CHES 2018 side channel contest CTF – Solution of the AES Challenges

January 30, 2019

Aron Gohr, Sven Jacob, Werner Schindler

Bundesamt für Sicherheit in der Informationstechnik (BSI)
Godesberger Allee 185-189, 53175 Bonn, Germany
{Aron.Gohr,Sven.Jacob,Werner.Schindler}@bsi.bund.de

Abstract. Alongside CHES 2018 the side channel contest 'Deep learning vs. classic profiling' was held. Our team won both AES challenges (masked AES implementation), working under the handle AGSJWS. Here we describe and analyse our attack. We can solve the more difficult of the two challenges with 2 to 5 power traces, which is much less than was available in the contest. Our attack combines techniques from machine learning with classical techniques. The attack was superior to all classical and deep learning based attacks which we have tried. Moreover, it provides some insights on the implementation.

Key Words. Power analysis, machine learning, SAT solver.

1 Introduction

Around the CHES 2018 conference Riscure organised the side channel contest 'Deep learning vs. classic profiling'. The task given was to break by any means possible implementations of the DES, AES and RSA cryptographic algorithms. Implementation details were largely kept secret. It was announced, for example, that the AES implementation was masked but the concrete method was not made public. On July 3, 2018 Riscure released altogether $40,000$ power traces each for the DES and the AES implementation and 40 for the RSA implementation. With the power traces, also input, output and key data was published on the web. This was meant to allow the participants to develop and test their attacks. For each algorithm, two challenges (DES, AES) or three challenges (RSA) were released at the beginning of September. The challenges consisted of $1,000$ attack traces for each of the AES and the DES implementations and of 1 trace each for the RSA challenges. The goal was to submit the correct key as early as possible. The first correct submission 'captured the flag'.

We (team AGSJWS) focused on the AES challenges and won both. In Section 2 we describe our attack, which combines techniques from machine learning with using a SAT solver to derive the desired AES key from some partial key information extracted from the trace set. Furthermore, in Section 3 we evaluate our attack and show that even for the more difficult 'portability' challenge 2 to 5 power traces suffice to recover the key with a large probability of success

and at moderate computational cost. Our attack is more efficient than other attacks we considered, which applied deep learning techniques or classical techniques. Moreover, in Section 4 we show that our attack provides some insight into the implementation and sources of leakage. We also highlight some unexpected properties of the machine learned leakage extraction function.

2 Our attack on the AES implementation

2.1 Setup

The organizers provided four sets of power traces of a masked AES implementation (to be precise: AES-128), in the following denoted by Set 1 to Set 4, which allowed the participants to develop and test their attack. Each of these sets contained $10,000$ power traces. The power traces in Set 1 to Set 3 came from three different devices (denoted by A, B, C), and each power trace corresponded to encryption with a freshly chosen key. Set 4 contained power traces from Device C, which in contrast were generated with a single key shared by all traces. Plaintexts, ciphertexts and keys were known. No information was given on the countermeasures against power analysis deployed in the implementation beyond the fact that masking was used.

Later, two sets with attack traces were published. Set 5 (no-portability challenge) contained $1,000$ power traces from device C, Set 6 (portability challenge) contained $1,000$ power traces from a new device D. The task was to recover the particular keys.

2.2 Preliminary tests

In a first step we verified that the key expansion process was not masked. For this, we used a decision tree classifier trained using scikit-learn [1] to distinguish Set 3 and Set 4. The rationale here was that a non-randomized key schedule used with a fixed key should produce a consistent signal in the traces that can be used to recognise traces from Set 4, and that the main cipher operation was very likely to be randomized. Training these distinguishers was immediately successful. The signal was for this purpose partitioned into segments 1000 data points long, where only every tenth point of each segment was used for training and evaluation. We trained decision tree classifiers with maximal depth 5, using otherwise the default parameters of the DecisionTreeClassifier class in scikit-learn. Half of the samples in Set 3 and Set 4 were used for training, and half were withheld for validation.

