DES and Differential Power Analysis

The "Duplication" Method*

Louis Goubin and Jacques Patarin

Bull SmartCards and Terminals
68, route de Versailles - BP45
78431 Louveciennes Cedex - France
{L.Goubin, J.Patarin}@frlv.bull.fr

Abstract. Paul Kocher recently developped attacks based on the electric consumption of chips that perform cryptographic computations. Among those attacks, the "Differential Power Analysis" (DPA) is probably one of the most impressive and most difficult to avoid.
In this paper, we present several ideas to resist this type of attack, and in particular we develop one of them which leads, interestingly, to rather precise mathematical analysis. Thus we show that it is possible to build an implementation that is provably DPA-resistant, in a "local" and restricted way (*i.e.* when – given a chip with a fixed key – the attacker only tries to detect predictable local deviations in the differentials of mean curves). We also briefly discuss some more general attacks, that are sometimes efficient whereas the "original" DPA fails. Many measures of consumption have been done on real chips to test the ideas presented in this paper, and some of the obtained curves are printed here.

Note: An extended version of this paper can be obtained from the authors.

1 Introduction

This paper is about a way of securing a cryptographic algorithm that makes use of a secret key. More precisely, the goal consists in building an implementation of the algorithm that is not vulnerable to a certain type of physical attacks – so-called "Differential Power Analysis".

These DPA attacks belong to a general family of attacks that look for information about the secret key by studying the electric consumption of the electronic device during the execution of the computation. In this family, we usually distinguish between SPA attacks ("Simple Power Analysis") and DPA attacks.

In SPA attacks, the aim is essentially to guess – from the values of the consumption – which particular instruction is being computed at a certain time and with which input or output, and then to use this information to deduce some part of the secret. Figure 1 shows the electric consumption of a chip, measured during a DES computation on a real smartcard. The fact that the 16 rounds of the DES algorithm are clearly visible is a good sign that power analysis attacks may indeed provide information about what the chip is doing.

* Patents Pending

Ç.K. Koç and C. Paar (Eds.): CHES'99, LNCS 1717, pp. 158–172, 1999.
© Springer-Verlag Berlin Heidelberg 1999

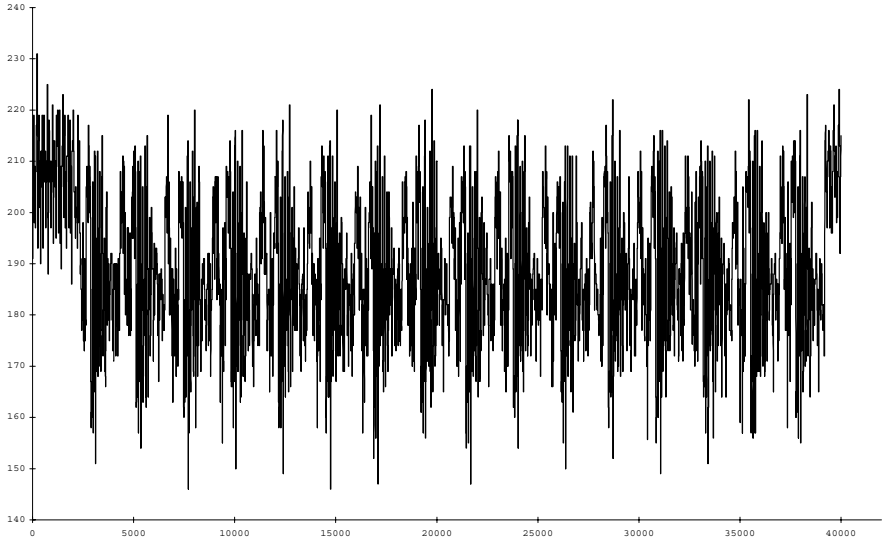


Fig. 1. Electric consumption measured on the 16 rounds of a DES computation

In DPA attacks, some differentials on two sets of average consumption are computed, and the attacks succeed if an unusual phenomenon appears – on these differentials of consumption – for a good choice of some of the key bits (we give details below), so that we are able to find out those key bits. What makes DPA attacks so impressive, when they work, is the fact that they can find out the secret key of a public algorithm (for example DES, but also many other algorithms) without knowing anything (nor trying to find anything) about the particular implementation of that algorithm. Implementations exist that are DPA-resistant (differentials do not show anything special) but not SPA-resistant (some critical information can be deduced from the consumption curves). On the contrary, other implementations exist that are SPA-resistant but not DPA-resistant (some critical information can be found by studying differentials of two mean curves of consumption). Finally, some implementations can be found that resist both types of attack (at least at the present), or none of them.

Throughout this paper, we study more particularly DPA and we will not deal any longer with SPA. Indeed, as we see below, DPA can easily be analyzed in a mathematical way (and not only in an empirical way). There exist many attacks based on the electric consumption. We do not claim to give here solutions to all the problems that may result from these attacks.

The cryptographic algorithms we consider here make use of a secret key in order to compute an output information from an input information. It may be a ciphering, a deciphering or a signature operation. In particular, all the material

described in this paper applies to "secret key algorithms" and also to the so-called "public key algorithms".

