Horizontal and Vertical Side-Channel Attacks against Secure RSA Implementations

Aurélie Bauer, Eliane Jaulmes, Emmanuel Prouff, and Justine Wild

ANSSI, 51, Boulevard de la Tour-Maubourg, 75700 Paris 07 SP, France
firstname.name@ssi.gouv.fr

Abstract. Since the introduction of side-channel attacks in the nineties, RSA implementations have been a privileged target. A wide variety of countermeasures have been proposed and most of practical attacks are nowadays efficiently defeated by them. However, in a recent work published at ICICS 2010, Clavier *et al.* have pointed out that almost all the existing countermeasures were ineffective if the attacks are performed with a *modus operandi* called *Horizontal*. Such attacks, originally introduced by Colin Walter at CHES 2001, involve a single observation trace contrary to the classical attacks where several ones are required. To defeat Horizontal attacks, the authors of the ICICS paper have proposed a set of new countermeasures. In this paper, we introduce a general framework enabling to model both Horizontal and classical attacks (called *Vertical*) in a simple way. This framework enables to enlighten the similarities and the differences of those attack types. From this formalism, we show that even if Clavier *et al.*'s countermeasures thwart existing attacks, they do not fully solve the leakage issue. Actually, flaws are exhibited in this paper and efficient attacks are devised. We eventually propose a new countermeasure.

1 Introduction

Side-Channel Analysis (SCA) is a cryptanalytic technique that consisting in exploiting the *side channel leakage* (*e.g.* the power consumption, the electromagnetic emanations) produced during the execution of a cryptographic algorithm embedded on a physical device. It uses the fact that this leakage is statistically dependent on the intermediate variables that are processed. Some of these variables are *sensitive* in the sense that they are related to secret data, thus reaching information on them enables efficient key recovery attacks [3, 9, 15]. Since the publication of the first attacks, many papers describing either countermeasures or attack improvements have been published (see [3,4,16] for example). Among these improvements, *higher-order SCA* are of particular interest. They extend the initial concept by considering a set of several instructions instead of a single one and circumvent many countermeasures proposed in the literature (*e.g.* [4, 10]). Another significant improvement has been proposed initially by Walter [19] and then studied more deeply by Clavier *et al.* in [5]. Essentially, it consists in a new *modus operandi* called *Horizontal*, in which sensitive information is extracted from a single measurement split into several parts. It differs from the classical *Vertical* mode where information is obtained from different algorithm executions. Horizontal mode applies when the same guessable sub-part of a secret is involved in many internal operations during the overall algorithm processing. This is particularly the case for RSA implementations where the exponentiation is composed of small multiplications that all depend on the same secret exponent sub-part. As noticed in [5, 19], a striking point is that classical countermeasures (*e.g.* the exponent or the message blinding techniques), efficient in the Vertical context, turn out to be almost ineffective in the Horizontal one. This makes the construction of new appropriate countermeasures a real issue for the security of embedded implementations of RSA and similar algorithms. An attempt to design such countermeasures thwarting both types of attacks has been done in [5]. Actually, we show in this paper that these countermeasures (Sections 4 and 5) do not completely remove the leakage, even if they thwart some attacks. To exhibit the flaws and, eventually, to propose new countermeasures (Section 6), we first introduce a framework enabling to formally study the resistance of an implementation against side channel attacks in both Horizontal and Vertical modes (Sections 2 and 3). This framework could be used to further analyse the security of other algorithms' implementations than RSA ones. On the other hand, the countermeasure proposed in this paper is a first step towards an efficient and effective security against Horizontal attacks. The definition of alternative countermeasures is a new open avenue for further research.

2 A Comprehensive Study of Side-Channel Analyses

In the following, a general framework is introduced which enables to describe most of the existing attacks in a similar way and to identify their core differences (actually the leakage pre-treatment, the leakage model and the statistical test).

2.1 A General Framework for Side-Channel Analyses

Notations. A realization of a random variable X is referred to as the corresponding lower-case letter x. A *sample* of observations of X is denoted by (x) or by $(x_i)_i$ when an indexation is needed. In this case, the global event is sumed up as $(x) \hookleftarrow X$. The i^{th} coordinate of a variable X (resp. x), viewed as a vector, is denoted by $X[i]$ (resp. $x[i]$). As usual, the notation $\mathbb{E}[X]$ refers to the mean of X. For clarity reasons we sometimes use the notation $\mathbb{E}_X[\cdot]$ to enlighten the variable over which the mean is computed.

All the attacks below target a variable $Z(k, X)$ defined as the output of a specific computation (*e.g.* a multiplication) performed by the device and parametrized by a secret sub-part k and a public variable X[1]. In the following, we shall use Z instead of $Z(k, X)$ if there is no ambiguity on k and X.

To recover information on k, the attacks are performed on a sample of observations related to the processing of Z by the device. Each of those observations, such as power consumption, electromagnetic emanations, and so on, is usually composed of several physical measurements corresponding to leakages at different times t_i. It can hence be viewed as a realization of a multivariate random variable $\boldsymbol{L}$ whose coordinates $\boldsymbol{L}[i]$ satisfy:

$$\boldsymbol{L}[i] = \varphi_i\big(Z\big) + \beta_i \ , \tag{1}$$

[1] We shall sometimes need to consider the known value as a pair of variables: in this case we will use the notation (X, Y) instead of X.

where φ_i only depends on the device behaviour at time t_i during the processing of Z and β_i is an independent Gaussian noise with zero mean and standard deviation σ_i. The function φ_i is *a priori* unknown. The index i will be sometimes omitted. In this case, it is assumed that the same function is associated to all the time indices.

An SCA is based on the *Hypothesis Testing principle* [13]. To make this test, a set of *prediction values* $\boldsymbol{h}_j$ are deduced from each hypothesis $\hat{k}$ on k and from the sample of implementation inputs (x_j) corresponding to the observations. This step involves a *leakage model function* $\mathbf{m}$ that must have been priorly chosen by the attacker (for instance based on its knowledge on the attacked device architecture). With this model function, the prediction values $\boldsymbol{h}_j$ are built s.t. $\boldsymbol{h}_j = \mathbf{m}(z(\hat{k}, x_j))$. Eventually, the adversary uses a distinguisher Δ to compare the $\boldsymbol{h}_j$ with the observations $\boldsymbol{l}_j \hookleftarrow \boldsymbol{L}|X = x_j$.

