Machine learning and classical side-channel analysis in a CTF competition

Yongbo Hu[1], Yeyang Zheng[1], Pengwei Feng[1], Lirui Liu[1],Chen Zhang[1], Aron Gohr[2], Sven Jacob[2], Werner Schindler[2], Ileana Buhan[3], Karim Tobich[3]

[1] Goodix, China,
email: lastnamefirstname@goodix.com
[2] Bundesamt für Sicherheit in der Informationstechnik (BSI), Germany
email: firstname.lastname@bsi.bund.de
[3] Riscure, Netherlands
email: buhan@riscure.com

Abstract. Machine learning is nowadays supplanting or extending human expertise in many domains ranging from board games to text translation. Correspondingly, the use of such tools is also on the rise in computer security. Alongside CHES 2018, a side channel challenge was organised under the theme of 'Deep Learning vs Classical SCA' to test whether Deep Learning is presently widely used in the SCA community and whether it yields competitive results. The competition had 58 participants, it ran for three months, and a quantity of 35GB of data was used as a test sample. This paper presents the solutions of the teams that captured a flag and then discusses the results. While deep learning was used by neither team, other machine learning methods turned out to be very useful. The first contribution is a snapshot in time of the expertise in the community, and shows a clear bias towards classic SCA. The second contribution is the presentation of novel techniques for key extraction for the challenges proposed and a reference for black-box evaluation of crypto primitives by experts in the field. The third contribution is a baseline which can be used to further improve upon. Based on the results of this competition, we conclude that human expertise remains very important in the design of successful SCA attacks and machine learning can be a useful tool.
Section 2,3 and 4 of this report have been directly contributed by the winning teams; as a consequence, section 3 is essentially identical to the previous eprint `https://eprint.iacr.org/2019/094/20190131:230649` [8] authored by A. Gohr, S. Jacob and W. Schindler.

Keywords: key recovery, deep learning, machine learning, side channel analysis, black-box evaluation

1 Introduction

Evaluating the resilience of cryptographic algorithms to side channel analysis (SCA) is required by several national and private certification schemes. A typical SCA evaluation requires a complex end-to-end procedure: an initial leakage assessment stage, an optional dimensionality reduction phase, a signal processing step (e.g., static alignment, filtering, re-sampling), the construction of a stochastic model of the leakage that is specific to the situation at hand, and finally the application of one or several attack methods.

Machine learning (ML) might increase the effectiveness of such a work-flow in various ways, on the one hand by making some steps easier to execute and on the other hand by allowing for the exploitation of leakage signals that a human SCA expert might not consider. In terms of comparison to classical SCA attacks, the features that distinguish ML based attacks are the use of general-purpose tools for the construction of the leakage model and the limited assumptions made about the nature of the leakage. SCA-specific expertise, however, may or may not still be leveraged.

Since 2017, several national and private schemes mandate the application of deep learning in side channel analysis (deep SCA) in the certification of secure products, which places deep SCA and ML-supported SCA more generally into the category of fundamental, required analysis techniques. However, to this date there is no objective comparison between the strength of deep SCA and classic SCA. For a meaningful benchmark, one needs a specifically designed trace set with traces collected on multiple devices which allow training a network, allow key extraction with classic SCA and cover a variety of possible SCA targets. Furthermore, the work factor of performing both classical SCA and machine-learning based SCA on the same targets is a significant obstacle for such a comparison for any single team. The results of the comparison will depend on the tools used and on the expertise of the person doing the analysis with either classic or deep SCA. To get closer to an answer to the question which of the two techniques is more widely used by the community, we set up an experiment. This paper presents an overview of the solutions developed by the teams to win at least one flag in the competition and discusses the results and their significance for the community.

Description of the experiment. In the summer of 2018, a new traceset was released that allows key recovery with classic and deep SCA covering three cryptographic primitives. For three months researchers in the community used their skills and knowledge to extract the keys. The experiment was set-up as a competition open for individuals and teams.

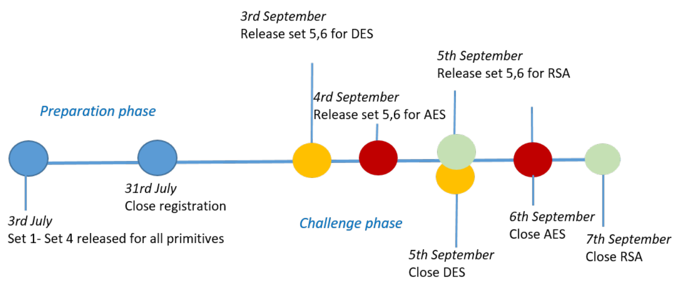

Figure 1: Timeline for the experiment. The preparation phase begins on the 3rd of July and ends on the 3rd of September when the challenge phase begins.

The challenges consist of three primitives: two symmetric key algorithms (hardware DES, software AES-128) and a public key algorithm (RSA-512). For each primitive, traces are collected from four physical devices (denoted by A, B, C, D), and two different keys needed to be extracted in the challenge phase. The first key tests the recovery of traces from a device known to the attacker, which we refer to as *non-portability*. This is the classic case, where the adversary has physical access to the target device, of a key he does not know. Recovery of the second key tests the case of an adversary who has no access to the target devices and uses other devices from the same class to profile. We refer to this scenario as *portability*.

Any technique is in scope, in the minimum amount of time possible. The model for scoring is "winner takes it all" with all the points going to the participant who wins the challenge. When multiple participants find the full key the winner is chosen based on the submission time. Based on the timeline information in Figure 1, we note that the experiment is biased to favor participants who chose deep learning, with two months of preparation to build and test their network, vs a few days allowed for the participants who chose classic SCA. The traces are in HDF5 format and are available online [6]. While it would be feasible in principle to solve the DES and RSA challenges by cryptanalysis instead of side-channel analysis, such an approach would not have been competitive, since the team to first solve a challenge wins that challenge.

