Using the Joint Distributions of a Cryptographic Function in Side Channel Analysis

Yanis Linge[1,2], Cécile Dumas[1], and Sophie Lambert-Lacroix[2]

[1] CEA-LETI/MINATEC, 17 rue des Martyrs,
38054 Grenoble Cedex 9, France
yanis.linge@emse.fr cecile.dumas@cea.fr
[2] UJF-Grenoble 1 / CNRS / UPMF / TIMC-IMAG
UMR 5525, Grenoble, F-38041, France
Sophie.Lambert@imag.fr

Abstract The Side Channel Analysis is now a classic way to retrieve a secret key in the smart-card world. Unfortunately, most of the ensuing attacks require the plaintext or the ciphertext used by the embedded algorithm. In this article, we present a new method for exploiting the leakage of a device without this constraint. Our attack is based on a study of the leakage distribution of internal data of a cryptographic function and can be performed not only at the beginning or the end of the algorithm, but also at every instant that involves the secret key. This paper focuses on the distribution study and the resulting attack. We also propose a way to proceed in a noisy context using smart distances. We validate our proposition by practical results on an AES128 software implemented on a ATMega2561 and on the DPA contest v4 [28].

1 Introduction

The original work on Side Channel Analysis was done by Kocher in the early 90s [14]. He introduced two new attacks: the Simple Power Analysis (SPA) and the Differential Power Analysis (DPA). In 2004, Brier et al. [4] formalized the DPA and provided a statistical way to compare the leakage model and the power traces thanks to the Pearson correlation factor. Today, side channel attacks gather many methods to attack a device from its power consumption or electromagnetic radiations, such as high order techniques in presence of a masking countermeasure [17,13,18], collision attacks [26,3,7], Algebraic Side Channel Attacks [23,24,25], etc.

Side Channel Attacks are generally based on statistical properties and tend to compare two random variable groups. The first one is represented by all the points of the acquired traces, while the second one depends on the underlying cryptographic function. For example, the Correlation Power Analysis (CPA) [4] consists in the correlation between the device leakage at one instant and the possible value of one intermediate data. For the attack achievement, this value must be computable, *i.e.* it only depends on some few key bits and it is obtained

from the plaintext (or the ciphertext). When neither is known, the acquired trace can not be connected to any cryptographic algorithm data.

The two random variable groups may only be studied independently. It was interesting to us to assume that we do not have any prior knowledge of the plaintext and the ciphertext. In fact, many smart-card applications use cryptographic functions without outputting the plaintext and ciphertext. In this case no internal data can be guessed, even partially, and a classic attack like CPA is not conceivable. Today, a cryptographic algorithm is implemented on a smart-card in a secure way. Many methods exist for masking the data and restricting the leakage [12,1,19,9,22]. However, it may happen for speed reason that countermeasures are only present at the beginning and the end of the implementation, but not in the middle. It is also possible that no protection is positioned precisely because the plaintext and the ciphertext are not outputted by the chip. For example, the GENERATE AC command of the EMV application [29] computes an Application Cryptogram using the algorithm CBC-MAC. A usual way for ensuring the security of this cryptographic scheme against Side Channel Analysis is to protect the first DES of the first block and the last DES of the final Triple-DES.

Nevertheless, the acquired traces contain some leakage information and we presume that it is correlated to the data computed by the device. A trace comprises many instants that reflect the chip activity during the algorithm execution. Each instant may be considered as a real random variable. The variance of these variables tells us that some instants are noisier or linked to the variation of the input data.

Failing to associate one trace to its corresponding algorithm guessing value, we can still study separately the various instants and the different algorithm values that involve a part of the key. This forms the main idea of our proposition. We propose to extract some properties from the algorithm and from the trace and to compare them.

Section 2 presents theoretical aspects by considering the data variations in the cryptographic algorithm. Section 3 discusses how to relate them to the acquired signals and Section 4 provides the last needed tools. In Section 5, we develop the complete attack. Experiments illustrate the attack's efficiency and its interest in Section 6 before the conclusion.

2 Study of the Variations in a Cryptographic Algorithm

An algorithm may be decomposed in small functions that mostly use a part of the secret key.

We will denote by g one of these functions and by $k^\star$ the involved part of the secret key, named subkey.

$$\begin{aligned} g : A \times K &\longrightarrow B \\ (a, k) &\longmapsto b = g(a, k) \end{aligned}$$

We propose to study how the output b varies when the input a is uniformly distributed in A. More precisely, we are interested in the leakage caused by a

and b. Let's denote by $L(z)$ the leakage induced by the handling of the data z, as Rivain did in his thesis [20]. $L(z)$ is comprised of the *leakage function* φ and the *leakage noise* B.