The overall set of SCA is usually split in two subclasses. The first one, called *simple Side-Channel Analysis*, contains all attacks where observations only need to be done on a single value of the public input parameter (this implies that all the x_j are equal to a same value, say x). This set contains S-PA [14], S-EMA [8,18] or S-TA (Timing Analysis) [14]. The second subclass, called *advanced Side-Channel Analysis*, includes attacks where observations of the targeted internal processing must be done for several public input parameters. In particular, it contains *univariate SCA* attacks such as DPA [15], CPA [3] or MIA [9] and *multivariate SCA* attacks such as HO-DPA [15, 17] or HO-MIA [1]. We give hereafter a more formal description of those two subclasses.

Simple SCA. The class of simple SCA includes all Vertical or Horizontal SCA where the adversary makes observations for a single algorithm input. Table 1 provides a description of a simple Side-Channel Analysis[2].

1. Choose a value x for X.
2. Measure a sample $(\boldsymbol{l}_j)_j \hookleftarrow (\boldsymbol{L}|X = x)$ of N leakages.
3. Select a distinguisher Δ and choose a model function $\mathbf{m}$.
4. For each hypothesis $\hat{k}$ on k, compute $\boldsymbol{h} = \mathbf{m}(z(\hat{k}, x))$.
5. For each $\hat{k}$, compute $\Delta[\hat{k}] = \Delta\big[(\boldsymbol{l}_j)_j, \boldsymbol{h}\big]$.
6. Deduce from $\Delta[\cdot]$ information on k.

Table 1. Simple Side-Channel Analysis

Remark 1. In theory, simple SCA may be conducted with a single observation. In practice however, it is often necessary to use several observations of the processing for the same variable x in order to reduce the noise impact. In this case, the statistical distinguisher Δ may for instance involve a preliminary step consisting in averaging the observations sample.

[2] In contexts where the adversary is not allowed to choose the algorithm input but knows it, the first step just aims at fixing the input value for the rest of the attack.

Advanced SCA. All the attacks where observations must involve different inputs belong to the advanced SCA category. This kind of attacks follows the outlines given in Table 2.

1. Get N measurements $(\boldsymbol{l}_j, x_j)_j \hookleftarrow (\boldsymbol{L}, X)$.
2. Select a distinguisher $\varDelta$ and choose a model function $\mathbf{m}$.
3. For each hypothesis $\hat{k}$ on k build a set of predictions h_j such that $\boldsymbol{h}_j = \mathbf{m}(z(\hat{k}, x_j))$.
4. For each $\hat{k}$, compute $\varDelta[\hat{k}] = \varDelta\Big[(\boldsymbol{h}_j)_j, (\boldsymbol{l}_j)_j\Big]$
5. Deduce from $\varDelta[\cdot]$ information on k.

Table 2. Advanced Side-Channel Analysis

Remark 2. Depending on the statistical treatment processed by the distinguisher, the latter one may include a particular leakage post-processing $\mathcal{E}$. This post-treatment may be used to select some particular points in the leakage traces and, possibly, to combine them. For instance, in a second-order advanced SCA involving the mutual information as distinguisher, the function $\mathcal{E}$ can be defined such that $\mathcal{E}\big(\boldsymbol{L}\big) = \big(\boldsymbol{L}[p], \boldsymbol{L}[q]\big)$ for some constant indices (leakage times) p and q. In a second-order advanced SCA involving the correlation coefficient as distinguisher, $\mathcal{E}$ may be defined such that $\mathcal{E}\big(\boldsymbol{L}\big) = (\boldsymbol{L}[p] - \mathbb{E}(\boldsymbol{L}[p])) \cdot (\boldsymbol{L}[q] - \mathbb{E}(\boldsymbol{L}[q]))$. Moreover, the choice of the model function must be done in accordance with the distinguisher (see *e.g.* [17] and [9]).

2.2 Leakage Measurements and Observations

In the literature, two main ways have been defined to get the observations $\boldsymbol{l}_j$ during the first step of the attacks in Tables 1 and 2. The first method simply consists in executing the implementation several times (with the same input in simple SCA or with several ones in advanced SCA) and in defining $\boldsymbol{l}_j$ as the observation related to the j^{th} algorithm execution. Those attacks are called *Vertical*. The second method refers to attacks where a single execution is needed and where each $\boldsymbol{l}_j$ corresponds to the observation of a processing at a different time period during this execution. In this case, the index j refers to the time period. The underlying assumption is that all the observations rely on the same internal calculus of $Z(k, X)$, parametrized by a same secret k and different known values x_j in advanced SCA, or a constant one x in simple SCA. Attacks corresponding to this *modus operandi* are called *Horizontal*. Figure 1 illustrates the notations and the differences between the two *modus operandi*.

All the attacks discussed in Section 2.1 can be either Vertical or Horizontal[3]. Even if the Horizontal or Vertical characteristic of an SCA has no impact on

[3] Possibly, the observations acquisition phase may mix horizontal and vertical techniques. In this case, the attack will be termed *Rectangle*.

