How to Fool a Black Box Machine Learning Based Side-Channel Security Evaluation

Charles-Henry Bertrand, Olivier Bronchain,
Gaëtan Cassiers, and François-Xavier Standaert

ICTEAM Institute, UCLouvain, Louvain-la-Neuve, Belgium

Abstract. Machine learning and deep learning algorithms are increasingly considered as potential candidates to speed up and automate black box side-channel security evaluations. Inspired by the literature on machine learning security, we put forward that it is easy to conceive implementations for which such black box security evaluations will incorrectly conclude that recovering the key is difficult, while an informed evaluator / adversary will reach the opposite conclusion (i.e., that the device is insecure given the amount of measurements available).

Introduction

The security certification of cryptographic products is a time-consuming and expensive task that implies practical testing by specialized labs. As a result, various approaches have been proposed to speed up this process while maintaining as much confidence as possible in the evaluation outcomes. A popular example of this trend is CRI's non-specific leakage detection test introduced in [13,9] and further discussed in [21,27,11]. It checks whether the leakages of a cryptographic implementation with fixed plaintexts (and key) differ from the ones obtained with random plaintexts (and fixed key). Over the last years, the use of machine learning and deep learning algorithms has been gaining traction as a promising alternative for more efficient and automated security evaluations [16,15,17,18,19,7,25,33]. Yet, while the pros and cons of standard leakage detection tests have been critically discussed in various publications (e.g., [29,5,34]), much less is known about the possible limitations of side-channel security evaluations based on machine learning and deep learning algorithms (e.g., in terms false positives, false negatives, ...).

In parallel, the general issue of machine learning in adversarial settings is an increasingly discussed topic as well [22]. Frequently considered attacks include adversarial examples [2] and data poisoning [3]. (Less relevant examples for the following discussion include model stealing [32] and membership inference attacks [28]). As illustrated in Figure 1(a), the goal of an adversarial example is to craft an observation with small (hard to notice) modifications so that it is misclassified. As illustrated in Figure 1(b), the goal of data poisoning is to add (possibly mislabeled) observations in a training set in order to modify the separation between classes so that some test observations are misclassified. Based on this state-of-the-art, a natural question is whether standard issues in machine learning security may limit the coverage of side-channel security evaluations based on machine learning and deep learning algorithms?

As such, adversarial examples and data poisoning do not seem directly applicable to the side-channel evaluation setting, where observations are typically collected in a trusted and controlled environment. Yet, we show in the following that a very related issue, that we denote as *cheating labels*, may easily fool a black box side-channel security evaluation based on generic classifiers such as Multi-Layer Perceptrons (MLP) or Random Forests (RF). As illustrated in Figure 1(c), the idea behind cheating labels is to confuse the training phase with observations inherently related to two labels: *sensitive labels* (e.g., x & y on the figure) that the target device is trying to hide, and *cheating labels* (e.g., A and B on the figure) that it is trying to make obvious to the classifier. In contrast with the previous examples, such cheating labels have a direct application in a security evaluation setting. Just imagine an implementation made of two designs: one designs leaks "a lot" about some random key; the other leaks "much less" about the real key used to encrypt; and both keys are related by some simple function (e.g., a XOR with some value Δ) so that the profiling of one key cannot be separated from the other.

Based on simulated experiments and actual measurements, we show that cheating labels can be easily instantiated. We discuss an example where two designs run in parallel on the same chip: one design is unprotected and manipulates a misleading key, the other one is masked and manipulates the real (sensitive) one.

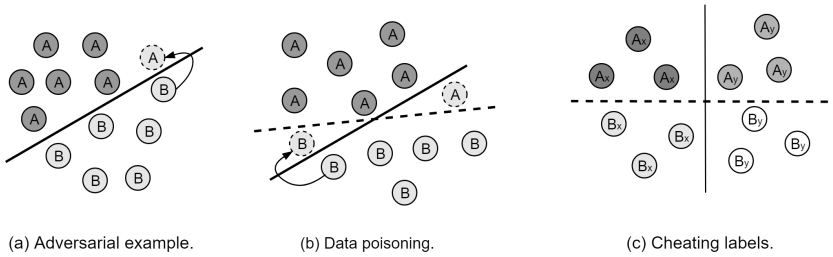


Fig. 1: Machine learning (in)security examples.

In this context, black box security evaluations based on MLP and RF will not converge towards the correct key, while an informed evaluator knowing that the correct key is manipulated by a masked implementation will easily succeed with simple tools such as a Moments-Correlating Profiled DPA (MCP-DPA) [23]. We then discuss how to circumvent the problem of cheating labels by profiling over several misleading keys.