We consider the random variables $\varphi(a)$ and $\{\varphi(g(a,k))\}_{k\in K}$ and the joint probability distributions $\{(\varphi(a),\varphi(g(a,k)))\}_{k\in K}$. We denote for each subkey k the joint probability distribution by $S(g,k) = \{p_{i,j} | i \in \{0,\ldots,n\}, j \in \{0,\ldots,m\}\}$ where $p_{i,j}$ is the probability that $\varphi(a) = i$ and $\varphi(g(a,k)) = j$.

For calculating the distribution of the subkey k, we compute $\varphi(a)$ and $\varphi(g(a,k))$ for each a in A. Then by counting the occurrence of the values $i = \varphi(a)$ and $j = \varphi(g(a,k))$, we obtain the probability for $(\varphi(a),\varphi(g(a,k)))$ to be equal to (i,j).

As $\varphi(g(a,k))$ depends on k, each distribution also depends on the subkey value k. If we get the distribution for an unknown subkey $k^\star$ and if each subkey k matches with a unique distribution[3], we are able to guess the value of $k^\star$ by comparing the distributions; for example, if we consider that g is defined by:

$$\begin{array}{rcl} g : \{0,1\} \times \{0,1\} & \longrightarrow & \{0,1\} \\ (a,k) & \longmapsto & b = a \oplus k \end{array}$$

and φ by:

$$\begin{array}{rcl} \varphi : \{0,1\} & \longrightarrow & \{0,1\} \\ a & \longmapsto & a \end{array}$$

If $k = 0$, $\varphi(a) = \varphi(g(a,k))$. If $k = 1$, $\varphi(a) = 1 - \varphi(g(a,k)) \bmod 2$.
The distributions $S(g,0)$ and $S(g,1)$ are drawn in Table 1. These two distributions are definitely different. So given any distribution, we are able to determine what subkey k was used to produce it.

$k = 0$

$\varphi(b)$ \ $\varphi(a)$	0	1
0	1/2	0
1	0	1/2

$k = 1$

$\varphi(b)$ \ $\varphi(a)$	0	1
0	0	1/2
1	1/2	0

Table 1: Joint distribution of $(a, b = a \oplus k)$ in $\mathbb{Z}/2\mathbb{Z}$

We have computed the theoretical distributions for several functions that involve a part of the key, like the *exclusive-or* between bytes, the DES S-boxes [27] and the AES SubBytes function [8]. All present some differences, even if the non linear functions present more differences. As the distributions of a function do not depend on a device, they may be pre-computed.

If the distributions are easily distinguished, that means the function g could be a good choice for an attack. However, an attacker will face two problems.

[3] This property is true for most cryptographic functions like DES S-boxes or AES SubBytes.

First, he must be able to get a distribution that will be compared to the theoretical ones, knowing he has access only to some traces. We suppose that he can acquire many traces he wants and that the trace number is sufficient for a uniform distribution of the input g function. This assumption is not restrictive because the cryptographic functions generally try to achieve this property. For obtaining a relevant distribution, the attacker needs also to locate the instants corresponding to the handling of the variables a and b. As the traces contain some information but also noise, he will only be able to *estimate* the frequency of the couple $(\varphi(a) = i, \varphi(b) = j)$ denoted $f_{i,j}$. We name $S_d = \{f_{i,j} | i \in \{0, \ldots, n\}, j \in \{0, \ldots, m\}\}$ the estimated distribution of the device. A solution for getting it is proposed in the next section.

The second problem the attacker faces is the need for a method to compare two distributions:

- $S(g, k)$, which is theoretical. It is issued from the preliminary study and depends on the function g and a key guess k.
- S_d, which is estimated. It is computed from the traces and related to the device.

Section 4 examines the existing distances and selects the most promising ones for comparing $S(g, k)$ and S_d.

For the rest of this article, we consider that the function $\varphi(z)$ represents the Hamming weight of z, *i.e.* the number of 1 in binary representation. The function g is composed of the AES operations: AddRoundKey followed by SubBytes, *i.e.* the data a represents a state byte before the AddRoundKey and the data b a state byte after the SubBytes. Notice that other φ models and other g functions could be studied.

3 How to Estimate the Distribution of the Device

We suggest here a method to first identify the suitable instants and then estimate the Hamming weight values that they represent. These instants, named points of interest, are denoted by *PoI*.