2 The "Differential Power Analysis" Attacks

The "Differential Power Analysis" attacks, developped by Paul Kocher and Cryptographic Research (see [1]), start from the fact that the attacker can get many more information (than the knowledge of the inputs and the outputs) during the execution of the computation, such as for instance the electric consumption of the microcontroller or the electromagnetic radiations of the circuit. The "Differential Power Analysis" (DPA to be brief) is an attack that allows to obtain information about the secret key (contained in a smartcard for example), by performing a statistical analysis of the electric consumption records measured for a large number of computations with the same key. Let us consider for instance the case of the DES algorithm (Data Encryption Standard). It executes in 16 steps, called "rounds". In each of these steps, a transformation F is performed on 32 bits. This F function uses eight non-linear transformations from 6 bits to 4 bits, each of which is coded by a table called "S-box". The DPA attack on the DES can be performed as follows (the number 1000 used below is just an example):

Step 1: We measure the consumption on the first round, for 1000 DES computations. We denote by E_1, ..., E_{1000} the input values of those 1000 computations. We denote by C_1, ..., C_{1000} the 1000 electric consumption curves measured during the computations. We also compute the "mean curve" MC of those 1000 consumption curves.

Step 2: We focus for instance on the first output bit of the first S-box during the first round. Let b be the value of that bit. It is easy to see that b depends on only 6 bits of the secret key. The attacker makes an hypothesis on the involved 6 bits. He computes – from those 6 bits and from the E_i – the expected (theoretical) values for b. This enables to separate the 1000 inputs E_1, ..., E_{1000} into two categories: those giving $b = 0$ and those giving $b = 1$.

Step 3: We now compute the mean MC' of the curves corresponding to inputs of the first category (*i.e.* the one for which $b = 0$). If MC and MC' show an appreciable difference (in a statistical meaning, *i.e.* a difference much greater than the standard deviation of the measured noise), we consider that the chosen values for the 6 key bits were correct. If MC and MC' do not show any sensible difference, we repeat step 2 with another choice for the 6 bits.

Note: In practice, for each choice of the 6 key bits, we draw the curve representing the difference between MC and MC'. As a result, we obtain 64 curves, among which one is supposed to be very special, *i.e.* to show an appreciable difference, compared to all the others.

Step 4: We repeat steps 2 and 3 with a "target" bit b in the second S-box, then in the third S-box, ..., until the eighth S-box. As a result, we finally obtain 48 bits of the secret key.

Step 5: The remaining 8 bits can be found by exhaustive search.

Note: It is also possible to focus (in steps 2, 3 and 4) on the set of the four output bits for the considered S-boxes, instead of only one output bit. This is what we actually did for real smartcards. In that case, the inputs are separated into 16 categories: those giving 0000 as output, those giving 0001, ..., those giving 1111. In step 3, we may compute for example the mean MC' of the curves corresponding to the last category (*i.e.* the one which gives 1111 as output). As a result, the mean MC' is computed on approximately $\frac{1}{16}$ of the curves (instead of approximately half of the curves with step 3 above): this may compel us to use a number of DES computations greater than 1000, but it generally leads to a more appreciable difference between MC and MC'.

We presented in figures 2 and 3 two mean curves, resulting from steps 2 and 3, for a classical implementation of DES on a real smartcard (with '1111' as target output of the first S-box and with 2048 different inputs, even if we noted that 512 inputs are sufficient). A detailed analysis of the 64 obtained curves (that we cannot all print here, due to the lack of place) shows that the one corresponding to the correct choice of the 6 key-bits can easily be detected (it contains much greater peaks than all the others).

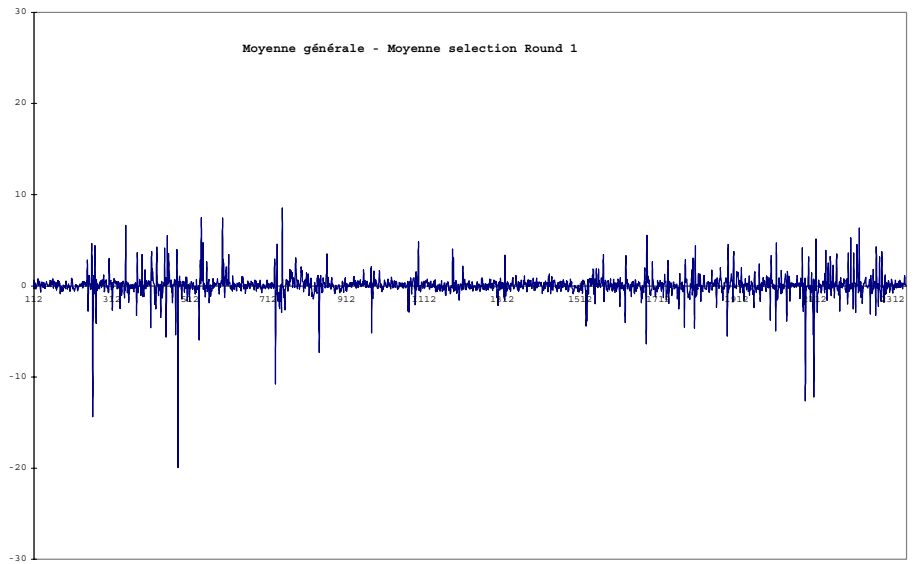


Fig. 2. An example of difference of the curves MC and MC' when the 6 bits are false

