Side-Channel Analysis of ANSSI's protected AES implementation

Internship Report

Université Catholique de Louvain, Louvain-la-Neuve, Belgium

Yann Aguettaz
1^{st} year Master Student
Computer Science
ENS de Lyon, Lyon, France

Supervised by:
François-Xavier Standaert
Olivier Bronchain
Crypto group, ICTEAM, UCLouvain
Louvain-la-Neuve, Belgium

May to July, 2021

Contents

0 The internship – in brief

My internship took place in the crypto group, part of the Institute of Information and Communication Technologies, Electronics and Applied Mathematics (ICTEAM) at UCLouvain in Belgium. It lasted 12 weeks between May and July 2021. Due to the sanitary situation, the few first weeks were done at a distance, but thankfully I was able to go there from mid-June to finish it. It is much easier both emotionally and content-wise to be able to reach a colleague by just walking to his office or sharing a

coffee or hot chocolate. The whole team was very welcoming, and I enjoyed working there.

In particular, I would like to thank Charles, with whom I shared my office, who was very helpfull planning my arrival in this distrubed context, and Olivier who helped me all along my internship, whenever I had a question or didn't understand something. I am also grateful to Davide, Balazs, Loïc, Gaëtan and the others for the shared beers and the interesting discussions in the cafetaria. Lastly, I extend my gratitude to François-Xavier which accepted me as an intern to work on an interesting subject which I knew very little about.

1 Introduction

1.1 Side-Channel Anaylsis

The security of a cryptographic device can be analysed from two angles. On the one hand, classical cryptanalysis focuses on the security of the underlying algorithm, seen as some mathematical function of a plaintext, a key and some randomness bits. On the other hand, when the algorithm gets embeded into a physical device, another source of possible insecurity appears: the physical implementation. The latter are studied by the field of physical security.

Side-Channel Attacks[1] are a class of physical attacks which exploit physical leakages of the device, such as response time, power consumption or electomagnetic radiation, to retrive some secret information (usually a secret key) manipulated in the system. They are usually non-invasive, meaning that they do not require to destroy the device, and relatively easy to carry, therefore they pose a real threat to a device's security, and are–or should be–taken into consideration when designing such a device.

When building an attack, we may, depending on the security model, have access to a copy of the device which we can modify as we want, or have full knowledge of the inner working of the device. Once our attack is built, the goal is to retrieve the (fixed) secret information on a device we don't control, by mean of the information leaked through the side channels over several operations of it. Usually the goal is to minimize the number of observations we need to retrieve the information.

Cryptographic algorithms implemented in the obvious, most direct way are often very sensitive to Side-Channel Attacks, as it is very natural for the power consumption to be correlated with the inner secret information, due do the representation of

[1] SCA for short

information in a physical device. Therefore, it is necessary to develop countermeasures to hide actual information and try to obscure its relationship with the side-channel leakages.

1.2 ASCAD

Sice-Channel security being dependent on every specific component in the device, and relations between them, for instance regarding information speed through the device, it is hard to theoretically evaluate. Therefore, the way a device's security is checked is often by running top-of-the-art attacks on it to see how well they perform. This however yields a new challenge: determining what a top-of-the-art attack is.

ANSSI[2] thus settled to propose a common adversary against which to compare different SCA, in the same fashion as the Machine Learning community uses the MNIST[3] database to compare handwritten digits recognitions algorithms. In 2018, they thus introduced the ASCAD[4] database[7]. It is a set of power consumption measurements of $60,000$ encryptions on a protected[5] implementation of AES[6], with $100,000$ measurements per encryption, each trace being labeled by the plaintext, key and internal randomness bits used, so as to simulate an environnment where we fully control the device.

However, as the dataset became increasingly studied, its difficulty to attack decreased and it is now a reputably easy target. Furthermore, ANSSI also released an other, more protected implementation of AES in 2019[7]. This prompted Olivier Bronchain and François-Xavier Standaert to try and attack this implementation, using their own, non released, measurements[2]. Subsequently, ANSSI decided to release a new public measurements database on this implementation, named ASCADv2[5], in order to propose a new standard for fair comparison of Side-Channel Attacks. They also developed several Deep Learning attacks on the dataset.