Cautionary notes. Admittedly, cheating labels are in a sense not new. They can be connected either with the problem of label noise in machine learning / deep learning [12], or with the problem of model variability in side-channel analysis [26]. The only difference with these previous works is the adversarial aspect. Yet, we believe our results come as a healthy reminder that while machine learning and deep learning can be very effective tools to attack implementations with limited knowledge, their use as an evaluation tool has to be coupled with a minimum understanding of the target implementation. In other words, it is already known that a worst-case security evaluation cannot be fully unprofiled / unsupervised [35].[1] We show that even in profiled / supervised setting, an evaluation cannot generally succeed in a purely black box setting.

Besides, a recent work advocated for the possibility that adversarial examples can be used as a basis for side-channel countermeasures [24]. The goal of this paper is quite the opposite: we do not to propose cheating labels as a countermeasure against side-channel attacks and our results rather aim to mitigate the optimism for the potential of machine learning and deep learning as black box evaluation tools. Our view is that countermeasures should provide security independent of the attack / evaluation strategy, and that a single (e.g., machine learning / deep learning) tool is unlikely to provide strong theoretical guarantees, especially in a black box context. So this study has to be seen as a complementary to discussions such as [6], which shows that there are realistic implementations for which black box automated security evaluations can be much less efficient than informed ones. We show that there are (less realistic) implementations where black box automated security evaluations cannot succeed at all.

Overall, the goal of this paper is therefore to stimulate a necessary discussion on the advantages and limitations of automated analyzes in the side-channel evaluation context, exactly as it happened for leakage detection. Putting forward critical case studies such as cheating labels is paramount for this purpose, since it provides a concrete basis to understand and mitigate the risks of overstated security claims.

1 Background

In this section, the necessary material for the rest of the paper is presented. Namely, the notations are first introduced. Then, various side-channel distinguishers are recalled, starting with MCP-DPA and followed by machine learning techniques. Finally, the masking countermeasure we use is briefly described.

1.1 Notations.

We denote a plaintext byte as p, a key byte as k and the target intermediate value of our attack as y. Random variables are denoted with bold capital letters such as $\boldsymbol{X}$. The dth-order raw statistical moments are defined as $\mathrm{M}_x^d = \mathsf{E}[\boldsymbol{X}^d]$ where $\mathsf{E}[\cdot]$ is the expectation operator. The dth-order central moments are defined as $\mathrm{CM}_x^d = \mathsf{E}[(\boldsymbol{X} - \mu)^d]$ with $\mu = \mathsf{E}[\boldsymbol{X}]$. The first-order moment M_x^1 is the mean of the distribution & the second-order central moment CM_x^2 is its variance.

[1] Which applies to non-profiled machine learning based evaluations such as [31] as well.

1.2 Side-channel evaluation tools

Moments-Correlating (Profiled) DPA [23]. MCP-DPA is a side-channel attack method based on a chosen statistical moment. In a profiling phase, a statistical moment is estimated for each targeted intermediate value Y by using a set of leakages with known keys k and plaintexts p. For example, estimating the mean of the output of the first Sbox, $y = \mathsf{Sbox}(k \oplus p)$, gives $\hat{\mu}_{p,k} = [\mu_1, \mu_2, \ldots, \mu_y]$ where μ_y denotes the mean value of the leakage for the target value y. The vector $\mathrm{M}^d_{p,k}$ (i.e., in this example, $\hat{\mu}_{p,k}$) obtained is used to recover the correct key which is selected according to:

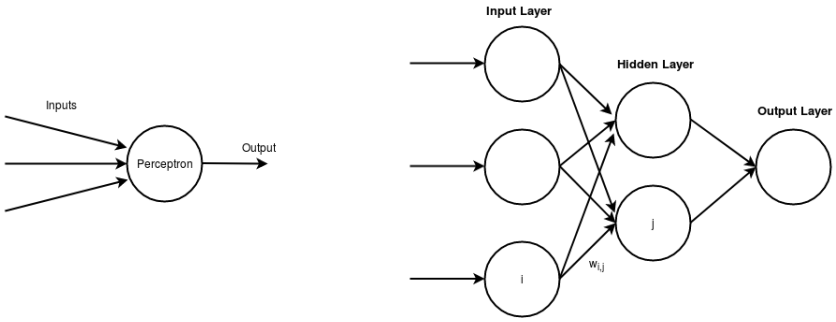
$$\hat{k} = \mathsf{argmax}_{k^*} \ \rho(M^d_{k^*,p}, (L_{p,k})^d), \tag{1}$$

where $L_{p,k}$ is the vector of leakages produced when manipulating a known plaintext p with an unknown key k. So MCP-DPA just estimates Pearson's correlation coefficient between a model corresponding to statistical moments of order d and the actual leakages raised to the same power. If the attack is successful, the best correlation is obtained for the correct key guess.