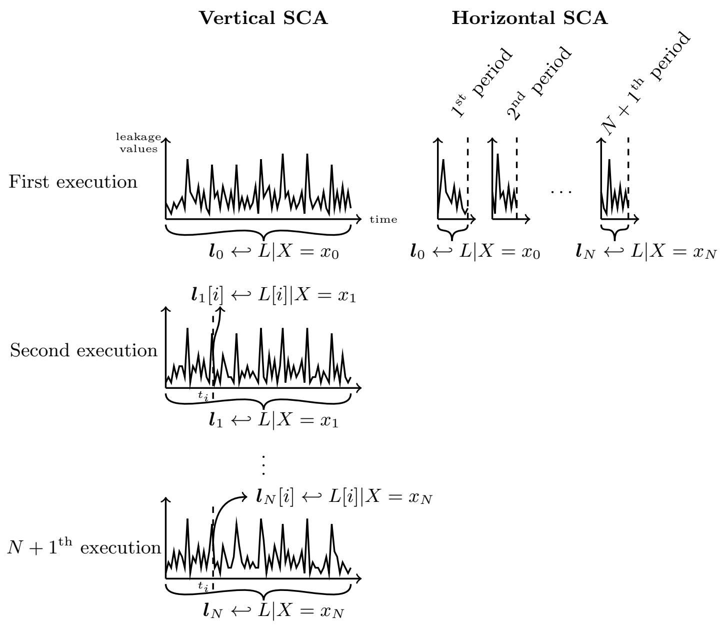

Vertical SCA
Horizontal SCA
1st period
2nd period
N + 1th period
leakage values
time
First execution
Second execution
N + 1th execution
$l_0 \hookleftarrow L|X = x_0$
$l_0 \hookleftarrow L|X = x_0$
$l_N \hookleftarrow L|X = x_N$
$l_1[i] \hookleftarrow L[i]|X = x_1$
t_i
$l_1 \hookleftarrow L|X = x_1$
$l_N[i] \hookleftarrow L[i]|X = x_N$
$l_N \hookleftarrow L|X = x_N$
. . .

Fig. 1. Vertical and Horizontal SCA

the attack steps themselves (as described in Tables 1 and 2), it impacts the implementation security analysis. Indeed, we will see in Section 4 that a countermeasure may become ineffective when going from one category of attacks to another one. We illustrate this in the context of secure RSA implementations.

2.3 Taxonomy

Based on the discussions conducted in previous sections, we propose here a general taxonomy for simple and advanced side-channel attacks. To name an attack we propose to use the convention [XXX]-[YYY]-[ZZZ] where:

- XXX equals either S for simple SCA or is a reference to the statistical tool for advanced SCA (*e.g.* C for Correlation, MI for Mutual Information, ML for Maximum Likelihood, LR for Linear Regression, etc.). In case of multivariate SCA, we propose to pad the order/dimension followed by O at the left of the distinguisher letter.
- YYY is an acronym referring to the leakage type; PA for Power Analysis, EMA for Electromagnetic Analysis, TA for Timing Attacks, etc.
- ZZZ is optional and may be used to specify if the attack is profiled or not. In this case, ZZZ is replaced by P (for Profiling) or UnP (for UnProfiling). For instance, Template attack is a ML-PA-P attack.

Of course, all those attacks can be applied either on a Vertical or Horizontal mode. Figure 2 illustrates the taxonomy for some existing attacks.

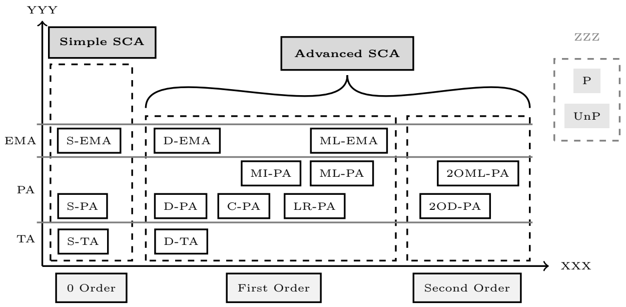


Fig. 2. Side-Channel Attacks

In the following sections, we focus on Horizontal SCA in the RSA context. We will recall the existing attacks and will discuss about the effectiveness of the Vertical SCA countermeasures against Horizontal SCA.

3 RSA Context

3.1 Operation Flows in RSA Exponentiations

The execution flow of an RSA implementation is usually viewed as the succession of only two different operations: a modular squaring and a modular multiplication respectively denoted by O_0 and O_1. For convenience, we will assume that both operations are bivariate and defined such that $\mathrm{O}_i(X,Y) = X^iY^{\bar{i}} \cdot Y$. For instance, the left-to-right *Square and Multiply* algorithm parametrized by a d-bit long secret k (the most significant bit is assumed to be equal to 1) and a public modulus n, operating on a message X can be associated to the following sequence:

$$Y \leftarrow \mathrm{O}_{f(0)}(X,Y), Y \leftarrow \mathrm{O}_{f(1)}(X,Y), \cdots, Y \leftarrow \mathrm{O}_{f(N)}(X,Y) \ , \tag{2}$$

where Y is the updated intermediate result (initially set to 1), N denotes the value $d + \mathrm{HW}(k)$ and the binary function f is defined as:

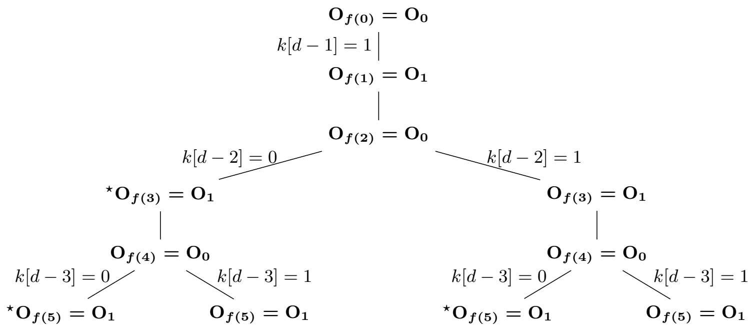
$$f(j) = \begin{cases} j & \text{, if } j = 0,1 \\ \overline{f(j-1)} \cdot k\big[d-1-\sum_{i=0}^{j-2} \overline{f(i)}\big] & \text{, otherwise.} \end{cases} \tag{3}$$

The operations' flow in (2) is illustrated on Figure 3.