The goal of the internship is to develop attacks of the same type as Bronchain *et Al*'s, but on the new dataset, to be able to compare them with others attacks, including the Machine Learning attacks presented in the paper that introduces the dataset. Bronchain *et Al*'s attacks rely on purely statistical methods, and belong to a class of attacks that are called Template Attacks.

[2] **A**gence **N**ationale de la **S**écurité des **S**ystèmes d'**I**nformation, the french computer security agency
[3] http://yann.lecun.com/exdb/mnist/
[4] **A**NSSI's **SCA D**atabase
[5] with additional countermeasures
[6] Advanced Encryptiown Standard
[7] https://github.com/ANSSI-FR/SecAESSTM32

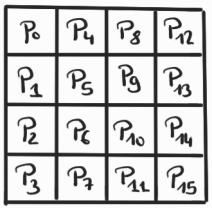
2 The target

We present the target, which is a protected implementation of AES. First, we present the overall operation of classical AES, and then the countermeasures that have been implemented to protect it against SCA. Lastly, we present the dataset we have access to and with which we will build and assess our attack.

2.1 AES

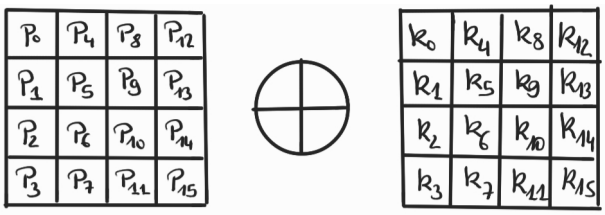
AES is a symmetric encryption standard established after a call to proposal by the U.S. NIST[8] in 2001, and one of the most widely used nowadays.

The most common version, and the one used in the target implementation, uses a 128 bits key to encrypt 16 bytes of data. It starts by organizing the 16 bytes of plaintext in a 4x4 grid, top-to-bottom left-to-right. The same is done with the key, which is expanded into 11 different keys by an operation called the Key Expansion. All subsequent computations are performed in the gallois field $GF(256)$.



AddRoundKey

The first step is to byte-wise xor the plaintext state with the first round key

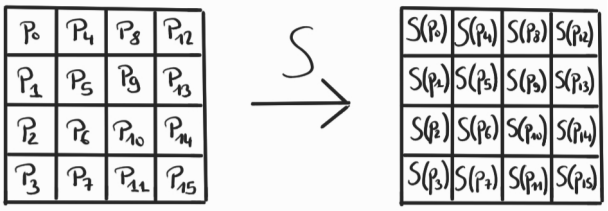


10 rounds

The algorithm then consists of 10 rounds, each containing 4 steps as follows:

1. SubBytes (SBox)

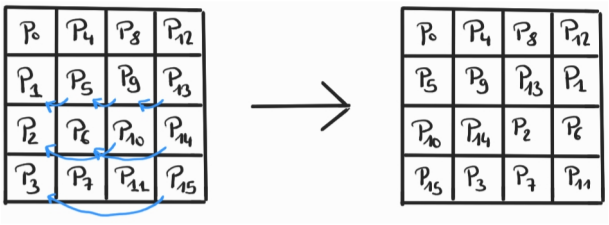
Each byte of the current state is replaced by its mapping via the *AES Sbox*, which is a derangement of $GF(256)$.



[8] National Institute of Standards and Technology

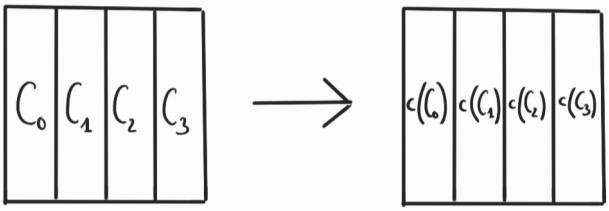
2. ShiftRows

Each row of the current state is shifted left by its index, meaning that the first row is left unchanged, the second is shifted one time, etc.



3. MixColumns

Each column is multiplied by a fixed polynomial. This step is skipped for the last round.



4. AddRoundKey

As in the first step, the current state it byte-wise xored with the round key.