Evaluating our decision tree classifiers, we found that classification accuracy varied strongly with the data segment considered but that distinguishing Set 3 and Set 4 was very easy for some data segments, reaching classification accuracies up to 99.7 percent. Looking at validation accuracy as a function of data segment number, a pattern emerged that suggested that a strong key-dependent signal could be found at the very beginning of the trace, maybe related to the device reading in key data, and during an operation, likely a key expansion step, that

is executed before each round transformation. The validation accuracy of these distinguishers by data segment number can be found in Figure 1.

The same conclusion can be reached by comparing point-wise variances between Set 3 and Set 4. In the parts of the traces that we suspect correspond to the key schedule, measured current varies much more in Set 3 than in Set 4. This is naturally expected if we assume that these parts of the traces perform unmasked key schedule operations. With a fixed key (Set 4), oberserved variance in power usage will mostly be noise in this case (although there might also be some non-random signals from the measuring equipment or environmental triggers), whereas for random keys (Set 3) a key-dependent varying leakage signal is added on top of the noise.

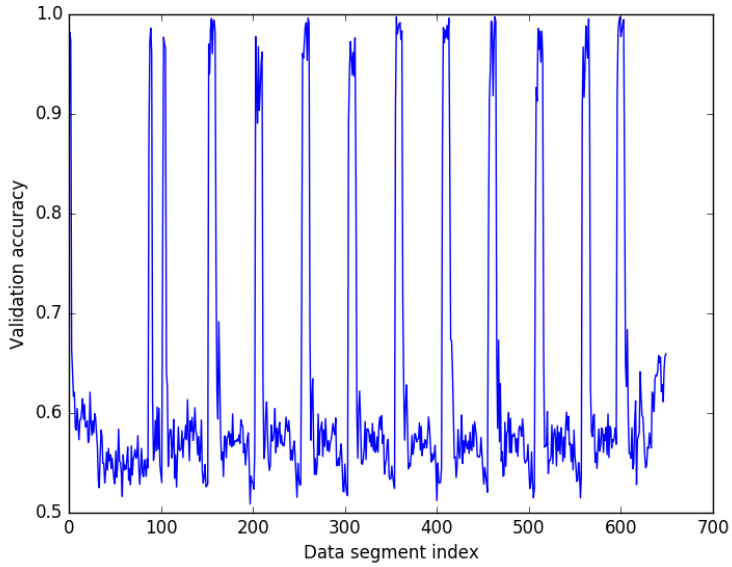

Fig. 1. Distinguishing Set 3 and Set 4 by decision tree classifiers. The dataset is partitioned into windows of size 1000 and the validation accuracy of a decision tree classifier trying to distinguish both sample sets is shown. Since Set 3 and Set 4 target the same device and differ chiefly in Set 4 having a constant key, success in distinguishing between samples of both sets is heuristically indicative of key leakage. Significant leakage is visible at the beginning of the trace and at 12 subsequent peaks. While distinguishing power is significantly larger than random guessing throughout the trace, it seems plausible that in between the visible peaks of leakage, the remaining success of the decision tree classifier may be explainable by other factors such as changes in operating conditions between both sets of measurements.

We then reduced the size of the traces in Set 3 by discarding all the data points with index $\neq 0 (\mathrm{mod}\ 10)$. For these reduced traces, we calculated for every data point the correlation between the Hamming weight of the key and the power

usage at this data point. The results convinced us that targeting the Hamming weights of the subkey bytes was feasible.

2.3 Attack: Phase I

Goals and setup The aim of Phase I was to guess the Hamming weights of the 176 ($= 11 * 16$) bytes of the 11 round keys. We assumed a weight model, i.e. we guessed that the Hamming weight of extended key bytes would leak from some parts of the trace, preferably with an affine leakage function and a manageable noise contribution to the signal.

Approaches studied We tried various machine learning methods based on neural networks and decision trees. We found that owing to the relatively small number of samples provided, overfitting was a significant problem for many network architectures. Two approaches in particular fared well with regards to naturally controlling overfitting. One was a deep convolutional network designed to process the entire reduced trace while having a relatively small number of weights. The second was a very simple design with just one input and one output layer and only linear activations. These designs worked reasonably well without manual elimination of uninteresting parts of the sample. Decision tree classifiers also showed some promise. The linear design showed the best learning, so we selected it for further development.

