On Practical Second-Order Power Analysis Attacks for Block Ciphers

Renato Menicocci[1], Andrea Simonetti[2], Giuseppe Scotti[2], and Alessandro Trifiletti[2]

[1] Fondazione Ugo Bordoni
Viale del Policlinico 147, 00161 Roma, Italy
rmenicocci@fub.it

[2] Dipartimento di Ingegneria dell'Informazione, Elettronica e Telecomunicazioni*
Via Eudossiana 18, 00184 Roma, Italy
{scotti,simonetti,trifiletti}@die.uniroma1.it

Abstract. We propose a variant for a published second-order power analysis attack [1] on a software masked implementation of AES-128 [2]. Our approach can, with reduced complexity, produce the same result as the original one, without requiring any additional tool. The validity of the proposed variant is confirmed by experiments, whose results allow for a comparison between the two approaches.

Keywords: Side channel attacks, Second order power analysis, Block ciphers, AES.

1 Introduction

Side channel attacks [3] [4] is a large collection of attack techniques targeting a *cryptographic device* as opposed to techniques targeting the implemented *cryptographic algorithm*. Side channel attacks are based on monitoring the physical activity of a cryptographic device and aim at recovering secret data by exploiting information leakage originated by *side channels* such as power consumption and electromagnetic radiation. In this paper, we focus on side channel attacks based on monitoring the power consumption of a cryptographic device, which are known as *power analysis attacks* [5]. Power analysis attacks rely on *data dependence* exhibited by the power consumption of the attacked cryptographic device. In a typical power analysis attack, after measuring the relevant power consumption (*power trace collecting*), the measurement material is suitably analyzed to infer on the secret information (key) controlling the cryptographic algorithm. Notice that the cryptographic device is assumed to operate under known conditions (the implemented cryptographic algorithm and its input and/or output are assumed to be known to the attacker).We focus on an analysis technique known as *Correlation Power Analysis* (CPA) [6]. In its elementary form, called *first-order*

* This work was partially supported by the European Commission under the Sixth Framework Programme (Project SCARD, Contract Number IST-2002-507270).

(or *standard*) CPA, this technique exploits the single samples of the collected power traces. *Higher-order* CPA is instead characterized by the combination of multiple power samples. Here, we focus on *second-order* CPA, where two power samples are combined to extract the relevant information. Against power analysis attacks several countemeaures have been proposed [3] [4]. For software implementations of symmetric block ciphers, a commonly adopted countermeasure relies on the concept of *Boolean masking* [7]. The application of this concept can produce, without huge costs, *masked* implementations which are resistant to attacks based on first-order CPA. For a period of time, this has been considered a satisfactory solution to the problem of power analysis attacks, based on the high costs of higher-order attacks able to break masked implementations.

In [1], some *practical* second-order power analysis attacks are presented which are applicable to physical implementations of a number of block ciphers. Namely, these attacks are devised for masked implementations which are resistant to first-order power analysis attacks. Some experimental results supporting the presented attacks are also shown in [1] for a case study based on AES-128 encryption algorithm [2] (or AES-128, in short). The presented attacks, following [8], require the execution of a *pre-processing step* and a *correlation step*, where the *pre-processed* power traces are examined by a standard CPA.

In this paper we present a variant of some of the attacks devised in [1] which aims to get the same result with a lower complexity. As in [1], the new attack proposed here is described for a masked implementation of AES-128. Moreover, the new attack is successfully experimented for a software masked implementation of a variant of AES-128. We stress the fact that, for the results to be shown here, this variant is at all equivalent to AES-128.

The organization of the paper is as follows. In Section 2 we give a short description of the previous attacks in [1]. In Section 3 we introduce our new attacks and give an analysis of their feasibility. In Section 4 we show some experimental results for both the previous and the new attacks. Finally, some conclusions appear in Section 5. Some additional materials appear in the appendices. In Appendix 5 we recall the specifications of AES-128. In Appendix 5 we describe an example of first-order power analysis attack on AES-128 based on CPA. In Appendix 5 we briefly recall the concept of Boolean masking as a countermeasure against first-order CPA. In Appendix 5 we detail the variant of AES-128 which we used as our reference encryption algorithm. Finally, in Appendix 5 we describe the masked implementation of our reference encryption algorithm which we used for the experiments reported in Section 4.

2 Previous Results

The basic idea in the attacks devised in [1] is to use the Hamming weight of $d_1 \oplus d_2$ to predict $|P(t_{d_1}) - P(t_{d_2})|$, where $\oplus$ denotes the XOR between data of equal length and $P(t_{d_i})$ denotes, for the attacked implementation, the power consumption at the time instant when d_i is processed. This idea is shown to make sense under the assumption that the Hamming weight of d can be used to predict $P(t_d)$. Observe that, in the given context, d is a masked quantity.

Basically, the attacks to be described consist in 1. Executing a proper transformation of the relevant power traces (*pre-processing step*), and 2. Executing a standard CPA [6] using the transformed power traces (*correlation step*).

The pre-processing step transforms each original power trace P, extending over L time instants named $1, 2 \ldots, L$, into a *modified power trace* P^*, extending over $L^* = (L-1) \times L/2$ *modified time samples*. P^* is formed by concatenating $|P(t) - P(1)|$ for $2 \leq t \leq L$, $|P(t) - P(2)|$ for $3 \leq t \leq L$, and so on up to $|P(t) - P(L-1)|$ for $t = L$. Clearly, the modified power trace certainly contains the relevant difference $|P(t_{d_1}) - P(t_{d_2})|$ provided that both t_{d_1} and t_{d_2} are covered by the original power trace.