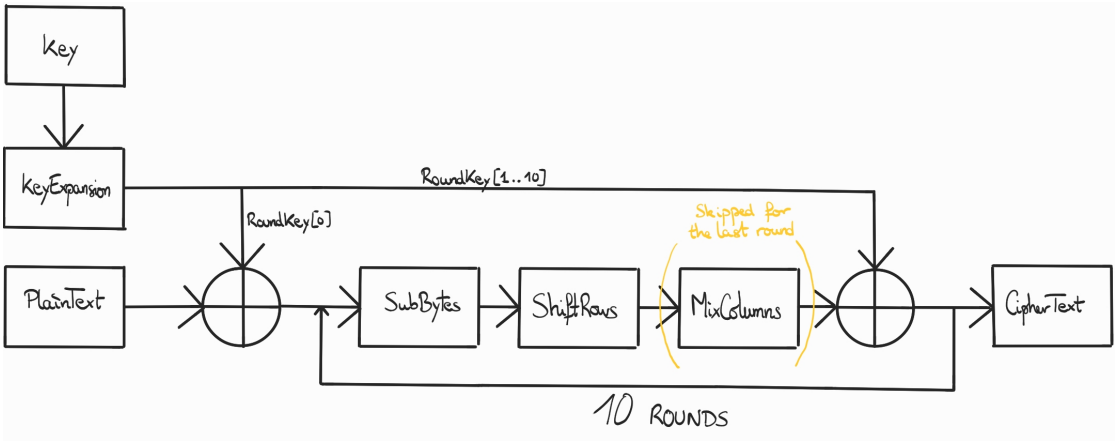


Figure 1: Full overview of AES

2.2 Protection

In order to try to protect the target implementation from SCA, two countermeasures are implemented.

2.2.1 Masking

Masking was introduced by Fumaroli *et Al* in 2010[3]. The principle is that the AES state (the 16 bytes matrix) is never manipulated directly, but instead replaced by some bijective transformation of it at each stage. Depending on the requirements of each

step of the algorithm, that transformation might change in time.

In the target implementation, it is masked using a multiplicative mask $\alpha \in GF(256)\backslash\{0\}$, and several additive masks $\beta \in GF(256)$, as such:

$$\text{maskedState} = (\alpha \otimes \text{state}) \oplus \beta$$

The multiplicative mask α is equal to some random value r_m, drawn at random for each encryption, and fixed for the whole execution.

On the other hand, the additive one β can be, depending of the current stage of the algorithm, equal to:

- $state_M[i]$, a byte from an other state table, on which each AES operation is also applied during the execution. It therefore changes after each operation.
- r_{in}, a random value drawn at random for each encryption, used at the entry of nonlinear operations (i.e Sbox) where the previous mask is locally cancelled.
- r_{out}, another random value drawn at random, used at the output of such operations.

2.2.2 Shuffling

The idea of shuffling is to perform some independant operations in a random order, in order to obscure which value is manipulated at which instant for an attacker. In this implementation, such a shuffling is performed thanks to two permutations, drawn randomly for each encryption:

- A 4-permutation is used to determine the order in which the polynomial multiplications are done in the MixColumns step.
- A 16-permutation is used to determine the order in which the bytes are processed in every other step.

2.3 The Dataset

The dataset is composed of the result of the measurements of the power consumption over $800,000$ encryptions, with 1M samples per execution. Such a set of 1M power values is called a trace. Additionnaly, we are provided, for each encryption, with the plaintext, key, ciphertext and masks used. Each of these data (except the ciphertext that depends on the others) are drawn uniformly for each encryption. This results in a dataset of over 800GB, split into 8 files of roughly 100GB.

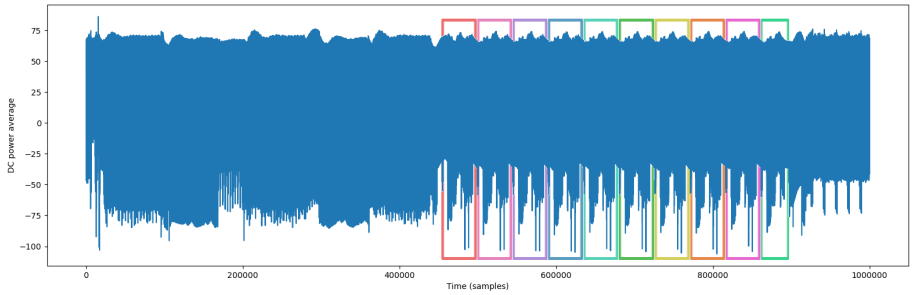


Figure 2: Average of 1000 traces. We can already observe the 10 AES rounds.

