# Technical analysis of the masked AES implementation

LSC, ANSSI

January 6, 2021

## Contents

## 1 Introduction

This document describes the side-channel characterization of a secure AES encryption implementation. We perform this study on the ChipWhisperer target based on a STM32F303RCT7 chip (using a Cortex-M4 core). Using the ChipWhisperer as a board for side-channel characterization is explained by an easy access to power consumption and clean signal acquisition chain. Furthermore, this board natively supports the *undercloking* of the Cortex-M4 target using an external oscillator: slowing down the core frequency to 4 MHz allows to use reasonable sampling rates during the acquisition.

This evaluation was performed on power consumption traces captured through an oscilloscope, sampling 100.000.000 samples by second. The obtained traces consist in 2.000.000 samples, encompassing the whole AES implementation. Figure 1 shows two main steps: the first one is the pre-processing, and the second one is the execution of the raw AES rounds. More details are given in Section 2.

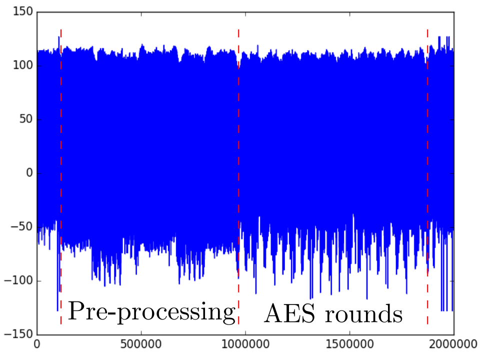

Figure 1: Power consumption trace of the AES encryption.

No resynchronization step has been performed on these traces. Nonetheless, visual inspection indicates that no significant desynchronization occurs. To illustrate this, Figure 2 displays a temporal zoom on three different traces.

A first acquisition campaign of 50.000 traces of the rolled version (*ie.*, the default version) is used to perform the caracterisation step as well as the first-order resilience assessment. For the second and third order assessments, we used a second campaign of 100.000 traces of the unrolled version. The following table summarizes the results. The analyses are further detailed in the rest of this document.

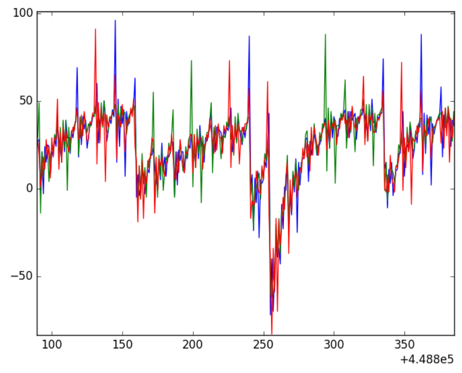

Figure 2: Three traces on a short temporal window.

| | Known permutation | | | Unknown permutation | | |
|---|---|---|---|---|---|---|
| Known | $\mathsf{r_m}, \mathsf{r_{out}}$ | $\mathsf{r_m}$ | None | $\mathsf{r_m}, \mathsf{r_{out}}$ | $\mathsf{r_m}$ | None |
| Order 1 | $\approx 1.000$ | $> 50.000$ | $> 50.000$ | $\approx 20.000$ | $> 50.000$ | $> 50.000$ |
| Order 2 | N/A | $> 8.000$ | $> 100.000$ | N/A | $> 100.000$ | $> 100.000$ |
| Order 3 | N/A | N/A | $> 100.000$ | N/A | N/A | $> 100.000$ |

Figure 3: Number of traces needed for a successful first-order, second-order, and thirs-order attack, depending on the masks known to the attacker.

## 2 Secured AES: Affine masking

The version of the AES is an affine implementation according to the paper *Affine masking against high-order side channel analysis* [**?**]. Each byte state of the AES state, denoted $\mathsf{state}[i]$ for the $i$-th byte, is manipulated under the form $\mathsf{r_m} \times \mathsf{state}[i] \oplus \mathsf{state_M}[i]$: $\mathsf{r_m}$ is a non zero random byte, $\mathsf{state_M}[i]$ is initialized with 16-random bytes denoted $\mathsf{M}[0] \dots \mathsf{M}[15]$, and $\times$ denotes the multiplication over the AES finite field [**?**]. For each round, the AES operations are applied to both $\mathsf{state}$ and $\mathsf{state_M}$.

Similarly, the key schedule operation is also protected by manipulating each byte of the key state, denoted $\mathsf{state_K}[i]$, under an affine form $\mathsf{r_m}' \times \mathsf{state_K}[i] \oplus \mathsf{state}'_\mathsf{M}[i]$ where $\mathsf{state}'_\mathsf{M}$ is initialize with 16-random bytes denoted $\mathsf{M}'[0] \dots \mathsf{M}'[15]$ independent from the aes random bytes $\mathsf{M}[0] \dots \mathsf{M}[15]$. As the key schedule is performed independently from encryption and decryption operations, before using each masked key byte, it has to be changed into $\mathsf{r_m} \times \mathsf{state_K}[i] \oplus \mathsf{state}'_\mathsf{M}[i]$ (same multiplicative byte $\mathsf{r_m}$) to have a consistent computation.

To sum up, the AES encryption/decryption and the key schedule each use 19 random bytes as input:

- 16 bytes for Boolean masking $\mathsf{M}[0] \dots \mathsf{M}[15]/\mathsf{M}'[0] \dots \mathsf{M}'[15]$ that initialized the masking state $\mathsf{state_M}/\mathsf{state_K}$;

- 1 multiplicative byte $\mathsf{r_m}/\mathsf{r_m}'$;
- 2 bytes $\mathsf{r_{in}}$, $\mathsf{r_{out}}$ / $\mathsf{r_{in}}'$, $\mathsf{r_{out}}'$ (two boolean masking bytes used for SubByte operation).

To make this report more readable, we provide the correspondance Table 1 between the notations in this report ("Here") and in the "assembly code".

| Assembly code | Here | size |
|---|---|---|
| `key_rin[0]` | $\mathsf{r_{in}}'$ | 1 |
| `key_rout[0]` | $\mathsf{r_{out}}'$ | 1 |
| `key_rmult[0]` | $\mathsf{r_m}'$ | 1 |
| `key_maskedState[i]` | $\mathsf{r_m} \times \mathsf{state_K}[i] \oplus \mathsf{state'_M}[i]$ | 1 |
| `key_masksState[i]` | $\mathsf{state'_M}[i]$ | 1 |
| `key_gtab` | $\mathsf{GTab}$ | $16 \times 16$ |
| `key_permIndicesMC` | $\mathsf{permIndicesMC}$ | 4 |
| `aes_state[i]` | $\mathsf{r_m} \times \mathsf{state}[i] \oplus \mathsf{state_M}[i]$ | 1 |
| `aes_state2[i]` | $\mathsf{state_M}[i]$ | 1 |
| `aes_rin[0]` | $\mathsf{r_{in}}$ | 1 |
| `aes_rout[0]` | $\mathsf{r_{out}}$ | 1 |
| `aes_rmult[0]` | $\mathsf{r_m}$ | 1 |
| `aes_gtab` | $\mathsf{GTab}$ | $16 \times 16$ |
| `aes_permIndices` | $\mathsf{permIndices}$ | 16 |
| `aes_permIndicesBis` | $\mathsf{permIndicesBis}$ | 16 |
| `aes_permIndicesMC` | $\mathsf{permIndicesMC}$ | 4 |
| `aes_permIndicesBisMC` | $\mathsf{permIndicesMCbis}$ | 4 |
| `aes_sboxMasked` | $\mathsf{Sbox_m}$ | $16 \times 16$ |

Table 1: Notations correspondance between this report and the assembly code. Sizes are in number of bytes.

We can distinguish three main steps:

1 Pre-processing: before the AES rounds, the following operations in this order are performed : loading of inputs, computation of the table $\mathsf{GTab}$ (multiplication by $\mathsf{r_m}$), computation of the affine sbox denoted $\mathsf{Sbox_m}$, key schedule with affine masking.

2 AES rounds: 10 AES rounds with affine masking of the state and subkeys.

3 Post-processing: after AES rounds, the state $\mathsf{state}$ is unmasked by removing the Boolean mask $\mathsf{state_M}$ and by removing the multiplicative factor $\mathsf{r_m}$ (multiplying by $\mathsf{r_m}^{-1}$).