To determine the most interesting instants of our traces, we used the variance. The higher the variance, the more favorable the instant, because it represents either the maximal variability of the noise or the maximal variability of some data spend by the algorithm. Determining the *PoI* is an important part of our proposal. In some other cases, the attacker may need more advanced techniques like in [10,2,11], but this simple way is initially sufficient.

After selecting some *PoI*, we have to decide from their amplitude value what the corresponding Hamming weight value is.

In [23], Renauld and Standaert used a Bayesian template described in [6] to retrieve the Hamming weight of the targeted variables. Template attacks are very efficient at obtaining a good approximation of the Hamming weight of the data. However, they require complete access to a device similar to the targeted one.

We suggest here a method that does not recognize the exact Hamming weight value, but allows a reasonable estimation for a low time and memory complexity by using the only traces of the targeted device. The estimation we present is close to the value of the Hamming weight of the targeted data.

Let $Y(t)$ be a set of M measured values corresponding to the same instant t of M traces. We sort this set in an ascending order. As we suppose that the data values of the cryptographic algorithm are uniform, this implies a particular distribution of the Hamming weight values. If n is the maximal Hamming weight value, among the M elements, $\frac{M \times C_n^p}{2^n}$ elements have a Hamming weight p. The elements of $Y(t)$ are classified knowing this distribution. The maximal Hamming weight value is assigned to the highest elements of the set $Y(t)$ and the minimal Hamming weight value to the smallest one.

For example, if 100 values represent the leakage of two bits, we associate the Hamming weight of:

- 2 for the $\frac{100 \times C_2^2}{2^2} = 25$ greater elements
- 1 for the $\frac{100 \times C_2^1}{2^2} = 50$ following elements
- 0 to the $\frac{100 \times C_2^0}{2^2} = 25$ smaller elements

Indeed, the way to classify the elements depends on the leakage quality. This method is effective if the noise is low. So by reducing the noise, for example thanks to the cumulant of order 4 [15], the estimation will be better. Using this simple method, we obtain an approximation of the Hamming weight of the targeted variable. Further, one error in the ranking generally gives a Hamming weight close to the real one. Some total mistakes may occur and perturb the estimated distribution. In this case, more traces will be necessary for obtaining an estimated distribution close to the theoretical distribution corresponding to the true key. We also can choose to sort the M elements in fewer groups by reducing φ. For example, φ may represent the most significant bit, *i.e.* $\varphi(z) = 0$ if the higher significant bit is zero, 1 otherwise. This implies fewer errors, but the function φ is less precise and the theoretical distributions will be more similar. This technique has a low complexity in time and memory but only gives an estimation. This method can be easily replaced by method based on clustering and machine learning.

Since we have a way to approximate the Hamming weight of the input and the output of the AES SubBytes, we can compute an estimated distribution S_d. We need now explain how to compare two distributions.

4 How to Compare Two Distributions

To confront the theoretical distribution $S(g,k)$ and an estimated distribution S_d, the first idea is to use the well-known χ^2 distance between them defined as:

$$\chi^2(S(g,k),S_d) = \sum_{i=0}^{i=n}\sum_{j=0}^{j=m} \delta(p_{i,j},f_{i,j}) \tag{1}$$

The distance between $p_{i,j}$ and $f_{i,j}$ is defined by:

$$\delta(p_{i,j},f_{i,j}) = \begin{cases} \frac{(p_{i,j}-f_{i,j})^2}{p_{i,j}} & ,p_{i,j} \neq 0 \\ 0 & ,p_{i,j} = f_{i,j} \\ \infty & ,p_{i,j} = 0 \neq f_{i,j} \end{cases} \tag{2}$$

Unfortunately, this distance does not allow errors in the estimated distribution S_d. Indeed, a theoretical distribution generally presents a lot of zero values $p_{i,j}$, so a small mistake in the estimated Hamming weight can cause a non-zero value for the corresponding $f_{i,j}$. Thus the distance between S_d and $S(g,k^\star)$ will be infinite with only one error.

So we need to find another distance. In [5], Cha proposes a comprehensive study of different distances between two distributions. We have tested all the 65 distances presented in this article by using the theoretical distributions based on the Hamming weight leakage and the AES SubBytes function.

When trying to match a distribution that is well estimated to the theoretical ones, most distances give similar results and return the good subkey with few samples. But if the device distribution is not well estimated because of the presence of errors for some samples, some distances give better results than others. We simulated 50% erroneous samples[4] to obtain biased device distributions and tried all the distances to compare each estimated distribution to the theoritical ones. We kept the four following best distances, that are those that on average lead to a successful attack.