To defeat simple SCA against RSA implementations, a classical countermeasure is to insert dummy multiplications in order to have a *regular* algorithm. This leads to the definition of the so-called *Square and Multiply Always* algorithm which may be associated with the sequence below where each square is followed by a multiplication whatever the secret k:

$$Y_1 \leftarrow \mathrm{O}_0(X,Y_1), Y_{k[d-1]} \leftarrow \mathrm{O}_1(X,Y_1), Y_1 \leftarrow \mathrm{O}_0(X,Y_1), Y_{k[d-2]} \leftarrow \mathrm{O}_1(X,Y_1), \cdots,$$
$$Y_1 \leftarrow \mathrm{O}_0(X,Y_1), Y_{k[0]} \leftarrow \mathrm{O}_1(X,Y_1) \ , \tag{4}$$

with Y_0 denoting a garbage variable and Y_1 a working register initially set to 1 (and playing the same role as Y in (2)).


$\mathbf{O}_{f(0)} = \mathbf{O}_0$
$k[d-1] = 1$
$\mathbf{O}_{f(1)} = \mathbf{O}_1$
$\mathbf{O}_{f(2)} = \mathbf{O}_0$
$k[d-2] = 0$
$k[d-2] = 1$
$^{\star}\mathbf{O}_{f(3)} = \mathbf{O}_1$
$\mathbf{O}_{f(3)} = \mathbf{O}_1$
$\mathbf{O}_{f(4)} = \mathbf{O}_0$
$\mathbf{O}_{f(4)} = \mathbf{O}_0$
$k[d-3] = 0$
$k[d-3] = 1$
$k[d-3] = 0$
$k[d-3] = 1$
$^{\star}\mathbf{O}_{f(5)} = \mathbf{O}_1$
$\mathbf{O}_{f(5)} = \mathbf{O}_1$
$^{\star}\mathbf{O}_{f(5)} = \mathbf{O}_1$
$\mathbf{O}_{f(5)} = \mathbf{O}_1$

*O denotes a dummy operation

Fig. 3. First Loops of Square and Multiply Always algorithm

Remark 3. An improved version of the Square and Multiply Always algorithm, based on the *Montgomery Ladder* trick [11], is often preferred as it is more resistant to the so-called *Safe-Error* attacks [20]. In this version, there is no garbage variable and $Y_{k[i]}$ is used in the subsequent operation even if $k[i]$ is equal to 0. We point out here that this version and the Square and Multiply Always algorithm have exactly the same vulnerabilities with respect to advanced SCA. Indeed, in both cases, each loop iteration in the exponentiation processes the same operations and only the memory manipulation is different.

The granularity of the sequence descriptions in (2) and (4) is not fine enough to investigate advanced SCA. Those attacks indeed require the identification of intermediate results depending on small sub-parts of the input parameters. To enable such an identification, the execution flows must be rewritten as a succession of operations on ω-bit words[4]. Let us assume that modular squarings and multiplications are implemented with the schoolbook multiplication called *Long Integer Multiplication* (LIM for short) followed by a Barrett reduction (for self-contentedness we recall the LIM algorithm in Appendix A). The variables X and Y are then represented as base-2^{ω} vectors[5] $(X[a])_{0\leq a\leq t}$ and $(Y[b])_{0\leq b\leq t}$ with $t = \lfloor \frac{\log_2(X)}{\omega} \rfloor$. After this rewriting, we get the following decomposition of an operation O_i, where we only exhibited the intermediate base-2^{ω} multiplications $Z[a,b] \leftarrow X[a]^{i} Y[a]^{\bar{i}} \cdot Y[b]$:

3.2 Attacks Targets

When applied against the operations' sequences (2) or (4), advanced SCA aim at recovering all the bits of k one after another from the left to the right. Here, we assume that the most significant bit of k is 1 and we show in this section and

[4] The value ω typically depends on the device architecture and is usually equal to 8, 16 or 32.

[5] Without loss of generality, we assume that X and Y have the same length t. This possibly implies that the binary representation of one of them has been left-padded with 0s.

$$\begin{array}{cccc}
Z[0,0]\leftarrow X[0]^iY[0]^{\bar{i}}\cdot Y[0], & Z[0,1]\leftarrow X[0]^iY[0]^{\bar{i}}\cdot Y[1], & \cdots, & Z[0,t]\leftarrow X[0]^iY[0]^{\bar{i}}\cdot Y[t] \\
Z[1,0]\leftarrow X[1]^iY[1]^{\bar{i}}\cdot Y[0], & \cdots & \cdots, & Z[1,t]\leftarrow X[1]^iY[1]^{\bar{i}}\cdot Y[t] \\
\vdots & \vdots & \vdots & \vdots \\
Z[t,0]\leftarrow X[t]^iY[t]^{\bar{i}}\cdot Y[0], & \cdots & \cdots, & Z[t,t]\leftarrow X[t]^iY[t]^{\bar{i}}\cdot Y[t]
\end{array} .$$

Fig. 4. Decomposition of the operation $\mathrm{O}_i(X,Y) = X^i \cdot Y^{\bar{i}} \cdot Y$

the next one how advanced SCA succeed in recovering the value of $k[d-2]$. The attacks may further be repeated to fully recover k. To simplify the notations, we denote the secret bit $k[d-2]$ by s. In a classical left-to-right Square and Multiply algorithm, s is involved for the first time in the operation $\mathrm{O}_{f(3)}$. In this case, one can develop the operands of $\mathrm{O}_{f(3)}$ in terms of s and X. Actually, according to (3) we have $f(3) = s$, which means that $\mathrm{O}_{f(3)}$ corresponds to the processing $X^2 \cdot X^{2-s}$ (*i.e.* $Y = X^2$ in Figure 4). In a left-to-right Square and Multiply Always algorithm, the value s impacts on the fifth operation. Indeed, depending on s, the result of the fourth operation has either been put into the working register or in the garbage register. As a consequence, the fifth operation (which is always a squaring O_0) corresponds to the processing $X^{2+s} \cdot X^{2+s}$ (*i.e.* $Y = X^{2+s}$ in Figure 4). Eventually, depending on the algorithm we deduce that the elementary base-2^ω multiplications $Z[a,b]$ satisfy[6]:

- Square and Multiply (operation $\mathrm{O}_{f(3)}$ in (2)):
$$Z[a,b] = X^2[a] \cdot X^{2-s}[b] \ . \tag{5}$$
- Square and Multiply Always (fifth operation in (4)):
$$Z[a,b] = X^{2+s}[a] \cdot X^{2+s}[b] \ . \tag{6}$$

Equations (5) and (6) show that each intermediate result $Z[a,b]$ depends on s. This implies that the observation $\boldsymbol{L}_{a,b}$ related to the manipulation of $Z[a,b]$ by the device leaks information on s. To exploit this leakage in a vertical advanced SCA, the pair of indices (a,b) is fixed and the observations are measured for **different** values $x \hookleftarrow X$ of the algorithm input. In an Horizontal advanced SCA, the observations are performed for a **single** value $x \hookleftarrow X$ but different pairs of indices $(a,b) \in [0;t] \times [0;t]$ (in the latter case, a and b are viewed as random variables and will be denoted by capital letters).