Shuffling: Many sub-operations on the AES state are performed in a random order: each byte is processed in a random order with pre-computed permutation based on the input random values. For AES rounds: $\mathsf{permIndicesMC}$ is used to shuffle MixColumns of $\mathsf{state}$ and $\mathsf{permIndicesMCbis}$ to shuffle MixColumns of

$\mathsf{state_M}$. Permutation $\mathsf{permIndicesMC}$ (resp. $\mathsf{permIndicesMCbis}$) is computed from the initial value $\mathsf{state_M}[0] = \mathsf{M}[0]$ (resp. $\mathsf{state_M}[1] = \mathsf{M}[1]$). The permutation $\mathsf{permIndices}$ is used to shuffle SubBytes and ShiftRows of $\mathsf{state}$ and the permutation $\mathsf{permIndicesBis}$ is used to shuffle SubBytes and ShiftRows of $\mathsf{state_M}$. These permutations are computed from $\mathsf{M}[0], \mathsf{M}[1], \mathsf{M}[2], \mathsf{M}[3]$.

For key schedule: the four permutations are similar. The permutation $\mathsf{permIndicesMC}$ is used to shuffle the manipulation of the words' coordinates of the masked key state (the term word referring here to 4-bytes vector) and $\mathsf{permIndicesMCbis}$ is used to shuffle the manipulation of the words' coordinates of $\mathsf{state'_M}$. The former permutation is computed from $\mathsf{M'}[0]$, while the latter one is computed from $\mathsf{M'}[1]$. The permutation $\mathsf{permIndices}$ is used to shuffle SubBytes and ShiftRows of the masked key state and $\mathsf{permIndicesBis}$ is used to shuffle SubBytes and ShiftRows of the key mask state $\mathsf{state'_M}$. These permutations are computed from $\mathsf{M'}[0], \mathsf{M'}[1], \mathsf{M'}[2], \mathsf{M'}[3]$.

## 3 Characterization phase

In this section, we perform several Signal-to-Noise Ratio (SNR) computations in order to identify univariate leakage samples related to the manipulation of different sensitive values.

### 3.1 Key bytes

We perform a SNR targeting each key byte. We superpose the leakage and SNR curves for the first key byte in Figure 4 to locate where the SNR peaks are located during AES computation. For each key byte, the results evidence one leakage point, which highlights the key manipulation during its masking (`Load_masterKey` function).

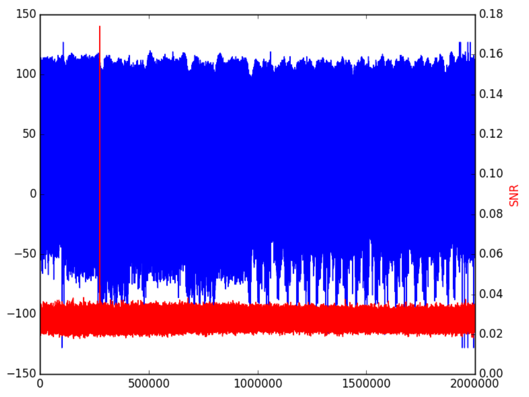

Figure 4: SNR targeting key byte 0 (in red), superposed to one leakage trace. SNR computed using 10.000 traces.

## 3.2 Plaintext bytes

Similarly, we perform a SNR targeting each plaintext byte as shown in Figure 5. The results evidence three leakage points. The first leakage is explained by the loading of the plaintext in the target. The second leakage is explained by the loading of the plaintext in the context (`Load_data` function). The third leakage is explained by the masking of the plaintext (`Map_in_G` function).

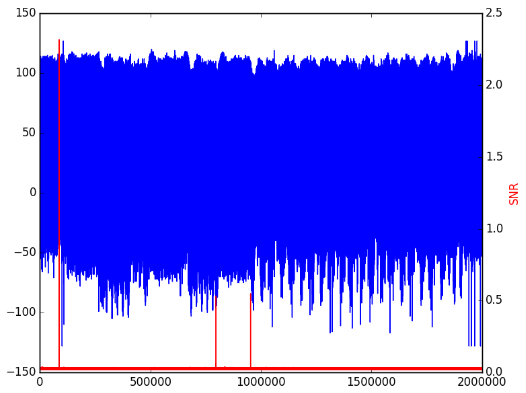

Figure 5: SNR targeting plaintext byte 0 (in red), superposed to one leakage trace. SNR computed using 10.000 traces.

## 3.3 Ciphertext bytes

Similarly, we perform a SNR targeting ciphertext bytes as shown in Figure 6. Leakages appear during the unmasking operation (`Multiplicative_unmasking` function) and the serial writing of the bytes: when the ciphertext is loaded in the output variable.

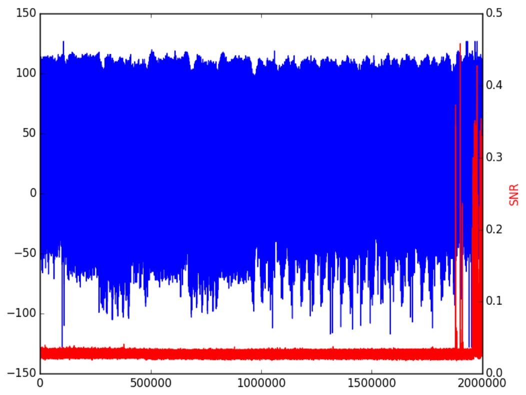

Figure 6: SNR targeting ciphertext byte 0 (in red), superposed to one leakage trace. SNR computed using 10.000 traces.

### 3.4 Mask bytes

Similarly, we perform a SNR targeting each mask byte. We detail the results hereafter, depending on the usages of each mask in the implementation.

#### 3.4.1 State masks

Bytes $\mathsf{M}[0]$ to $\mathsf{M}[15]$, $\mathsf{r}_{\mathsf{in}}$, $\mathsf{r}_{\mathsf{out}}$ and $\mathsf{r}_{\mathsf{m}}$ are used for the masking of the state of the encryption. We characterize hereafter their leakages.

Figure 7 and Figure 8 illustrate the results when targeting bytes $\mathsf{M}[0]$ and $\mathsf{M}[1]$. These bytes are used for computing both the shuffling permutation used in the substitution layer of the AES, and the permutations used in the linear layer. They are also used as linear masks. Leakages appear during the loading of the bytes (`Load_random` function). Then, leakages appear during the computation of the shuffling permutation of the substitution layer, `Compute_permIndices_Tables`, and the computation of the permutations in the linear layer (`Compute_permIndices_Tables_MC` functions). Another leakage peak appears when the masks are loaded just before the encryption (`Load_data` function), and a smaller peak appears when the state is masked (`Xor_states` function). Finally, we observe nine peaks corresponding to the 9 `MixColumns` operations.

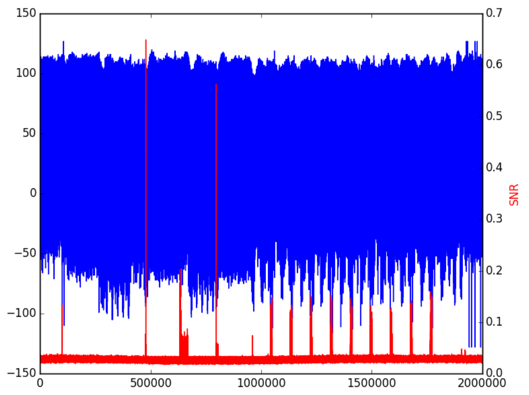

Figure 7: SNR targeting mask byte M[0] (in red), superposed to one leakage trace. SNR computed using 10.000 traces.

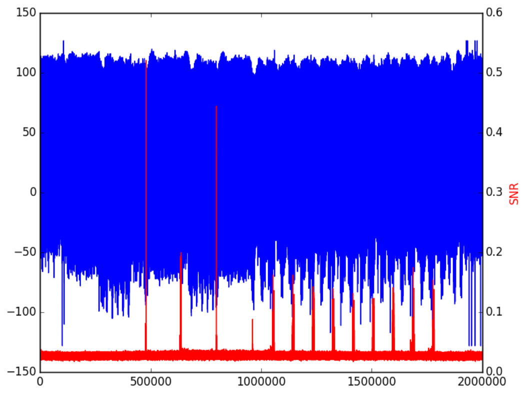

Figure 8: SNR targeting mask byte M[1] (in red), superposed to one leakage trace. SNR computed using 10.000 traces.