Model structure Our final model θ can be described as a single-layer perceptron with a particular step function as activation. Concretely, θ is the concatenation of an affine function $f : \mathbb{R}^{65000} \to \mathbb{R}^{176}$ with componentwise application of the step function

$$s : \mathbb{R} \to \mathbb{N}^2, s(x) := (\lfloor x \rfloor, \lceil x \rceil).$$

Hence, we have $\theta := \overrightarrow{s} \circ f$, where $\overrightarrow{s}$ is componentwise application of s to a vector in $\mathbb{R}^n$ (with $n = 176$). Given a trace v with extended AES key k, the output tuple $\theta(v)_i$ is interpreted as a *top2-guess* for the Hamming weight of the i-th extended key byte k_i. We say that the guess for the extended key byte k_i is *true* if $k_i \in \theta(v)_i$ and otherwise say that it is *false*.

Training The step function was chosen manually. Connection weights were learned by Ridge regression, i.e. by minimizing the expected value of the error term

$$\epsilon(f, x, y) := \|f(x) - y\|^2 + \alpha \|f\|^2.$$

Here, $f(x) = Ax + b$ and $\|f\|^2 := \|A\|^2$, where $\| \cdot \|$ denotes the L2 norm ($A \in \mathbb{R}^{176 \times 65000}$, $b, y \in \mathbb{R}^{176}$, $x \in \mathbb{R}^{65000}$, $\alpha \in \mathbb{R}$). The average is taken over a training set X of N input values and the corresponding set of output vectors Y.This is a special case of Tikhonov regularization. The regularization term α was chosen by grid search with generalized cross validation using the GridCV class in scikit-learn [1]. In this step X and Y were set to the union of Set 1 and Set 2. The classifier so trained was tested using Set 3 and Set 4. We found that classifier performance was easily sufficient to solve the portability challenge, and

with the approach thereby validated restarted training using the same methods but using all three training sets Set 1, Set 2 and Set 3 to obtain a new set of weights that was expected to be even more robust to change of device or operating conditions. The resulting classifier was validated against Set 4. While validation against Set 4 had the disadvantage of testing against a set with just one key and against a device already used in training, given the viability of the general approach used we were confident that the resulting classifier would be better than the one trained only on two training datasets. Grid search for the regularization parameter α chose $\alpha = 2^{14}$. Reasonable regularisation parameters would be a lot smaller if traces were normalised before processing, but in our tests normalisation did not seem to help prediction accuracy.

2.4 Combining traces

Our classifier θ takes as input a single trace and outputs a vector of 176 top-2 guesses for the Hamming weights of the extended key bytes. It is intuitive that prediction accuracy can be improved by averaging the outputs of f over many traces, as this is expected to reduce the noise component of the signal. Since f is a linear function, this averaging can equivalently be performed at the level of input data. Hence, when given a set S of n traces to predict, our approach is to calculate $t_{\text{av}} := \frac{\sum_{v \in S} v}{n}$ and produce the top-2 prediction $\theta(t_{\text{av}})$.

2.5 Attack: Phase II

Phase I provided 176 top-2 guesses for the round key bytes. In Phase II we applied a SAT solver to solve a system of non-linear equations which is given by the AES key expansion algorithm and restrictions on the values of the expanded key bytes given by our top-2 guesses. We then removed a randomly selected subset of 20 key bytes ('drop out'), and we only gave the top-2 guesses of the remaining 156 key bytes to the SAT solver.

This approach tolerates a small number of false top-2 guesses in Phase I as explained below.

The probability that a uniformly distributed byte has Hamming weight m or $m+1$ is $\leq 126/256 < 0.5$ (with the probability of 126/256 being reached for $m \in \{3, 4\}$). Hence each top-2 guess provides more than 1 bit information. A straight-forward (admittedly heuristic) argumentation suggests that 156 top-2 guesses from attack Phase I should determine the AES key uniquely. This conclusion matches with our experiments.