The correlation step executes a standard CPA using the modified power traces and predictions of the form $h_w(d_1 \oplus d_2)$, h_w being the Hamming weight function.

Two of the attacks presented in [1] are briefly described here for the case of a masked implementation of AES-128 (see Appendix 5). Both these attacks assume that in the masked implementation of AES-128 a masked byte transformation *SBox′*, transforming masked input bytes into masked output bytes, replaces the original byte transformation SBox. Namely, it is assumed that, on input $x \oplus m_{in}$, SBox′ produces $y \oplus m_{out}$, where y is the SBox output on input x, that is, $\text{SBox}'(x \oplus m_{in}) = \text{SBox}(x) \oplus m_{out}$.

The *one masked table look-up attack* [1] assumes that $m_{in} = m_{out} = m$ for any SBox′ operation. The Hamming weight of $(x \oplus m) \oplus (y \oplus m) = x \oplus \text{SBox}(x)$ is then used to predict $|P(t_{x \oplus m}) - P(t_{y \oplus m})|$. This produces two possible attacks targeting a single SBox′ operation either at round R1 or at round R10. Under the hypothesis of known random plaintexts (or ciphertexts), a single application of the attack at round R1 (or R10) reveals one selected byte of K^0 (or of K^{10}). Observe that the selected round key byte is revealed by CPA at the cost of an appropriate exhaustive search over 2^8 guesses, where a guess consists of a round key byte. We notice that, apparently, there is no need to assume that m is the same for any SBox′ operation: It seems sufficient to assume that, for the SBox′ operation corresponding to the selected byte of K^0 (or of K^{10}), the input mask equals the output mask (or also that the XOR between these masks is known).

The *two masked table look-ups attack* [1] assumes that m_{in} is independent of m_{out} and that the same pair (m_{in}, m_{out}) is used for any SBox′ operation. This time, the Hamming weight of $(y_1 \oplus m_{out}) \oplus (y_2 \oplus m_{out}) = \text{SBox}(x_1) \oplus \text{SBox}(x_2)$ is used to predict $|P(t_{y_1 \oplus m_{out}}) - P(t_{y_2 \oplus m_{out}})|$. This produces two possible attacks targeting either two SBox′ operations at round R1 or one SBox′ operation at round R1 and one at round R10. Under the hypothesis of known random plaintexts (or both plaintexts and ciphertexts), a single application of the attack at round R1 (or at both R1 and R10) reveals two selected bytes of K^0 (or one selected byte of K^0 and one selected byte of K^{10}). In the first case, during CPA, the above prediction takes the form $h_w(\text{SBox}(p_i \oplus g_i) \oplus \text{SBox}(p_j \oplus g_j))$, where g_i and g_j denote guesses about the unknown selected bytes, k_i^0 and k_j^0, of K^0, and p_i and p_j denote the corresponding plaintext bytes. In the second case, the above prediction takes the form $h_w(\text{SBox}(p_i \oplus g_i) \oplus (c_j \oplus g_j))$, where g_i and g_j denote guesses about the unknown selected bytes, k_i^0 and k_j^{10}, of K^0 and K^{10}, and p_i and c_j denote the corresponding plaintext and ciphertext bytes. Observe that, in both cases, the two selected round key bytes are revealed by CPA at the cost of an appropriate exhaustive search over 2^{16} guesses, where a guess consists of a pair of round key bytes.

Notice that, apparently, there is no need to assume that the same pair (m_{in}, m_{out}) is used for any SBox$'$ operation: It seems sufficient to assume that, for the SBox$'$ operations corresponding to the two selected bytes of K^0 (or one selected byte of K^0 and one selected byte of K^{10}), the output masks equal each other. Notice also that the complexity of reconstructing all the 16 bytes of K and its dependence on the hypotheses about the used mask are not addressed in this paper.

3 New Attacks

In this section we present a variant of the above attack targeting two masked table look-ups (see Section 2). This variant aims to replace the original exhaustive search over 2^{16} guesses by two exhaustive searches over 2^8 guesses. The new attack is, as before, described for a masked implementation of AES-128. Two attack forms are considered where the input mask and the output mask associated to an SBox$'$ operation are assumed to be independent of each other.

The new attack essentially consists of two correlation steps, the second of which depends on the results provided by the first one. The pre-processing step, which is defined exactly as before, can be common to the two correlation steps. The only point to recall is that the original power traces have to cover the relevant time instants used in the correlation steps.

3.1 First Form of Attack

The first form of the attack assumes plaintext knowledge. The first correlation step targets the generation of the input for two SBox$'$ operations at round R1. The second correlation step targets the outputs produced by the two SBox$'$ operations involved in the first correlation step. At the end of the second correlation step, two bytes of K^0 are revealed.