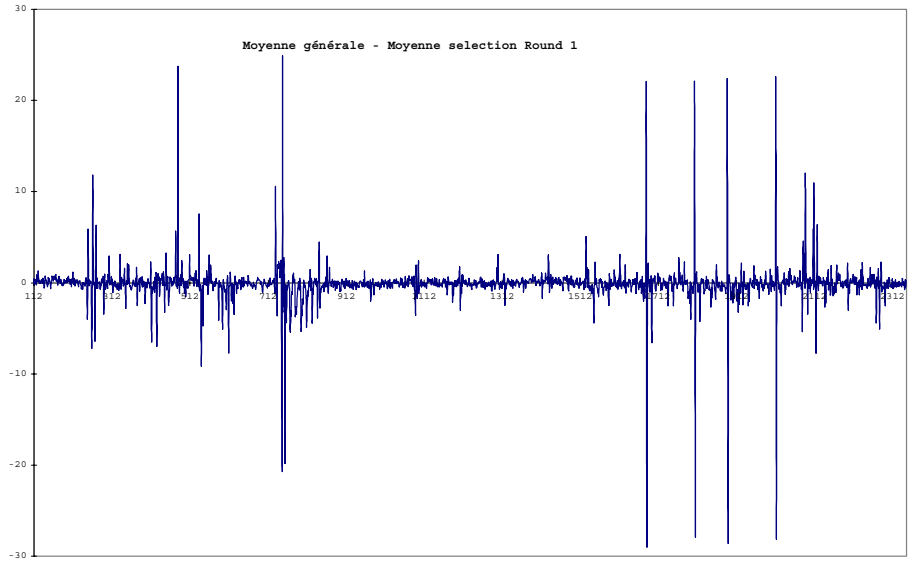


Fig. 3. Difference of the curves MC and MC' when the 6 bits are correct

This attack does not require any knowledge about the individual electric consumption of each instruction, nor about the position in time of each of these instructions. It applies exactly the same way as soon as the attacker knows the outputs of the algorithm and the corresponding consumption curves. It only relies on the following fundamental hypothesis:

Fundamental hypothesis*: There exists an intermediate variable, that appears during the computation of the algorithm, such that knowing a few key bits (in practice less than 32 bits) allows us to decide whether two inputs (respectively two outputs) give or not the same value for this variable.*

All the algorithms that use S-boxes, such as DES, are potentially vulnerable to the DPA attack, because the "natural" implementations generally remain within the hypothesis mentioned above.

3 Securing the Algorithm

Several countermeasures against DPA attacks can be conceived. For instance:

1. Introducing random timing shifts, so that the computed means do not correspond any longer to the consumption of the same instruction. The crucial point consists here in performing those shifts so that they cannot be easily eliminated by a statistical treatment of the consumption curves.

2. Replacing some of the critical instructions (in particular the basic assembler instructions involving writings in the carry, readings of data from an array, etc) by assembler instructions whose "consumption signature" is difficult to analyze.
3. For a given algorithm, giving an explicit way of computing it, so that DPA is provably unefficient on the obtained implementation. For instance, for a DES-like algorithm, we detail in section 4 how to compute the non-linear transformations of the S-boxes in order to avoid some DPA attacks.

In the present paper, we essentially study the third idea because it leads to a quite precise mathematical analysis. We give in this section a general method to implement an algorithm with a secret key so as to avoid the DPA attacks described above. The basic principle consists in programming the algorithm so that the fundamental hypothesis above is not true any longer (*i.e.* an intermediate variable never depends on the knowledge of an easily accessible subset of the secret key).

The Main Idea

In this paper, we mainly study how this can be done by using the following main idea: replacing each intermediate variable V, occuring during the computation and depending on the inputs (or the outputs), by k variables V_1, ..., V_k, such that V_1, V_2, ..., V_k allows us – if we want – to retrieve V. More precisely, to guarantee the security of the algorithm in its new form, it is sufficient to choose a function f satisfying the identity $V = f(V_1, ..., V_k)$, together with the two following conditions:

Condition 1: *From the knowledge of a value v and for any fixed value i, $1 \leq i \leq k$, it is not feasible to deduce information about the set of the values v_i such that there exist a $(k-1)$-uple $(v_1, ..., v_{i-1}, v_{i+1}, ..., v_k)$ satisfying the equation $f(v_1, ..., v_k) = v$.*

Condition 2: *The function f is such that the transformations to be performed on V_1, V_2, ..., or V_k during the computation (instead of the transformations usually performed on V) can be implemented without calculating V.*

First example for condition 1: If we choose $f(v_1, ..., v_k) = v_1 \oplus v_2 \oplus ... \oplus v_k$, where $\oplus$ denotes the bit-by-bit "exclusive-or" function, condition 1 is obviously satisfied, because – for any fixed index i between 1 and k – the considered set of the values v_i contains all the possible values and thus does not depend on v.

Second example for condition 1: If we consider some variable V whose values lie in the multiplicative group of $\mathbf{Z}/n\mathbf{Z}$, we can choose the function $f(v_1, ..., v_k) = v_1 \cdot v_2 \cdot ... \cdot v_k \bmod n$, where the new variables v_1, v_2, ..., v_k also have values in the multiplicative group of $\mathbf{Z}/n\mathbf{Z}$. Condition 1 is also obviously true because – for any fixed index i between 1 and k – the considered set of the values v_i contains all the possible values and thus does not depend on v.

We then "translate" the algorithm by replacing each intermediate variable V depending on the inputs (or the outputs) by the k variables V_1, ..., V_k. In the following sections, we study how conditions 1 and 2 can be achieved in the case of the DES or RSA algorithms.

4 The DES Algorithm: First Example of Implementation for DPA Resistance

In this section, we consider the particular case of the DES algorithm. We choose here to separate each intermediate variable V, occuring during the computation and depending on the inputs (or the outputs), into two variables V_1 and V_2 (*i.e.* we take $k = 2$). Let us choose the function $f(v_1, v_2) = v = v_1 \oplus v_2$ (see the first example of section 3), which satisfies condition 1. From the construction of the algorithm, it is easy to see that the transformations performed on v always belong to one of the five following categories:

1. permutation of the bits of v;
2. expansion of the bits of v;
3. "exclusive-or" between v and another variable v' of the same type;
4. "exclusive-or" between v and a variable depending only on the key;
5. transformation of v using a S-box.