The Pseudoalgorithm below sketches our attack.

Pseudoalgorithm

```
Determine 176 top-2 guesses (Attack Phase I)
Repeat (Attack Phase II)
  - select randomly 20 top-2 guesses ('drop-out')
  - input the remaining 156 top-2 guesses into the SAT
    solver
  - terminate the SAT solver if it is unlikely that
```

```
        a solution exists
    until the SAT solver finds the key
```

As SAT solver, we used CryptoMinisat 5.0.1 [2] via the python interface given by the pycryptosat package (version 0.1.4) [3]. Search on a particular SAT instance was terminated if the number of conflicts encountered during search exceeded 300000 (thus limiting the size of the search tree explored by the SAT solver). This took on average 20 seconds on our machine. Solutions were usually found within that time frame.

2.6 Results

In the contest the neural network in Phase I delivered 176 correct top-2 key guesses for both Set 5 (no-portability challenge) and Set 6 (portability challenge), in each case using all 1000 challenge traces available. The SAT solver needed some seconds to find the correct keys.

3 Reducing the number of traces

After the contest we had a closer look at our attack. Our results are explained below. The most interesting question, of course, was how many power traces our attack requires at least to recover the key (in a reasonable time, with modest computing resources).

For the Set 5 (no-portability challenge) in most cases even a single trace turned out to be sufficient. In the following we consider Set 6 (portability challenge).

We already know that the attack will succeed if all the false top-2 guesses are contained in the drop-out. Of course, if there are m false guesses this probability equals

$$\text{Prob(all } m \text{ false top-2 guesses in the drop-out)} = \frac{\binom{20}{m}}{\binom{176}{m}}\,, \tag{1}$$

and the expected number of trials in the attack Phase II (= no. of systems of non-linear equations) is given by the reciprocal of (1) divided by a constant c (empirically close to 1) that is given by the likelihood that attack Phase II will abort on a solvable SAT instance.

We divided the $1,000$ attack traces from Set 6 into non-overlapping subsets of $N = 2, 3, 4, 5$ power traces. Table 1 shows the empirical cumulative distribution of false top-2 guesses for different sample sizes. Table 2 shows the expected number of trials in Phase II and the expected execution time under the assumption that each trial needs 20 seconds. We furthermore assume that our abort condition does not lose a significant fraction of solvable instances. Both assumptions are approximately true with single-threaded execution on our machine and our chosen abort condition at 300000 conflicts. For the solution of difficult instances, parallelization is of course desirable and easy to accomplish for our algorithm.

These results show that our attack is expected to be nearly almost successful for $N = 5$ if we are willing to spend less than half an hour of CPU time. If we spend 44 hours then even for sample size $N = 2$ more than half of the attacks will be successful.

number of false top-2 guesses					
	0	≤ 1	≤ 2	≤ 3	≤ 4
$N = 2$	1%	8%	24%	41%	57%
$N = 3$	10%	38%	63%	82%	91%
$N = 4$	31%	60%	84%	96%	98%
$N = 5$	45%	83%	95%	99%	100%

Table 1. Number of false top-2 guesses (cumulative, empirical results, among 176 top-2 guesses)

number of false top-2 guesses					
	0	1	2	3	4
E(# systems of eq.)	1	9	81	784	7973
E(exec. time)	20 sec	3 min	27 min	4.3 h	44 h

Table 2. Expected no. of trials in Attack Phase II and the expected execution time depending on the number of false top-2 guesses (among all 176 top-2 guesses)

4 Anatomy of the classifier

We have examined the weights of our trained model to see whether anything can be learned from them. We find that the sixteen bytes of the original key cause a strong power signal in three active regions of the trace, while the remaining bytes of the expanded key have one active region of the trace each. The three active regions for bytes 0 and 1 are shown in Figure 2.