The first correlation step relies on the assumption that the input masks associated to the two relevant SBox$'$ operations have the same value m_{in}. In this case, the Hamming weight of $(x_1 \oplus m_{in}) \oplus (x_2 \oplus m_{in}) = x_1 \oplus x_2$ can be used to predict $|P(t_{x_1 \oplus m_{in}}) - P(t_{x_2 \oplus m_{in}})|$. During CPA, this prediction takes the form $h_w((p_i \oplus g_i) \oplus (p_j \oplus g_j))$, that is $h_w((p_i \oplus p_j) \oplus g_{\delta_{i,j}})$, where $g_{\delta_{i,j}}$ is a guess for the unknown XOR between the two selected bytes, k_i^0 and k_j^0, of K^0, and p_i and p_j denote the two corresponding plaintext bytes. The value of $k_i^0 \oplus k_j^0$ is revealed by CPA at the cost of an appropriate exhaustive search over 2^8 guesses.

Notice that, for the value of $k_i^0 \oplus k_j^0$, the above correlation step could produce a (small) set of candidates, $S_{i,j}$, which is expected to be constituted by bytes *close* to $k_i^0 \oplus k_j^0$, where close is to be understood in the sense of Hamming distance. Observe also that the number of candidates is expected to reduce to just 1 (the right value of $k_i^0 \oplus k_j^0$) when a sufficient number of traces is used. A further discussion of all this is presented in Section 3.3, where it is also shown that the above exhaustive search could be reduced to 2^7 guesses.

The second correlation step is analogous to the correlation step for the *two masked table look-ups* attack at round R1 of [1]. So, it assumes that the output masks associated to the two relevant SBox′ operations have the same value. The difference is that, this time, the correlation step is executed with the help of the results provided by the first correlation step about the value of $k_i^0 \oplus k_j^0$. This allows to replace the guess (g_i, g_j) by the guess $(g_i, g_i \oplus \tilde{g}_{\delta_{i,j}})$, where $\tilde{g}_{\delta_{i,j}} \in S_{i,j}$. The values of k_i^0 and k_j^0 are then revealed by CPA at the cost of an appropriate exhaustive search over 2^8 guesses. Observe that, to be on the safe side, a linear factor due to the cardinality of $S_{i,j}$ should be considered for the cost of this correlation step.

3.2 Second Form of Attack

The second form of the attack assumes ciphertext knowledge. The first correlation step targets the output for two SBox′ operations at round R10. The second correlation step targets the generation of the input for the two SBox′ operations involved in the first correlation step. Again, for the two relevant SBox′ operations, it is assumed that the input masks have the same value m_{in} and that the output masks have the same value m_{out}. At the end of the second correlation step, two bytes of K^{10} are revealed.

The first correlation step is based on using the Hamming weight of $(y_1 \oplus m_{out}) \oplus (y_2 \oplus m_{out}) = y_1 \oplus y_2$ to predict $|P(t_{y_1 \oplus m_{out}}) - P(t_{y_2 \oplus m_{out}})|$. Namely, this prediction takes the form $h_w((c_i \oplus g_i) \oplus (c_j \oplus g_j))$, that is $h_w((c_i \oplus c_j) \oplus g_{\delta_{i,j}})$, where $g_{\delta_{i,j}}$ is a guess for the unknown XOR between the selected bytes, k_i^{10} and k_j^{10}, of K^{10}, and c_i and c_j denote the corresponding ciphertext bytes. The description of this step can be readily completed by adapting what has been said for the first step of the first form of attack (see 3.1).

The second correlation step is based on using the Hamming weight of $(x_1 \oplus m_{in}) \oplus (x_2 \oplus m_{in}) = x_1 \oplus x_2$ to predict $|P(t_{x_1 \oplus m_{in}}) - P(t_{x_2 \oplus m_{in}})|$. Namely, this prediction takes the form $h_w(\text{SBox}^{-1}(c_i \oplus g_i) \oplus \text{SBox}^{-1}(c_j \oplus g_j))$, where SBox^{-1} denotes the byte transformation which inverts the SBox byte transformation. Again, the guess (g_i, g_j) is formed by taking advantage of the results about $k_i^{10} \oplus k_j^{10}$ provided by the first correlation step. The description of this step can be readily completed by adapting what has been said for the second step of the first form of attack (see 3.1).

3.3 Expectable Effects of a First Correlation Step Targeting Two Masked Key Additions

In [6], a probabilistic power model is considered where the power consumption due to the processing of data D is assumed to be linear in the Hamming distance between D and R, where R is a constant, and the effectiveness of CPA is shown for the problem of determing the value of R when this is unknown. In the following, some of the results established in [6] are suitably adapted to the case of interest.

Assume the relevant power consumption can be modelled as $P = ah_w(D) + B$, where D is binary string of s independent and uniformly distributed bits, h_w is the Hamming weight function, a is a scale factor, and B is a random variable independent of D. Then, some results can be established for the correlation coefficient, $\rho(P, H)$, between P and $H = h_w(D)$, and for the correlation coefficient, $\rho(P, H_q)$, between P and $H_q = h_w(D \oplus E_q)$, E_q being any s-bit string with exactly q ones ($h_w(E_q) = q$). Namely,

$$\rho(P, H) = (a\sqrt{s})/\sqrt{sa^2 + 4\sigma_B^2} \ ; \tag{1}$$

$$\rho(P, H_q) = \rho(P, H)(s - 2q)/s \ . \tag{2}$$

Observe that the sample correlation coefficients computed between *physical* and *prediction vectors* during CPA (see Appendix 5) are estimates of the above correlation coefficients. Thus, under the given model, (2) can be used to anticipate what happens in practice when computing the sample correlation coefficient between a physical vector corresponding to unknown data and a prediction vector based on wrong data.