Choice of the primitives. We chose classical primitives, familiar to the community, which cover both hardware and software engines, symmetric and public key algorithms. A limiting factor in the choices made was the size of the traces. The current set has a size of 35GB of data. The first challenge is a hardware DES implementation on a basic card ZC 5.4 with no (noticeable) countermeasures. The second primitive is a software AES implemented on an ARM-CORTEX M0 with random, boolean masking added to the round key. The third primitive is an RSA-512 algorithm with random exponent blinding implemented on an ARM-CORTEX M0 platform. As the tracesets remain public, we limit the description of the countermeasures.

Description of the tracesets. For each primitive four sets of traces are provided, in the following denoted by Set 1 to Set 4, which allowed the participants to develop and test their attack. The power traces in Set 1 to Set 3 came from three different devices 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. In the challenge phase, two sets with attack traces were published, Set 5 (no-portability challenge) with traces collected from device C, Set 6 (portability challenge) with traces collected from device D. The task was to recover the particular keys.

2 The DES Challenge

Standard CPA[1][2], a specific form of Classical Profiling, is used to solve DES challenge. The round structure for DES algorithm is illustrated in Figure 2.

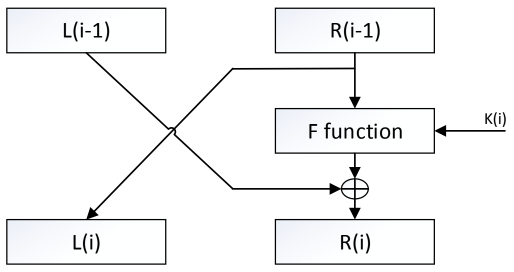

Figure 2: Round structure of DES

2.1 Preliminary tests

The power traces for the DES Challenge are not aligned, which is shown in Figure 3, so it is necessary to properly align the traces with standard static alignment. Since the acquisition frequency of the sample is not provided, the x-axes of the relevant Figures are labeled as "Samples".

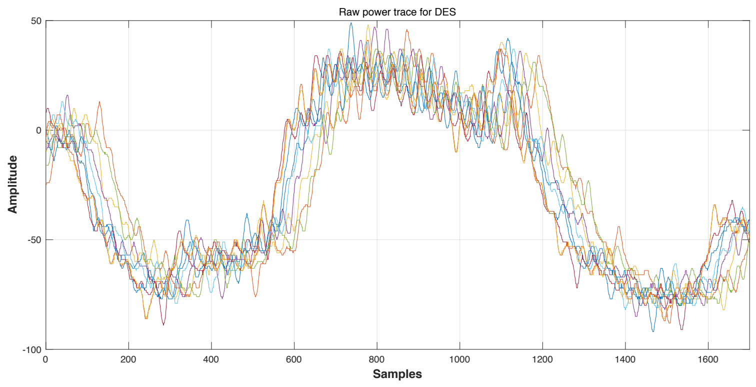

Figure 3: The first 10 raw traces, real time versus the power amplitude,from DES learning traces.

After the alignment, we perform leakage assessment and as a result, we find that in the switching of the DES state register from $R0$ to $R1$ the Hamming distance $HD(R0, R1)$ leaks. We assume a linear model of the power consumption T defined as $T(t) = \xi HD(R0, R1) + l$, where l indicates the noise present in the traces. Let ξ denote the linear parameter. The correlation between the model and the power traces is defined as $corr(t) = Pearson\_Correlation(HD(R0, R1), T(t))$. With the formula of correlation, the leakage assessment can be obtained during profiling with all the traces for learning from set 1 to set 3, as is shown in Figure 4. Because of the leakage shown in Figure 4, both template attack [3] and standard CPA are applicable, but the latter one is less complicated and this is why it is preferred. However, considering only 1K traces are available in the attack phase, the simplified profiling,which indicates the selection of high leakage points, still plays an essential part in increasing the attacking confidence. And in this way, the POI(Points Of Interests), defined as $Tsets : \{t|corr(t) > threshold\}$, can be filtered out. Let $OFFSET$ be the offset of the learning traces and attack traces, the POI selection for attack traces are defined as:$OFFSET + Tsets$ as shown in Figure 5.

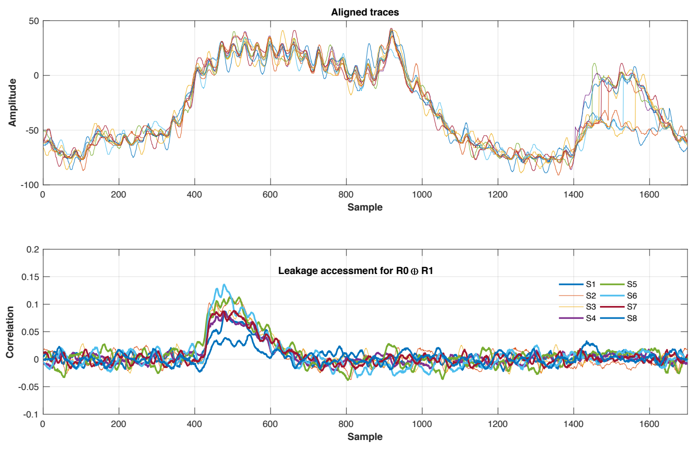

Figure 4: Leakage assessment on HD(R0,R1) for learning traces

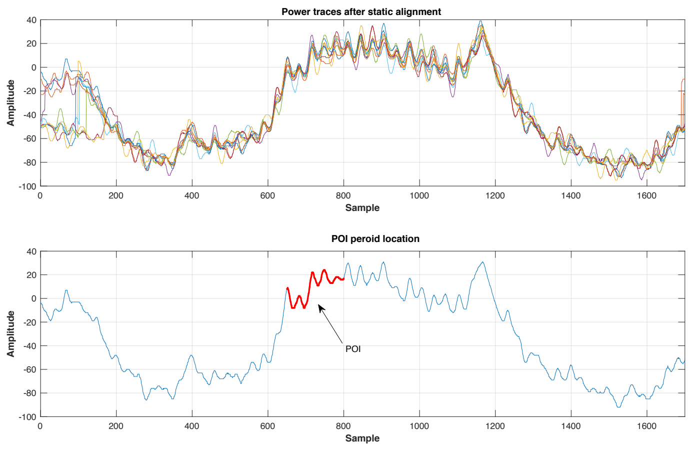

Figure 5: Aligned traces and the POI selection for attack-phase traces

2.2 Attack phase

At attack phase, the Pearson Correlation calculation between the 1K power traces and the intermediate data for different key hypothesis per Sbox can be performed for both portability traces and the non-portability traces. Figure 6 illustrates the attack results of 8 Sboxes with non-portability traces.