We see that the active regions for byte 0 and byte 1 have significant overlap. Further, we see that the classifier assumes a negative relationship between the power usage in some parts of the trace and the Hamming weight of both byte 0 and byte 1. For some of the relevant points, single-point regression also shows a negative correlation. In other points of the trace, negative coefficients for the prediction of byte 0 seem to conincide with positive values for byte 1. A possible explanation might be that the negative coefficients are an adaptation minimizing crosstalk between the power signatures of consecutive bytes. In general, the curves for single-point regression and the coefficients of our classifier have a similar shape, but do not follow each other entirely.

It is natural to assume that the coefficients in the inactive regions of the trace are just noise, i.e. that they do not contribute to predictive accuracy. However, this assumption seems to be wrong, as the following experiment demonstrates.

Denote by t_i the i-th trace of Set 4 and let $\pi : \mathbb{Z}_{10000} \to \mathbb{Z}_{10000}$ be a random permutation. Let $t_i[a : b)$ denote the subtrace consisting of the data points with indices $\{a, a + 1, \ldots, b - 1\}$ of trace t_i and let $||$ be concatenation of arrays. We created a set of *hybridized* traces t'_i by setting $t'_i := t_i[0 : 30000)||t_{\pi(i)}[30000 : 65000)$. This set of traces will in the sequel be called *hybridized Set 4*. All traces in Set 4 have the same key and correspond to the same device, so if prediction of Hamming weights is a largely local operation, we would expect that a loss in prediction performance when our classifier is applied to hybridized traces instead

