High-order Attacks against the Exponent Splitting Protection

Frédéric Muller[1] and Frédéric Valette[2]

[1] HSBC-France Frederic.Muller@m4x.org
[2] CELAR, RENNES, France
Frederic.Valette@m4x.org

Abstract. Exponent splitting is a classical technique to protect modular exponentiation against side-channel attacks. Although it is rarely implemented due to efficiency reasons, it is widely considered as a highly-secure solution. Therefore it is often used as a reference to benchmark new countermeasure proposals.
In this paper, we make new observations about the statistical behavior of the splitting of the exponent. We look at the correlations between the two shares, and show an important imbalance. Later, we show how to use this imbalance in higher-order attacks (mostly based on address-bit, safe-error and fault analysis). We also present experimental results to estimate their feasibility.

1 Introduction

Modular exponentiation is frequently used by public-key cryptosystems, for example RSA [17] or DSA [16]. However, data manipulated during these computations should generally be kept secret, since any leakage of information (even only a few bits of secret information) may be useful to an attacker. For example, during the generation of an RSA signature by a cryptographic device, the secret exponent is used to transform an input related to the message into a digital signature via modular exponentiation.

In recent years, many methods have been proposed to attack these algorithms, using a physical source of information, instead of the usual cryptographic inputs and outputs. The first important result was due to Kocher who suggested to use timing information to retrieve secret keys manipulated by the cryptographic operations [14]. Another interesting idea was proposed by Boneh *et al.* who suggested to modify the physical environment of a cryptographic device to create a fault during the computations [3]. Faulty results sometimes leak information about the secret key.

These attacks, generally called **side-channel attacks**, may represent an important threat for systems. Indeed it is often assumed that cryptographic devices are tamper-resistant, while naive implementations often leak information about the secrets stored and manipulated by the device. Many attacks, either passive (like Kocher's timing attack) or active (like Boneh *et al.*'s fault attack), have been studied, and some generic countermeasures were proposed. Among the possible protection methods, an interesting direction [4], inspired by the well-known secret sharing techniques [18] consists in splitting the secret data in two (or more) shares. Then two (or more) separate computations are performed (one on each share), such that the actual output can be retrieved from the different results. This idea, initially introduced by Chari *et al.* in [4] was further developed by Clavier and Joye in the case of modular exponentiation [6]. Similar methods also exist for secret-key algorithms [10] and for scalar multiplication on elliptic curves [19].

For modular exponentiation, it is called **the exponent splitting method** and is widely considered as a secure solution to thwart side-channel attacks. However its inefficiency (it roughly doubles the execution time) is an important limitation in practice. Recent countermeasures (see [5] for instance) often use the exponent splitting method as a reference to evaluate the security level they achieve.

In this paper, we make new observations about the statistical behavior of the sharing method. As a result, the two separate modular exponentiations have strong correlations. Later, we exploit these correlations in higher-order side-channel attacks, *i.e.* attacks that analyse simultaneously the physical information at two different instants in the computation. More precisely, we describe 4 new higher-order attacks against this countermeasure. They work when all the exponentiation are protected against Simple Power Analysis (SPA), and can even defeat some extra randomization countermeasures. Three of the four attacks are active attacks, and as such require the injection of faults during the cryptographic computations.

Our paper is constructed as follows : first, we remind the Exponentiation Splitting method, as well as several popular side-channel attack techniques in this context. Then, we describe our new results : we start by our new statistical observations, and we continue by suggesting three new high-order fault attacks.

2 The Exponent Splitting Countermeasure

The idea to share a secret in several parts was first introduced by Shamir in [18] for a cryptographic purpose. Later, Chari *et al.* suggested to split a cryptographic computation in several shares [4], in such a way that :

- The actual output can be retrieved from the outputs of each partial computation.
- One needs to attack the scheme as many times as the number of shares in order to retrieve the secret.

In particular, they argued that this approach was a reasonable countermeasure against side-channel attacks. For instance, randomizing the splitting algorithm allows to counter attacks based on statistical analysis.

More specifically in the case of modular exponentiation, Clavier and Joye introduced the idea of **exponent splitting** to thwart side-channel attacks [6]. Similar ideas were described in [19] in the case of scalar exponentiation on elliptic curves. Besides the switching from multiplicative to additive notation, the idea of both methods is essentially the same.

2.1 Definition

Let us consider a secret exponent, noted

$$d = \sum_{i=0}^{n-1} d_i.2^i$$

In many cryptographic algorithms (RSA for instance), one needs to raise some input M to the power d, modulo some large number N. The result is noted :

$$S = M^d \bmod N = \prod_{i=0}^{n-1} M^{d_i.2^i} \bmod N$$

The main idea of the splitting technique is to pick a random r (smaller than d)[1] and to compute the value $r^* = d - r$. Then, one computes separately ($S_r = M^r \bmod N$) and ($S_{r^*} = M^{r^*} \bmod N$) from which it is easy to recover S by :

$$S = S_r \cdot S_{r^*} = M^{(r+r^*)} \bmod N = M^d \bmod N$$

A natural idea is that, since any of the two exponentiations consists in basically raising M to a random exponent, it is sufficient to protect one of the two exponentiations against side-channel attacks.