3.3 Horizontal Attacks

In this section, we are interested in Horizontal analyses such as the *Big Mac attack* [19] and the *Horizontal Correlation Analysis* [5]. For a fixed value $x \hookleftarrow X$ but various pairs (a,b), we assume that the adversary observes the device behavior $\boldsymbol{l}_{a,b}$ during the processing of the intermediate results $z[a,b]$.

[6] We alert the reader on the fact that, in this paper, we make a distinction between the notations $X^i[a]$ and $(X[a])^i$: the first one denotes the $(a+1)^{\text{th}}$ coordinate of the base-2^ω representation of the value X^i, whereas the second one denotes the rising at the power i of the $(a+1)^{\text{th}}$ coordinate of the base-2^ω representation of the value X.

Big Mac Attack. This attack is a *Collision Analysis*, designed in the case of the Square and Multiply algorithm when the adversary does not known the exponentiation input x. The principle consists in recovering the secret key k from the most significant bit to the least significant one. According to Equation (5), elementary operations involved in $\mathrm{O}_{f(3)}$ during the modular exponentiation can be either of the shape $x^2[a] \cdot x^2[b]$ when s equals 0, or of the form $x^2[a] \cdot x[b]$ when s equals 1. As a consequence, if the attacker is able to determine whether the leakage traces $\boldsymbol{l}_{a,b}$, involved in this operation, correspond to multiplications by $x^2[b]$ or by $x[b]$, then the value of s will easily be recovered. In order to make this distinction, the adversary performs a collision attack between the traces $\boldsymbol{l}_{a,b}$ corresponding to $\mathrm{O}_{f(3)}$ and the traces $\boldsymbol{l}'_{a,b}$ related to another multiplication involving x as operand (*e.g.* the operation $\mathrm{O}_{f(1)}$ which defines the multiplication of 1 by the input x in the Square and Multiply algorithm). To this purpose, the attacker uses for instance the average leakages $(\frac{1}{t+1}\sum_a \boldsymbol{l}_{a,b})_b$ and $(\frac{1}{t+1}\sum_a \boldsymbol{l}'_{a,b})_b$, and after selecting a distinguisher Δ, *e.g.* the *Euclidean Distance*, computes the value $\Delta((\frac{1}{t+1}\sum_a \boldsymbol{l}'_{a,b})_b, (\frac{1}{t+1}\sum_a \boldsymbol{l}_{a,b})_b)$ in order to validate or invalidate the hypothesis $s = 1$. As explained before, the Big-Mac attack has originally been described as a *Collision Analysis* for unknown exponentiation inputs and a non-regular Square and Multiply algorithm[7] (see bold notations on right-hand sided leaf in Figure 5).

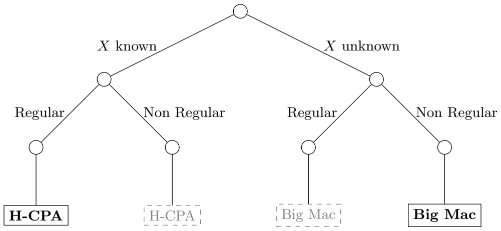


Fig. 5. Big-Mac attack and Horizontal CPA classification

Horizontal C-PA. Contrary to the previous attack, this one has been described in the context of *Atomic Square and Multiply* implementations, and also applies to the *Square and Multiply Always* algorithm (Sequence (4)) when the input x is known to the adversary. To recover the key-bit s corresponding to the variables $Z[a, b]$ defined in (6), the attacker involves a well-chosen model function (*e.g.* the Hamming weight) and for each key-bit hypothesis $\hat{s} \in \{0, 1\}$, computes the set of predictions $\boldsymbol{h}_{a,b} = \mathbf{m}(Z[a, b])$ (where s is replaced by $\hat{s}$ in (6)). Eventually, the Pearson coefficient ρ is chosen as distinguisher and the discrimination is done by processing $\rho[(\boldsymbol{h}_{a,b})_{a,b}, (\boldsymbol{l}_{a,b})_{a,b}]$. The applicability of this attack has been illustrated on Figure 5, see bold notations for the original description of the attack.

[7] This principle can be extended to *Sliding Windows* implementations, see Walter's original paper [19].

Extension of these attacks. Even if the Big Mac Attack has been initially introduced for unknown exponentiation inputs, it can of course be adapted to known entries. Indeed, another way to proceed consists in using the model function $\mathbf{m} : a, \hat{s} \mapsto \frac{1}{t+1}\sum_a \hat{\varphi}(x^2[a]\cdot x^{1+\hat{s}}[b])$ where $\hat{\varphi}$ is chosen according to the device specificities (*e.g.* $\hat{\varphi}$ equals to the Hamming weight function). In the framework proposed in Section 2, the *Big Mac Attack* can thus become an *Horizontal ED-PA* (with ED standing for *Euclidean Distance*). This could be illustrated on Figure 5 by adding Big Mac in each leaf of the left hand-sided sub-tree. In addition, one can also use Big Mac attack to target regular implementations, such as modular exponentiations using the Square and Multiply always algorithm. Indeed, Equation (6) shows that operation $\mathrm{O}_{f(4)}$ is either $x^2[a]\cdot x^2[b]$ when s equals 0, or $x^3[a]\cdot x^3[b]$ when s equals 1. In that case, the attacker can average on the second multiplications operands instead of the first one (as done in the previous attacks). This leads to the computation of the values $(\frac{1}{t+1}\sum_b \boldsymbol{l}'_{a,b})_a$ coming from operation $\mathrm{O}_{f(3)}$ (which corresponds to $x^2[a]\cdot x[b]$) and $(\frac{1}{t+1}\sum_b \boldsymbol{l}_{a,b})_a$ issued from $\mathrm{O}_{f(4)}$. From that point and as before, the attacker evaluates a distinguisher Δ (*e.g.* the Euclidean Distance) to determine whether the multiplication has been performed with $x^2[a]$ or with $x^3[a]$, which leads to recover s. Eventually, the same process can be applied to guess the following remaining bits of the secret key k. This extension of the Big Mac attack is illustrated on Figure 5 by the grey-dotted box.