The experimental results presented in [1] show, for the analyzed case, that a positive correlation exhists between a physical vector consisting of samples of the modified power traces and a prediction vector based on Hamming weight computation.

Based on this, we adopt for the relevant *modified power consumption* P^* (see Section 2) a model analogous to the one presented before. Namely, we put $P^* = a^*h_w(D^*) + B^*$, where D^* is a binary string of s independent and uniformly distributed bits, produced by the XOR between two binary strings of s independent and uniformly distributed bits, a^* is a scale factor, and B^* is a random variable independent of D^*. Notice that the experimental results in [1] show that a positive scale factor (a^*) can be used to model the case studied there. Based on (2), if $H^* = h_w(D^*)$ and $H_q^* = h_w(D^* \oplus E_q)$, we get

$$\rho(P^*, H_q^*) = \rho(P^*, H^*)(s - 2q)/s \ . \tag{3}$$

Now, we can motivate the introduction of the set $S_{i,j}$ for the attack presented in Section 3.1. Observe that the following considerations also apply, *mutatis mutandis*, to the attack presented in Section 3.2. When targeting the generation of the input for two SBox$'$ operations at round R1 (see Section 3.1), we produce, during CPA for the first correlation step, the predictions $h_w((p_i \oplus p_j) \oplus g_{\delta_{i,j}})$, where $g_{\delta_{i,j}}$ is a guess for the unknown XOR between the two selected bytes, k_i^0 and k_j^0, of K^0, and p_i and p_j denote the two corresponding plaintext bytes.

Observe that only using the right guess $\hat{g}_{\delta_{i,j}} = k_i^0 \oplus k_j^0$ produces the right inputs $(p_i \oplus p_j) \oplus \hat{g}_{\delta_{i,j}}$ to the prediction function h_w. If a wrong guess, at Hamming distance q from the right one, is used, then the wrong inputs provided to the prediction functions are at Hamming distance q from the right ones.

Based on the previous analysis of the correlation coefficient, we see that the right guess could turn out to be not so distinguishable, by CPA, from wrong guesses sufficiently close to it, where close is to be understood in the sense of Hamming distance. For example, for $s = 8$, (3) gives $\rho(P^*, H_1^*) = 0.75\rho(P^*, H^*)$ and $\rho(P^*, H_2^*) = 0.5\rho(P^*, H^*)$. In Section 4, we give explicit results about the CPA effects of wrong guesses having Hamming distances one and two from the right one.

A consequence of (3) is

$$\rho(P^*, H_{s-q}^*) = -\rho(P^*, H_q^*) \ . \tag{4}$$

For the attacks of interest here (see Section 3.1 and Section 3.2), this refers to the effects of using two guesses at maximal Hamming distances from each other during the first correlation step. Two such guesses are here called *companion guesses*. Observe that two companion guesses produce inputs at maximal Hamming distances, 8, from each other to the prediction function h_w. As a result, the two predictions are the complement to 8 of each other. Then, the correlation curves produced during CPA can be divided into pairs of *companion curves*, where a pair of companion curves is originated by a pair of companion guesses. Equality (4) anticipates that companion curves are, apart from the sign, very close to each other. As a consequence, the exhaustive search over the set of the relevant guesses could be restricted to half its cardinality by using a subset of guesses where companion guesses are not present. Observe that the effects of taking this approach depend on the knowledge of the sign of the scale factor a^*. In fact, if this sign is unknown, it is required to use a CPA based on the absolute value of the correlation curves and CPA can only provide pairs of candidate right guesses. It is easy to see that this limitation is absent in the case where the sign of a^* is known. We anticipate that, as it was confirmed by experiments, a^* was expected to be positive in the case studied here.

Notice that the closeness of companion correlation curves anticipated by (4) can be made more precise. In fact, a result analogous to (4) holds when replacing the correlation coefficient by its estimate provided by the sample correlation coefficient. In fact, for any pair of sample vectors U and V, both of N entries, if $r(U, V)$ denotes the sample correlation coefficient between U and V, then, if V' denotes the sample vector where each entry is obtained by complementing to any constant z the omologous entry of V, then we have $r(U, V') = -r(U, V)$. This can be readily proved by comparing the following expressions for sample vectors of N entries, where subscripts and their limits are omitted for simplicity

$$r(U,V) = \frac{N\sum UV - \sum U \sum V}{\sqrt{[N\sum U^2 - (\sum U)^2][N\sum V^2 - (\sum V)^2]}} \ ; \tag{5}$$

$$r(U,V') = \frac{N\sum(U(z-V)) - \sum U \sum(z-V)}{\sqrt{[N\sum U^2 - (\sum U)^2][N\sum(z-V)^2 - (\sum(z-V))^2]}} \ . \tag{6}$$

4 Experimental Results

In this Section we show a comparison between the *two masked table look-ups attack* (see Section 2) and the *first form of attack* we proposed in Section 3.1. We executed both these attacks by using the HW/SW environment described in the sequel.

We attacked a variant of AES-128 encryption algorithm, here called *AESv*. Notice that the structure of AESv is at all consistent with both the input stage and the output stage of AES-128. To experiment the selected second-order attacks, we designed, in standard C language, a masked implementation of AESv, here called *mAESv*, targeting an 8-bit microcomputer platform (8051 compatible) with clock frequency of 3.686 MHz. AESv and mAESv are respectively described in Appendix 5 and in Appendix 5. For convenience, the fragment of C code corresponding to the operations (two masked SBox operations at round R1) whose execution is captured in a power trace is reported in Figure 1. In the used measurement set-up, the execution of this code fragment takes $\approx$15.5 μsec and is within an acquisition window of 20 μsec (500 points at 25 MSamples/sec by a basic oscilloscope (8 bit resolution, 500 MHz bandwidth)).