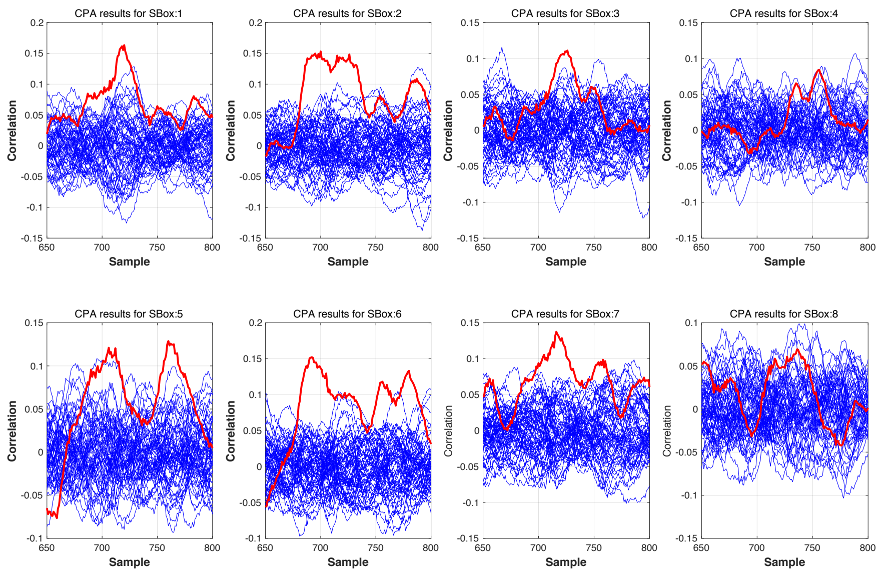

Figure 6: CPA results for 8 SBOXes

The correct key for each Sbox is defined as below

$$k = argmax_k\{corr(k,t), k = 0, 1...63\} \tag{1}$$

2.3 Results

In Figure 6, the correlation curves the of correct key hypothesis are easy to distinguish except for S4 and S8. The side channel part of the attack only tries to recover the first subkey. Here we do not have good results for two Sboxes(S4 and S8), meaning these have to be brute-forced. Also, the first subkey only yields 48 bits of information on the full 56 bits root key, so another 8 bits remaining key bits have to be brute-forced. This gives the desired $2^6 * 2^6 * 2^8 = 2^{20}$ remaining complexity.

All in all, the attack provides a fast and comprehensive solution for DES challenge. We summarize the work-flow in Figure 7. Furthermore, this method only learns the POI off-line using learning traces and attacks the on-line 1K traces without any other information and knowledge, so it is very robust to device change. This helps to solve both portability and non-portability challenge with one unified approach.

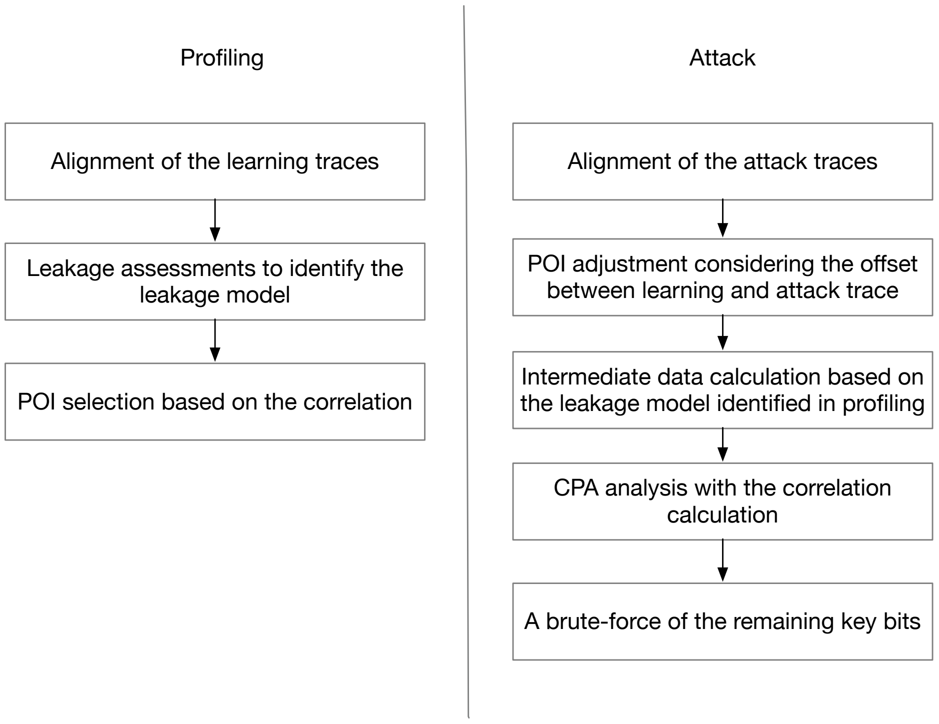
Profiling
Alignment of the learning traces
Leakage assessments to identify the leakage model
POI selection based on the correlation
Attack
Alignment of the attack traces
POI adjustment considering the offset between learning and attack trace
Intermediate data calculation based on the leakage model identified in profiling
CPA analysis with the correlation calculation
A brute-force of the remaining key bits

Figure 7: Work-flow for the attack of DES challenge

What can be done to potentially optimize the attack further about the DES challenges is listed as below

1. To improve the CPA efficiency with some pre-process since the leakage is not a single point one;
2. To perform template attack[3] or deep learning with the leakage;
3. To exploit the leakage of $R1 \oplus R2$ and partial key bit from Figure 6 with CPA to reduce the remaining key entropy.