Applying Horizontal C-PA from the Square and Multiply always implementation to the non-regular one is obvious (see also the grey-dotted box on the scheme).

As seen in this section, the Big Mac attack and the Horizontal C-PA can both be applied not only in their original contexts but against Square and Multiply and Square and Multiply always implementations. Their success indeed does not depend on the structure of the exponentiation algorithm. They can moreover be applied in both known input and unknown input modes. In the sequel, we use those observations to argue that the countermeasures proposed to defeat Horizontal C-PA, are in fact sensible to Big Mac like attacks.

4 Existing Countermeasures

The most popular countermeasures against Vertical advanced SCA is the *exponent blinding* and the multiplicative/additive *message blinding* (*e.g.* [6, 14]). The first countermeasure implies that all the observations in the adversary hands correspond to different secrets/exponents. The second countermeasure implies that no intermediate variable depends on the algorithm input. In the two cases, it becomes impossible to make predictions and the Vertical first-order advanced SCA fail.

The exponent blinding countermeasure perfectly illustrates that the effectiveness of a countermeasure may totally change when passing from Vertical to Horizontal contexts. Indeed, when the exponent randomization is applied, all the variables $Z[a, b]$ defined as in (5) or (6) will depend on the same masked bit $\tilde{s}$. As a consequence, the Horizontal advanced SCA described in Section 3 will succeed in recovering it. As the knowledge of the blinded exponent provides the adversary with the same capabilities as knowing s itself (*e.g.* it can produce the same signatures) the attack may be considered as successful. In [5], the authors also argue that message blinding thwart Horizontal attacks only if the bit-length λ of the random value R is greater than 32 bits. For smaller values of λ, an efficient attack is indeed exhibited. As a consequence of the exponent blinding ineffectiveness and of the message blinding inefficiency, there is a real lack of countermeasures against Horizontal attacks. This led Clavier *et al.* to propose the following three countermeasures [5]:

- *Blind Operands in LIM.* The first countermeasure proposed in [5] consists in applying a *full blinding* on the words $X[a]$ and $Y[b]$, *i.e.* to substitute in the LIM algorithm the operation $X[a] \cdot Y[b]$ by $(X[a] - R_1)(Y[b] - R_2) + R_1 \cdot Y[b] + R_2 \cdot X[a] - R_1 \cdot R_2$, where R_1 and R_2 are two ω-bit random values. For efficiency reasons, the authors propose to compute once the values $R_1 \cdot Y[b]$, $R_2 \cdot X[a]$ and $R_1 \cdot R_2$ and to store them. The complexity of this countermeasure is $(t+1)^2+2(t+1)+1$ ω-bit multiplications (the unprotected LIM requires $(t+1)^2$ multiplications) and $4\log_2(n) + 2\omega$ bits of additional storage, where n is the RSA modulus.
- *Randomize One Loop in LIM and Blind.* The second countermeasure in [5] starts from the first one and mixes it with a randomization of the order in which the words $X[a]$ are involved in the LIM. This method consists in using a permutation vector applied to the words $X[a]$ and in masking the words $Y[b]$. This countermeasure requires $(t+1)^2 + t + 1$ ω-bit multiplications and $2\log_2(n)$ bits of additional storage.
- *Randomize the Two Loops in LIM.* This countermeasure is a variant of the second one. In this case, the authors fully randomize the order of the processings of the $Z[a, b]$ variables. As an advantage no operand in the LIM needs to be blinded anymore. However, the drawback is that two random permutation vectors have to be stored. No extra ω-bit multiplication compared to the unprotected LIM is needed here.

In the next section, we argue that the three countermeasures below do not fully hide the first-order leakage and we exhibit efficient attacks.

5 Attacks Against Horizontal SCA Countermeasures

The attacks presented in this section are described in the Square and Multiply Always setting but are straightly applicable in the classical Square and Multiply setting.

Blind Operands in LIM. In this case, the variable $Z[a, b]$ in (6) becomes:

$$\tilde{Z}[a, b] = (X^{2+s}[a] - R_1) \cdot (X^{2+s}[b] - R_2) \ . \tag{7}$$

According to (1), the observation $\boldsymbol{l}_{a,b}$ of the $\tilde{Z}[a, b]$ processing satisfies $\boldsymbol{l}_{a,b} \hookleftarrow \varphi_{a,b}\big(\tilde{Z}[a, b]\big) + \beta_{a,b}$, where it can be checked that $\sum_a \tilde{Z}[a, b]$ depends on $X^{2+s}[b]$, and hence on s. This dependency can be exploited in a Horizontal C-PA by correlating the means $\overline{\boldsymbol{l}}_{\cdot,b} = \frac{1}{t+1}\sum_a \boldsymbol{l}_{a,b}$ with the predictions:

$$\boldsymbol{h}_b = \tfrac{1}{t+1}\textstyle\sum_a \mathbf{m}_{a,b}\big(X^{2+\hat{s}}[a] \cdot X^{2+\hat{s}}[b]\big) \quad , \tag{8}$$

for $\hat{s} = 0$ and $\hat{s} = 1$ and for $\mathbf{m}_{a,b}$ being an estimation of the unknown function $\varphi_{a,b}$. The rationale behind the definition of $\boldsymbol{h}_b$ in (8) may be found in the extended version of this paper [2]. As illustrated in Figure 6, it may be observed that the attack is still effective if the mask R_1 is different for all the words $X^{2+s}[a]$ since the leakages are averaging over indices a to compute the predictions $\boldsymbol{h}_b$.