The main interest of this method is that is allows choosing which moment will be exploited to recover the key, which is usually becoming the optimal strategy as the noise in the measurements increases [10]. In the following investigations, it will be particularly handy to explicitly distinguish based on the sensitive labels (despite the cheating ones may leak "more" in some sense). Concretely, we will only need MCP-DPA of order 1, which targets the means of leakages (i.e., the first-order statistical moment) and MCP-DPA on order 2, which targets the variances of leakages (i.e., the second-order central moment).

Multi-Layer Perceptrons (MLPs). Before explaining MLPs, let us introduce what is a *perceptron*. A perceptron is a linear classifier and represents the simplest neural network model: an n-input $(x_1, x_2, .., x_n)$ perceptron with one output o is defined by n weights $(w_1, w_2, .., w_n)$, a bias w_0 and an activation function f (in our experiments the ReLU function, $\mathsf{f}(x) = \max(0,x)$, was used). A perceptron is illustrated in Figure 2a. The output of the perceptron is computed as:

$$o = \mathsf{f}(w_0 + \sum_{i=1}^{n} w_i x_i).$$



(a) Perceptron. (b) MLP architecture.

Fig. 2: Perceptron and MPL architecture.

During the training phase, weights are initialized at zero (or randomly set to some small value). Then, thanks to a learning algorithm, they are optimized in order to minimize a loss function (which is a function quantifying the error between the prediction of the algorithm and the actual ground truth, for instance the mean squared error). One way to achieve this is to apply the *gradient descent algorithm* to optimize weights.

A Multilayer Perceptron (MLP) is then simply defined as a specific combination of perceptrons allowing to build more complex classifiers. Figure 2b illustrates its typical architecture. The input is propagated from

the left to the right and each unit (i.e., perceptron) of a layer is connected to units of the previous layer. In this figure, each neuron of a layer is connected to all the neurons of the previous layer: this architecture is thus called a fully connected network. In Figure 2b, we observe three types of layers which constitute MLPs:

- **The input layer** makes the intermediate between the input data and the first hidden layer (it just passes the data to the hidden nodes).
- **Hidden layers** allow one to introduce non-linearity in the model in order to fit non-linear separable datasets. According to the complexity of the targeted problem, the number of hidden layers and the number of neurons have to be adjusted. Using too much neurons could lead to an overfitting of the model, while not enough neurons could fail to create an accurate model.
- **The output layer** is the last layer of the network. Outputs of its neurons map directly to labels which have to be predicted. Here these labels are the intermediate values y targeted by the side-channel attack.

The goal of the MLP training phase is to find optimal weights for all neurons of the architecture. For more information about this training, see [4].

Random Forests. Decision trees are classification models based on the sequential application of simple binary rules. They are structured as a directed tree: starting at the root, the tree is traversed towards a leaf by selecting an edge at each node according to a simple rule. The leaf nodes are the class labels, which are the possible values for the target intermediate variable in the side-channel attack context. Each node's rule is a threshold test on the leakage sample.

In the profiling phase, training data is used in order to build the decision tree. First, the dataset is presented to the root and split based on a test over the leakage sample that most effectively discriminates sets associated to different target values. Each new subset created is associated to one of the child nodes of the root, and the process is repeated on each new subset in a recursive way until the child node contains only samples associated to the same target value, or the gain to split again the set becomes less than some threshold. Finally, one assigns a target intermediate value to each leaf.

Next, when a new leakage sample is sent to the decision tree during the attack phase, it is first presented to the root and forwarded to one of its child nodes depending on the result of the edge's test. The process is repeated until a leaf is finally reached. A decision tree is illustrated in Figure 3.

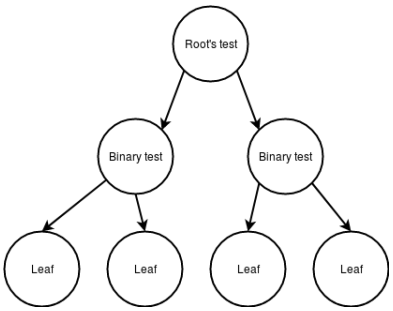


Fig. 3: Decision tree.

A Random Forest (RF) is constituted of many decision trees, each of them trained with a different subset of the training dataset. The output of the random forest is computed through a majority vote among its classification trees outputs. In our experiments, the number of trees in the random forest was set to 200.

1.3 Domain-Oriented Masking.

Masking is a well-known countermeasure against side-channel attacks, of which the goal is to randomize all intermediate values that are manipulated by the algorithm into several shares, so that an adversary is forced to target the shares jointly to recover sensitive information [8].