Figures 9 and 10 illustrate the results when targeting bytes M[2] and M[3]. These bytes are used for computing the shuffling permutation used in the substitution layer, and as linear masks. The leakages are similar to those of bytes M[0] and M[1], except that no leakage is observed during the `MixColumns` operations, nor during the computation of the permutations used in the linear layer.

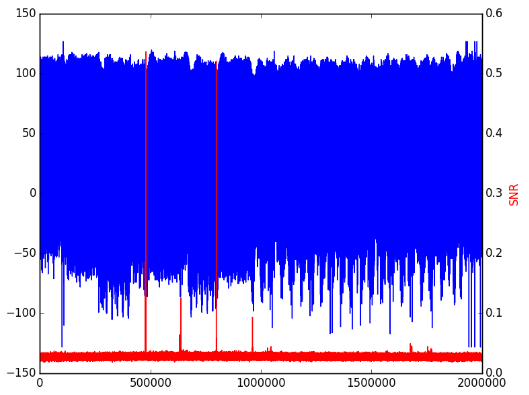

Figure 9: SNR targeting mask byte M[2] (in red), superposed to one leakage trace. SNR computed using 10.000 traces.

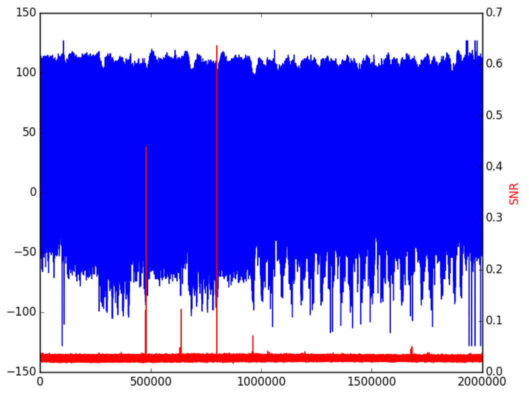

Figure 10: SNR targeting mask byte M[3] (in red), superposed to one leakage trace. SNR computed using 10.000 traces.

Figures 11 and 12 illustrate the results when targeting bytes M[4] and M[15]. These bytes are used as linear masks. Similar results are obtained for bytes M[5] to M[14]. The leakages are similar to the one obtained for previous bytes, but are now only observed for the loadings and masking of the state.

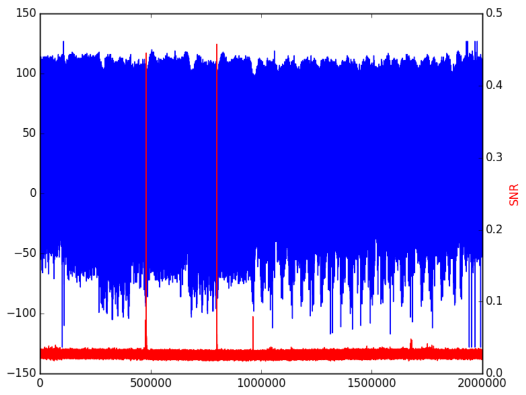

Figure 11: SNR targeting mask byte M[4] (in red), superposed to one leakage trace. SNR computed using 10.000 traces.

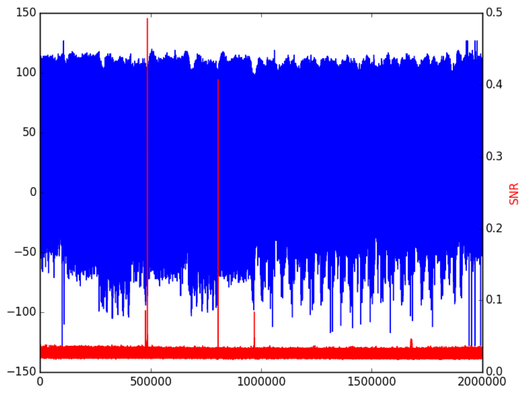

Figure 12: SNR targeting mask byte M[15] (in red), superposed to one leakage trace. SNR computed using 10.000 traces.

Figure 13 illustrates the results when targeting $\mathsf{r}_{\mathsf{in}}$. This byte is used as the input mask of the substition layer. Leakage first appears during the loading of the randoms (`Load_data`). Strong leakage can then be observed during the re-computation of the substitution table (`Compute_Affine_sboxMasked` function). Finally 10 peaks appear during the substitution steps of each round of the AES.

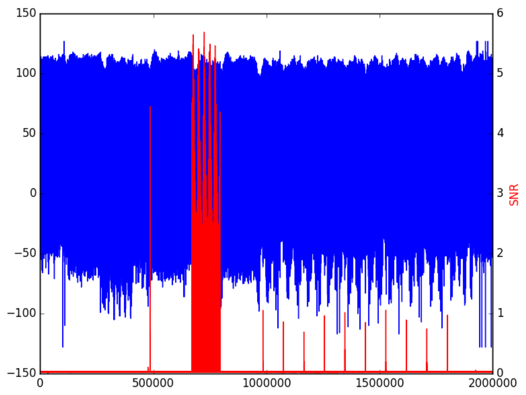

Figure 13: SNR targeting mask byte $\mathsf{r_{in}}$ (in red), superposed to one leakage trace. SNR computed using 10.000 traces.

Figure 14 illustrates the results when targeting $\mathsf{r_{out}}$. This byte is used as the output mask of the subtitution layer. Leakages appear at similar time samples as for $\mathsf{r_{in}}$.

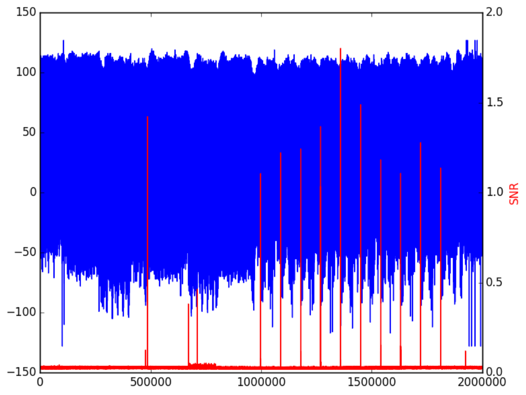

Figure 14: SNR targeting mask byte $\mathsf{r_{out}}$ (in red), superposed to one leakage trace. SNR computed using 10.000 traces.

Figure 15 illustrates the results when targeting $\mathsf{r_m}$. This byte is used as the multiplicative mask of the substitution layer. Leakage once again appear when the randoms are loaded (`Load_data` function). Leakage also appear during the computation of the multiplicative table (`Compute_GTab` function), and during the computation of the substitution box (`Compute_Affine_sboxMasked` function).

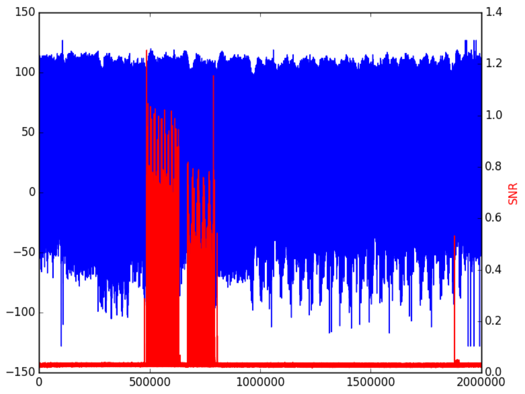

Figure 15: SNR targeting mask byte $\mathsf{r_m}$ (in red), superposed to one leakage trace. SNR computed using 10.000 traces.

### 3.4.2 Key masks

Bytes $\mathsf{M}'[0]$ to $\mathsf{M}'[15]$, $\mathsf{r_{in}}'$, $\mathsf{r_{out}}'$ and $\mathsf{r_m}'$ are used for the masking of the key. We characterize hereafter their leakages. Figure 16 illustrates the results when targeting byte $\mathsf{M}'[0]$. These bytes are used for computing the shuffling permutation used in the key expansion. They are also used as linear masks.

Observed peaks correspond to the loading of the random in the target, loading of the masks (`Load_random_key` function), masking of the key (`Load_masterKey`), and the key expansion (`KeyExpansion_masked` function), due to the involvment of the mask byte in the computation of the shuffling permutation (`Compute_permIndices_Tables_MC` function).

