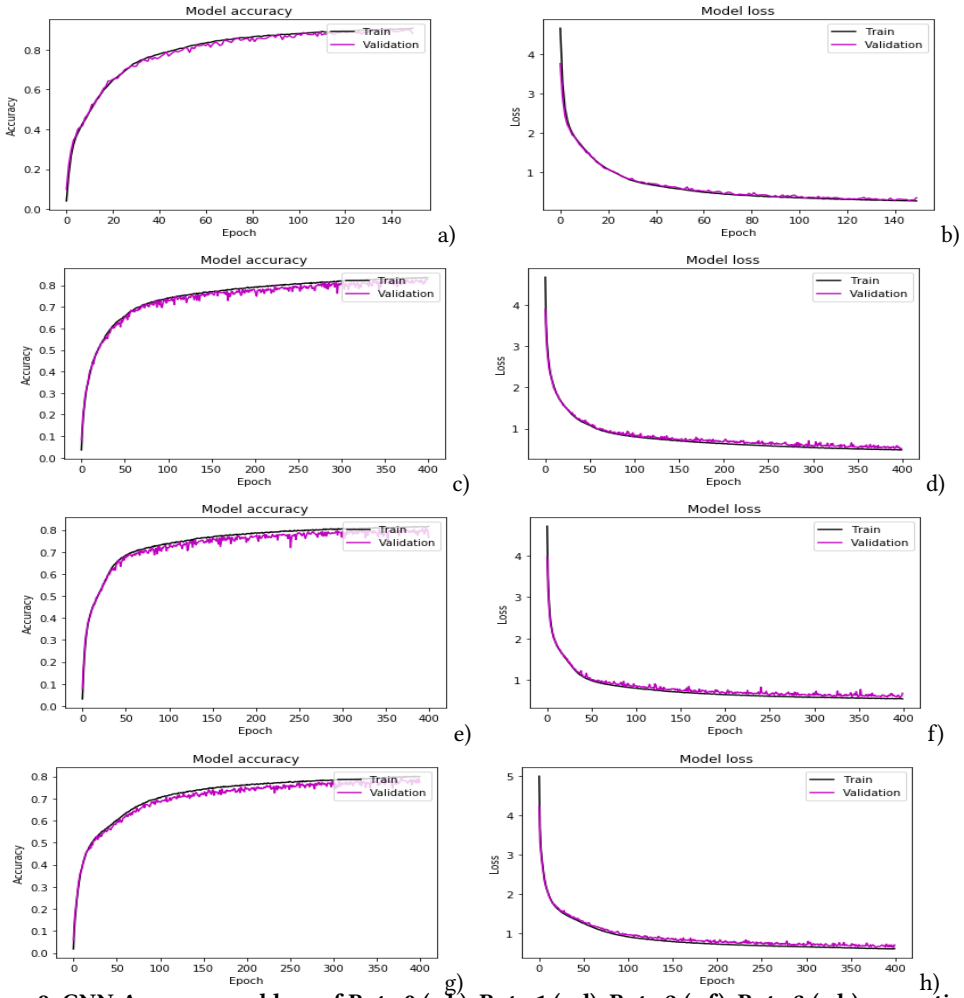

Figure 9: CNN:Accuracy and loss of Byte 0 (a,b), Byte 1 (c,d), Byte 2 (e,f), Byte 3 (g,h) respectively

with lower power traces and without time-synchronization. On the other hand, CNN model demanded more epochs and time to build models that weren't as robust as that of MLP. The CNN model might not be the optimal solution for retrieving key bytes from the power leakage since we need to see the whole data to predict. When using CNN, the model tends to overfit the data.

6 FUTURE WORK

For the future scope of this work, the leakage model can be changed to get a better accuracy for the key bytes with even lower correlation. Also We can train a general purpose model to crack all the key bytes at the same time; however, we need more traces for training. Moreover, we can shift the traces in time dimension and then try to solve the problem. This is done by inserting delay in the AES code. By doing this, we can further expand our dataset.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Hunter Stuckey Amin Sarihi. 2020. Attacking AES using Deep Learning. https://github.com/hss127/machinelearningcs519

[3] Eric Brier, Christophe Clavier, and Francis Olivier. 2004. Correlation power analysis with a leakage model. In *International workshop on cryptographic hardware and embedded systems*. Springer, 16–29.

[4] Joan Daemen and Vincent Rijmen. 1999. AES proposal: Rijndael. (1999).

[5] Debayan Das, Anupam Golder, Josef Danial, Santosh Ghosh, Arijit Raychowdhury, and Shreyas Sen. 2019. X-DeepSCA: Cross-device deep learning side channel attack. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6.

[6] Jasper van Woudenberg Guilherme Perin, Baris Ege. 2018. Lowering the bar:deep learning forside-channel analysis. (2018).

[7] Sunghyun Jin, Suhri Kim, HeeSeok Kim, and Seokhie Hong. 2020. Recent advances in deep learning-based side-channel analysis. *ETRI Journal* (2020).

[8] Paul Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential power analysis. In *Annual International Cryptology Conference*. Springer, 388–397.

[9] Takaya Kubota, Kota Yoshida, Mitsuru Shiozaki, and Takeshi Fujino. 2019. Deep Learning Side-Channel Attack Against Hardware Implementations of AES. In *2019 22nd Euromicro Conference on Digital System Design (DSD)*. IEEE, 261–268.

[10] Colin O'Flynn and Zhizhang David Chen. 2014. Chipwhisperer: An open-source platform for hardware embedded security research. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 243–260.

[11] Yuval Yarom and Katrina Falkner. 2014. FLUSH+ RELOAD: a high resolution, low noise, L3 cache side-channel attack. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*. 719–732.

[12] Enrique Zabala. 2008. The Rijndael Animation. http://www.formaestudio.com/rijndaelinspector/.