In the case we will consider next, each intermediate value y is concealed by a random value m called the mask. The mask m is generated by the device and changes from one execution to another. We use Boolean masking, therefore the masked intermediate value is defined as $y_m = y \oplus m$. In order to exploit the leakage of a (securely) masked implementation, the adversary typically has to estimate a higher-order statistical moment, which is the main ingredient leading to security improvements [10]. In the unprotected (resp., 2-share) implementations we will consider, an adversary will therefore have to perform a first-order (resp., 2nd-order) MCP-DPA to successfully recover the key. Performing linear operations on shared data is straightforward (one can just apply the operation share by share). Non-linear operations are more tricky and there is a wide literature investigating solutions to perform secure multiplications with minimum overheads (in time, space and randomness). We will next use the Domain-Oriented Masking (DOM) scheme of Gross et al. [14], which comes with a convenient generic open source HDL code.

2 Cheating labels and instantiation

The objective of following investigations is to confuse a training phase with observations inherently related to two labels: *sensitive labels* (i.e., the real labels that the device is trying to hide) and *cheating labels* (i.e., misleading labels that the device is trying to make obvious to the classifier). Assuming that the profiling and target devices are different (which is usually the case in practice), one can then expect that an attack trained with cheating labels will not converge towards the correct sensitive key in its online phase, since the relationship between sensitive labels and cheating labels is device-dependent.

Our proposed instantiation of cheating labels is illustrated in Figure 4. It consists in two encryption designs running in parallel on the same device. The first one is an unprotected design using a key $k \oplus \delta$ (with δ a device-specific value between 0 and 255). The second one is a 2-share masked design using the sensitive key k. Since the two designs (and the two shares of the masked design) are running in parallel, a simple model for their power consumption (assuming leakages proportional to Hamming weights of the manipulated data) is: $\mathsf{L}(k, p, s, \delta) = \mathrm{HW}(\mathsf{Sbox}(k \oplus \delta \oplus p)) + \mathrm{HW}(\mathsf{Sbox}(k \oplus p) \oplus m) + \mathrm{HW}(m) + \epsilon$, where HW(.) is the Hamming weight function, m the mask and $\epsilon \sim \mathcal{N}(0, \sigma^2)$ is a zero-mean Gaussian noise of variance σ^2.

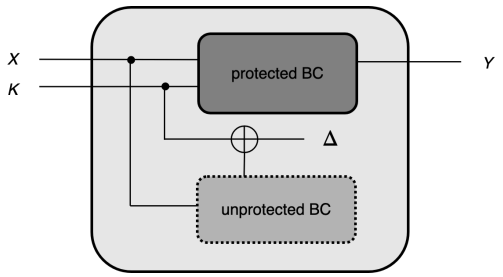


Fig. 4: Cheating labels (general principle).

With sufficiently noisy leakages, we can expect that the unprotected design will leak significantly more and in a more obvious manner than its masked counterpart. In particular, and as previously mentioned, a first-order (resp., 2nd-order) MCP-DPA should be sufficient to break the first (resp., second) design. By contrast, in case of a machine learning (or any other) tool targeting directly the full distribution, the fact that both keys are related by a device-specific δ should lead to primarily model the leakage of the unprotected, misleading design. Therefore, if a model trained on a device with a δ value δ_m is next used to attack a device using δ_a, it is expected that the attack will be unsuccessful (recovering $k \oplus \delta_m \oplus \delta_a$ instead of k).

3 Simulated experiments

We now provide a couple of simulated experiments demonstrating the theoretical applicability of cheating labels. For this purpose, we assume Hamming weight leakages of the form given in the previous section and

consider an adversary who has full control of a (simulated) training device for the profiling phase. We used a noise variance of 10 and allowed a profiling with 2,000 simulated measurements per sensitive value y (out of 256 possibilities). Then, during the attack phase, we simply consider a different δ to generate the leakages. The target intermediate value is the (first-round, first) S-box output of an AES implementation.

For each simulated attack, 20 independent sets of traces were generated, from which we estimated the average key rank (i.e., the guessing entropy), which is a usual metric for side-channel security evaluations [30]. Precisely, after each attack, the rank of a key k^* is defined as:

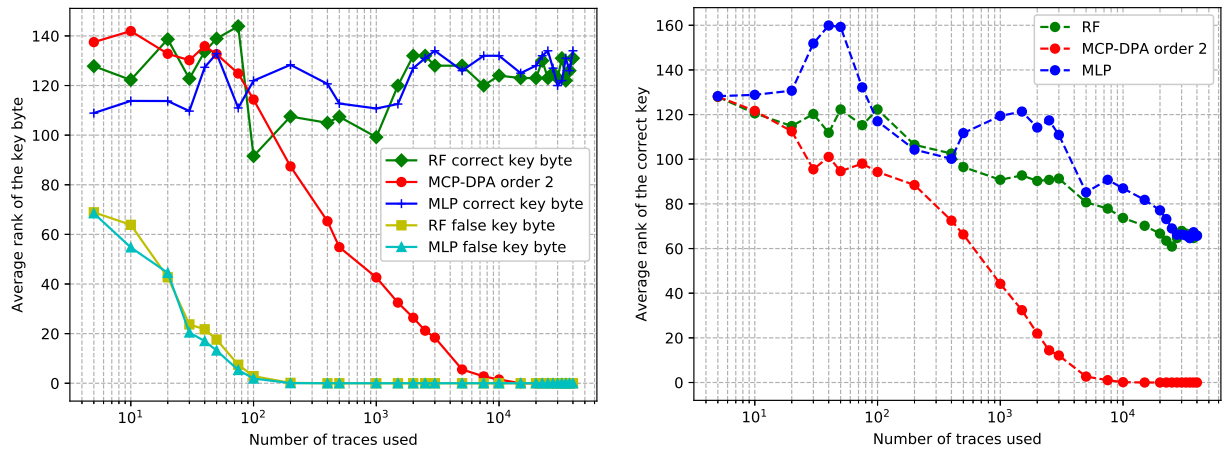
$$rank(k^*) = |\{k \in \mathcal{K} | d[k] > d[k^*]\}|,$$

where $d[k]$ denotes the score (here the likelihood) given to the key k.

Model parameters Our machine learning based attacks use the scikit-learn library [1]. All parameters of the methods under investigation have been selected thanks to a grid search, by evaluating performance of the attack (i.e., the number of traces required to recover the correct key byte and computational resources required for it) against both an unmasked and a masked simulated implementation. For each combination of parameters, 20 datasets independent from the training set were used in order to avoid overfitting, and to enable meaningful comparisons. Value of deltas for profiling sets and each attack set were chosen randomly but stayed consistent among the same set. The parameters used in our experiments are given below:

- Multi-layer perceptrons :
  - Number of hidden layers: 2 – values tested: [1,2,3,4];
  - Number of neurons per layer: 40 – values tested :[10,20,30,40,50];
  - Output layer: 256 neurons.
- Random Forest:
  - Number of trees: 200 – values tested [100,200,300];
  - Maximum depth of the tree: 15 – values tested [10,15,20].

Profiling with a single δ. Our first experiment covers the case where our profiling set contains a single device, hence a single δ, which is different from the one of the attacked device. Figure 5a represents the average rank of the correct key byte for an increasing number of traces used by the attacker.



(a) Profiling with a single δ.

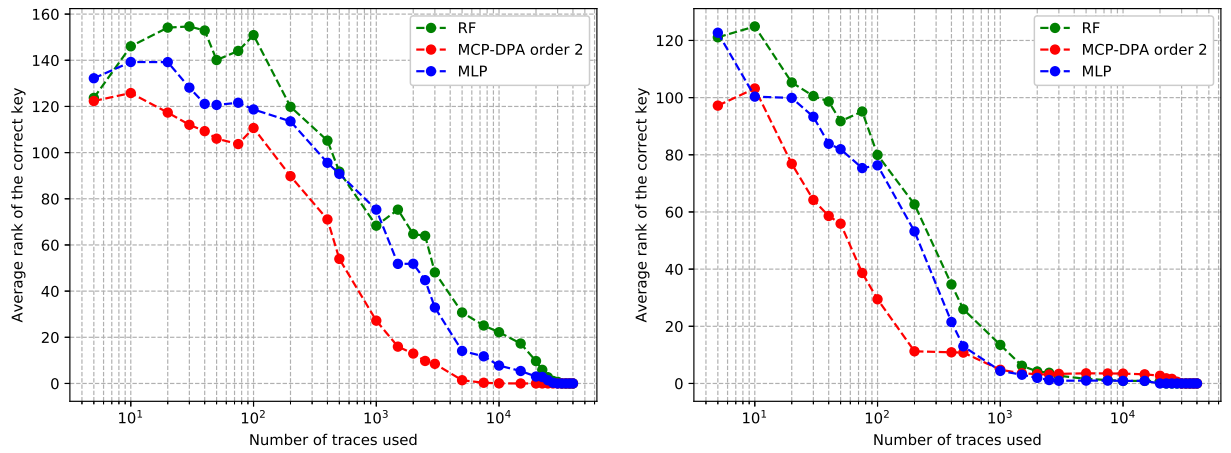
(b) Profiling with 10 different δ's.

Fig. 5: Simulated analyzes (I).