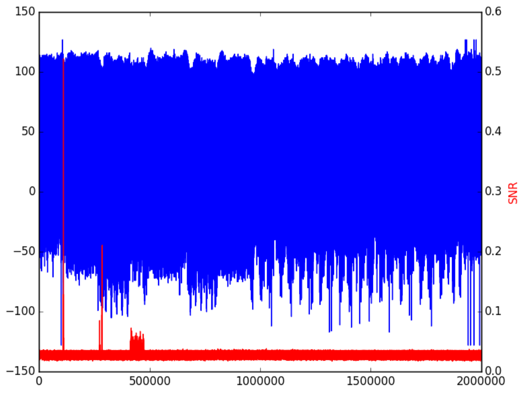

Figure 16: SNR targeting mask byte $\mathsf{M}'[0]$ (in red), superposed to one leakage trace. SNR computed using 10.000 traces.

Figures 17 and 18 illustrate the results when targeting bytes $\mathsf{M}'[1]$ and $\mathsf{M}'[2]$.

Similar results are observed when targeting bytes $\mathsf{M}'[3]$ to $\mathsf{M}'[15]$. These bytes are not used for the computation of the shuffling permutation. However, they serve as linear masks during the key expansion, and, as such, a small peak appears at the beginning of the `KeyExpansion_masked` function. The other peaks are similar to the ones appearing for byte $\mathsf{M}'[0]$.

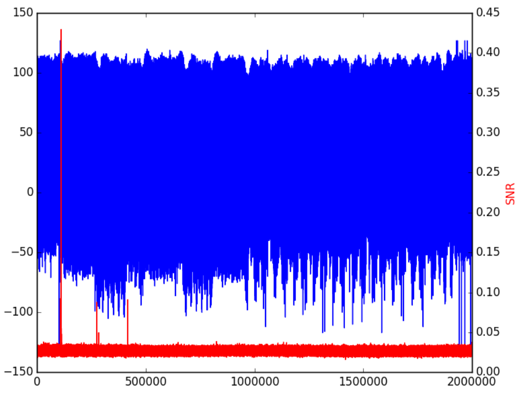

Figure 17: SNR targeting mask byte $\mathsf{M}'[1]$ (in red), superposed to one leakage trace. SNR computed using 10.000 traces.

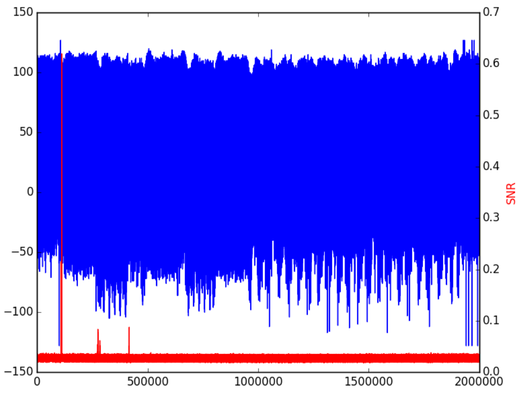

Figure 18: SNR targeting mask byte $\mathsf{M}'[2]$ (in red), superposed to one leakage trace. SNR computed using 10.000 traces.

Figure 19 illustrates the results when targeting $\mathsf{r_{in}}'$. This byte is used as the input mask of the substition layer. Leakages appear at the same time samples as for bytes $\mathsf{M}'[1]$ to $\mathsf{M}'[15]$, and also during the recomputation of the Sbox and the key expansion.

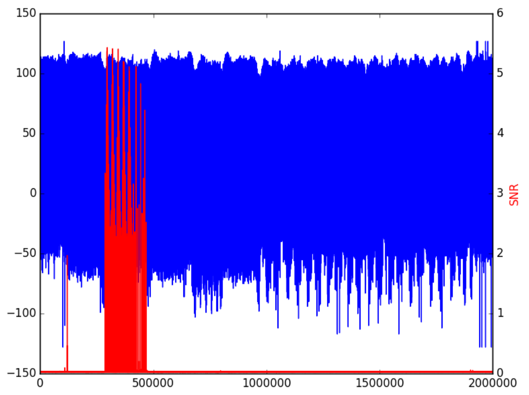

Figure 19: SNR targeting mask byte $\mathsf{r_{in}}'$ (in red), superposed to one leakage trace. SNR computed using 10.000 traces.

Figure 20 illustrates the results when targeting $\mathsf{r_{out}}'$. This byte is used as the output mask of the subtitution layer. Leakages appear at the same time samples as for bytes $\mathsf{M}'[1]$ to $\mathsf{M}'[15]$, and also during the recomputation of the Sbox and the key expansion.

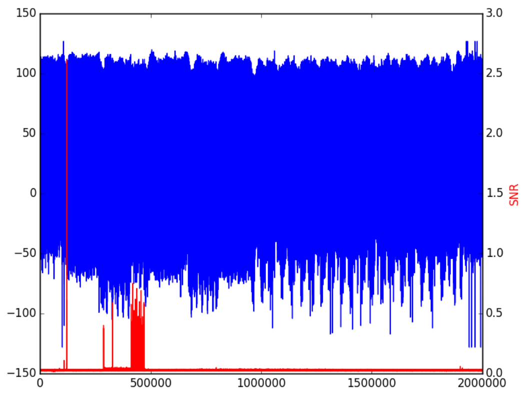

Figure 20: SNR targeting mask byte $\mathsf{r_{out}}'$ (in red), superposed to one leakage trace. SNR computed using 10.000 traces.

Figure 21 illustrates the results when targeting byte $\mathsf{r_m}'$. This byte is used as the multiplicative mask of the substitution layer. Leakages appear at the same time samples as the previous bytes, and also during the computation of the multiplication table `Gtab`. Finally, a peak appears just before the encryption, when the value is loaded.

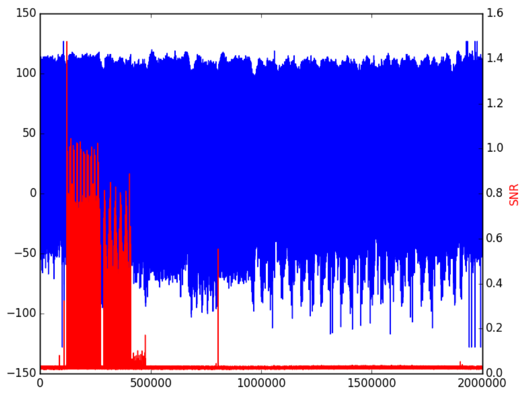

Figure 21: SNR targeting mask byte $\mathsf{r_m}'$ (in red), superposed to one leakage trace. SNR computed using 10.000 traces.

## 3.5 Sbox output bytes

### 3.5.1 Raw bytes

Figure 22 illustrates the results when targeting a Sbox output byte without taking into account the output mask (ie. for each byte $i$, the value $S[\mathsf{P}[i] \oplus \mathsf{K}[i]]$ is targeted) and there is no observable leakage.

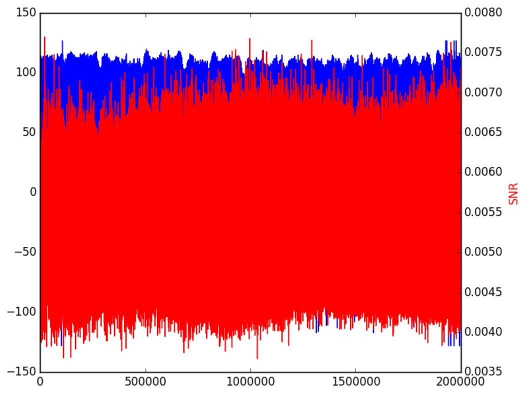

Figure 22: SNR targeting Sbox output byte 0 (in red), superposed to one leakage trace. SNR computed using 50.000 traces.

### 3.5.2 Multiplied by mask

Figure 23 illustrates the results when targeting a byte of the Sbox output, multiplied by the multiplicative mask, ie., for each byte $i$, the value $\mathsf{r_m} \times S[\mathsf{P}[i] \oplus \mathsf{K}[i]]$ is targeted. No significant leakage is observed.