Simulation Results. Experiments have been performed to check that the predictions $\boldsymbol{h}_b$ in (8) are consistent with the means $\bar{\boldsymbol{l}}_{\cdot,b}$ of the observations $\boldsymbol{l}_{a,b}$ when the hypothesis $\hat{s}$ on s is valid. Indeed, in that case a correlation peak can be observed, making the adversary guess the right key-bit value s. Then, processing iteratively, the whole secret k can finally be recovered. The results that have been obtained are sumed up in Figure 6 for a model function $\mathbf{m}_{a,b}$ and φ defined as the Hamming weight. Each point on the curve corresponds to the smallest number t needed to achieve a 90% success rate, according to the noise standard deviation. Attacks are reported for an architecture size ω in $\{8, 16, 32\}$.

Randomize One Loop in LIM and Blind. In this case, a random permutation α over $[0; t]$ is generated before each new exponentiation and the variable $Z[a, b]$ in (6) becomes:

$$\tilde{Z}[a, b] = X^{2+s}[\alpha(a)] \cdot (X^{2+s}[b] - R) \ . \tag{9}$$

The randomization of the manipulations of the words $X^{2+s}[a]$ does not modify the value of the sum $\sum_a \tilde{Z}[a, b]$. As a consequence, it does not change the fact that - as for countermeasure 1 - this sum depends on $X^{2+s}[b]$ and hence on s. The previous attack against Countermeasure 1 can hence be still applied and its efficiency is the same.

Randomize the Two Loops in LIM. In this case, two random permutations α and β defined over $[0; t]$ are generated and we have:

$$\tilde{Z}[a, b] = X^{2+s}[\alpha(a)] \cdot X^{2+s}[\beta(b)] \ . \tag{10}$$

To attack this countermeasure, a solution consists in performing the same attack as against the first and the second countermeasure exhaustively for all the possible permutations β. This attack stays efficient as long as t is reasonably small (lower than 16).

The first attack presented in this section shows that protection strategies based on masking are not promising. A possibility could be to use a different pair of masks for each internal multiplication but the cost of such a countermeasure would be prohibitive. The randomization of operations order seems to be more interesting but the attacks exhibited in this section point out that they must be carefully specified. In the following section we propose such a specification where the operations order randomization is done globally over all the indices.

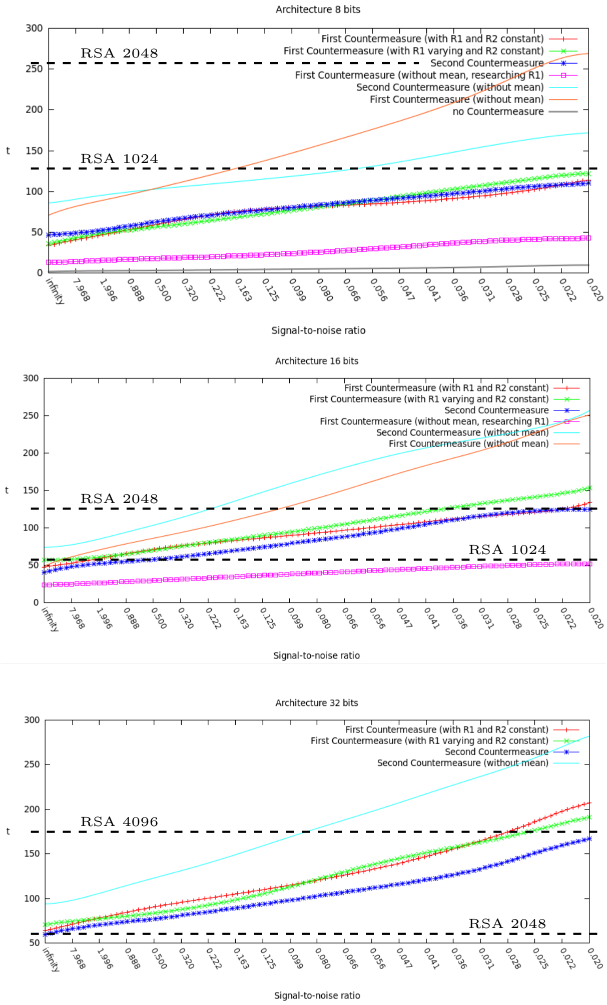


Fig. 6. Evolution of the ω-bit length t of X and Y (y-axis log-scaled) to achieve a 90% success rate depending on the noise standard deviation σ (x-axis log-scaled).

6 New Countermeasure

Here, we propose to randomize the two loops (over a and b) in the LIM *simultaneously*. For such a purpose, we first need to efficiently[8] generate a random permutation over $\{(a,b); a,b \in [0;t]\}$. To achieve good efficiency, we propose to generate such a random permutation using Algorithm 1, following the same idea as in [7]. Even if there is no formal proof that this method enables to generate random permutations that are indistinguishable from perfectly random ones, we

[8] This step may be very costly even for small values of the parameter t [12].

are confident about this in practice. The number of permutations that can be generated thanks to Algorithm 1 is around $(t+1)^{2\ell}$ where ℓ is the number of random values in the input of Algorithm 1. This number is sufficiently high to prevent attacks involving exhaustive search.