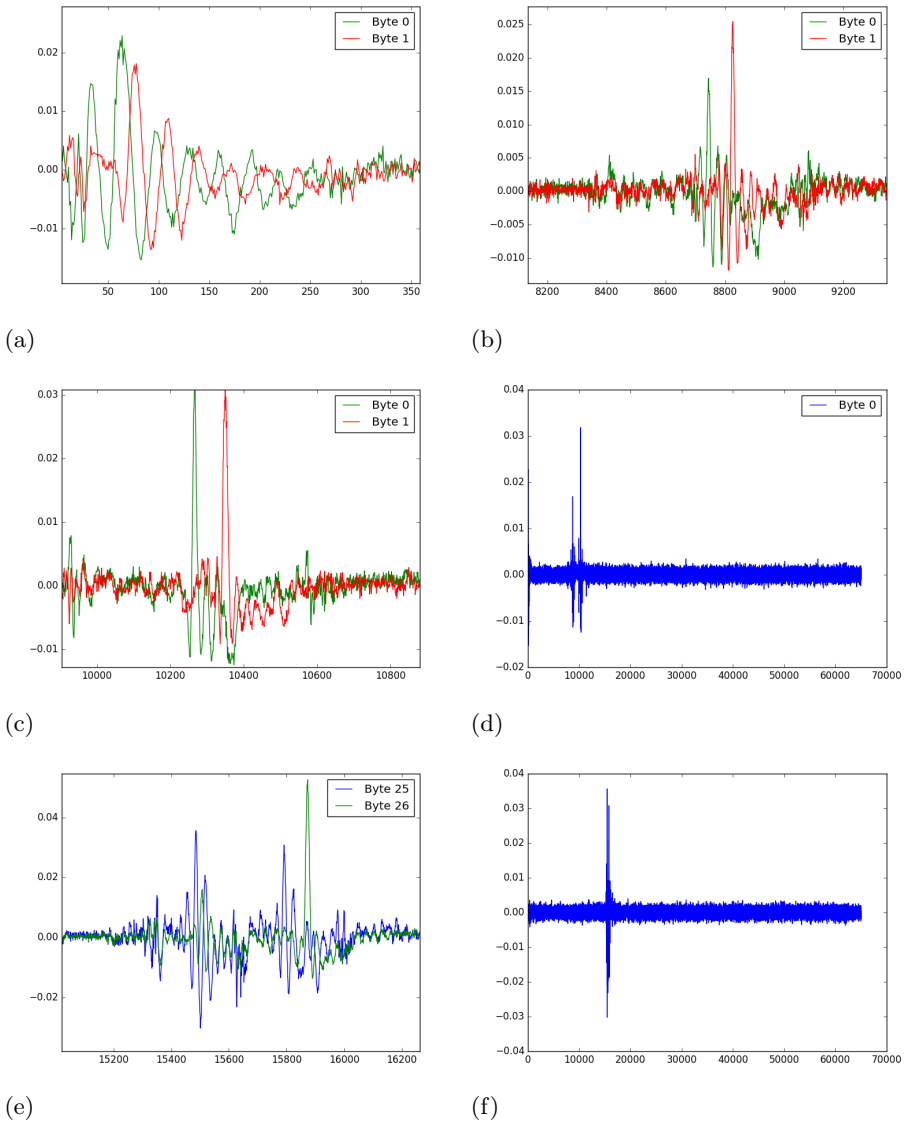

Fig. 2. (All) The horizontal axis denotes the index within a reduced trace. Traces are reduced by using only every tenth data point. The vertical axis gives the numerical weights of our predictor. (a-c) Weights for the Hamming weight prediction of byte 0 (green) and byte 1 (red) of the AES key in our best classifier. Only the active areas of the trace are shown. (d) All weights for byte 0. (e) Weights for the prediction of bytes 25 (blue) and 26 (green). Only the active part of the trace is shown. (f) All weights for byte 25.

of regular traces will mostly happen around the splicing point, i.e. affect maybe a few bytes of the key but leave most untouched.

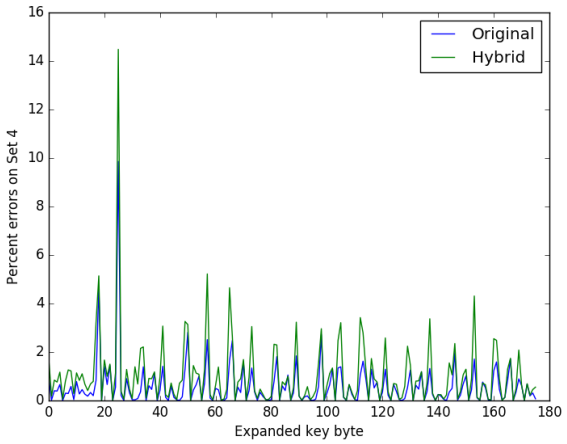

Fig. 3. Error rates of our main model by byte for Set 4 and hybridized Set 4. Hybridization is seen to affect predictive performance across the whole trace.

However, it turns out that top-2 prediction error on hybridized Set 4 is strictly higher than on Set 4 for 156 of the 176 bytes predicted. Figure 3 shows top-2 prediction error rates on both sets as a function of the byte predicted. We hypothesize that the coefficients in inactive regions of the trace serve to adapt the model against change of device or environmental conditions or possibly other global influences on prediction accuracy such as small alignment deviations.

We tested this hypothesis by looking at the combined output of the inactive weights for byte 25 of the trace. Byte 25 of the extended key is relatively difficult to predict using our approach, as is shown also in Figure 3. Visual examination of the weight vector suggested identifying indices 15000 to 17000 of the trace as the active part of the trace for the purposes of byte 25 prediction. We then zeroized the corresponding weights (i.e. weights 15000 to 17000) in our predictor and computed the combined outputs of the remaining weights for all 30000 training traces (i.e. the output of the hamming weight predictor with the internal weights changed as described). Denote by $o(t) \in \mathbb{R}$ the output value so computed for input the trace t.

Recall that for a vector $v \in \mathbb{R}^n$ the mean of v is defined by setting

$$m(v) := \frac{1}{n} \sum_{i=1}^{n} v_i$$

and for $n > 1$ the empirical standard deviation[1] is given by

$$s(v) := \sqrt{\frac{\sum_{i=1}^{n}(v_i - m(v))^2}{n-1}}.$$

Further, for a vector $v \in R^n$ with $n > 2$ and $s(v) \neq 0$ we define its *normalisation* to be $\nu(v) := (v - m(v))/s(v)$, where addition between vectors and scalars is defined in the natural way, i.e. component-wise.