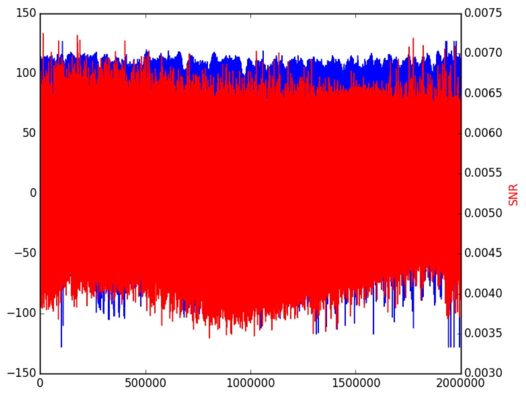

Figure 23: SNR targeting the multiplicatively masked Sbox output byte 0 (in red), superposed to one leakage trace. SNR computed using 50.000 traces.

### 3.5.3 Affine masked

Figure 24 illustrates the results when targeting a byte of the Sbox output, affinely masked, ie., for each byte $i$, the value $\mathsf{r_m} \times S[P[i] \oplus K[i]] \oplus \mathsf{r_{out}}$ is targeted. We remark that 50.000 traces are not enough to highlight a leakage of this value. This result is expected because of the shuffling countermeasure.

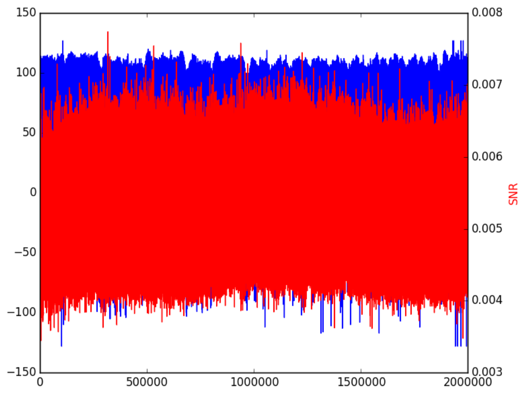

Figure 24: SNR targeting the affinely masked Sbox output byte 0 (in red), superposed to one leakage trace. SNR computed using 50.000 traces.

## 3.6 Unshuffled output bytes

### 3.6.1 Raw bytes

We use the knowledge of the mask bytes to inverse the shuffling $Sh$ performed during the substitution phase. For each index $i$, we target the value $S[\mathsf{P}[Sh[i]] \oplus$

$\mathsf{K}[Sh[i]]]$. No significant leakage can be observed. Figure 25 illustrates the results.

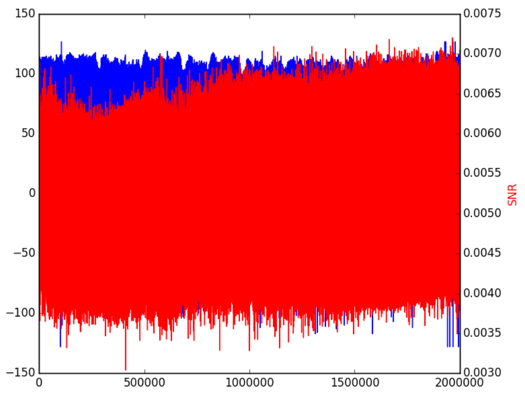

Figure 25: SNR targeting the unshuffled Sbox output byte 0 (in red), superposed to one leakage trace. SNR computed using 50.000 traces.

### 3.6.2 Multiplicately masked

We use the knowledge of the mask bytes to inverse the shuffling $Sh$ performed during the substitution phase. For each index $i$, we target the value $\mathsf{r_m} \times S[\mathsf{P}[Sh(i)] \oplus \mathsf{K}[Sh(i)]]$. A small amount of leakage is present, at the time of the manipulation of $\mathsf{r_m}$. This is easily explained since the random variables $\mathsf{r_m} \times S[\mathsf{P}[Sh(i)] \oplus \mathsf{K}[Sh(i)]]$ and $\mathsf{r_m}$ are not independent. Figure 26 illustrates the results.

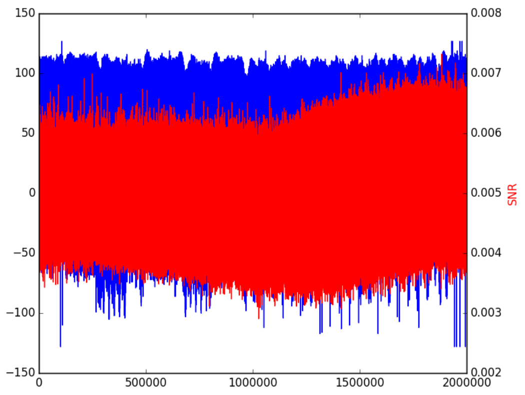

Figure 26: SNR targeting the unshuffled multiplicatively masked Sbox output byte 0 (in red), superposed to one leakage trace. SNR computed using 50.000 traces.

### 3.6.3 Affinely masked

We use the knowledge of the mask bytes to inverse the shuffling $Sh$ performed during the substitution phase. For each index $i$, we target the value $\mathsf{r_m} \times S[\mathsf{P}[Sh(i)] \oplus \mathsf{K}[Sh(i)]] \oplus \mathsf{r_{out}}$. A significant leakage is observed after the first substitution layer. Figure 27 illustrates the results.

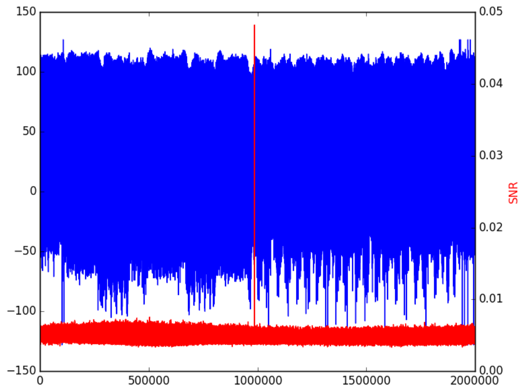

Figure 27: SNR targeting the unshuffled affinely masked Sbox output byte 0 (in red), superposed to one leakage trace. SNR computed using 50.000 traces.

## 4 First-order correlation analysis

In this section, we analyse the effectiveness of the implemented countermeasures against first-order side-channel analysis based on the linear correlation coefficient (Correlation Power Analysis CPA). For this study, we will target the Sbox output value.

We separate this study in two parts. In the first part, we will characterize the effectiveness of a CPA against this implementation, by considering an attacker with priviledged knowledge. In the second part, we will evaluate the effectiveness of a CPA against this implementation without such knowledge.

### 4.1 Priviledged knowledge

We first study the setting where the attacker knows the random masks used to compute the permutation.

In order to characterize the effectiveness of a CPA against this implemenation, we use the random masks to recompute the permutation $Sh(i)$ for each byte index $i$. We then perform an attack targeting the value
$Z_{\hat{\mathsf{K}}}[i] = S[\mathsf{P}[Sh(i)] \oplus \mathsf{K}[Sh(i)] \oplus \hat{\mathsf{K}}]$, where $\hat{\mathsf{K}}$ is an hypothesis on a value of one byte. The attack is successful if the best hypothesis returned by the attack is 0.

#### 4.1.1 Affinely masked Sbox output

We suppose the knowledge of both the multiplicative mask and the output mask. This knowledge allows for the prediction of the value $\mathsf{r_m} \times Z_{\hat{\mathsf{K}}}[i] \oplus \mathsf{r_{out}}$ for any hypothesis $\mathsf{K}$. Depending on the index byte, the attack succeeds with around 1.000 traces. Figure 28 illustrates the result of the attack targeting byte 0, using 1.000 traces.

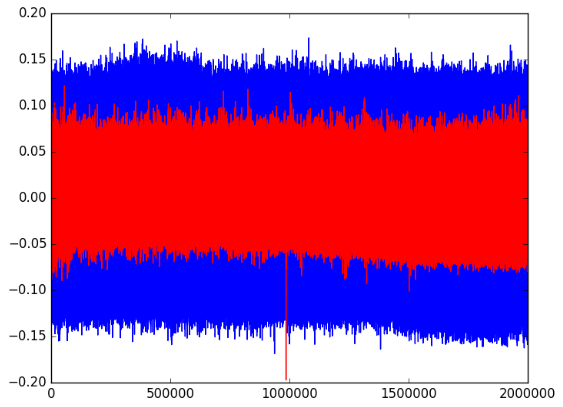

Figure 28: Correlation coefficients obtained when targeting $\mathsf{r_m} \times Z_{\hat{\mathsf{K}}}[i] \oplus \mathsf{r_{out}}$, for every value of $\hat{\mathsf{K}}$. Correct hypothesis is plotted in red.