3 The AES Challenge

In this section, we describe our attack on the AES challenge. The exposition here given is up to some minor editing identical to the previous eprint [8] contributed by team AGSJWS describing the same solution.

3.1 The attack

3.1.1 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 [4] 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 8.

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), observed variance in power usage will mostly be noise in this case, which may include non-random signals from the measuring equipment or environmental triggers. For random keys (Set 3), on the other hand, a key-dependent varying leakage signal is added on top of the noise.

We then reduced the size of the traces in Set 3 by discarding all the data points with index $\neq 0 \pmod{10}$. 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.

3.1.2 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

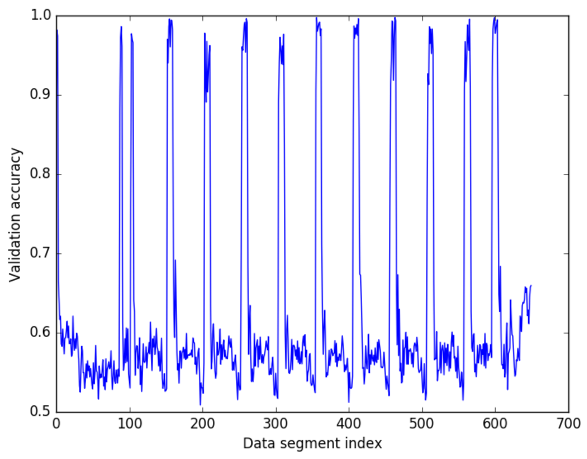

Figure 8: Distinguishing Set 3 and Set 4 by decision tree classifiers. The dataset is partitioned into windows of size 1000 and the validation accuracy of a decision tree classifier trying to distinguish both sample sets is shown. Since Set 3 and Set 4 target the same device and differ chiefly in Set 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.

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. In other words, 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 the *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 error term $\|AX + b - Y\|_2^2 + \alpha\|A\|_2^2$, where $f(x) = Ax+b$ and where X is a set of training traces, while Y is the set of the corresponding Hamming weight vectors ($A \in \mathbb{R}^{176\times65000}$, $b, Y \in \mathbb{R}^{176}$, $x \in \mathbb{R}^{65000}$, $\alpha \in \mathbb{R}$). 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 [4]. 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.

3.1.3 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 $t_{\mathrm{av}} := \frac{\sum_{v \in S} v}{n}$ and produce the top-2 prediction $\theta(t_{\mathrm{av}})$.

3.1.4 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 a 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 $\leq 126/256 < 0.5$ ('=' holds for $m = 3, 4$). 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 [5] via the python interface given by the pycryptosat package (version 0.1.4) [7]. 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.

3.1.5 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). The SAT solver needed some seconds to find the correct keys.

3.2 Further Investigations

3.2.1 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

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

and the expected number of trials in the attack Phase II (= no. of systems of non-linear equations) is given by the reciprocal of (2) 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.

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
$N = 2$	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%

3.2.2 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 9.