- The distance based on the Inner Product defined by:

$$d_{IP}(S(g,k),S_d) = 1 - \sum_{i=0}^{i=n}\sum_{j=0}^{j=m} p_{i,j}.f_{i,j} \tag{3}$$

- The distance based on the Harmonic Mean defined by:

$$d_{HM}(S(g,k),S_d) = \begin{cases} 1 - 2.\sum_{i=0}^{i=n}\sum_{j=0}^{j=m} \frac{p_{i,j}.f_{i,j}}{p_{i,j}+f_{i,j}} & ,p_{i,j}+f_{i,j} \neq 0 \\ 0 & ,p_{i,j}+f_{i,j} = 0 \end{cases} \tag{4}$$

[4] An erroneous sample is obtained by adding a white noise to the true value.

– The χ^2 Pearson distance

$$d_{\chi^2_P}(S(g,k),S_d) = \begin{cases} \sum_{i=0}^{i=n}\sum_{j=0}^{j=m}\frac{(p_{i,j}-f_{i,j})^2}{f_{i,j}} & ,f_{i,j}\neq 0 \\ 0 & ,f_{i,j}=p_{i,j} \\ \infty & ,f_{i,j}=0\neq p_{i,j} \end{cases} \quad (5)$$

– The distance of Kullback-Leiber

$$d_{KL}(S(g,k),S_d) = \begin{cases} \sum_{i=0}^{i=n}\sum_{j=0}^{j=m} p_{i,j}.ln(\frac{p_{i,j}}{f_{i,j}}) & ,f_{i,j}\neq 0 \\ 0 & ,f_{i,j}=0 \end{cases} \quad (6)$$

With the study of the distributions of two variables related to the cryptographic algorithm, the method presented in Section 3 for estimating an equivalent distribution related to the device and the distances introduced here, we are now able to establish an attack in order to retrieve the subkey $k^\star$.

5 The Proposed Attack

Our attack consists of four phases. **First**, we get the pre-computed theoretical distributions that are not device dependent. This part is described in Algorithm 1.

Algorithm 1 Computation of the theoretical distributions.

procedure COMPUTATION OF $S(g,k)(g : K \times A \to B\ ,\ N = |K|)$
  for $k \in K$ **do**
    $S(g,k) \leftarrow 0$ ▷ $S(g,k) \in \{0 \dots n\} \times \{0 \dots m\}$
    for $a \in A$ **do**
      $S(g,k)(\varphi(a),\varphi(g(a,k))) \leftarrow S(g,k)(\varphi(a),\varphi(g(a,k))) + \frac{1}{|A|}$
    end for
  end for
return $S(g,k)$
end procedure

In the second step, we detect some *PoI* thanks to the variance criteria. We denote the found *PoI* by $t_a \in T_a$ for the input of our function g and by $t_b \in T_b$ for the output. **The third step** consists of extracting the estimated Hamming weights of each $Y(t_a)$ and each $Y(t_b)$ and computing several estimated distributions $S_d(t_a,t_b)$ for each $t_a \in T_a$ and $t_b \in T_b$ with the method described in Section 3. **Finally**, we compute all the distances between the theoretical distributions and the different estimated distributions. The secret subkey for the couple of *PoI* (t_a,t_b) will be given by:

$$k^\star = \underset{k\in K}{\operatorname{argmin}}(d(S(g,k),S_d(t_a,t_b)))$$

Algorithm 2 Our proposal Attack

procedure ATTACK(M estimated Hamming weight pairs (a_i, b_i))
$g : K \times A \to B$, $N = |K|$
  for $k \in K$ **do**
    $S(g,k) \leftarrow 0$ ▷ $S(g,k) \in \{0 \dots n\} \times \{0 \dots m\}$
    for $a \in A$ **do** ▷ Compute the theoretical distributions
      $S(g,k)(\varphi(a), \varphi(g(a,k))) \leftarrow S(g,k)(\varphi(a), \varphi(g(a,k))) + \frac{1}{|A|}$
    end for
  end for
  $S_d \leftarrow 0$ ▷ $S_d \in \{0 \dots n\}$ x $\{0 \dots m\}$
  for i **from** 0 **to** $M-1$ **do** ▷ Compute the estimated distribution
    $S_d(a_i, b_i) \leftarrow S_d(a_i, b_i) + \frac{1}{M}$
  end for
  $key \leftarrow 0$
  for $k \in K$ **do** ▷ Compare the estimated distribution to the theoretical distributions
    if $d(S(g,k), S_d) < d(S(g,key), S_d)$ **then**
      $key \leftarrow k$
    end if
  end for
return key
end procedure

We name the number of samples M, the number of possible keys $N = |K|$. We describe our attack in Algorithm 2.