We collected 20000 power consumption traces corresponding to a selected temporal portion of the encryption of 20000 random plaintexts. The 20000 traces, originally extending over 500 time samples, were then extended to 124750 modified time samples by the pre-processing step common to the two attacks to be compared.

The execution of the *single* correlation step involved in the *two masked table look-ups attack*, looking for two specific key bytes, produced the results summarized in Figure 2, which shows the bounds, taken over the 124750 available points, of all the 2^{16} correlation curves against the used number of modified power traces, where this number is taken at step 100. Notice that two couples of bounds are depicted in Figure 2: One couple, in black, is relative to the correlation curve corresponding to the right guess for the relevant couple of key bytes, here called the *right correlation curve*, whereas the other couple, in grey, is relative to the remaining correlation curves (the *wrong* ones). It results that, to make the right curve distinguishable from the wrong ones, at least 7200 modified power traces have to be considered (see the *crossing point* in Figure 2). The crossing point is defined as the point from which the ratio between the maximum value of the right curve and the maximum value among the wrong curves is always greater than 1. Namely, such a *right-to-wrong ratio* equals 1.035 and 1.471 for 10000 and 20000 power traces, respectively. A specific comparison, over the 124750 available points, between the right correlation curve (in black) and the bounds of the remaining wrong ones is shown in Figure 3, for the case where 20000 modified power traces are exploited. Observe that, as it was experimentally confirmed, the cost of this single correlation step is expected to be a linear combination of both the number of guesses (2^{16}), the number of points in the modified power traces, and the number of power traces.

The execution of the *first form of attack*, to be compared with the previous one, involved two correlation steps, with the second step depending on the first one.

The *first* correlation step, looking for an XOR difference between two specific key bytes, produced the results summarized in Figure 4, which shows the bounds, taken over the 124750 available points, of all the 2^8 correlation curves against the used number of modified power traces, where this number is taken at step 100. Again, two couples of bounds are depicted in Figure 4: One couple, in black, is relative to the correlation curve corresponding to the right guess for the XOR between the relevant couple of key bytes (the *right correlation curve*), whereas the other couple, in grey, is relative to the remaining correlation curves (the *wrong* ones). It results that, to make the right curve distinguishable from the remaining wrong ones, at least 3900 modified power traces have to be considered (the *crossing point* in Figure 4 is defined as before). The relevant right-to-wrong ratio, which is defined as before, equals 1.082 and 1.196 for 10000 and 20000 power traces, respectively (see also the discussion below). A specific comparison, over the 124750 available points, between the right correlation curve (in black) and the bounds of the remaining wrong ones is shown in Figure 5, for the case where 20000 modified power traces are exploited. Observe that, as it was experimentally confirmed, the cost of this first correlation step is expected to be a linear combination of both the number of guesses (2^8), the number of points in the modified power traces, and the number of power traces.

Some of the effects announced in Section 3.3 were practically observed. Figure 6 and Figure 7 show how Figure 4 changes when removing from the set of the wrong guesses the ones corresponding, respectively, to XOR differences at Hamming distance 1 (8 guesses) and Hamming distances 1 and 2 (36 guesses) from the right one. Moreover, we compared the amplitude, c, of the correlation peak corresponding to the right guess with both the average amplitudes, $c_{1,a}$ and $c_{2,a}$, and the maximum amplitudes, $c_{1,max}$ and $c_{2,max}$, of the correlation peaks corresponding, respectively, to the wrong guesses at Hamming distance 1 and at Hamming distance 2 from the right one. For 10000 traces, we have $c_{1,a}/c = 0.7659$ and $c_{1,max}/c = 0.9245$, wheeas $c_{2,a}/c = 0.5703$ and $c_{2,max}/c = 0.7484$. For 20000 traces, we have $c_{1,a}/c = 0.7524$ and $c_{1,max}/c = 0.8360$, whereas $c_{2,a}/c = 0.5113$ and $c_{2,max}/c = 0.6569$. Observe that the computed ratios $c_{1,a}/c$ and $c_{2,a}/c$ seem to agree with the anticipation in Section 3.3 (see (3)). Notice also that the above values for $c_{1,max}/c$ are the inverse of the right-to-wrong ratios, 1.082 and 1.196, given before for 10000 and 20000 traces, respectively.

Finally, observe that we did not explore the possibility, described in Section 3.3, of reducing the cardinality of the set of guesses from 2^8 to 2^7. One effect of this is visible in Figure 5, where the companion curve of the right curve constraints the bounds for the wrong curves, among which it is included, to show, locally, a behavior which is the opposite of the one shown by the right curve.