The first two cases correspond to linear transformations on the bits of the variable v. For these ones, condition 2 is thus very easy to satisfy: we just have – instead of the transformation usually performed on v – to perform the permutation or the expansion on v_1, then on v_2, and the identity $f(v_1, v_2) = v$, which was true before the transformation, is also true afterwards.

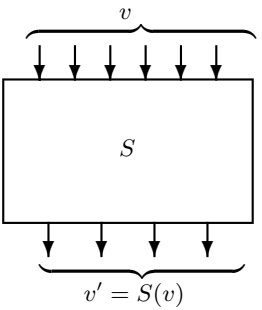
In the same way, in the third case, we just have to replace the computation of $v'' = v \oplus v'$ by the computation of $v_1'' = v_1 \oplus v_1'$ and $v_2'' = v_2 \oplus v_2'$. The identities $f(v_1, v_2) = v$ and $f(v_1', v_2') = v'$ give indeed $f(v_1'', v_2'') = v''$, so that condition 2 is true again.

As concerns the exclusive-or between v and a variable c depending only on the key, condition 2 is also very easy to satisfy: we just have to replace the computation of $v \oplus c$ by $v_1 \oplus c$ (or $v_2 \oplus c$) and that gives condition 2.

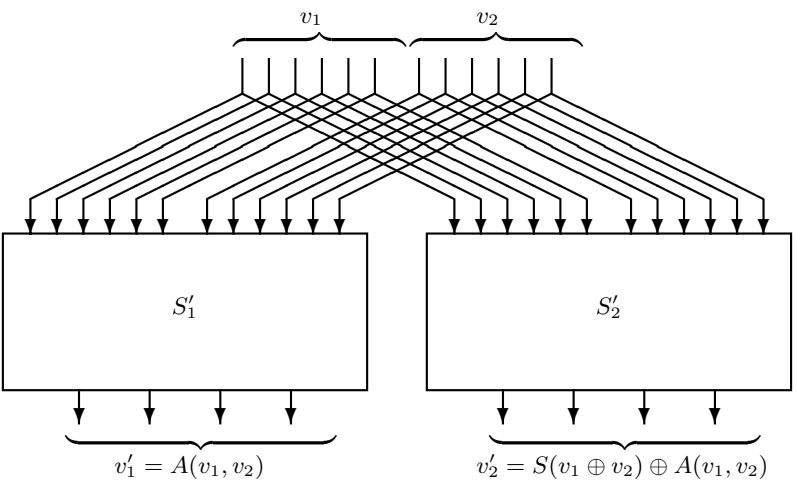
Finally, instead of the non-linear transformation $v' = S(v)$, given under the form of a S-box (which in that example has 6-bits inputs and 4-bits outputs), we implement the transformation $(v_1', v_2') = S'(v_1, v_2)$ by using two new S-boxes (each of which sending 12 bits onto 4 bits). In order to keep the identity $f(v_1', v_2') = v'$, we may choose:

$$(v_1', v_2') = S'(v_1, v_2) = (A(v_1, v_2), S(v_1 \oplus v_2) \oplus A(v_1, v_2)).$$

where A denotes a *randomly* chosen *secret* transformation from 12 bits to 4 bits (see figure 4). The first of the new S-boxes corresponds to the table of the transformation $(v_1, v_2) \mapsto A(v_1, v_2)$, and the second one corresponds to the table of the transformation $(v_1, v_2) \mapsto S(v_1 \oplus v_2) \oplus A(v_1, v_2)$. Thanks to the randomly



Initial implementation: the predictable values v and v' appear in RAM at some time



Modified implementation: the values $v = v_1 \oplus v_2$ and $v' = v'_1 \oplus v'_2$ never explicitely appear in RAM

Fig. 4. Standard transformation of a S-box

chosen function A, condition 1 is satisfied. Moreover, the use of tables allows us to avoid the computation of $v_1 \oplus v_2$, so that condition 2 is also true.

The solution presented in this section is quite realistic for chips that compute DES in hardware (and are not embedded in a card), or for PCs, because – in those cases – enough memory is available. More precisely, the size of the memory required to store the S-boxes is 32 Kbytes for the method described in this section. It is too much for smartcards, for which specific variations using less memories are described in section 5 below.

5 Smartcard Implementations of DES

First Variation

In order to reduce the ROM used by the algorithm, it is quite possible to use the same random function A for the eight S-boxes (of the initial description of the DES), so that we have only nine (new) S-boxes (*i.e.* 18 Kbytes) to store in ROM, instead of sixteen S-boxes.

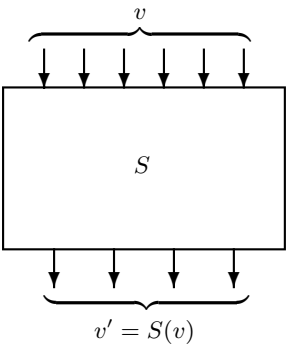
Second Variation

In order to reduce the size of the ROM needed to store the S-boxes, we can also use the following method: instead of each non-linear transformation $v' = S(v)$ of the initial implementation, given under the form of a S-box (with 6-bits inputs and 4-bits outputs in the case of the DES), we implement the transformation $(v'_1, v'_2) = S'(v_1, v_2)$ by using two S-boxes, each of which sending 6 bits onto 4 bits. The initial implementation of the computation $v' = S(v)$ is replaced by the two following successive computations:

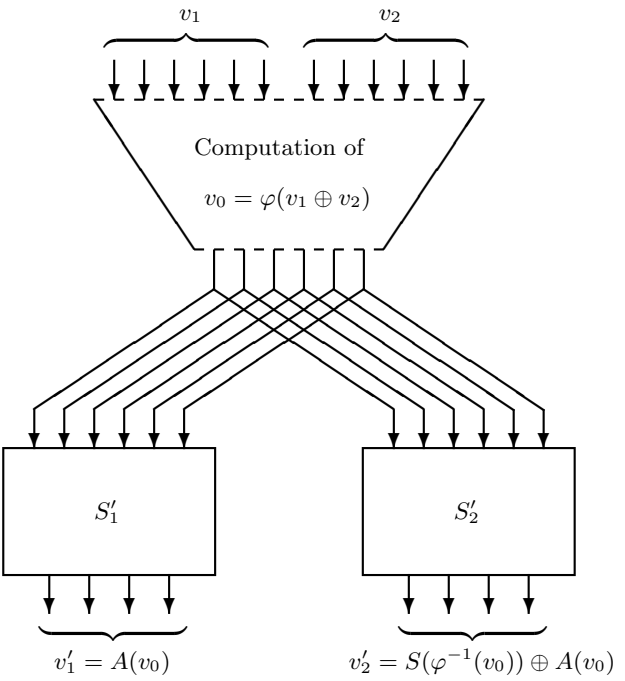
- $v_0 = \varphi(v_1 \oplus v_2)$
- $(v'_1, v'_2) = S'(v_1, v_2) = (A(v_0), S(\varphi^{-1}(v_0)) \oplus A(v_0))$

where φ is a *bijective* and *secret* function from 6 bits to 6 bits and where A denotes a *random* and *secret* transformation from 6 bits to 4 bits. The first of the two new S-boxes corresponds to the table of the transformation $v_0 \mapsto A(v_0)$ and the second one corresponds to the table of the transformation $v_0 \mapsto S(\varphi^{-1}(v_0)) \oplus A(v_0)$. From this construction, the identity $f(v'_1, v'_2) = v'$ is always true. Thanks to the random function A, condition 1 is satisfied. Moreover, the use of tables allows us to avoid the computation of $\varphi^{-1}(v_0) = v_1 \oplus v_2$, so that condition 2 is also true. This solution (shown in figure 5) requires 512 bytes to store the S-boxes.

In order to satisfy condition 2, it remains to choose the bijective transformation φ such that the computation of $v_0 = \varphi(v_1 \oplus v_2)$ is feasible without computing $v_1 \oplus v_2$. We give below two examples of possible choice for the function φ.



Initial implementation: the predictable values v and v' appear in RAM at some time



Modified implementation: the values $v = v_1 \oplus v_2$ and $v' = v'_1 \oplus v'_2$ never explicitely appear in RAM

Fig. 5. Transformation of a S-box (second variation)

Exemple 1: a linear bijection

We choose φ as a *linear secret* and *bijective* function from 6 bits to 6 bits (we consider the set of the 6-bits values as a vectorial space of dimension 6 on the finite field $\mathbf{F}_2$ with 2 elements). In practice, choosing φ is equivalent to choosing a *random* and *invertible* 6×6 *matrix* whose coefficients are 0 or 1. With this choice of φ, it is easy to see that condition 2 is satisfied. Indeed, to compute $\varphi(v_1 \oplus v_2)$, we just have to compute $\varphi(v_1)$, then $\varphi(v_2)$ and finally to compute the "exclusive-or" of the two obtained results.

For instance, the matrix $\begin{pmatrix} 1&1&0&1&0&0 \\ 1&1&0&1&0&1 \\ 0&1&1&0&1&0 \\ 1&1&1&0&1&0 \\ 0&1&1&1&1&0 \\ 0&0&1&1&0&1 \end{pmatrix}$ is invertible. It corresponds to the linear bijection φ from 6 bits to 6 bits defined by $\varphi(u_1, u_2, u_3, u_4, u_5, u_6) = (u_1 \oplus u_2 \oplus u_4, u_1 \oplus u_2 \oplus u_4 \oplus u_6, u_2 \oplus u_3 \oplus u_5, u_1 \oplus u_2 \oplus u_3 \oplus u_5, u_2 \oplus u_3 \oplus u_4 \oplus u_5, u_3 \oplus u_4 \oplus u_6)$.

Let $v_1 = (v_{1,1}, v_{1,2}, v_{1,3}, v_{1,4}, v_{1,5}, v_{1,6})$ and $v_2 = (v_{2,1}, v_{2,2}, v_{2,3}, v_{2,4}, v_{2,5}, v_{2,6})$. To compute $\varphi(v_1 \oplus v_2)$, we successively compute:

- $\varphi(v_1) = (v_{1,1} \oplus v_{1,2} \oplus v_{1,4}, v_{1,1} \oplus v_{1,2} \oplus v_{1,4} \oplus v_{1,6}, v_{1,2} \oplus v_{1,3} \oplus v_{1,5},\ v_{1,1} \oplus v_{1,2} \oplus v_{1,3} \oplus v_{1,5}, v_{1,2} \oplus v_{1,3} \oplus v_{1,4} \oplus v_{1,5}, v_{1,3} \oplus v_{1,4} \oplus v_{1,6})$
- $\varphi(v_2) = (v_{2,1} \oplus v_{2,2} \oplus v_{2,4}, v_{2,1} \oplus v_{2,2} \oplus v_{2,4} \oplus v_{2,6}, v_{2,2} \oplus v_{2,3} \oplus v_{2,5}, v_{2,1} \oplus v_{2,2} \oplus v_{2,3} \oplus v_{2,5}, v_{2,2} \oplus v_{2,3} \oplus v_{2,4} \oplus v_{2,5}, v_{2,3} \oplus v_{2,4} \oplus v_{2,6})$

Then we compute the "exclusive-or" of the two obtained results.