We can observe that the key is easily recovered by the 2nd-order MCP-DPA. By contrast, neither the MPL-based nor the RF-based attacks succeed to discriminate the sensitive key. In fact, both machine learning

based methods converge towards a key $k \oplus \delta_1 \oplus \delta_2$ (called "false key byte" in Figure 5a), by combining deltas of the attack device (say δ_1) and the profiling device (say δ_2). This experiment demonstrates the theoretical applicability of cheating labels: by designing an implementation manipulating two related keys, and forcing the leakage of a misleading key (i.e., corresponding to a cheating label) to be both larger in amplitude and easier to exploit, the leakage related to the sensitive label remains hard to capture during profiling.

Profiling with multiple δ's. As a complement to our first experiment, we investigate a natural option to mitigate the risk of cheating labels. Namely, we repeat the previous attacks after profiling over multiple devices, each of them with a different (randomly generated) δ. Figure 5b represents the average rank of the correct key byte for an increasing number of traces used by the attacker considering a profiling over 10 devices (not including the correct δ). Figure 6a represents the same quantity when profiling over all the δ's.



(a) Profiling with 256 different δ's.

(b) Profiling with 10 different δ's including the correct one.

Fig. 6: Simulated analyzes (II).

We observe that profiling over multiple δ's gradually makes the cheating labels appear as random noise that is easy-to-model for machine learning tools (rather than a fixed key-dependent secret). When profiling with 10 δ's, correct labels are not yet correctly classified but the rank of the correct key decreases to approximately 70. When profiling with 256 δ's, all misleading labels become possible (and equally likely) and machine learning tools therefore discriminate the correct key (which is the only secret left).

Profiling with the good δ's. Eventually, we repeated the profiling with 10 δ's, this time including the δ of the device targeted in the online attack in the profiling set. Results in Figure 3 show that in this case, both the MLP and the RF recover the correct key with less traces.

From the previous simulations, we conclude that machine learning based attacks can circumvent cheating labels in case the correct δ is part of their profiling set, and that the attacks will succeed faster in case the subset of δ's including the correct one used for profiling remains small (which limits the noise).

4 Actual measurements

We now confirm the previous conclusions based on actual measurements. We first describe our measurement setup and then discuss the effect of cheating labels on different Points-of-Interest (POIs) of our traces.

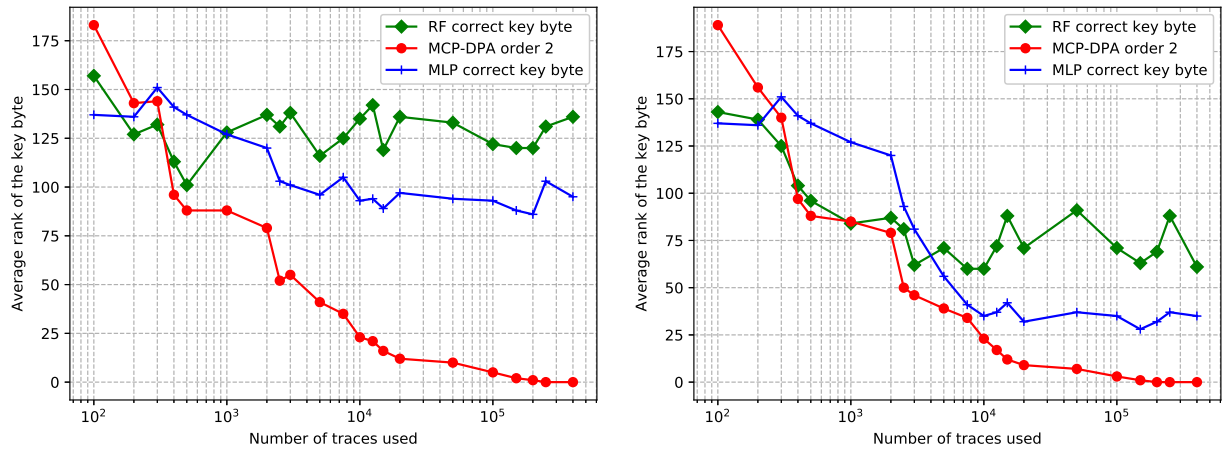
Measurement setup. An actual prototype of cheating labels has been implemented on a `Xilinx Kintex-7 FPGA` placed on a `Sakura-X` side-channel evaluation board.[2] The power consumption is measured on a 1[Ohm]

[2] http://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-X.html

resistor placed between the power supply and the target `FPGA` running at 4[MHz]. This signal is sampled with a `PicoScope 5000 Series` at a rate of 500[MSamples/s] with 12-bit precision. Hence, 125 samples are available within each cycles. The module implementing cheating label is similar to Figure 4 where a protected and an unprotected implementation are running in parallel. Both are derived from the open-source DOM protected AES instantiated with two shares.[3] The protected one is fed with fresh randomness generated from an AES-based PRG. The unprotected one is strictly the same excepted that is fed with a constant as randomness. This ensures that both are synchronous in their manipulation of the sensible variables y, meaning that both 1st- and 2nd-order leakages should be exploitable in the same samples.