3 The Attack

In our security model, we have full knowledge of the inner workings of the device, and we have access to a copy, here represented by the dataset, which we ran with a lot of different keys and plaintexts.

The attack we build, largely inspired by Bronchain *et Al*'s, and that they qualify of "Countermeasures Dissection", begins by targeting the internal randomness of the countermeasures, like the masks values (or "shares" in the vocabulary of SCA). The first goal is to build a probability model over the possible values of each share, independently. Once done, we hold the most likely value of each share as true, and try to retrieve the key with this hypothesis.

3.1 Building blocks

We first present some statistical tools which are used to build the attack.

3.1.1 Signal-to-Noise Ratio

The first step of the attack is to reduce the dimensionality of the problem, because treating the whole traces is very computationally expensive. We use SNR as a mean to identify points along the traces where the power consumption is most correlated with the targeted data. As the name suggests, it is defined as the ratio of the signal over the noise, and in this context, for a share modeled as a random variable X (e.g the additive mask) and a leakage L_X, equal to[4]:

$$\text{SNR} = \frac{\text{Signal}}{\text{Noise}} = \frac{\mathbb{V}\text{ar}_{x \leftarrow X}\left[\mathbb{E}(L_x)\right]}{\mathbb{E}_{x \leftarrow X}\left[\mathbb{V}\text{ar}(L_x)\right]}$$

The goal is to compute that value for each of the 1M time samples. For each of the targeted variable, this metric allows us to rank the samples from least to most informative about the data, and thus to easily restrain our problem to several thousand points instead of the initial million.

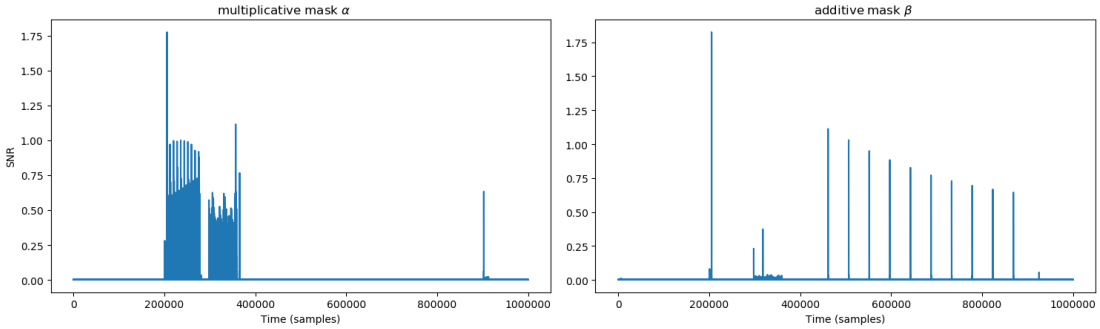


Figure 3: Example: SNR for two of the shares

For instance, on the above plots, we can see when the value of the shares are correlated with the power consumption, which correspond to when the shares are manipulated. It is coherent with what we expect, as the additive mask is used at the beginning to precompute some tables for the masked operations, and to mask the state, and at the end to cancel the masking, but never in between. On the other hand, the multiplicative mask is canceled locally once per round, so there are 10 peaks during the execution.

We then select so-called points of interest by selecting the indexes where the SNR is maximized.

3.1.2 Linear Discriminant Analysis Classifier & Gaussian Template

We then use LDA Classification to select the meaningful features of the signal by projecting the leakages in a space of smaller dimension k[8]. It models the probability law of the leakages according to the variables under the form of a gaussian distribution, such that

$$f(l|x) = \frac{1}{\sqrt{(2\pi)^k \cdot |\Sigma|}} \cdot \exp\left[\frac{1}{2}\left(W \cdot l - \mu_x\right)\Sigma\left(W \cdot l - \mu_x\right)^T\right]$$

where W is a dimensionality reduction matrix, k is the number of classes, $\mu_x = \mathbb{E}(W \cdot l_x)$ the mean of the leakage in the new space, and $\Sigma = \mathbb{C}\text{ov}(W \cdot l_x - \mu_x)$.