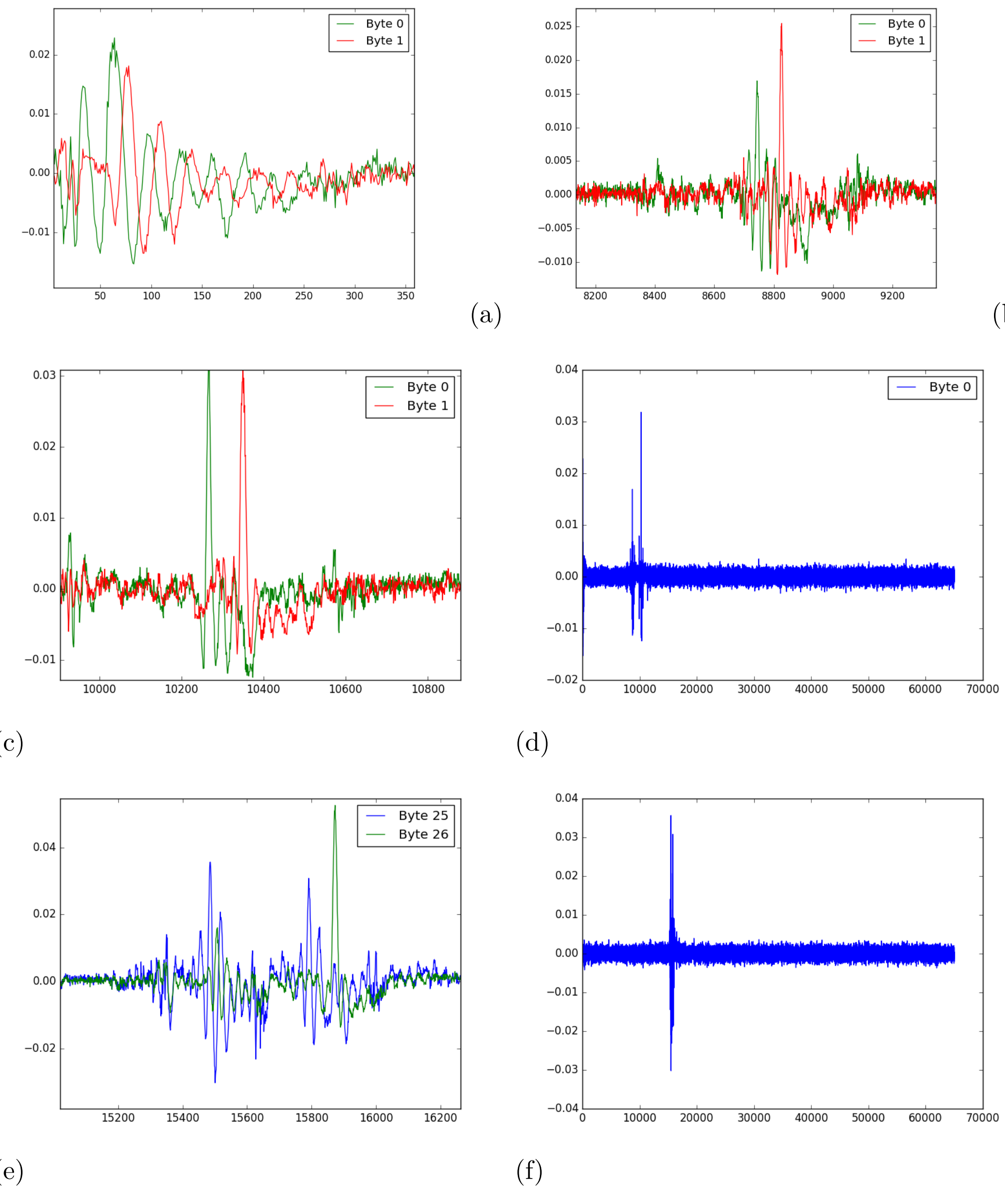

Figure 9: (a-c) Coefficients for the Hamming weight prediction of byte 0 (green) and byte 1 (red) of the AES key in our best classifier. Only the active areas of the trace are shown. The x-axis indicates the index of each coefficient. (d) All coefficients for byte 0. (e) Weights for the prediction of bytes 25 (blue) and 26 (green). Only the active part of the trace is shown. (f) All weights for byte 25.

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)

	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

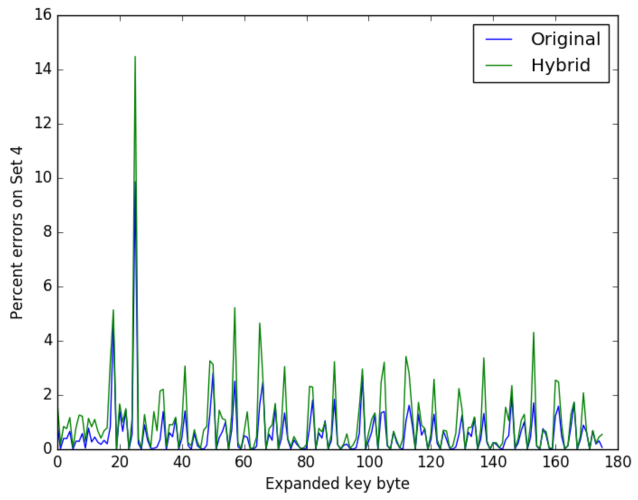

Figure 10: Error rates of our main model by byte for Set 4 and hybridized Set 4. Hybridization is seen to affect predictive performance across the whole trace.

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 $\pi : \mathbb{Z}_{10000} \rightarrow \mathbb{Z}_{10000}$ 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 of regular traces will mostly happen around the splicing point, i.e. affect maybe a few bytes of the key but leave most untouched.

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 10 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.

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 10. 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. Denote by $o(t)$ 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 $Y25 := \nu((o(t_i))_{0 \leq i \leq 30000})$ and $\sigma := \nu((s(t_i))_{0 \leq i \leq 30000})$ are in close correspondence to each other for $t_i \in$ Set 1 $\cup$ Set 2 $\cup$ Set 3 (see Figure 11). 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 11, 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.

3.3 Concluding remarks on the AES challenge

We have shown how a combination of a linear classifier and a SAT solver can break a protected AES implementation by power analysis with a very small number of traces. To develop our attack, the adversary does not need to know much about the implementation. Analyzing the solution, some knowledge about the implementation is, however gained; for instance, we see evidence of an unprotected initial key setup phase and an unprotected key schedule. 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

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

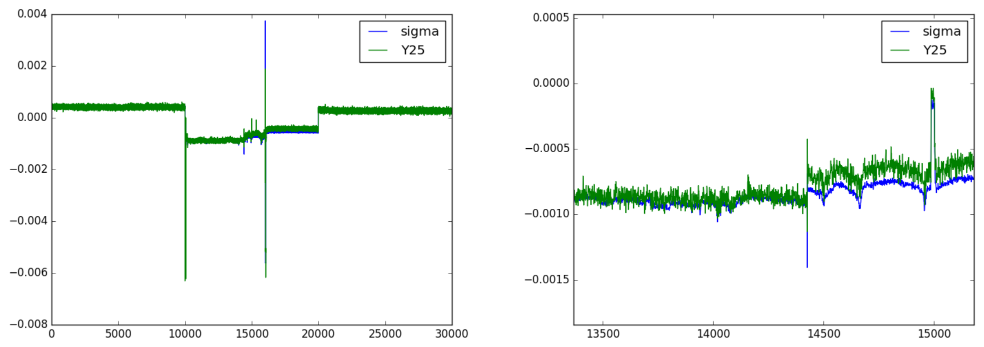

Figure 11: Normalised empirical standard deviations (sigma) for the traces of Set 1 to Set 3 compared to normalised output of our predictor restricted to inactive areas of the trace for extended key byte 25 (Y25). Normalised output closely follows the standard deviation for the trace. Traces 0-9999 correspond 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.

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.

4 The RSA challenge

4.1 Preliminary tests

Template attack[3], a Classical Profiling attack, has been chosen for RSA challenge. Considering the different patterns, a first partitioning of the trace into three parts can be found visually as illustrated in Figure 12. The head part of the trace identifies some pre-processing of RSA, followed by a modular exponentiation part which is the main function of RSA decryption and a tail part with some post-processing.