As a pre-processing, we evaluated Mangard' Signal-to-Noise Ratio (SNR) on the measured traces [20], which allowed us to identify POIs. For illustration, we selected two POIs (with lower and higher SNRs) for our experiments. For the rest, we used the same parameters as selected for our simulated analyzes and we next consider only the case where a single (incorrect) δ is used for profiling (since the impact of profiling over more δ's and possibly the correct one follows the same pattern as in the previous section).

POI with lower SNR. The results of our various attacks against the POI with lower SNR are in Figure 7a, where we can notice a behavior essentially similar to the one of our simulations. Namely, only the MCP-DPA of order 2 succeeds in recovering the key. So this sample typically corresponds to a situation where the leakage of the cheating labels dominates. We assume this is due to a large-enough noise so that the 1st-order information "dominates". That is, as discussed in [10], Section 4.2, for large enough noise (so low enough SNR), the best adversarial strategy is always to target the lowest-order statistical moment of the leakage distribution, which is only generated by the cheating labels.



(a) Attacks exploiting a POI with low SNR.

(b) Attacks exploiting a POI with high SNR.

Fig. 7: Real measurements.

POI with higher SNR. The results of our various attacks against the POI with higher SNR are in Figure 7b. This time the effectiveness of machine learning based attacks significantly improves: one still cannot recover the key in full, but its rank is noticeably decreased. We assume this difference to be due to a less dominant 1st-order leakage, due to a lower level of noise in the traces. That is, as discussed in [10], Section 4.2, when the noise decreases, the complexity of exploiting the different statistical moments of the leakage distribution becomes more similar. This explains why the first-order leakages of our cheating labels become less dominant in front of the sensitive second-order leakages.

[3] https://github.com/hgrosz/aes-dom

5 Conclusions

Both our simulations and experimental results confirm that cheating labels can be an effective way to fool black-box machine learning based side-channel security evaluations, and how a more specific profiling can circumvent them. In this respect, we note that profiling over 256 δ's as in this paper is not overly expensive, but one could naturally design more expensive implementations mixing masking schemes at more security orders to make the profiling more expensive.

More generally, the examples given in this paper may admittedly look somewhat artificial, since it is unlikely that any concrete adversary has for sole purpose to fool a security evaluation. Yet, we believe that they show an important risk of shortcoming that all black box security evaluations (and importantly, not only the ones exploiting machine learning / deep learning) may encounter. Namely, when ignoring important implementation details, one may incorrectly conclude that some sensitive operations are hard (or even impossible) to profile, hence missing them in the online attack phase of the evaluations.

Acknowledgments. Gaëtan Cassiers and François-Xavier Standaert are respectively Research Fellow and Senior Associate Researcher of the Belgian Fund for Scientific Research (FNRS-F.R.S.). This work has been funded in part by the ERC project 724725.

References

1. Scikit-learn library. http://scikit-learn.org/stable/.
2. M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar. Can machine learning be secure? In AsiaCCS, pages 16–25. ACM, 2006.
3. B. Biggio, B. Nelson, and P. Laskov. Poisoning attacks against support vector machines. In ICML. icml.cc / Omnipress, 2012.
4. C. M. Bishop. Pattern recognition and machine learning, 5th Edition. Information science and statistics. Springer, 2007.
5. O. Bronchain, T. Schneider, and F. Standaert. Multi-tuple leakage detection and the dependent signal issue. IACR Trans. Cryptogr. Hardw. Embed. Syst., 2019(2):318–345, 2019.
6. O. Bronchain and F. Standaert. Side-channel countermeasures' dissection and the limits of closed source security evaluations. IACR Trans. Cryptogr. Hardw. Embed. Syst., 2020(2):1–25, 2020.
7. E. Cagli, C. Dumas, and E. Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing. In CHES, volume 10529 of LNCS, pages 45–68. Springer, 2017.
8. S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. In CRYPTO, volume 1666 of LNCS, pages 398–412. Springer, 1999.
9. J. Cooper, E. D. Mulder, G. Goodwill, J. Jaffe, G. Kenworthy, and P. Rohatgi. Test vector leakage assessment (tvla) methodology in practice. In International Cryptographic Module Conference (ICMC 2013), page 13.
10. A. Duc, S. Faust, and F. Standaert. Making masking security proofs concrete - or how to evaluate the security of any leaking device. In EUROCRYPT (1), volume 9056 of LNCS, pages 401–429. Springer, 2015.
11. F. Durvaux and F. Standaert. From improved leakage detection to the detection of points of interests in leakage traces. In EUROCRYPT (1), volume 9665 of LNCS, pages 240–262. Springer, 2016.
12. B. Frénay and M. Verleysen. Classification in the presence of label noise: A survey. IEEE Trans. Neural Netw. Learning Syst., 25(5):845–869, 2014.
13. G. Goodwill, B. Jun, J. Jaffe, P. Rohatgi, et al. A testing methodology for side-channel resistance validation. In NIST non-invasive attack testing workshop, volume 7, pages 115–136, 2011.
14. H. Groß, S. Mangard, and T. Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In TIS@CCS, page 3. ACM, 2016.
15. A. Heuser and M. Zohner. Intelligent machine homicide - breaking cryptographic devices using support vector machines. In COSADE, volume 7275 of LNCS, pages 249–264. Springer, 2012.
16. G. Hospodar, B. Gierlichs, E. D. Mulder, I. Verbauwhede, and J. Vandewalle. Machine learning in side-channel analysis: a first study. J. Cryptographic Engineering, 1(4):293–302, 2011.
17. L. Lerman, S. F. Medeiros, G. Bontempi, and O. Markowitch. A machine learning approach against a masked AES. In CARDIS, volume 8419 of LNCS, pages 61–75. Springer, 2013.
18. L. Lerman, R. Poussier, G. Bontempi, O. Markowitch, and F. Standaert. Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In COSADE, volume 9064 of LNCS, pages 20–33. Springer, 2015.