To retrieve the probability of a value x of the share, we can then apply Bayes law, such that

$$\mathbb{P}[x|l] = \frac{f(l|x)}{\sum_{x'} f(l|x')}$$

3.1.3 Perceived Information

To assess the precision of the classification model, we use the notion of perceived information, which corresponds to the amount of information that can be extracted from the data with this model[1], and is defined, for a random variable $X \in \mathcal{X}$ and a model $M : \mathcal{L} \to \mathcal{X}$, as

$$PI = H(X) + \sum_{x \in \mathcal{X}} \mathbb{P}(x) \cdot \sum_{l \in \mathcal{L}} \mathbb{P}(l|x) \cdot \log_2 m(x|l)$$

This value is upper bounded by the entropy, and should be maximized to retrieve as much information as possible.

3.2 The attack

We now combine the previously exposed elements to build the attack. We target the internal state of the circuit just after the SubBytes operation in the first round. This value depends on the key, the plaintext, the additive and multiplicative masks, and the 16-permutation over the state bytes for the SBox operation. Therefore, we first need to retrieve both masks, and the permutation indexes.

3.2.1 SNR

We have already shown in figure 3 that we are able to identify the points where the two masks that we are interested in. We still need to retrieve the permutation.

The AES state at this point is $(\text{SBox}[\text{plaintext} \hat{} \text{ key}] \otimes \alpha) \oplus \beta$, which is a 16 bytes value, so we can compute 16 SNRs to retrieve the points where the bytes are manipulated. However, due to the shuffling on the order of these 16 bytes, we learn nothing if we naively try to compute the SNR of each byte of the state separately.

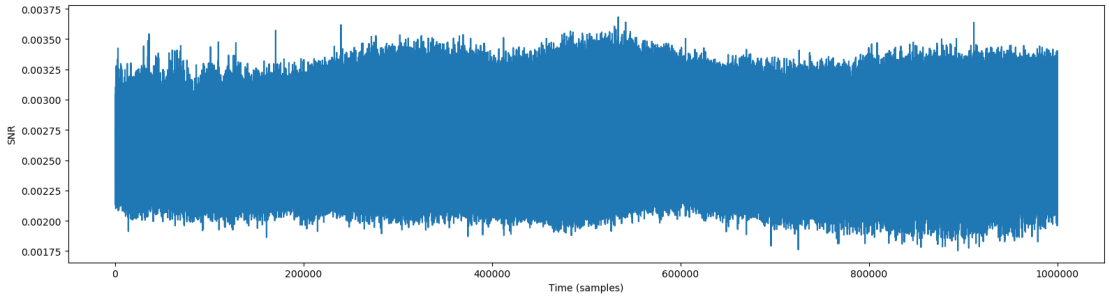


Figure 4: SNR for the first byte of the AES state at the target point

By taking the permutation (which is retrievable with the four first mask bytes) in consideration, it is on the other hand possible to focus rather on the first manipulated byte than on the first state byte, and doing so we obtain an exploitable SNR. We do the same for each of the 16 indexes.