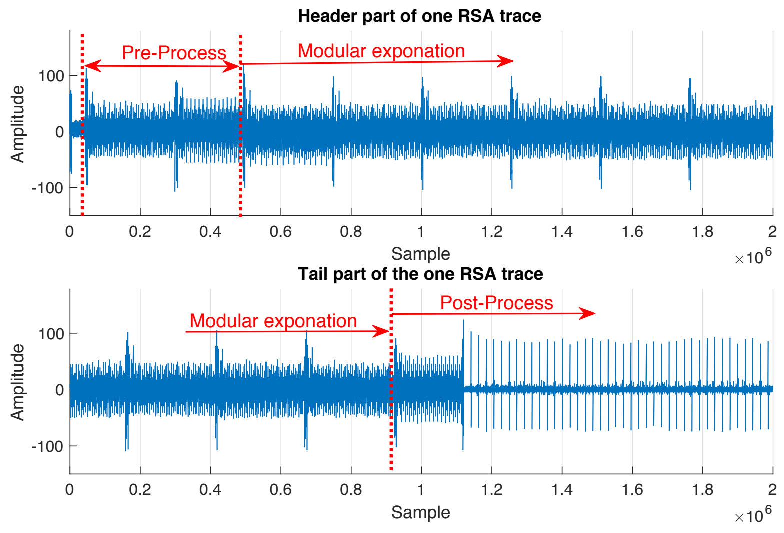

Figure 12: Simple power analysis for RSA

The head part includes two operations, the first operation may be the blinding process of private d as in Eq.(3) and the latter may indicate the conversion to Montgomery-Domain as in Eq.(4).

$$d' = d + r\phi(N) \tag{3}$$

$$T(x) = Mont(x, R^2) \tag{4}$$

The tail of the trace, the post processing, includes a time consuming communication part, a CPU-Call to output the results of decryption and one modular operation, which indicates the transform of data from Montgomery-Domain to standard representation as below

$$T(x) = Mont(x, 1) \tag{5}$$

The rest of the trace, containing the modular exponentiation, is comprised of two kinds of operations, which are modular multiplication and square. When counting total operation number, totally 810 operations are found with the first trace in set 1 as an example. Moreover,$d' = d + r\phi(N)$ is 544 bits since $\phi(N)$ is 512 bits and r is 32 bits for this case, and we have

$$Ratio = \frac{810}{512 + 32} = 1.489 \tag{6}$$

Hence the modular exponentiation is the basic right-to-left or left-to-right algorithm without always modular multiplication square countermeasures, otherwise the *Ratio* above should be nearly 2. Furthermore, when calculating the operation number for a specific RSA calculation in learning traces, we can make the conclusion that the algorithm is as below.

4.2 Profiling phase

To get more information about every operation, either a 544 bits modular multiplication or a 544 bits square, the trace for each one is extracted out from the raw trace and and we will hereafter refer to these sub-traces as segment-traces. The necessary alignment of traces is performed leveraging the pattern marked in red and marked as *Alignment target*. In other words, each segment-traces will start from a specific pattern marked with *Alignment target* and last for 260K sample points. These segment-traces will correspond to an operation, either the modular multiplication or the square. After successful alignment, the leakage

Algorithm 1 The left-to-right modular exponentiation

```
Require: $M, d = \{d_{n-1}, d_{n-1}, ...d_1, d_0\}$
Ensure: $M^d$
1: out=1;
2: for $i = n - 1$ to 0 do
3:     $out = out^2$
4:     if $d_i == 1$ then
5:         $out = out * M$
6:     end if
7: end for
8: return $out$
```

assessment, i.e. the correlation between the tag of the operation($tag = 1$ indicates a 544 bits square, $tag = 0$ indicates a 544 bits modular multiplication) and the segment-traces, can be performed, as is shown in Figure 13.

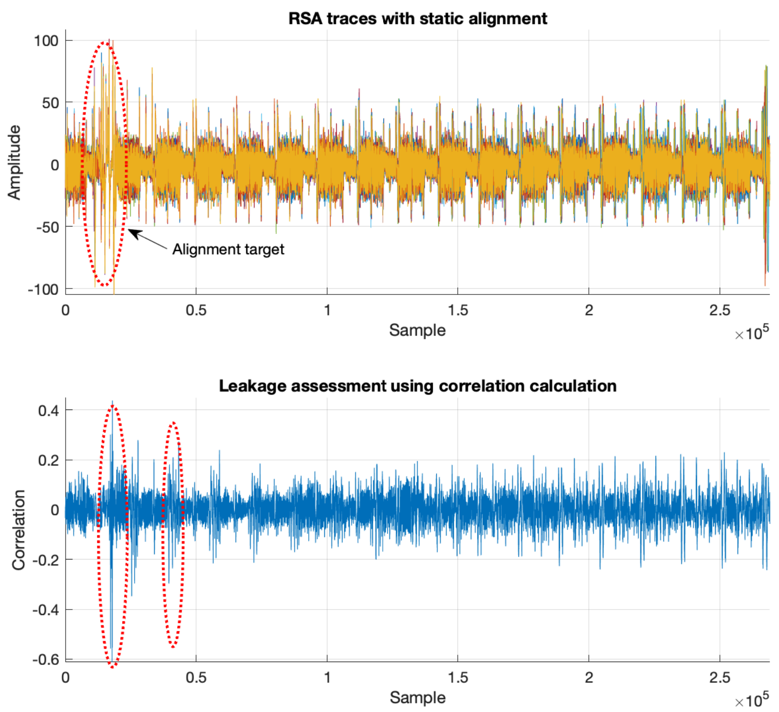

Figure 13: Above figure: The first 10 segment-traces for the first 10 operations with alignment; Below figure: Correlation between the operations and segment-traces for leakage assessment

Significant leakage points, marked with the first red circle in Figure 13, represent different operations for the storage of out and M, with notations as in Algorithm 1. As out and M are stored in different memories, access to out is sufficient for the modular square operation while an extra access to storage is mandatory for the modular multiplication operation.

In the experiment with learning traces, only taking advantage of the leakage points marked in the first red circle to build the template, attacking the same trace sets as used in profiling phase works, however attacking other traces does not present very high accuracy. So in order to effectively retrieve the whole key with higher confidence, more POIs with more leakage are needed. Please notice the leakage points marked in the second red circle in Figure 13, they locate in small operations within a 544 bits operation. A total of 16 small operations exists in each 544 bit modular multiplication or square, which may be caused by the implementation of dividing 544 bits modular multiplication into smaller size of operations. However, the specific implementation does not impact to our attack.