Exemple 2: a quadratic bijection

We choose φ as a *quadratic secret* and *bijective* function from 6 bits to 6 bits. Here, "quadratic" means that each bit of the output is given by a polynomial function of *total degree two* of the 6 bits of the input (which are identified to 6 elements of the finite field $\mathbf{F}_2$). In practice, we may choose the function φ defined by $\varphi(x) = t(s(x)^5)$, where s is a secret linear bijection from $(\mathbf{F}_2)^6$ to $\mathcal{L}$, t is a secret linear bijection from $\mathcal{L}$ to $(\mathbf{F}_2)^6$ and $\mathcal{L}$ denotes an algebraic extension of degree 6 over the finite field $\mathbf{F}_2$. The bijectivity of this function φ follows from the fact that $a \mapsto a^5$ is a bijection on the extension $\mathcal{L}$ (whose inverse is $b \mapsto b^{38}$). To be convinced that condition 2 is still satisfied, just notice that we can write:

$$\varphi(v_1 \oplus v_2) = \psi(v_1, v_1) \oplus \psi(v_1, v_2) \oplus \psi(v_2, v_1) \oplus \psi(v_2, v_2),$$

where the function ψ is defined by $\psi(x, y) = t(s(x)^4 \cdot s(y))$.

For instance, if we identify $\mathcal{L}$ to $\mathbf{F}_2[X]/(X^6 + X + 1)$ and if we choose s and t whose matrices are $\begin{pmatrix} 1&1&0&1&0&0 \\ 1&1&0&1&0&1 \\ 0&1&1&0&1&0 \\ 1&1&1&0&1&0 \\ 0&1&1&1&1&0 \\ 0&0&1&1&0&1 \end{pmatrix}$ and $\begin{pmatrix} 0&1&0&0&1&1 \\ 1&1&0&1&0&0 \\ 1&0&1&0&1&1 \\ 0&1&1&1&0&0 \\ 1&0&1&0&1&0 \\ 0&0&1&0&1&1 \end{pmatrix}$ with respect to the basis

$(1, X, X^2, X^3, X^4, X^5)$ of $\mathcal{L}$ over $\mathbf{F}_2$ and to the canonical basis of $(\mathbf{F}_2)^6$ over $\mathbf{F}_2$, we obtain the following quadratic bijection φ from 6 bits to 6 bits:

$$\varphi(u_1, u_2, u_3, u_4, u_5, u_6) = (u_2u_5 \oplus u_1u_4 \oplus u_4 \oplus u_6 \oplus u_6u_2 \oplus u_4u_6 \oplus u_2 \oplus u_5 \oplus u_3 \oplus u_4u_3, u_2u_5 \oplus u_5u_1 \oplus u_1u_4 \oplus u_4 \oplus u_6 \oplus u_4u_5 \oplus u_2 \oplus u_3 \oplus u_3u_1, u_2u_5 \oplus u_5u_1 \oplus u_6u_5 \oplus u_1u_4 \oplus u_3u_5 \oplus u_1 \oplus u_4u_6 \oplus u_6u_3 \oplus u_4u_3 \oplus u_3u_1, u_1u_4 \oplus u_2u_3 \oplus u_6u_1 \oplus u_4u_6 \oplus u_5 \oplus u_6u_3 \oplus u_4u_3, u_5u_1 \oplus u_1u_4 \oplus u_6 \oplus u_3u_5 \oplus u_4u_5 \oplus u_1 \oplus u_6u_1 \oplus u_4u_6 \oplus u_3 \oplus u_6u_3 \oplus u_4u_2, u_4 \oplus u_6 \oplus u_3u_5 \oplus u_1 \oplus u_4u_6 \oplus u_6u_3).$$

To compute $\varphi(v_1 \oplus v_2)$, we use the function $\psi(x, y) = t(s(x)^4 \cdot s(y))$ from 12 bits to 6 bits, which gives the 6 output bits from the 12 input bits as follows:

$$\psi(x_1, x_2, x_3, x_4, x_5, x_6, y_1, y_2, y_3, y_4, y_5, y_6) = (x_3y_5 \oplus x_6y_2 \oplus x_6y_3 \oplus x_6y_4 \oplus x_3y_1 \oplus x_6y_1 \oplus x_1y_3 \oplus x_1y_5 \oplus x_5y_2 \oplus x_5y_5 \oplus x_5y_1 \oplus x_6y_6 \oplus x_1y_6 \oplus x_1y_2 \oplus x_1y_4 \oplus x_2y_1 \oplus x_2y_2 \oplus x_4y_4 \oplus x_3y_3 \oplus x_3y_6 \oplus x_4y_3 \oplus x_5y_3, x_4y_5 \oplus x_3y_1 \oplus x_6y_1 \oplus x_2y_5 \oplus x_5y_1 \oplus x_6y_6 \oplus x_1y_6 \oplus x_1y_2 \oplus x_2y_1 \oplus x_2y_2 \oplus x_4y_1 \oplus x_4y_4 \oplus x_3y_3, x_6y_2 \oplus x_6y_3 \oplus x_6y_4 \oplus x_6y_5 \oplus x_3y_1 \oplus x_6y_1 \oplus x_2y_5 \oplus x_5y_1 \oplus x_1y_6 \oplus x_1y_1 \oplus x_1y_2 \oplus x_1y_4 \oplus x_2y_1 \oplus x_2y_4 \oplus x_4y_2 \oplus x_2y_6 \oplus x_3y_4 \oplus x_5y_3, x_3y_1 \oplus x_6y_2 \oplus x_2y_6 \oplus x_5y_3 \oplus x_5y_4 \oplus x_5y_6 \oplus x_6y_3 \oplus x_2y_3 \oplus x_4y_6 \oplus x_6y_5 \oplus x_1y_3 \oplus x_5y_5 \oplus x_2y_4 \oplus x_4y_2 \oplus x_4y_5 \oplus x_3y_5 \oplus x_4y_3 \oplus x_6y_1 \oplus x_4y_1, x_3y_1 \oplus x_6y_6 \oplus x_5y_3 \oplus x_5y_6 \oplus x_5y_2 \oplus x_1y_5 \oplus x_1y_1 \oplus x_1y_2 \oplus x_2y_1 \oplus x_2y_3 \oplus x_3y_6 \oplus x_6y_5 \oplus x_1y_3 \oplus x_2y_4 \oplus x_3y_3 \oplus x_4y_5 \oplus x_2y_5 \oplus x_6y_1 \oplus x_4y_1 \oplus x_6y_4 \oplus x_3y_2, x_6y_6 \oplus x_4y_4 \oplus x_5y_4 \oplus x_5y_6 \oplus x_6y_3 \oplus x_1y_6 \oplus x_1y_1 \oplus x_1y_2 \oplus x_2y_1 \oplus x_6y_5 \oplus x_2y_4 \oplus x_4y_2 \oplus x_4y_5 \oplus x_3y_5 \oplus x_6y_1 \oplus x_6y_4).$$