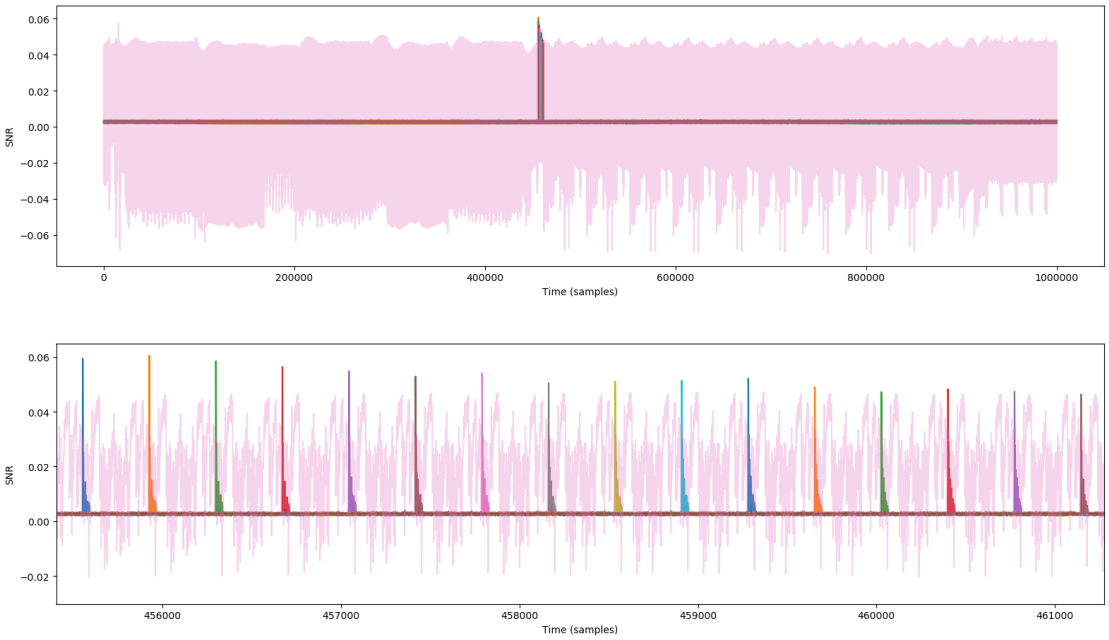
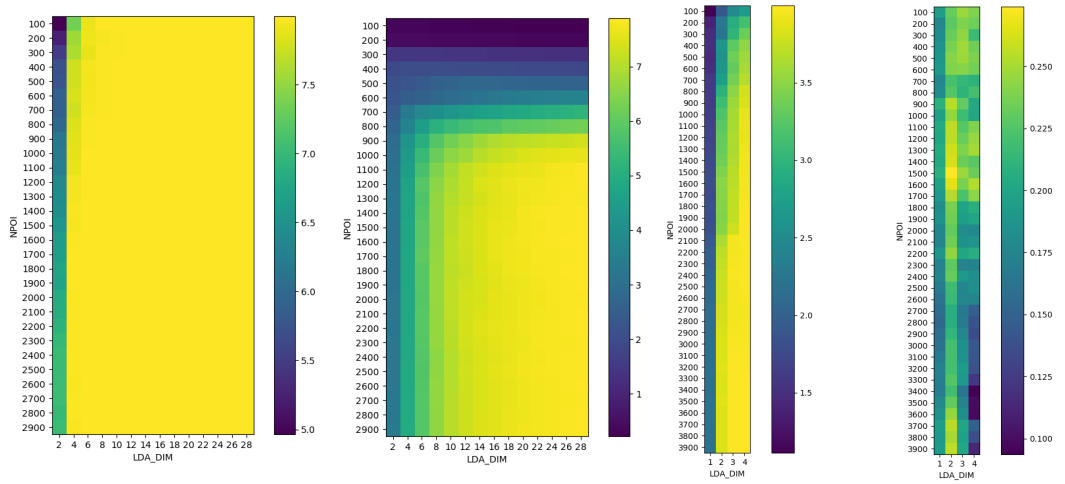


Figure 5: SNRs for the 16 bytes, grouped by order of manipulation rather than position in the AES state. In background is the scaled down average of the traces: the peaks are within the first AES round. The second figure is a zoom on the peaks

3.2.2 LDA - PI

There are two main parameters we can tune to raffine the result of the classification: the number of points of interest we keep after the SNR step, and the number of internal dimensions of the LDA Classifier. We plot here the value of the potential information depending on these two parameters, knowing that we want to maximize it, and it is upper bounded by the log of the number of possible values.



(a) multiplicative mask α 7.99/7.99 (b) additive mask β 7.99/8 (c) indexes 3.99/4 (d) values 0.3/8

We see that the accuracy on the multiplicative mask is really good, even with few points of interest. We need slightly more work on the additive mask ans the permutation indexes, but they are also retrieved with great accuracy. The result on the permuted values is not so great, and there could probably be some improvement there, but we will still proceed with these value.

3.2.3 Retrieving the key

Once we have attacked all the shares that are of interest to us, we can recover the actual key. We actually build a probability model for each byte of the key, and then check the rank of the actual key if we enumerate the 16-bytes tuples in decreasing order of probability.

To compute the probability distribution for a given byte, we begin by enumerating all 256 possible values of this byte. We then compute, for each trace in our attack, the value of the corresponding byte in the AES state at the target point, which is equal to $(\text{SBox}\,[\text{plaintext} \,\hat{}\, \text{key}] \otimes \alpha) \oplus \beta$, that we obtain with this key byte value and while

assuming that the most probable choice for the masks α and β are the true values. We can give a score to this byte value and for each particular trace by looking at the probability that our model on the permuted value assigned to the value we just calculated. Lastly, we agregate these score over all attack traces by taking the mean of the logs of the scores of each trace. We now have a score for each possible byte value, for this particular byte.