19. H. Maghrebi, T. Portigliatti, and E. Prouff. Breaking cryptographic implementations using deep learning techniques. In SPACE, volume 10076 of LNCS, pages 3–26. Springer, 2016.
20. S. Mangard. Hardware countermeasures against DPA ? A statistical analysis of their effectiveness. In CT-RSA, volume 2964 of LNCS, pages 222–235. Springer, 2004.
21. L. Mather, E. Oswald, J. Bandenburg, and M. Wójcik. Does my device leak information? an a priori statistical power analysis of leakage detection tests. In ASIACRYPT (1), volume 8269 of LNCS, pages 486–505. Springer, 2013.
22. P. D. McDaniel, N. Papernot, and Z. B. Celik. Machine learning in adversarial settings. IEEE Security & Privacy, 14(3):68–72, 2016.
23. A. Moradi and F. Standaert. Moments-correlating DPA. In TIS@CCS, pages 5–15. ACM, 2016.
24. S. Picek, D. Jap, and S. Bhasin. Poster: When adversary becomes the guardian - towards side-channel security with adversarial attacks. In CCS, pages 2673–2675. ACM, 2019.
25. S. Picek, I. P. Samiotis, J. Kim, A. Heuser, S. Bhasin, and A. Legay. On the performance of convolutional neural networks for side-channel analysis. In SPACE, volume 11348 of LNCS, pages 157–176. Springer, 2018.
26. M. Renauld, F. Standaert, N. Veyrat-Charvillon, D. Kamel, and D. Flandre. A formal study of power variability issues and side-channel attacks for nanoscale devices. In EUROCRYPT, volume 6632 of LNCS, pages 109–128. Springer, 2011.
27. T. Schneider and A. Moradi. Leakage assessment methodology - extended version. J. Cryptographic Engineering, 6(2):85–99, 2016.
28. R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. In IEEE Symposium on Security and Privacy, pages 3–18. IEEE Computer Society, 2017.
29. F. Standaert. How (not) to use welch's t-test in side-channel security evaluations. In CARDIS, volume 11389 of LNCS, pages 65–79. Springer, 2018.
30. F. Standaert, T. Malkin, and M. Yung. A unified framework for the analysis of side-channel key recovery attacks. In EUROCRYPT, volume 5479 of LNCS, pages 443–461. Springer, 2009.
31. B. Timon. Non-profiled deep learning-based side-channel attacks with sensitivity analysis. IACR Trans. Cryptogr. Hardw. Embed. Syst., 2019(2):107–131, 2019.
32. F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Stealing machine learning models via prediction apis. In USENIX Security Symposium, pages 601–618. USENIX Association, 2016.
33. F. Wegener, T. Moos, and A. Moradi. DL-LA: deep learning leakage assessment: A modern roadmap for SCA evaluations. IACR Cryptology ePrint Archive, 2019:505, 2019.
34. C. Whitnall and E. Oswald. A critical analysis of ISO 17825 ('testing methods for the mitigation of non-invasive attack classes against cryptographic modules'). In ASIACRYPT (3), volume 11923 of LNCS, pages 256–284. Springer, 2019.
35. C. Whitnall, E. Oswald, and F. Standaert. The myth of generic dpa...and the magic of learning. In CT-RSA, volume 8366 of LNCS, pages 183–205. Springer, 2014.