By using these formulas, we successively compute $\psi(v_1, v_1)$, $\psi(v_1, v_2)$, $\psi(v_2, v_1)$ and $\psi(v_2, v_2)$. Finally, we compute the "exclusive-or" of the four obtained results.

Third Variation

To further reduce the size of the ROM needed to store the S-boxes, we can apply simultaneously the ideas of both variations 1 and 2: we use the second variation, with the same secret bijection φ (from 6 bits to 6 bits) and the same secret random function A (from 6 bits to 6 bits) in the new implementation of each non-linear transformation given by a S-box. This variation thus requires only 288 bytes to store the S-boxes. We have applied the Differential Power Analysis on real smartcard implementations of this third variation. Two examples of differential mean curves (with 2048 inputs and with '1111' as target output of the first S-box) are presented in figures 6 and 7. A precise analysis of the 64 curves given by the DPA (see note after step 3, in section 2) shows that none of them appears to be "very special", compared to the others, so that we can say that this implementation resists the DPA attack (at least in its basic form, see appendix 2 for a possible generalization that could still be dangerous).

Fourth Variation

In this last variation, instead of implementing the transformation $(v_1', v_2') = S'(v_1, v_2)$ (which replaces the non-linear transformation $v' = S(v)$ of the initial implementation, given by a S-box) by using two S-boxes, we perform the computation of v_1' (respectively v_2') by using a simple algebraic function (*i.e.* the bits

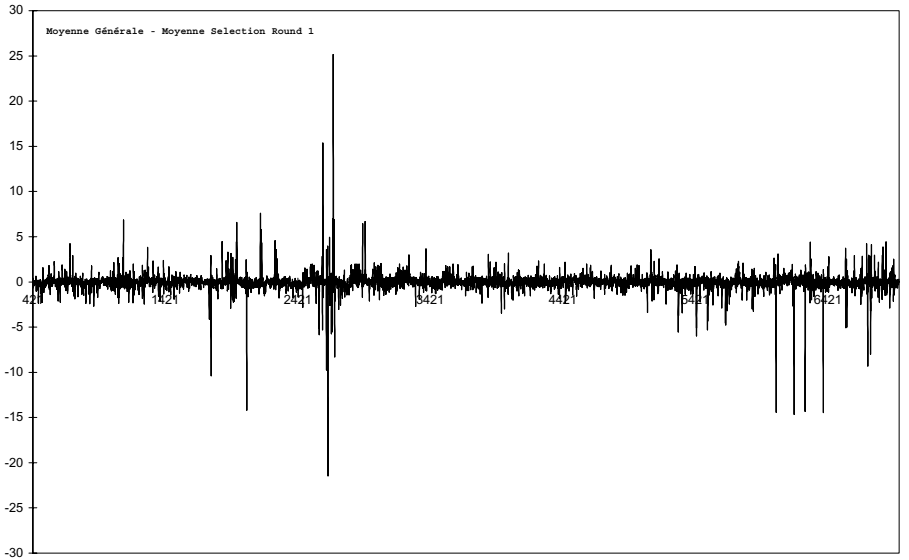


Fig. 6. An example of difference of the curves MC and MC' when the 6 bits are false

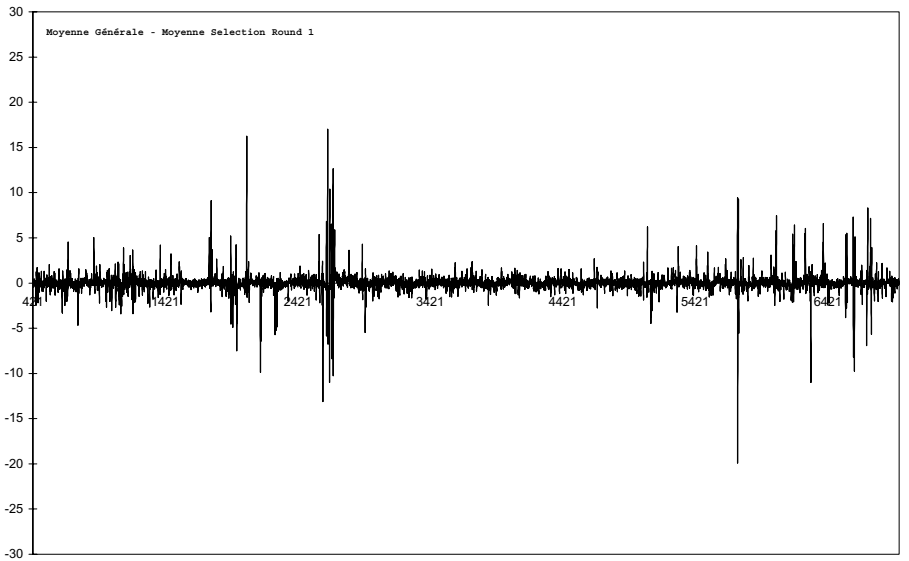


Fig. 7. Difference of the curves MC and MC' when the 6 bits are correct

of v'_1 (respectively v'_2) are given by a polynomial function of total degree 1 or 2 of the bits of v_1 and v_2), then we compute v'_2 (respectively v'_1) by using a table. This enables to reduce again the needed ROM for the implementation. This last variation requires only 256 bytes to store the S-boxes.