Doing this for each of the 16 key bytes, we obtain 16 probability distributions over $GF(256)$. We can then enumerate the 16-tuples of bytes by decreasing order of probability until we find the actual key.

3.3 Possible improvements

3.3.1 True permutations

After training the LDA on the permutation indexes, we are left with 16 probability distributions over $[[0, 15]]$, predicting the index at which each byte is manipulated. However, if we treat them naively, i.e we take the index of maximum probability for each position, we have no guarantee that the 16 values we choose actually form a permutation of $[[0, 15]]$, as there may be repetitions. A possible improvement is to condider the **permutation** of highest probability instead of the **tuple** of highest probability. Indeed, when attacking over 2700 traces, we found that about 7% of the predicted tuples aren't permutations. However, quick tests showed that the accuracy improvement was very slight, so we didn't continue using it, as it represented an additional complexity not worth the gain.

4 Results

Finally, we can assess the peformance of our attack. In practice, one would enumerate the keys in decreasing order of probability, based on the marginal distributions of each bytes, until the correct key (i.e the one that successfully retrieves the secret messages) is found.

Here, we use key rank estimation techniques from [6] to estimate the rank of the correct key without doing it in practice, which is impossible to do for the small number of traces. We thus are able to plot an estimation of the rank of the correct key based on the number of traces considered.

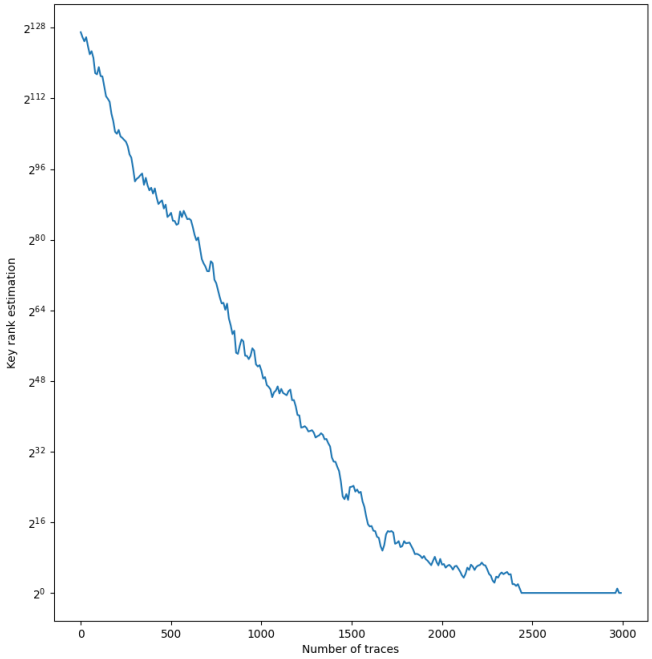


Figure 7: Key rank estimation depending on the number of traces

We see that we recover the correct key as our first guess from less than 2500 traces. However, we obtain an exploitable with much less traces. For instance, with 1400 traces, there are about 2^{32} keys to enumerate, which is doable in practice in a reasonable time.

Besides, these results are of about the same order as those obtained by Bronchain *et Al'* in [2], who recover the full key with about two thousands traces. It seems that the Deep Learning approaches presented in [5] perform better, with about two hundred traces needed. However, they are also more complex to setup, while template attacks are well known and fairly easy to build, while still remaining powerful.

One difference between this attack and the one developped by Bronchain *et Al'* is the targeted point: while we target the output of the SBox of the first round, they target the output of the MixColumns of the first round. It would also be interesting to have been able to compare the two approaches, to see if one is a better choice than the other. Indeed, the MixColumns step being only shuffled with a 4-permutation, it is a-priori easier to retrieve.

A Code of the attack

The code used to perform the attack is hosted in the following GitHub repository:

https://github.com/yann-a/ASCADv2-Internship

It uses SCALib[9], a Python library that runs Rust under the hood and is dedicated to Side-Channel Attacks, mainly developped by Olivier Bronchain.