Hence if the alignment of the traces can be based on small operations, more POIs may potentially appear within each small operation. This speculation can be verified after the advanced alignment and leakage assessment afterwards as is shown in Figure 14. The root cause for the leakage of small operations may be the different storage access of the small operations between 544 bits modular multiplication and square. We call the resulting alignment procedure *advanced alignment*.

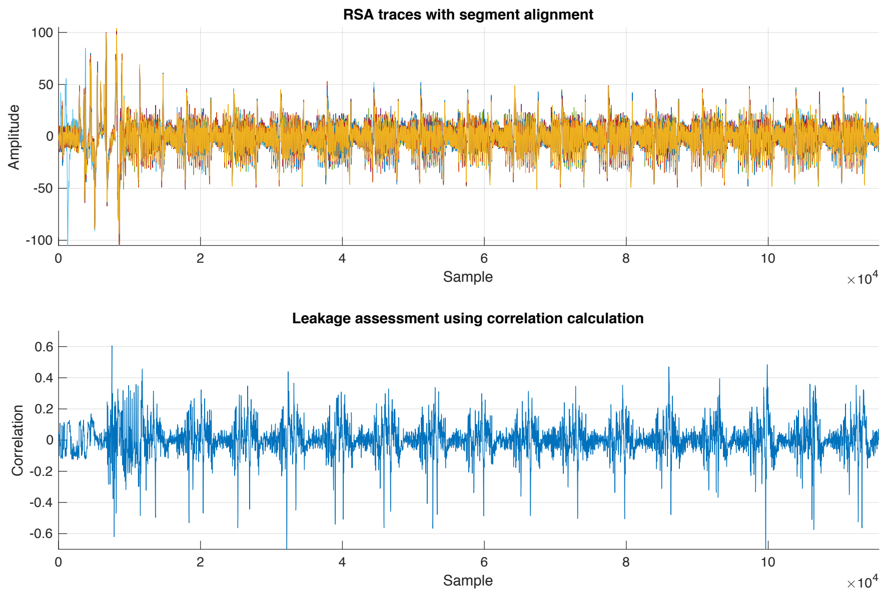

Figure 14: Above figure: The first 10 segment-traces(either a 544 bits modular multiplication or a square) with advanced alignment; Below figure: Correlation between the operations and segment-traces for leakage assessment

After 804 POIs have been selected which satisfy $|correlation| > 0.3$, the profiling for template attack[3], modeling with $\{u(i,t)(i = 0, 1), C^{-1}\}$, is performed for Challenge 5,6 and an extra 7 as below. In the whole attack, $operation = 1$ indicates a 544 bits modular square and $operation = 0$ indicates a 544 bits modular multiplication.

$$\begin{aligned} u(i,t) &= \frac{\sum_{operation=i} T(t)}{\sum_{operation=i} 1} \\ C &= \frac{Cov_{operation=0}T(t) + Cov_{operation=1}T(t)}{2} \end{aligned} \tag{7}$$

4.3 Attack phase

Let m be the operation number for one Challenge trace and let $\{O_0, O_1, \dots O_{m-2}, O_{m-1}\}$ be the operation series. Let $\{T(j), j = 0, 1, 2 \dots m-1\}$ denote the segment-traces after pre-processing, exactly the same as what we did for profiling, the probability for attack phase and the operation series can be obtained as below.

$$\begin{aligned} P_i^j &= (T(j) - u(i)) * C^{-1} * (T(j) - u(i)), i = 0, 1 \\ O_j &= argmax_k(P_k^j), j = 0, 1 \dots m-1, k = 0, 1 \end{aligned} \tag{8}$$

A final step of key brute-force computation, a decryption-encryption algorithm with public key (N, e), can be applied to confirm the private key after retrieving the operation series O_i in attack phase. N is provided and $e = 65537$ is verified after trying both $e = 3$ and $e = 65537$ with the first private key in learning traces.

Algorithm 2 Brute force using decryption-encryption for RSA

```
Require: $O = O_0, O_1, \dots O_{m-2}, O_{m-1}$
Ensure: $privatekey : d'$
    Msg=Random()
 2: for ($O1$ $and$ $O$ $have$ $less$ $than$ $k$ $bits$ $difference$) do
        $Key = ConvertMultiplicationAndSquareToRootKey(O1)$
 4:     if $(Msg^{Key})^e == Msg$ then
            return $Key$
 6:     end if
    end for
 8: return NULL
```

Then the whole RSA key for Challenge Round-5 and Round-6 can be retrieved with a maximum of 3 bits error tolerance which indicates an error probability of $\frac{3}{1.5*(512+32)} = 0.37\%$. The expected computational effort for this error correction is then given by

$$\binom{544*1.5}{0} + \binom{544*1.5}{1} + \binom{544*1.5}{2} + \binom{544*1.5}{3} = 26831985 = 2^{24.7} \tag{9}$$

modular exponentiations.

In conclusion, the work-flow of this attack is summarized as Figure 15. What's more, this proposal describes a solution to RSA challenge which can retrieve the whole RSA private key without any knowledge of the implementation with a single decryption power trace for both Challenge Round 5 and 6. This attack can be applied to the RSA implementation with countermeasures of exponentiation blinding, message blinding or both, because this is purely a single trace attack. Moreover, the attack provide a results benchmark and work-flow to exploit the extra smaller operation within the big modular operation, e.g. 544 bits modular multiplication or square in this case.