Our algorithm complexity is:

- First step: one multiplication and $N \cdot |A|$ additions for the computation of the theoretical distributions.
- Second step: one multiplication and M additions for the computation of the estimated distribution.
- Third step: $N \cdot n \cdot m$ multiplications and $1 + N \cdot n \cdot m$ additions to compute N distances based on the Inner Product.

We notice that the complexity of the attack is low. The first step is performed once and for all. The cost of the attack depends on the sample number for the second step and on the key number for the last step.

6 Experimentations

6.1 Unprotected software implementation on ATMega2561

To validate our Hamming weight estimation and our attack, we have targeted a software AES on an ATMega2561. For this implementation, the internal representation of the data is based on eight bits. The system is not very vulnerable and the acquiered traces are a bit noisy since SPA is not practical and CPA succeeds from 800 traces.

First, we need to ensure that our methodology for estimating the Hamming weight gives good results. We have acquired 1,000 samples on our device. The different AES steps can be distinguished on the traces (see Figure 1). We consider the nine instants with a variance greater than 10 times the average variance (see Figure 2 that is synchronized with Figure 1.).

As some *PoI* are poorly located regarding the region of the trace identified as the function AddRoundKey followed by SubBytes, we are able to identify some unusable *PoI*. For example, one of the *PoI* is localized at the beginning of the trace. Four *PoI* are situated in the SubBytes (SB) region and may therefore correspond to the input of the targeted function. The four remaining *PoI* are positioned in the MixColumn (MC) region and may be associated with the output of the SubBytes function.

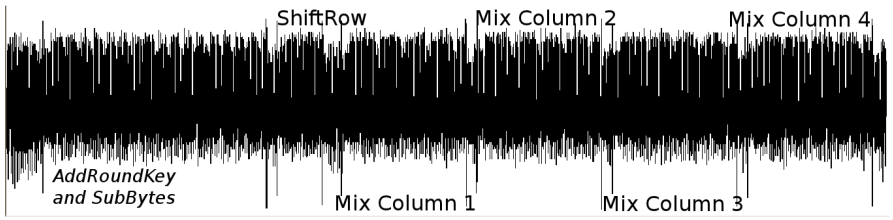

Figure 1: Electromagnetic emanation signal from an ATMega2561 during the execution of the first round of an AES128 software implementation.

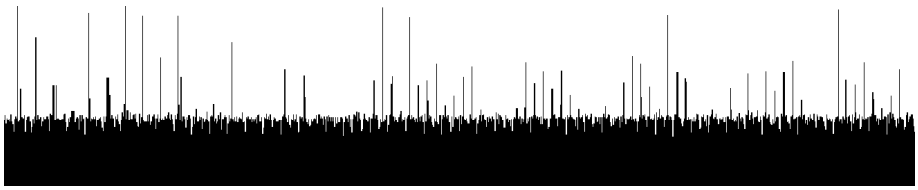

Figure 2: Variance obtained for 1,000 electromagnetic emanation signals from an ATMega2561 during the execution of the first round of an AES128 software implementation.

For validating the Hamming weight estimation method we compare the estimated values to the theoretical ones. Table 2 (resp. Table 3) presents the per-

centage of good Hamming weight estimations for the four *PoI* in the SubBytes region (resp. in the MixColumn region) for all the state bytes.

PoI	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
SB region : 0	24%	21%	28%	18%	78%	22%	24%	29%	23%	24%	29%	24%	23%	21%	24%	25%
SB region : 1	21%	19%	20%	27%	24%	21%	25%	24%	26%	21%	29%	22%	24%	24%	28%	23%
SB region : 2	25%	24%	21%	27%	22%	29%	81%	26%	31%	21%	24%	28%	27%	29%	26%	24%
SB region : 3	22%	23%	24%	68%	21%	27%	23%	25%	23%	31%	25%	21%	23%	25%	21%	22%

Table 2: Percentage of good Hamming weight estimations for the input of the targeted function for all the state bytes.