The *second* correlation step, looking for a couple of key bytes having a given XOR difference, produced the results summarized in Figure 8, which shows the bounds, taken over the 124750 available points, of all the 2^8 correlation curves against the used number of modified power traces, where this number is taken at step 100. Again, two couples of bounds are depicted in Figure 8: One couple, in black, is relative to the correlation curve corresponding to the right guess for the relevant couple of key bytes (the *right correlation curve*), whereas the other couple, in grey, is relative to the remaining correlation curves (the *wrong* ones). It results that, to make the right curve distinguishable from the remaining wrong ones, at least 6200 modified power traces have to be considered (the *crossing point* in Figure 8 is defined as before). The relevant right-to-wrong ratio, which is defined as before, equals 1.035 and 1.748 for 10000 and 20000 power traces, respectively. A specific comparison, over the 124750 available points, between the right correlation curve (in black) and the bounds of the remaining wrong ones is shown in Figure 9, for the case where 20000 modified power traces are exploited. Observe that, as it was experimentally confirmed, the cost of this second correlation step is, as the first one, expected to be a linear combination of both the number of guesses (2^8), the number of points in the modified power traces, and the number of power traces.

```
mState[ u ] = mSBox[ mState[ u ] ] ;
tmp = mState[ t ] ;
mState[ t ] = mSBox[ mState[ v ] ] ;
```

Fig. 1. C code fragment corresponding to two masked SBox operations at round R1

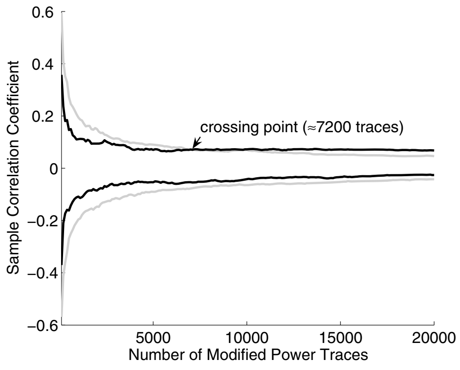

Fig. 2. Previous attack: Bounds for 2^{16} correlation curves (right one in black)

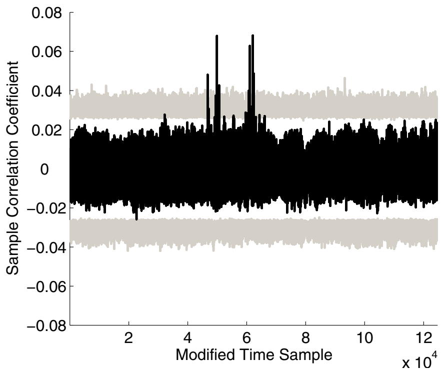

Fig. 3. Previous attack: Right correlation curve (in black) and bounds for wrong ones (20000 traces)

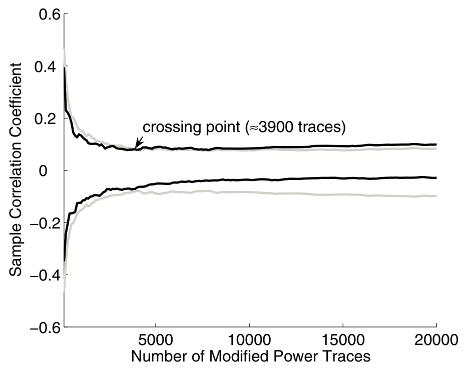

Fig. 4. First step of new attack: Bounds for 2^8 correlation curves (right one in black)

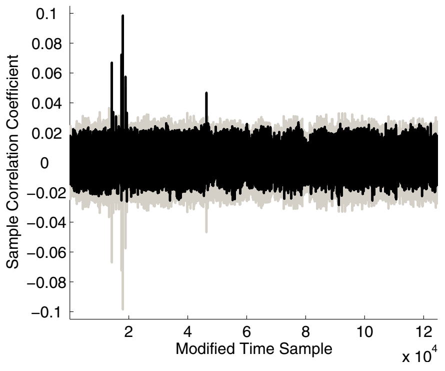

Fig. 5. First step of new attack: Right correlation curve (in black) and bounds for wrong ones (20000 traces)

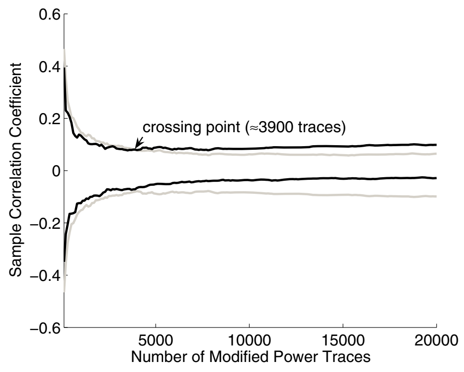

Fig. 6. First step of new attack: Bounds for 248 selected correlation curves (right one in black)

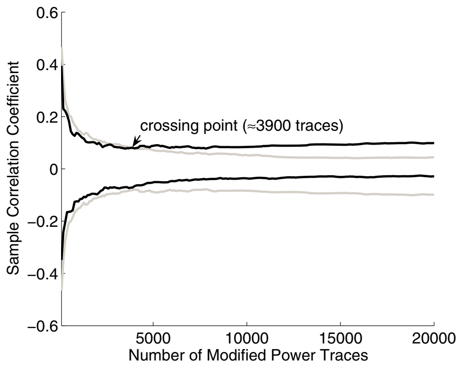

Fig. 7. First step of new attack: Bounds for 220 selected correlation curves (right one in black)