#### 4.1.2 Multiplicatively masked Sbox output

We suppose the knowledge of the multiplicative mask. This knowledge allows for the prediction of the value $\mathsf{r_m} \times Z_{\hat{\mathsf{K}}}[i]$ for any hypothesis $\hat{\mathsf{K}}$. The attack is unsuccessful using 50.000 traces. Figure 29 illustrates the result of the attack targeting byte 0, using 50.000 traces.

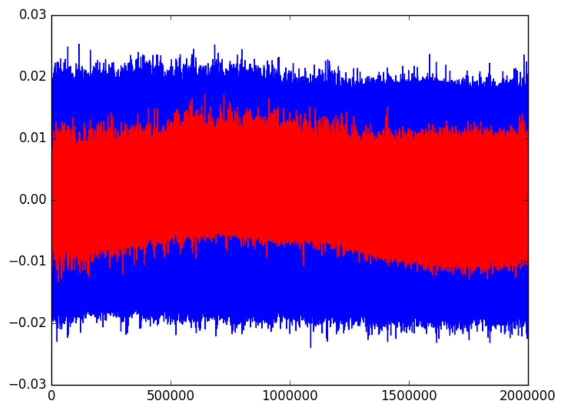

Figure 29: Correlation coefficients obtained when targeting $\mathsf{r_m} \times Z_{\hat{\mathsf{K}}}[i]$, for every value of $\hat{\mathsf{K}}$. Correct hypothesis is plotted in red.

### 4.1.3 Raw Sbox output

We suppose no prior knowledge (except the permutation), and predict the value $Z_{\hat{\mathsf{K}}}[i]$ for any hypothesis $\hat{\mathsf{K}}$. The attack is unsuccessful using 50.000 traces. Figure 30 illustrates the result of the attack targeting byte 0, using 50.000 traces.

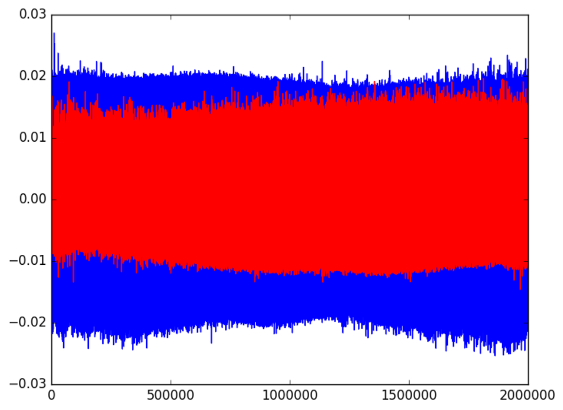

Figure 30: Correlation coefficients obtained when targeting $\mathsf{r_m} \times Z_{\hat{\mathsf{K}}}[i]$, for every value of $\hat{k}$. Correct hypothesis is plotted in red.

## 4.2 Unknown permutation, processed traces

We now study the setting where the attacker does not know the random masks used to compute the permutation.

We preprocess the traces in order to average the leakage over the different byte indices manipulation:

- for each index $i$ in $[0, 15]$, we define a small window $w_i$ of $\ell$ points around the SNR peak corresponding to the manipulation of $\mathsf{r_m} \times S[P[Sh(i)] \oplus K[Sh(i)]] \oplus \mathsf{r_{out}}$ in the characterization phase. In our experiments, the size of the window was arbitrarily fixed to $\ell = 11$.

- for each trace in our acquisition campaign, we compute the average window $m$ such that for each time sample $j$, $m[j] = \frac{1}{16} \sum_{i=0}^{15} w_i[j]$. We then consider $m$ as our reduced averaged trace of size $\ell$.

### 4.2.1 Affinely masked Sbox output

We suppose the knowledge of both the multiplicative mask and the output mask. This knowledge allows for the prediction of the value $\mathsf{r_m} \times S[\mathsf{P}[i] \oplus \hat{\mathsf{K}}] \oplus \mathsf{r_{out}}$ for any hypothesis $\hat{\mathsf{K}}$ on one byte of the secret key. Depending on the byte index, the CPA is successful using around 20.000 to 30.000 traces.

Figure 31 illustrates the results using 50.000 traces.

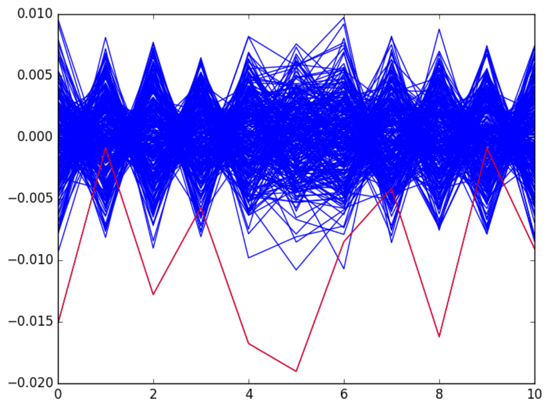

Figure 31: Correlation coefficients obtained when targeting $\mathsf{r_m} \times S[\mathsf{P}[i] \oplus \hat{\mathsf{K}}] \oplus \mathsf{r_{out}}$, on averaged traces, for every value of $\hat{\mathsf{K}}$. Correct hypothesis is plotted in red.

### 4.2.2 Multiplicatively masked Sbox output

With 50.000 traces, the attack does not succeed when targeting $\mathsf{r_m} \times S[\mathsf{P}[i] \oplus \hat{\mathsf{K}}]$. Figure 32 illustrates the results using 50.000 traces.

Figure 32 illustrates the results using 50.000 traces.

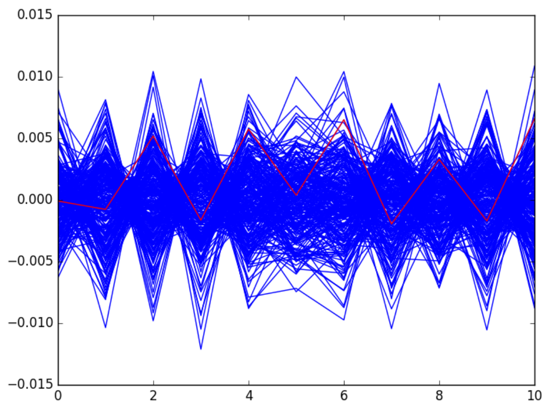

Figure 32: Correlation coefficients obtained when targeting $\mathsf{r_m} \times S[\mathsf{P}[i] \oplus \hat{\mathsf{K}}] \oplus \mathsf{r_{out}}$, on averaged traces, for every value of $\hat{\mathsf{K}}$. Correct hypothesis is plotted in red.

### 4.2.3 Raw Sbox output

With 50.000 traces, the attack does not succeed when targeting $S[\mathsf{P}[i] \oplus \hat{\mathsf{K}}]$. Figure 33 illustrates the results using 50.000 traces. Figure 33 illustrates the results using 50.000 traces.

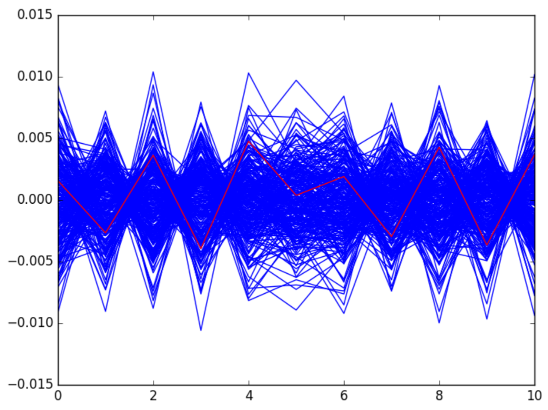

Figure 33: Correlation coefficients obtained when targeting $\mathsf{r_m} \times S[\mathsf{P}[i] \oplus \hat{\mathsf{K}}] \oplus \mathsf{r_{out}}$, on averaged traces, for every value of $\hat{\mathsf{K}}$. Correct hypothesis is plotted in red.

## 4.3 Unknown permutation, non-processed traces

For the sake of completeness, we also perform these experiments on raw unprocessed traces. No success is obtained when targeting $Z_{\hat{\mathsf{K}}}[i]$ or $\mathsf{r_m} \times Z_{\hat{\mathsf{K}}}[i]$ using 50.000 traces. Similarly, by targeting $\mathsf{r_m} \times S[\mathsf{P}[i] \oplus \hat{\mathsf{K}}] \oplus \mathsf{r_{out}}$, the attack does not succeed.