PoI	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MC region : 0	18%	21%	25%	27%	22%	24%	27%	29%	23%	22%	28%	20%	23%	21%	28%	29%
MC region : 1	25%	24%	21%	27%	22%	29%	73%	26%	31%	21%	24%	28%	27%	29%	26%	24%
MC region : 2	23%	25%	28%	75%	25%	25%	24%	28%	21%	24%	21%	24%	22%	24%	29%	23%
MC region : 3	24%	21%	28%	18%	64%	22%	24%	29%	23%	24%	29%	24%	23%	21%	24%	25%

Table 3: Percentage of good Hamming weight estimations for the output of the targeted function for all the state bytes.

As the implementation handles byte by byte, one *PoI* corresponds to, at most, one state byte. So the estimator shall associate many correct Hamming weight values to, at most, one state byte. For the other state bytes, the number of correct Hamming weight values is close to a random estimation. For example, we can remark that the first *PoI* in SubBytes region shall represent the state byte 4 before AddroundKey. Thus the estimator gives us a good approximation of six bytes, each one corresponding to one *PoI*.

We conclude that our estimator gives good results, at least for the trace instants where the variance is high. Thus an attacker could directly use the variance criteria and the identification of the trace blocks to choose the *PoI*. Of course, he does not have the means to verify the estimation method because he does not know the key value.

Finally, we have to validate the proposed attack. Luckily, the chosen *PoI* give Hamming weight values that correspond to the same state byte before and after the targeted function. So we expect to find the bytes 3, 4 and 6 of the subkey. Four *PoI* have been considered by region, so $4 \times 4 = 16$ pairs (t_a, t_b) have to be tested. The whole attack has been performed in a few seconds. In Table 4, we present the results of our attack for all pairs regarding the distance of the Inner Product. The pairs are sorted by the minimal distance to the true theoritical distribution. As only four *PoI* are selected in each region, we keep the four subkeys with a lower distance. As expected, we retrieve three bytes of the key and a wrong one.

PoI in SB region	*PoI* in MC region	Byte value	Distance	True?
2	1	54	0.0036051	✓
3	2	31	0.0036711	✓
0	3	61	0.0037023	✓
3	3	224	0.0037423	X
0	0	200	0.0037556	X
2	3	234	0.0037823	X
2	0	39	0.0037883	X
1	3	154	0.0037976	X
3	0	55	0.0037986	X
0	2	216	0.0038011	X
1	1	159	0.0038514	X
2	2	206	0.0038786	X
1	2	21	0.0038983	X
3	1	218	0.0039113	X
0	1	257	0.0039115	X
1	0	197	0.0039612	X

Table 4: Attack results for the 4×4 chosen *PoI*.

It is important to notice that every recorded key byte reduces the cryptographic security. Here an attacker must still guess the position of the recovered bytes and the missing bytes, knowing that he may get a wrong byte. So he would like to retrieve more key bytes. For that we propose to get more *PoI* by considering for each region the 50 instants with the highest variance. As the *PoI* number rises, the time required for the attack increases too. We need a few minutes to obtain the results showed in Table 5 where the 16 first ones are ranked regarding the distance based on the Inner Product. Ten key bytes are the real ones so we have still $10! \cdot 2^{6 \cdot 8} \approx 2^{70}$ keys to test. This number may seem huge, but it is possible to perform again our attack at another round and then combine the results.

PoI in SB region	*PoI* in MC region	Byte value	Distance	True?
8	23	31	0.035180	✓
42	18	23	0.035773	✓
33	45	54	0.035813	✓
1	8	228	0.035867	✓
16	49	191	0.035941	✓
12	38	138	0.035977	X
11	21	61	0.035996	✓
48	20	224	0.036023	X
5	19	61	0.036023	✓
28	12	207	0.036051	✓
13	33	25	0.036094	X
21	42	39	0.036121	X
9	34	197	0.036137	X
25	47	198	0.036137	✓
38	15	109	0.036187	X
17	31	145	0.036203	✓

Table 5: The top 16 attack results for the 50 *PoI* with the higher variance for each region.

We have also performed the attack by using the other distances of Cha's article [5]. This leads to worse results and validates the choice of the Inner Product distance.

The targeted implementation is eight bits, but it is also possible to attack a 16-bit implementation. This implies computing more distributions that are larger, but their computation can be performed in less than a half hour.

More, if the implementation provides a data masking countermeasure, our proposal can still be effective by targeting the AddRoundKey operation instead of the SubBytes.

6.2 DPAContest V4 [28]

In the 2013 summer, the DPA contest version 4 [28] has been released. It provides 100,000 samples corresponding to the first round and the beginning of the second round of an AES-256 software. So we focus only the first subkey.