B Measurement Setup

While exploring the dataset to build the attack, it appeared to us that it didn't seem uniform, especially between traces from the beginning and the end of the dataset. That difference is especially visible if we plot the average of each trace in the dataset on a graph, as below.

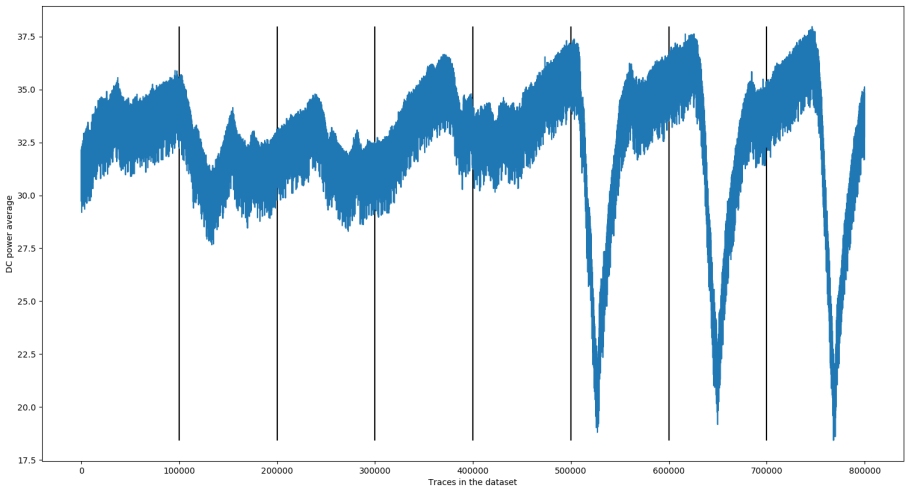


Figure 8: Average consumption of each trace in the dataset. Vertical line represent file separations.

The average is fluctuating, especially within the last 3 files, where the abrupt pits seem to indicate that something happened, for instance the ship might have been stopped, and not let to warm long enough before restarting the measurements. We warned people at ANSSI about this, as a faulty measurement setup might make a

[9] https://github.com/simple-crypto/SCALib

device whose security we are trying to assess look more secure than it actually is.

It could have been interesting to assess how much this irregularity in the measurements actually affects the good working of the attack, but here we just removed the three last files from our study.

References

[1] Olivier Bronchain, Julien M. Hendrickx, Clément Massart, Alex Olshevsky, and François-Xavier Standaert. Leakage certification revisited: Bounding model errors in side-channel security evaluations. Cryptology ePrint Archive, Report 2019/132, 2019. https://eprint.iacr.org/2019/132.

[2] Olivier Bronchain and François-Xavier Standaert. Side-channel countermeasures' dissection and the limits of closed source security evaluations. Cryptology ePrint Archive, Report 2019/1008, 2019. https://eprint.iacr.org/2019/1008.

[3] Guillaume Fumaroli, Ange Martinelli, Emmanuel Prouff, and Matthieu Rivain. Affine masking against higher-order side channel analysis. Cryptology ePrint Archive, Report 2010/523, 2010. https://eprint.iacr.org/2010/523.

[4] Stefan Mangard. Hardware countermeasures against dpa – a statistical analysis of their effectiveness. volume 2964, pages 222–235, 02 2004.

[5] Loïc Masure and Rémi Strullu. Side channel analysis against the anssi's protected aes implementation on arm. Cryptology ePrint Archive, Report 2021/592, 2021. https://eprint.iacr.org/2021/592.

[6] Romain poussier, François-Xavier Standaert, and Vincent Grosso. Simple key enumeration (and rank estimation) using histograms: an integrated approach. Cryptology ePrint Archive, Report 2016/571, 2016. https://eprint.iacr.org/2016/571.

[7] Emmanuel Prouff, Remi Strullu, Ryad Benadjila, Eleonora Cagli, and Cecile Dumas. Study of deep learning techniques for side-channel analysis and introduction to ascad database. Cryptology ePrint Archive, Report 2018/053, 2018. https://eprint.iacr.org/2018/053.

[8] François-Xavier Standaert and Cédric Archambeau. Using subspace-based template attacks to compare and combine power and electromagnetic information leakages. In *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th*

International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings, volume 5154 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2008.