## 4.4 Summary

The results in this section evidence the effectiveness of the implementation of the countermeasures against first-order side-channel attack. Assuming knowledge of the mask values, the shuffling countermeasure alone increases the number of necessary traces by a factor $\sim 20$. Furthermore, masking countermeasures seem to completely thwart first order side-channel attacks using 50.000 traces.

| | Known permutation | | | Unknown permutation | | |
|---|---|---|---|---|---|---|
| Known | $\mathsf{r_m}, \mathsf{r_{out}}$ | $\mathsf{r_m}$ | None | $\mathsf{r_m}, \mathsf{r_{out}}$ | $\mathsf{r_m}$ | None |
| # Traces | $\approx 1.000$ | $> 50.000$ | $> 50.000$ | $\approx 20.000$ | $> 50.000$ | $> 50.000$ |

Figure 34: Number of traces needed for a successful first-order attack, in the different settings.

# 5 Second-order correlation analysis

In this section, we analyse the effectiveness of the implemented countermeasures against second-order side-channel analysis based on the linear correlation coefficient (CPA). For this study, we will target the Sbox output value and consider a Hamming weight ($HW$) univariate leakage model.

In the remainder of this section, we will use:

- the centered product combination function $\mathcal{C}(x,y) = (x - \bar{x})(y - \bar{y})$ to preprocess the traces, where $\bar{x}$ (resp. $\bar{y}$) stands for the mean value of $x$ (resp. $y$), computed on all traces in our campaign;
- the corresponding function $f_{HW}(z) = \sum_m (HW(z \oplus m) - \overline{HW(z \oplus m)})$ $(HW(m) - \overline{HW(m)})$ to compute the predictions, where $\overline{HW(m)}$ (resp $\overline{HW(z \oplus m)}$) stands for the mean value of $HW(m)$ (resp. $HW(z \oplus m)$) computed on all possible values of $m$[1].

As in our first-order correlation analysis, we will split this study in two parts. The first part will consider priviledged knowledge on the mask values. The second part will not take this knowledge into account.

[1]Note that all possible values for $m$ are in $[0, 255]$.

## 5.1 Priviledged knowledge

We first study the setting where the attacker knows the random masks used to compute the permutation. In order to characterize the effectiveness of a CPA against this implementation, we use the random masks to recompute the permutation $Sh(i)$ for each byte index $i$.

For each byte index $i$, we use the characterization phase to isolate a window $w_{z\oplus m}$ of 11 points of interest corresponding to the manipulation of the Sbox output $\mathsf{r_m} \times S[P[Sh(i) \oplus K[Sh(i)]] \oplus \mathsf{r_{out}}$, and a window $w_m$ of 11 points of interest corresponding to the manipulation of the mask value $\mathsf{r_{out}}$. For each couple $(i, j)$ of points in $w_{z\oplus m} \times w_m$, we apply the combination function $\mathcal{C}(i, j)$. We then perform an attack targeting the value $Z_{\hat{k}}[i] = S[P[Sh(i)] \oplus K[Sh(i)] \oplus \hat{k}]$, where $\hat{k}$ is an hypothesis on a value of one byte.

### 5.1.1 Multiplicatively masked Sbox output

We suppose the knowledge of the multiplicative mask $\mathsf{r_m}$. This knowledge allows for the prediction of the value $\mathsf{r_m} \times Z_{\hat{k}}[i]$ for any hypothesis $\hat{k}$. The attack succeeds with around 8.000 traces to 20.000 traces. Figure 35 illustrates the results using 20.000 traces.

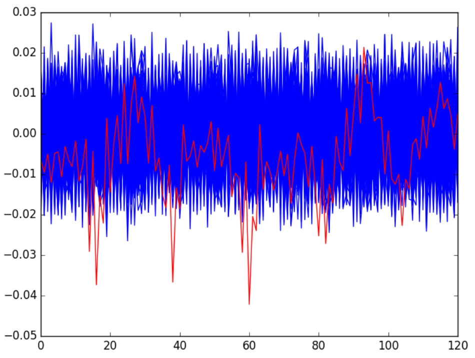

Figure 35: Correlation coefficients obtained when targeting $\mathsf{r_m} \times S[P[i] \oplus \hat{k}]$, for every value of $\hat{k}$. Correct hypothesis plotted in red. X-axis represents all 121 points combinations.

### 5.1.2 Raw Sbox output

We target the value $Z_{\hat{k}}[i]$ for any hypothesis $k$. The attack does not succeed using the 100.000 traces. Figure 36 illustrates the results using 100.000 traces.

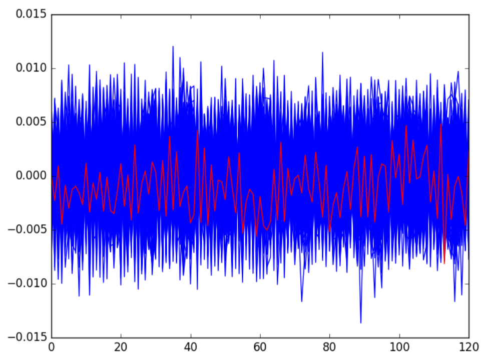

Figure 36: Correlation coefficients obtained when targeting $S[P[i] \oplus \hat{k}]$, for every value of $\hat{k}$. Correct hypothesis plotted in red. X-axis represents all 121 points combinations.

## 5.2 Unknown permutation, processed traces

We now study the setting where the attacker does not know the random masks used to compute the permutation. We preprocess the traces in order to average the leakage over the different byte indices manipulation:

- for each index $i$ in $[0, 15]$, we define a small window $w_i$ of $\ell$ points around the SNR peak corresponding to the manipulation of $\mathsf{r_m} \times S[P[Sh(i)] \oplus K[Sh(i)]] \oplus \mathsf{r_{out}}$ in the characterization phase. In our experiments, the size of the window was arbitrarily fixed to $\ell = 11$.
- for each trace in our acquisition campaign, we compute the average window $m$ such that for each time sample $j$, $m[j] = \frac{1}{16} \sum_{i=0}^{15} w_i[j]$. We then consider $m$ as our reduced averaged trace of size $\ell$.
- we concatenate to this trace a small window of $\ell$ points around the SNR peak corresponding to the manipulation of $\mathsf{r_{out}}$.

### 5.2.1 Multiplicatively masked Sbox output

We suppose the knowledge of the multiplicative mask $\mathsf{r_m}$. This knowledge allows for the prediction of the value $\mathsf{r_m} \times Z_{\hat{k}}[i]$ for any hypothesis $\hat{k}$. The attack is unsuccessful using 100.000 traces.

Figure 37 illustrates the results using 100.000 traces.

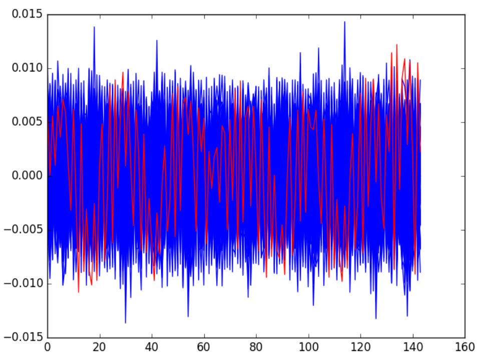

Figure 37: Correlation coefficients obtained when targeting $\mathsf{r_m} \times S[P[i] \oplus \hat{k}]$, for every value of $\hat{k}$. Correct hypothesis plotted in red. X-axis represents all 121 points combinations.

However, we observe that, for certain combinations of points (around abscissa 135), the correct hypothesis is indeed the most likely. Furthermore, the rank convergence of the key, plotted on Figure 38, might indicate that the attack could be successful when using more traces.

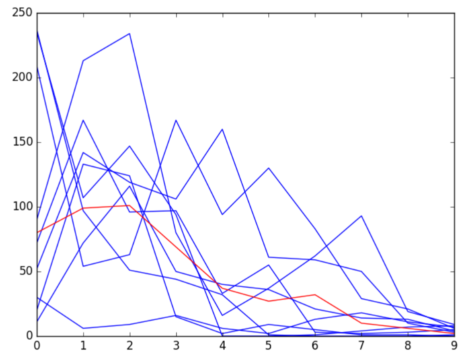

Figure 38: Rank of the ten best-ranked keys, by steps of 10000 measurements, from 10.000 to 100.000. Correct key plotted in red.