The proposed implementation is protected by using a countermeasure called RSM [21]. The RSM is a masking countermeasure that uses an unique 16-byte mask that is randomly rotated by an offset between 0 and 15 at each execution. So the AES is computed with 16 different random masks. First we have classed the traces obtained by the same mask offset thanks to a pattern detection by autocorrelation. This classification does not need to be very accurate because our methodology is resilient to distribution errors. Then we only consider one class of traces: about six thousand samples corresponding to the computation with a same, but unknown, mask offset j.

The SubBytes function in the AES is replaced by sixteen masked sboxes:

$$SB(X \oplus M_{i+j \bmod 16} \oplus K) \oplus M_{i+1+j \bmod 16}$$

where $M_0 \dots M_{15}$ are the bytes of the mask and i is the sbox number. As the value of the 16-byte mask is known, this function contains two unknowns values: the secret byte subkey K and also the value $i + j$.

We decide to retrieve these two values together by using our method. If the offset j is fixed, the knowledge of $i + j$ is an advantage as it gives the relative position between the finding bytes. This hugely reduces the number of remaining keys in the exhaustive search.

As previously we have selected 1,000 *PoI* by using the variance. But this time we use four distances:

- Inner Product
- Harmonic mean
- Pearson χ^2
- Kullback-Leiber

These four distances will give different results, but we expect that the good key byte will has a good rank for each distance. The idea is to compute the top 16 attack results for each distance and each pair of *PoI*. Then we keep only the results for each pair that appear for all distances. Finally the first 16 most frequent values are proposed for the subkey. We found 7 ordered bytes of the secret subkey. It is important to notice that here we only have $7 \cdot 2^{9 \cdot 8} \approx 2^{75}$ remaining subkeys.

We have applied the same attack on another class of traces by considering the other offset values and compared the different results. The same bytes of the subkey have been obtained. More secret key bytes could be retrieved with the acquisition of the AES next rounds.

7 Conclusion

We propose a promising Side Channel Attack based on the joint distributions of a cryptographic function. The great advantage is that it is not necessary to know the plaintext or the ciphertext. First, we have investigated the way the internal data varies between them and noticed that the joint distribution of two data sets highly depends on the secret key used by the algorithm. In parallel, we have proposed to deduce the real joint distribution from the acquired traces thanks to a simple Hamming weight estimator based on the statistical variance and the particular repartition of random variables. In order to compare this estimated distribution with the theoretical ones, we have tried several distances and chosen the more favorable to build an attack. In the ideal case where the Hamming weight estimation is always correct, our proposed attack is very effective and the true key is found with fewer than 30 samples. In the real world, the acquired signals are noisy and the estimator is not perfect. But, with more samples the attack still remains successful, even with 50% of estimation error.

Indeed, we have validated both the estimation method and the key recovery by applying our attack on two sets of acquisition. The first one contains 1,000 traces issued from an AES128 software implementation into an ATMega2561. The results show that 10 disordered key bytes can be retrieved without any knowledge of the plaintext or the ciphertext. The second one comes from the DPA contest v4. Here we found 7 ordered bytes of the key.

The software context is well adapted to estimate the joint distribution of two internal data sets because few bits are processed at the same time. Attacking a hardware implementation in this manner remains a challenge, as it involves a huge bit number. This represents an easy way to protect a cryptographic algorithm. However, our attack may be also relevant for non-cryptographic operations like masking or reverse engineering.