6 The RSA Algorithm

The "Power Analysis" attacks also threaten the classical implementations of the RSA algorithm. Indeed, these implementations often use the so-called "*square-and-multiply*" principle to perform the computation of $x^d \bmod n$. It consists in writing the binary decomposition $d = d_{m-1}2^{m-1} + d_{m-2}2^{m-2} + ... + d_1 2^1 + d_0 2^0$ of the secret exponent d, and then in performing the computation as follows:

1. $z \leftarrow 1$;
   For i going backwards from $m-1$ to 0 do:
2. $z \leftarrow z^2 \bmod n$;
3. if $d_i = 1$ then $z \leftarrow z \times x \bmod n$.

In this computation, we see that – among the successive values taken by the z variable – the first ones depend on only a few bits of the secret key d. The fundamental hypothesis that enables the DPA attack is thus satisfied. As a result, we can guess for instance the 10 most significant bits of d by studying the consumption measures on the part of the algorithm corresponding to i going from $m-1$ to $m-10$. We can then continue the attack by using comsumption measures on the part of the algorithm corresponding to i going from $m-11$ to $m-20$, which gives the 10 next bits of d, and so on. We finally find all the bits of the secret exponent d.

The method described in section 3 also applies to securing the RSA algorithm. We use here a separation of each intermediate variable V (whose values lie in the multiplicative group of $\mathbf{Z}/n\mathbf{Z}$), occuring during the computation and depending on the inputs (or the outputs), into two variables V_1 and V_2 (*i.e.* we take $k = 2$), and we choose the function $f(v_1, v_2) = v = v_1 \cdot v_2 \bmod n$. We already saw in section 3 (cf "second example") that this function f satisfies condition 1.

We thus replace x by (x_1, x_2) such that $x = x_1 \cdot x_2 \bmod n$ and z by (z_1, z_2) such that $z = z_1 \cdot z_2 \bmod n$ (in practice, we can for instance choose x_1 randomly and deduce x_2). Considering again the three steps of the "square-and-multiply" method, we perform the following transformations:

1. $z \leftarrow 1$ is replaced by $z_1 \leftarrow 1$ and $z_2 \leftarrow 1$;
2. $z \leftarrow z^2 \bmod n$ is replaced by $z_1 \leftarrow z_1^2 \bmod n$ and $z_2 \leftarrow z_2^2 \bmod n$;
3. $z \leftarrow z \times x \bmod n$ is replaced by $z_1 \leftarrow z_1 \times x_1 \bmod n$ and $z_2 \leftarrow z_2 \times x_2 \bmod n$.

It is easy to check that the identity $z = f(z_1, z_2)$ remains true all along the computation, which shows that condition 2 is satisfied.

Let us notice that the computations performed respectively on the z_1 variable and on the z_2 variable are completely independent. We thus can imagine to

perform the two computations either in a sequential way, or in an overlapped way, or simultaneously in the case of multiprogrammation, or simultaneously in different processors working concurrently.

7 Generalized Attacks

Recently, more general attacks were introduced, where the attacker tries to correlate different points of a power consumption curve. We have no place here to analyze in detail the effect of this idea on the "Duplication Method". However, it is possible to show that if each variable is splitted in, say, k variables, then the complexity of the implementation increases in $\mathcal{O}(k)$, while the complexity of the attack increases exponentially in k.

As concerns DES implementations, we also recommend, when it is possible, to use different S-Boxes for each smartcard (stored in EEPROM). In particular, this avoids some attacks which use a smartcard with a known key to help finding the key in another smartcard whose key is unknown.

8 Conclusion

In this paper, we investigate how the study of the electric consumption measures of an electronic device can be used by an attacker to get information about the secret key of the cryptographic algorithm computed by the chip. More precisely, we focus on the so-called Differential Power Attacks, which were recently introduced by Paul Kocher, and which use a statistical analysis of a set of consumption curves measured for many different inputs of the cryptographic algorithm.

We study more precisely how DPA attacks work, and what precise hypotheses they rely on. We then present several ways of securing cryptosystems. In particular, concrete examples of such countermeasures are described in the cases of DES and RSA, which are the most used cryptographic algorithms at the present.

To secure those algorithms, we essentially study the main idea that consists in splitting each intermediate variable, occuring in the computation, into two (or more) variables, such that the values of these new variables cannot be easily predicted. The obtained implementations can be proved to resist the "local" version of Differential Power Analysis (where the attacker only tries to detect local deviations in the differentials of mean curves). Nevertheless other attacks can be conceived, still using the analysis of electric consumption. We do not pretend to solve all security problems linked to these threats. These latter attacks are not only theoretical, since we found real products that are defeated by them, but it also shows that theoretical investigations have to be continued in that sensitive subject.

References

1. Paul Kocher, Joshua Jaffe, Benjamin Jun, *Introduction to Differential Power Analysis and Related Attacks*, 1998. This paper is available at http://www.cryptography.com/dpa/technical/index.html