### 5.2.2 Raw Sbox output

We target the value $Z_{\hat{k}}[i]$ for any hypothesis $\hat{k}$. The attack does not succeed using the 100.000 traces. Figure 39 illustrates the results using 100.000 traces.

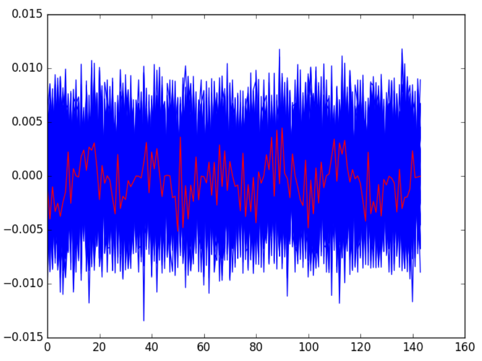

Figure 39: Correlation coefficients obtained when targeting $S[P[i] \oplus \hat{k}]$, for every value of $\hat{k}$. Correct hypothesis plotted in red. X-axis represents all 121 points combinations.

## 5.3 Unknown permutation, non-processed traces

For the sake of completeness, we also perform these experiments on raw unprocessed traces. No success is obtained when targeting $Z_{\hat{k}}[i]$ or $\mathsf{r_m} \times Z_{\hat{k}}[i]$.

No attack is successful using 100.000 traces.

## 5.4 Summary

This section evidences the effectiveness of the implemented countermeasures. The shuffling countermeasures increases the number of needed measurements by a factor of at least $\frac{100.000}{8.000} = 12,5$. It is however likely that this countermeasure does not suffice when increasing slightly the number of measurements. The effectiveness of the affine countermeasure is evidenced by the fact that a 2OCPA is achievable targeting the value $\mathsf{r_m} \times S[P[i] \oplus \hat{k}]$ but no attack is found targeting the value $S[P[i] \oplus \hat{k}]$ using 100.000 traces.

| | Known permutation | | Unknown permutation | |
|---|---|---|---|---|
| Known | $\mathsf{r_m}$ | None | $\mathsf{r_m}$ | None |
| # Traces | $\approx 8.000$ | $> 100.000$ | $> 100.000$ | $> 100.000$ |

Figure 40: Number of traces needed for a successful second-order attack, in the different settings.

# 6 Third-order correlation analysis

In this section, we analyse the resistance of the implementation against third order side-channel analysis based on the linear correlation coefficient (CPA).

For this study, we will target the Sbox output value and consider a Hamming weight ($HW$) univariate leakage model.

In the remainder of this section, we will use:

- the centered product combination function $\mathcal{C}(x, y, z) = (x-\overline{x})(y-\overline{y})(z-\bar{z})$ to preprocess the traces, where $\overline{x}$ (resp. $\overline{y}$, $\bar{z}$) stands for the mean value of $x$ (resp. $y$, $z$), computed on all traces in our campaign;
- the corresponding function $f_{HW}(z) = \sum_{m,m'}(HW(m' \times z \oplus m) - \overline{HW(m' \times z \oplus m)})(HW(m) - \overline{HW(m)})(HW(m') - \overline{HW(m')})$ to compute the predictions, where $\overline{HW(m)}$ (resp $\overline{HW(m' \times z \oplus m)}$, $\overline{HW(m')}$) stands for the mean value of $HW(m)$ (resp. $HW(m' \times z \oplus m)$, $HW(m')$) computed on all possible values of $m, m'$[2].

As in our first-order and second-order correlation analyses, we will split this study in two parts. The first part will consider priviledged knowledge on the mask values. The second part will not consider this knowledge.

## 6.1 Priviledged knowledge

We first study the setting where the attacker knows the random masks used to compute the permutation.

In order to characterize the effectiveness of a CPA against this implemenation, we use the random masks to recompute the permutation $Sh(i)$ for each byte index $i$.

### 6.1.1 Raw Sbox output

We target the value $Z_{\hat{k}}[i]$ for any hypothesis $k$. The attack does not succeed using the 100.000 traces. Figure 41 illustrates the results using 100000 traces.

[2]Note that all possible values for $m$ are in $[0, 255]$, while all possible values for $m'$ are in $[1, 255]$.

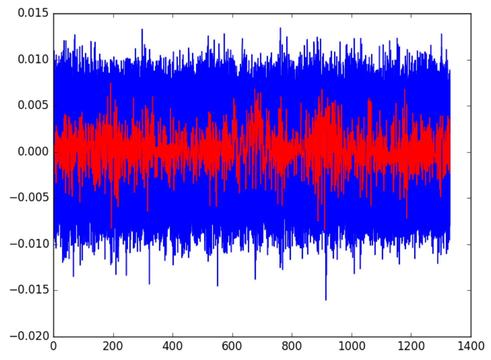

Figure 41: Correlation coefficients obtained when targeting $S[P[i] \oplus \hat{k}]$, for every value of $\hat{k}$. Correct hypothesis plotted in red. X-axis represents all 1331 points combinations.

## 6.2 Unknown permutation, processed traces

We now study the setting where the attacker does not know the random masks used to compute the permutation.

We preprocess the traces in order to average the leakage over the different byte indices manipulation:

- for each index $i$ in $[0, 15]$, we define a small window $w_i$ of $\ell$ points around the SNR peak corresponding to the manipulation of $\mathsf{r_m} \times S[P[Sh(i)] \oplus K[Sh(i)]] \oplus \mathsf{r_{out}}$ in the characterization phase. In our experiments, the size of the window was arbitrarily fixed to $\ell = 11$.
- for each trace in our acquisition campaign, we compute the average window $m$ such that for each time sample $j$, $m[j] = \frac{1}{16} \sum_{i=0}^{15} w_i[j]$. We then consider $m$ as our reduced averaged trace of size $\ell$.
- we concatenate to this trace a small window of $\ell$ points around the SNR peak corresponding to the manipulation of $\mathsf{r_{out}}$.

### 6.2.1 Raw Sbox output

We target the value $Z_{\hat{k}}[i]$ for any hypothesis $k$. The attack does not succeed using the 100.000 traces. The correct key rank does not seem to indicate that this number of traces is close to allow a success attack. Figure 42 illustrates the results using 100.000 traces.

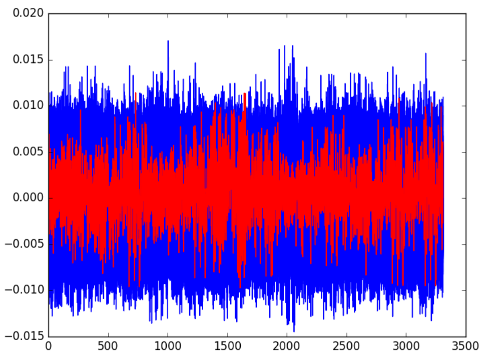

Figure 42: Correlation coefficients obtained when targeting $S[P[i] \oplus \hat{k}]$, for every value of $\hat{k}$. Correct hypothesis plotted in red. X-axis represents all 1331 points combinations.

## 6.3 Unknown permutation, non-processed traces

For the sake of completeness, we also perform these experiments on raw unprocessed traces. No success is obtained when targeting $Z_{\hat{k}}[i]$. No attack is successful using 100.000 traces. Figure 43 illustrates the results using 100.000 traces.

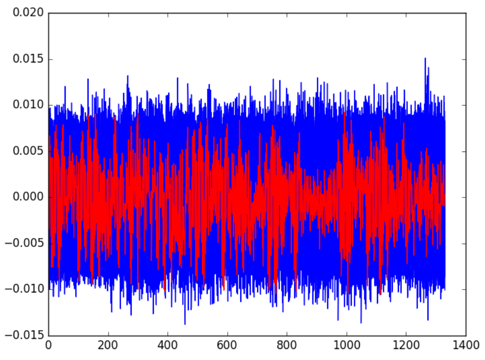

Figure 43: Correlation coefficients obtained when targeting $S[P[i] \oplus \hat{k}]$, for every value of $\hat{k}$. Correct hypothesis plotted in red. X-axis represents all 1331 points combinations.

## 6.4 Summary

No third order attack has been achieved using 100.000 traces.