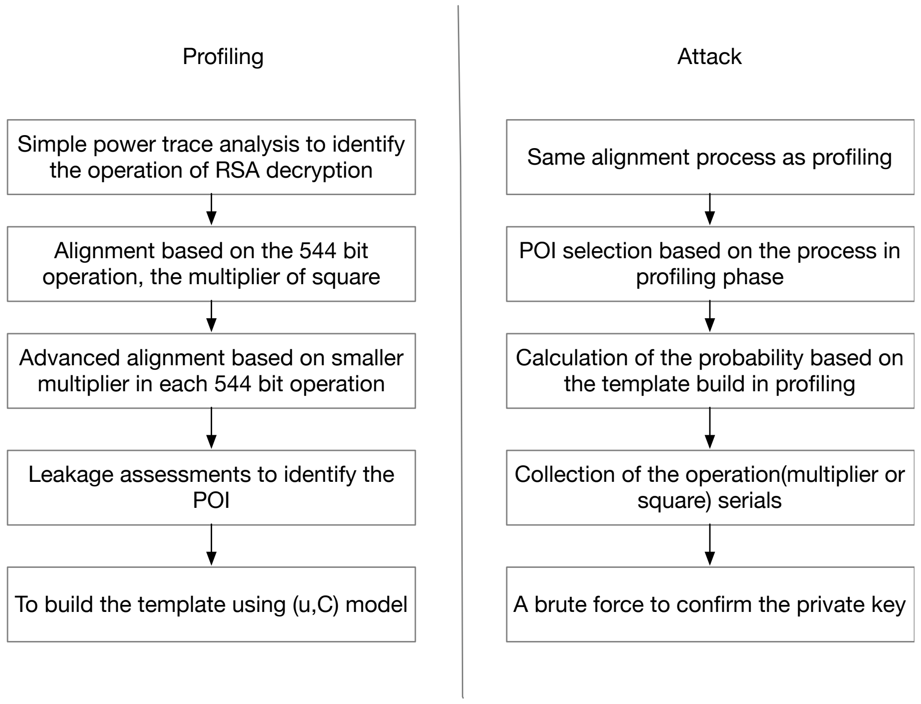
Profiling
Simple power trace analysis to identify the operation of RSA decryption
Alignment based on the 544 bit operation, the multiplier of square
Advanced alignment based on smaller multiplier in each 544 bit operation
Leakage assessments to identify the POI
To build the template using (u,C) model
Attack
Same alignment process as profiling
POI selection based on the process in profiling phase
Calculation of the probability based on the template build in profiling
Collection of the operation(multiplier or square) serials
A brute force to confirm the private key

Figure 15: Work-flow for the attack of RSA challenge

What's more, what we can research further about the RSA part at least includes three aspects.

1. To attack the private key masking operation which is $d' = d + r\phi(N)$.
2. To exploit the existence of the extra subtraction operation for modular multiplication or square.
3. To perform Deep Learning for the traces without too much pre-processing.

5 Conclusion

We use a combination of profiling, POI selection and leakage model assessment, for extracting the keys for the DES challenges. This results in a robust and relatively fast solution. All the techniques used can be classified as classic SCA. It remains to be seen if deep learning or other flavours of machine learning can further improve the performance of this attack.

For the AES challenges, the solution here discussed is based on a combination of domain knowledge and machine learning. Our ML model has 65000 free parameters and requires almost no preprocessing of the trace data prior to training or inference. We show that our model without manual intervention learned to take device properties into account for its predictions. The combined solution consisting of the leakage model, SAT solver and error correction is very efficient: using a modest amount of computation on a single desktop computer, we can solve the portability challenge using only a few traces. In the non-portability challenge, even a single trace usually suffices. Our results show that a combination of relatively simple machine learning techniques and modern constraint satisfaction tools can produce state-of-the-art results on this side channel challenge. DL models on this challenge will have to exploit the leakage extremely well to improve significantly on these results.

For the RSA challenges, the presented solution retrieves the whole key with a single trace without any knowledge of the implementation. We use simple trace analysis to identify the patterns. Next, to filter the leakage points we use advanced trace alignment for small multiplication based on automatic patter searching. To confirm the private key we use the template model and a simple brute force scheme. Same as the solution for AES challenges, this solution is a hybrid between classical side channel analysis and the machine learning (used for trace alignment).

Although we tipped the scale in favour of deep learning, by allowing a lot of time for preparation, we observe the participants who captured the flag preferred classic SCA, or made use a hybrid approach which combines machine learning and classic side channel analysis. We have no results on the performance of machine learning techniques on the hardware engine. This is an interesting finding, when certification bodies in the industry mandate the use of deep learning for high-end security evaluations.

The main contribution of this paper is therefore a snapshot in time, of the state of the art in the community with respect to the application of deep learning techniques for side channel analysis. The second contribution of this paper is an in-depth description of the most successful solutions for key extraction which is a learning opportunity for performing side channel analysis in a black-box scenario by experts in this field. We conclude from the presented solutions that human expertise remains very important in the design of successful attacks by improving it with the use of machine learning techniques. As a third contribution, the solutions presented in this paper serve both as a baseline for future research and as a possible template or other efficient side channel attack designs. Performance improvement may be possible here e.g. in terms of the number of traces required to retrieve the key, the amount of processing applied to the raw traces and the number of bits left for brute forcing.

Acknowledgements This work was partly supported by the European Unions H2020 programme under grant agreement number ICT-731591 (REASSURE).

References

[1] P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis: Leaking Secrets. International Cryptology Conference 1999.

[2] E. Brier, C. Clavier, F. Olivier. Correlation Power Analysis with a Leakage Model. Cryptographic Hardware and Embedded Systems, CHES 2004. Springer Berlin Heidelberg.

[3] Chari S, Rao J R, Rohatgi P. Template Attacks. International Workshop on Cryptographic Hardware and Embedded Systems. 2002.

[4] F. Pedregosa et al., *Scikit-learn: Machine Learning in Python*, Journal of Machine Learning Research, vol. 12, p. 282-2830, 2011

[5] 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

[6] Repository for the tracesets, https://chesctf.riscure.com/2018/training/, accessed 2019/12/01

[7] Pycryptosat homepage, https://pypi.org/project/pycryptosat/, accessed 2018/10/08

[8] A. Gohr, S. Jacob, W. Schindler. CHES 2018 Side Channel Contest CTF - Solution of the AES Challenges. IACR e-print report 2019/094, https://eprint.iacr.org/2019/094/20190131:230649