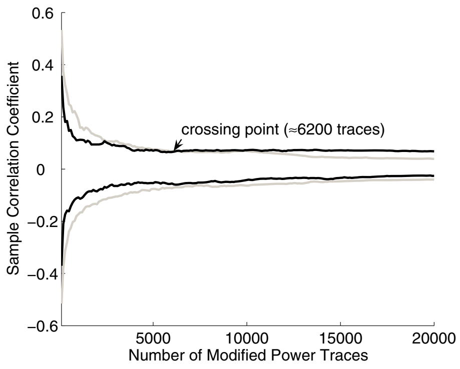

Fig. 8. Second step of new attack: Bounds for 2^8 correlation curves (right one in black)

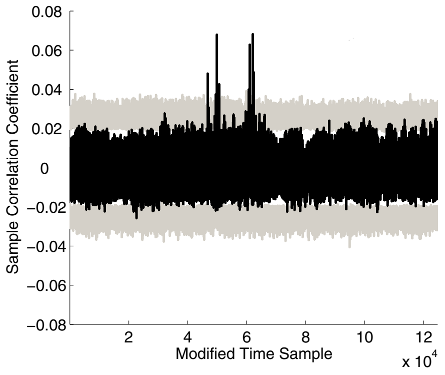

Fig. 9. Second step of new attack: Right correlation curve (in black) and bounds for wrong ones (20000 traces)

5 Conclusions

We have proposed a variant of a published second-order power analysis attack that, with reduced complexity, can produce the same result as the original one. Based on evidence produced by a basic measure environment, we have also shown the results of an experimental comparison between the two approaches.

References

1. Oswald, E., Mangard, S., Herbst, C., Tillich, S.: Practical Second-Order DPA Attacks for Masked Smart Card Implementations of Block Ciphers. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 192–207. Springer, Heidelberg (2006)
2. Daemen, J., Rijmen, V.: The design of Rijndael: The wide trail strategy explained. Springer, New York (2000)
3. Quisquater, J.-J., Koeune, F.: Side-channel attacks: state-of-the-art. In: CRYPTREC 2002 (2002), `http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/1047_Side_Channel_report.pdf`
4. Koeune, F., Standaert, F.: A Tutorial on Physical Security and Side-Channel Attacks. In: Aldini, A., Gorrieri, R., Martinelli, F. (eds.) FOSAD 2005. LNCS, vol. 3655, pp. 78–108. Springer, Heidelberg (2005)
5. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
6. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004)
7. Messerges, T.S.: Securing the AES Finalists Against Power Analysis Attacks. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 150–164. Springer, Heidelberg (2001)
8. Waddle, J., Wagner, D.: Towards Efficient Second-Order Power Analysis. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 1–15. Springer, Heidelberg (2004)

Appendix A: AES-128 Encryption Algorithm

We shortly recall the specifications of AES-128 encryption algorithm, corresponding to the *Advanced Encryption Standard* for 128-bit keys [2]. Apart from *KeyExpansion* (generating the needed 16-byte *round keys* from the given 16-byte key), the basic operations used in AES-128 encryption are: *AddRoundKey* (an XOR between a 16-byte *data* operand and a 16-byte *key* operand), *SubBytes* (where an invertible byte transformation called *SBox* is applied to all the bytes of a 16-byte operand), *ShiftRows* (a byte permutation applied to a 16-byte operand), and *MixColumns* (an invertible transformation consisting in 1. Dividing a 16-byte operand into 4 4-byte operands called *columns* and 2. *mixing* each of these columns, where mixing a column consists in computing, by a specific linear combination of the 4 bytes in the column, a new value for each of the 4 bytes.). The 16-byte ciphertext corresponding to given 16-byte plaintext P and key K is produced by 11 *transformation rounds*, each controlled by a specific round key. Here, the i-th round and the i-th round key are denoted, respectively, by Ri and $K^i, i = 0, 1, \ldots, 10$. Recall that $K^0 = K$. R0 consists of just the AddRoundKey operation with data operand P and key operand K^0. Ri, $1 \leq i \leq 9$, logically consists in cascading SubBytes, ShiftRows, MixColumns, and AddRoundKey with key operand K^i. R10 logically consists in cascading SubBytes, ShiftRows, and AddRoundKey with key operand K^{10}.

Appendix B: First-Order Correlation Power Analysis

A straightforward implementation of AES-128 (either software or hardware) is expected to be susceptible to attacks based on *Power Analysis*. We give here an example of a (*first-order*) attack based on *Correlation Power Analysis*(CPA) [6].

We suppose a sufficient number N of random plaintexts, all encrypted by the relevant implementation of AES-128 under the same unknown key $K = k_0, k_1, \ldots, k_{15}$, are given. We further suppose that the power traces corresponding to these encryptions are also given and that any power trace covers the same L time samples.

Under the hypothesis that, in the attacked implementation, the power consumption due to the processing of data d can be predicted by the Hamming weight of d, any key byte can be recovered by the following example *correlation trial*, which targets a single output from SBox at round R1.

First, a value g is guessed for the selected key byte k_i, $i \in \{0, 1, \ldots, 15\}$. Based on g and the N plaintexts, a *logical vector*, A_g, corresponding to the selected key byte is then generated as follows. Given the n-th plaintext ($1 \leq n \leq N$), by using its i-th byte, p_i, we form the n-th entry of A_g as SBox$(p_i \oplus g)$ (recall that $K^0 = K$). From A_g, a corresponding *prediction vector*, B_g, is generated by taking the Hamming weight of each entry of A_g.

From the N power traces, for each available time sample r ($1 \leq r \leq L$), we extract N values forming the *physical vector*, $F[r]$, and compute the sample correlation coefficient between $F[r]$ and B_g, so producing the *correlation curve* Q_g covering L time samples.