References

1. M.L. Akkar, C. Giraud. *An Implementation of DES and AES secure against some attacks*. CHES 2001, LNCS vol. 2162 pp. 309–318, 2001.
2. C. Archambeau, E. Peeters, F.-X. Standaert, J.-J. Quisquater. *Template Attacks in Principal Subspaces*. CHES 2006, LNCS, vol. 4249, pp. 1-14, 2006.
3. A. Bogdanov. *Improved Side-Channel Collision Attacks on AES*. SAC 2007, LNCS, vol. 4876, pp. 84-95, 2007.
4. E. Brier, C. Clavier, F.Olivier. *Correlation power analysis with a leakage model*. CHES 2004, LNCS, vol. 3156, pp. 16-29, 2004.
5. S.-H. Cha. (2007) *Comprehensive survey on distance/similarity measures between probability density functions*. International Journal of Mathematical Models and Methods in Applied Science, 1 (4), 300–307, 2007.
6. S. Chari, J. Rao, P.Rohatgi. *Template Attack*. CHES 2002, LNCS, vol. 2523, pp. 13-28, 2002.
7. C. Clavier, B. Feix, G. Gagnerot, M. Roussellet, V. Verneuil. *Improved Collision-Correlation Power Analysis on First Order Protected AES* CHES 2011, LNCS, vol 6917, pp 49-62, 2011.
8. J. Daemen, V. Rijmen. *AES proposal: Rijndael*, 1998.
9. B. Debraize. *Efficient and Provably Secure Methods for Switching from Arithmetic to Boolean Masking*. CHES'12, LNCS, vol. 7428, pp. 107–121, 2012.
10. B. Gierlichs, L. Batina, P. Tuyls, B. Preneel. *Mutual Information Analysis - A Generic Side-Channel Distinguisher*. CHES 2008, LNCS, vol. 5154, pp. 426-442, 2008.
11. B. Gierlichs, K. Lemke-Rust, C. Paar. *Templates vs. stochastic methods*. CHES 2006, LNCS, vol. 4249, pp. 15-29, 2006.
12. L. Goubin, J. Patarin. *DES and Differential Power Analysis - The duplication method*. CHES '99 LNCS vol. 1717, pp. 158-172, 1999.
13. M. Joye, P. Paillier, B. Schoenmakers. *On Second-Order Differential Power Analysis*. CHES 2005, LNCS vol. 3659, pp. 293–308, 2005.
14. P.C. Kocher, J. Jaffe, B. Jun. *Differential power analysis*. CRYPTO, pp. 388-397, 1999.
15. T.-H. Le, J. Clédière, C. Servière, J.-L. Lacoume. *Noise reduction in side channel attack using fourth-order cumulant*. IEEE Transactions on Information Forensics and Security, vol. 2, no. 4, pp. 710–720, 2007.

16. S. Mangard, E. Oswald, T. Popp. *Power Analysis Attack - Revealing the Secret of Smart Cards*. Springer, 2007.
17. T. Messerges. *Using Second-order Power Analysis to Attack DPA Resistant Software*. CHES 2000 LNCS vol. 1965, pp. 238-251, 2000.
18. E. Oswald, S. Mangard, C. Herbst, S. Tillich. *Practical Second-Order DPA Attacks for Masked Smart Card Implementations of Block Ciphers*. CT-RSA 2006 LNCS vol. 3860, pp. 192–207, 2006.
19. E. Oswald, S. Mangard, N. Pramstaller. *Secure and Efficient Masking of AES - A Mission Impossible?* Cryptology ePrint Archive, Report 2004/134, http://eprint.iacr.org/2004/134.
20. M. Rivain. *On the Physical Security of Cryptographic Implementations*. PhD thesis, University of Luxembourg, 2009.
21. M. Nassar, Y. Souissi, S. Guilley, J.-L. Danger. *RSM: A small and fast countermeasure for AES, secure against 1st and 2nd-order zero-offset SCAs*. DATE 2012, pp 1173-1178, 2012.
22. M. Rivain, E. Prouff, J. Doget. *Higher-Order Masking and Shuffling for Software Implementations of Block Ciphers*. CHES 2009, LNCS, vol. 5747, pp. 171–188, 2009.
23. M. Renauld , F-X.Standaert. *Algebraic Side-Channel Attacks*. Cryptology ePrint Archive, report 2009/279, http://eprint.iacr.org/2009/279.
24. M. Renauld, F.-X. Standaert, N. Veyrat-Charvillon. *Algebraic Side-Channel Attacks on the AES: Why Time also Matters in Differential Power Analysis*. CHES 2009, LNCS, vol. 5746, pp. 97-111, 2009.
25. M. Saied Emam Mohamed, S. Bulygin, M. Zohner, A. Heuser, M. Walter *Improved Algebraic Side-Channel Attack on AES* Cryptology ePrint Archive, report 2012/084, http://eprint.iacr.org/2012/084.
26. K. Schramm, T.J. Wollinger, C. Paar. *A new class of collision attacks and its application to Des*. FSE'03, LNCS, vol. 2887, pp. 206-222, 2003.
27. Federal Information Processing. *Data Encryption Standard. Standards Publication 46-1 National Technical Information Service, U.S. Dept. of Commerce, 1977*.
28. DPA contest v4. *http://www.dpacontest.org/v4/*
29. EMVCo *EMV Integrated Circuit Card Specifications for Payment Systems, Book 2, Security and Key Management, Version 4.3, November 2011*.