We then find that $\text{Y25} := \nu((o(t_i))_{0 \leq i \leq 30000})$ and $\sigma := \nu((s(t_i))_{0 \leq i \leq 30000})$ are in close correspondence to each other for $t_i \in \text{Set 1} \cup \text{Set 2} \cup \text{Set 3}$ (see Figure 4). This is slightly surprising, as the output of the inactive part of a trace is a linear function of the trace, whereas the empirical standard deviation is a nonlinear function. However, it is a differentiable function in the area of interest, so the traces may simply be close enough to each other in $\mathbb{R}^{65000}$ for this function to have a good linear approximation locally.

As can be seen from Figure 4, the standard deviation of a trace is an excellent tool for device discrimination, so the results of this test confirm our hypothesis that the predictor uses parts of the trace that do not directly carry information about the extended key bytes to take into account device properties or environmental factors.

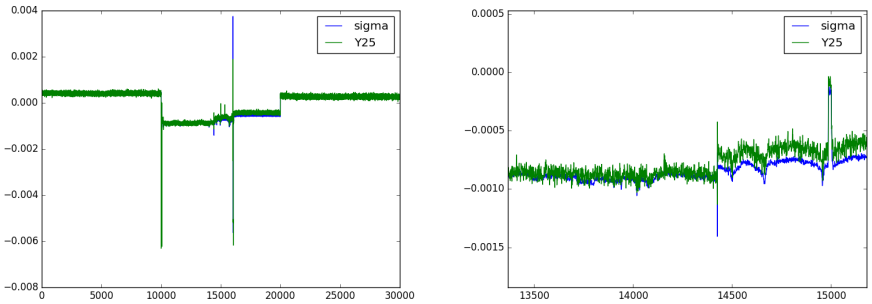

Fig. 4. Normalised empirical standard deviations (sigma) for the traces of Set 1 to Set 3 compared to normalised output of our predictor restricted to inactive areas of the trace for extended key byte 25 (Y25). Normalised output closely follows the standard deviation for the trace. Traces 0-9999 correspond to Set 1, traces 10000-19999 correspond to Set 2, and the remaining traces 20000-29999 correspond to set 3. *(Left)* Data for all 30000 training traces. *(Right)* Same data for a few traces within Set 2.

5 Conclusions

We have shown how a combination of linear classifiers and a SAT solver can break a protected AES implementation by power analysis with a very small number of

[1] Note that this treats each trace as though it were a vector of realisations of a random variable. Of course it isn't.

traces. To develop our attack, the adversary does not need to know much about the implementation. We hypothesize that our success is due to lack of masking of the key schedule. An additional signal is found right at the beginning of the trace, suggesting that initial key setup might also be unprotected. It would be interesting to know the physics underlying some negative correlations between power usage and Hamming weight of key bytes. A hypothesis related to cross-talk between signals of consecutive bytes may offer a partial explanation in some cases.

In terms of a comparison to classical profiling, machine learning is quite useful in this case study in terms of automatically taking care of relations between a wide range of observations and the variables of interest. Our classifier naturally combines the results of thousands of measurements into one Hamming weight estimate and we did in fact get significantly worse results when restricting our approach to small sets of time slices judged interesting in terms of key prediction (note that such restriction would have to be fairly drastic in order to be useful since we seek to predict all 176 Hamming weights of the extended key). On the other hand, it is generally easier in classical profiling to deal with overfitting, and models are more closely tied to pre-existing knowledge about the physics of the device under attack. There is, however, no hard boundary between both approaches and combining the advantages of both worlds may well be feasible in future attacks.

References

1. F. Pedregosa et al., *Scikit-learn: Machine Learning in Python*, Journal of Machine Learning Research, vol. 12, p. 2825-2830, 2011
2. M. Soos, K. Nohl, C. Castelluccia, *Extending SAT solvers to Cryptographic Problems*, Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009
3. Pycryptosat homepage, https://pypi.org/project/pycryptosat/, accessed 2018/10/08