We repeat the computation of the correlation curve for all possible guess values for k_i, so getting 2^8 correlation curves. The recognition of the *right guess* $g = k_i$ relies on the possibility of distinguishing Q_{k_i} from the remaining curves, corresponding to *wrong guesses*. This is usually done by searching for the correlation curve exhibiting the peaks with maximal (absolute) values.

The presented attack is said to be based on a *first-order* analysis, since the available samples of the relevant power traces are singly exploited, each independent of each other.

Appendix C: Thwarting First-Order Correlation Power Analysis by Masking

Against power analysis attacks a countermeasure called *masking* is usually adopted. Here, we consider the *Boolean masking* described in [7], which aims at thwarting the feasibility of first-order attacks by replacing *sensitive* data by *masked* data. Namely, data d is replaced by $d \oplus r$, r being a random *mask*, and each mask used for data randomization is assumed to be unknown to attackers. A *masked implementation* of a given algorithm is then one which transforms given input values into proper output values under the constraint of producing each intermediate value in a *masked* form. The objective of such an implementation is to make any intermediate value unrecognizable by a first-order processing of the power consumption of the corresponding device. A masked implementation can be characterized by both the number of independent mask bits it uses and the way these are used.

Appendix D: Reference Encryption Algorithm

AESv is formed by cascading the three standard transformation rounds R0, R1, and R10 of AES-128. AESv does not include the KeyExpansion operation and uses three independent 16-byte keys for its three transformation rounds. Notice that, as for the results to be shown here, AESv well represents both the input stage and the output stage of AES-128.

Namely, for given plaintext P and keys K_1, K_2, and K_3, a standard round R0 is first applied with data operand P and key operand K_1, which produces the *current* ciphertext C_1. Then, using K_2 as the key operand for AddRoundKey, the standard round R1 is applied, which produces the *current* ciphertext C_2. Finally, using K_3 as the key operand for AddRoundKey, the standard round R10 is applied, which produces the *final* ciphertext C.

Appendix E: Masked Implementation of the Reference Encryption Algorithm

mAESv uses the Boolean masking technique [7] to gain resistance to first-order power analysis attacks (observe that such a resistance was exhibited by mAESv in several experimental attacks).

Apart from plaintext (16 bytes) and keys (48 bytes), mAESv depends on a *key mask* M_K (16 bytes), an SBox *input mask* m_{in} (1 byte), an SBox *output mask* m_{out} (1 byte), and an *auxiliary mask* m_{aux} (1 byte) (see the description of MixColumns′ given below). To produce any intermediate masked data, it is always used one of a few specific linear combinations of the above masks. Observe that mAESv (briefly presented in the sequel) is not intended to provide an efficient implementation of the Boolean masking technique.

A basic role in mAESv is played by the masked byte transformation *SBox′*, transforming masked input bytes into masked output bytes. As proposed in [7], SBox′ depends on given masks m_{in} and m_{out}. On input $x \oplus m_{in}$, SBox′ produces $y \oplus m_{out}$, where y is the SBox output on input x ($\text{SBox}'(x \oplus m_{in}) = \text{SBox}(x) \oplus m_{out}$). In mAESv, a single SBox′, implemented as a table, supports all the applications of the associated masked transformation *SubBytes′*.

A *mask adaptation* block is used to change the mask acting on a given masked operand. mAESv uses two mask adaptation blocks, *MA1* and *MA2*, defined for 16-byte operands. For given masks M and M_n and masked input $X \oplus M$, these blocks produce $X \oplus M_n$. Namely, in MA1, $M = M_K$ and $M_n = m_{in}^{(16)}$, whereas, in MA2, $M = M_K \oplus m_{out}^{(16)}$ and $M_n = M_K$, where $x^{(16)}$ denotes a 16-byte operand formed by repeating 16 times the byte x.

A *mask removal* block is used to remove the mask acting on a given masked operand. mAESv uses one mask removal block, *MR*, defined for 16-byte operands. For given mask M and masked input $X \oplus M$, this block produces X. MR is used to *unmask* the masked ciphertext $C \oplus M_K$.

Observe that no changes are needed in ShiftRows. MixColumns is instead modified into its masked version MixColumns′. On masked input $X' = X \oplus m_{out}^{(16)}$, MixColumns′ produces $Y' = Y \oplus m_{out}^{(16)}$, where Y is the MixColumn output on input X. MixColumns′ makes use of the auxiliary mask m_{aux} to guarantee that no unmasked *sensitive* data are produced during the transformation of X' into Y'.

mAESv works as follows, based on plaintext P and masked keys K_1', K_2', and K_3', with $K_i' = K_i \oplus M_K$, $i = 1, 2, 3$. A first AddRoundKey between P and K_1' produces $C_1 \oplus M_K$. After that, the *masked round* formed by cascading MA1, SubBytes′, ShiftRows, MixColumns′, AddRoundKey–with key operand K_2', and MA2 is applied, which produces $C_2 \oplus M_K$. Then, the masked round formed by cascading MA1, SubBytes′, ShiftRows, AddRoundKey–with key operand K_3', and MA2 is applied, which produces $C' = C \oplus M_K$. Finally, C' is unmasked by MR to the final ciphertext C.

Each call of mAESv is preceeded by a preliminary phase where, starting from fresh masks (19 random bytes), the needed masked keys and masked transformations are prepared.