Algorithm 1 : Generation of Random Permutation (GRP)

```
Input: Two integers $t$ and $\ell$, a permutation $\alpha_0$ over $[0, (t+1)^2 - 1]$.
Output: A vector in $[0, (t+1)^2 - 1]$ (elements are represented in base $t+1$).
$(r_0, r_1, \dots, r_{\ell-1}) \leftarrow$ random elements in $\mathbb{Z}_{(t+1)^2}$
for $i$ from 0 to $\ell - 1$ do
    for $j$ from 0 to $(t+1)^2 - 1$ do
        $\alpha_{i+1}[j] \leftarrow \alpha_0[(\alpha_i[j] + r_i) \bmod (t+1)^2]$
return $\alpha_\ell$
```

The random permutation returned by Algorithm 1 can be used to randomize the manipulation of the words $X[a]$ and $Y[b]$ simultaneaously. A second random permutation P over the set of integers $1, 2, \dots, 2t+1$ must be used to avoid attacks in the carry propagation treatment. Eventually, we get the proposal of Algorithm 2 leading to a secure Long Integer Multiplication algorithm, where the carry registers $C[h]$ must be of bit-length $\omega + \log_2(t+1)$.

Algorithm 2 : Long Integer Multiplication with randomization of the two loops together.

```
Input: $X = (X[t], X[t-1], \dots, X[0])_{2^\omega}$, $Y = (Y[t], Y[t-1], \dots, Y[0])_{2^\omega}$, $p$.
Output: $\mathrm{LIM}(X, Y)$.
$\alpha_\ell = (\alpha, \beta) \leftarrow \mathrm{GRP}(t, p, \alpha_0)$
$P \leftarrow$ random permutation of $1, 2, \dots, 2t+1$.
for $a$ from 0 to $2t+1$ do
    $R[a] = C[a] = 0$
for $h$ from 0 to $(t+1)^2 - 1$ do
    $a \leftarrow \alpha[h]; b \leftarrow \beta[h]$
    $(U, V)_{2^\omega} \leftarrow R[a+b] + X[a] \cdot Y[b]$
    $R[a+b] \leftarrow V$
    $C[a+b+1] \leftarrow C[a+b+1] + U$
for $i$ from 1 to $2t+1$ do
    for $j$ from 1 to $2t+1$ do
        $s \leftarrow P[j]$
        if $s \geq i$ then
            $(U, V)_{2^\omega} \leftarrow R[s] + C[s]$
            $R[s] \leftarrow V$
            $C[s+1] \leftarrow C[s+1] + U$
            $C[s] \leftarrow 0$
return $R$
```

References

1. Lejla Batina, Benedikt Gierlichs, Emmanuel Prouff, Matthieu Rivain, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. Mutual Information Analysis: a Comprehensive Study. *J. Cryptology*, 24(2):269–291, 2011.
2. Aurélie Bauer, Éliane Jaulmes, Emmanuel Prouff, and Justine Wild. Horizontal and Vertical Side-Channel Attacks Against Secure RSA Implementations – Extended Version – . To appear on the Cryptology ePrint Archive., 2013.
3. Eric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In *CHES*, pages 16–29, 2004.
4. S. Chari, C.S. Jutla, J.R. Rao, and P. Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In M.J. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.
5. Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Horizontal Correlation Analysis on Exponentiation. In *ICICS*, pages 46,64, 2010.
6. Jean-Sébastien Coron. Resistance Against Differential Power Analysis For Elliptic Curve Cryptosystems. In *Cryptographic Hardware and Embedded Systems*, volume 1717 of *Lecture Notes in Computer Science*, pages 725–725. Springer Berlin / Heidelberg, 1999.
7. Jean-Sébastien Coron. A New DPA Countermeasure Based on Permutation Tables. In *SCN*, pages 278–292, 2008.
8. K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic Analysis: Concrete Results. In Ç.K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.
9. Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual Information Analysis. In Elisabeth Oswald and Pankaj Rohatgi, editors, *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 426–442. Springer, 2008.
10. L. Goubin and J. Patarin. DES and Differential Power Analysis – The Duplication Method. In Ç.K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES '99*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.
11. Marc Joye and Sung-Ming Yen. The Montgomery Powering Ladder. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*. Springer, 2003.
12. Donald E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973.
13. C.K. Koc. *Cryptographic Engineering*. Springer, 2008.
14. Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO*, pages 104–113, 1996.
15. Paul C. Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to Differential Power Analysis. *J. Cryptographic Engineering*, 1(1):5–27, 1998.
16. T.S. Messerges. Using Second-Order Power Analysis to Attack DPA Resistant software. In Ç.K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 238–251. Springer, 2000.
17. Emmanuel Prouff, Matthieu Rivain, and Régis Bevan. Statistical Analysis of Second-Order Differential Power Analysis. *IEEE Trans. Computers*, 58(6):799–811, 2009.
18. J.-J. Quisquater and D. Samylde. A new Tool for non-Intrusive Analysis of Smart Cards based on Electro-Magnetic Emissions, the SEMA and DEMA Methods. In *Presented at the rump session of Eurocrypt*, 2000.

19. Colin D. Walter. Sliding Windows Succumbs to Big Mac Attack. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*. Springer, 2001.
20. S.-M. Yen and M. Joye. Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis. *IEEE Transactions on Computers*, 49(9):967–970, 2000.

A Long Integer Multiplication

Let $X = (X[t], X[t-1], \ldots, X[0])_{2^\omega}$ denote the decomposition of an integer X in ω-bit words. The Long Integer Multiplication algorithm is:

Algorithm 3 : Long Integer Multiplication

```
Input: $X = (X[t], X[t-1], \ldots, X[0])_{2^\omega}$, $Y = (Y[t], Y[t-1], \ldots, Y[0])_{2^\omega}$.
Output: $\mathrm{LIM}(X, Y)$.
for $a$ from 0 to $2t+1$ do
    $R[a] \leftarrow 0$
for $a$ from 0 to $t$ do
    $C \leftarrow 0$
    for $b$ from 0 to $t$ do
        $(U, V)_{2^\omega} \leftarrow Z[a, b] = X[a] \cdot Y[b]$
        $(U, V)_{2^\omega} \leftarrow (U, V)_{2^\omega} + C$
        $(U, V)_{2^\omega} \leftarrow (U, V)_{2^\omega} + R[a+b]$
        $R[a+b] \leftarrow V$
        $C \leftarrow U$
    $R[a+t+1] \leftarrow C$
return $R$
```