2.2 Alternative solutions

Because r is purely random, it seems that this countermeasure offers a very high level of security. Alternative protection methods can be grouped in two classes :

- Those based on randomizing the input data (either M or d), prior to the exponentiation algorithm [7, 14]
- Those based on randomizing the exponentiation algorithm itself (see [5, 11]).

For many of these countermeasures used alone, some problems have been identified [8]. So it is customary to combine several countermeasures in implementations, provided it does not affect too badly the performances.

No attack is known against the exponent splitting method, even without additional countermeasure (some basic SPA-protection is still needed, as shown in the next Section). However, the exponent splitting is much less efficient than the alternative propositions, since it doubles the length of the computation. The goal of some recent proposals (see [5] for instance) is to reach the same level of security than exponent splitting, at a more reasonable cost.

[1] one could think of picking a random r smaller than $\varphi(N)$. Although this does not totally thwart our attacks, it changes the analysis as pointed out in Section 5.4

3 Some Usual Side-Channel Attacks

In this section, we describe some popular side-channel attacks against modular exponentiations. In general, one distinguishes between **passive attacks** where an attacker observes some physical variable in the environment, and **active attacks** where the physical environment is modified by the attacker.

Attacks can also be sorted according to which physical mean is used. As an example, many papers focus on power attacks, where the source of information is the power consumption of the cryptographic device. Regarding active attacks (*e.g.* fault attacks), it is not always specified which mean is used to inject a fault. Popular methods used in practice include light and power glitches.

3.1 SPA

Modular exponentiation is generally implemented using a sequence of squaring and multiplication modulo N. Simple Power Analysis (SPA) [15] is based on the natural idea that multiplication and squaring may not result in the same power consumption. It is therefore a passive attack, where one monitors power traces of a cryptographic device executing a modular exponentiation[2]. One expects to retrieve the sequence of squaring and multiplication that was actually executed, from the power traces.

In a naive implementation of modular exponentiation, the multiplication at step i is executed if and only if $d_i = 1$. Therefore **an attacker learns if** $d_i = 1$ **by simply looking if a multiplication was executed at step** i. It is quite simple to thwart SPA by always executing the squaring and the multiplication at step i. When $d_i = 0$, the multiplication is a useless operation, so the *"square-and-multiply always"* algorithm, as depicted in Figure 1 is slightly slower than a naive implementation. It is a very popular algorithm, often implemented in practice

Input: a message M, an n-bit integer $d = \sum_{i=0}^{n-1} d_i 2^i$
Output: M^d
$Q[0] = 1$
for i from $n-1$ down to 0
    $Q[0] = Q[0]^2$
    $Q[1] = Q[0] \times M$
    $Q[0] = Q[d_i]$
return $Q[0]$

Fig. 1. "Square-and-multiply always" algorithm, resistant against SPA

(sometimes in addition to other countermeasures). This SPA-protection remains

[2] SPA has been primarily developed as a power attack, however it adapts very simply to other physical sources of information, like electromagnetic radiations

a requirement for the security of Exponent Splitting. Otherwise an attacker can learn separately r and r^* by running the SPA twice and then reconstruct

$$d = r + r^*$$

However it may seem sufficient to protect only one of the two exponentiations if one considers only SPA. Indeed, since r is random, an attacker learns basically nothing about d if he obtains only r or $d - r$. However, protecting only one of the two modular exponentiations is not a very natural solution.

3.2 Fault Attacks

Cryptographic devices are often sensitive to perturbations of their environment [3]. Fault attacks are based on the assumption that the normal execution of the modular exponentiation can be modified by such physical perturbation. This goal is generally reached by light or power glitches, or temperature variations.

For instance, assume that an attacker is able to flip the value of the bit d_i during the exponentiation of the input M. Then, instead of the correct result S, the attacker obtains the "faulty" result :

$$S' = S \cdot M^{2^i}$$

if $d_i = 0$ and

$$S' = S \cdot M^{-2^i}$$

if $d_i = 1$. Therefore an attacker **learns one bit of the secret exponent, by comparing one correct and one faulty modular exponentiation**.

3.3 Safe errors attacks

Safe errors attacks can be viewed as an enhancement of fault attacks, adapted to thwart some countermeasures. The attacker uses the fact that some of the operations that are executed can be useless. For instance, in the "square-and-multiply always" algorithm of Figure 1, the multiplication is useless when $d_i = 0$.

If a fault is injected at this step of the computation, the result of the exponentiation will be $S' = S$ when $d_i = 0$ and an invalid value otherwise. Like for a basic fault attack, comparing one correct and one faulty modular exponentiation allows an attacker to learn one bit of d. Moreover, the underlying assumptions are much lighter : **it is easier to inject an arbitrary fault, than a fault that specifically flips one bit of the exponent**.

Both faults and safe-error attacks do not apply to exponent splitting, because learning one bit of r (or r^*) does not provide any information about d.

3.4 Address-bit attacks

The address-bit attack is a specific attack to target algorithms like the "square-and-multiply always" of Figure 1, where the fact that d_i is 0 or 1 does not affect the intermediate **values** that are computed, but affects instead the **addresses** that are manipulated. For instance, when $d_i = 1$, $Q[1]$ will be manipulated at the last stage of round i, while it is $Q[0]$ otherwise.

Power attacks [15] or ElectroMagnetic (EM) analysis [9] are often based on a correlation between the **manipulated data** and the physical source of information. However, it is also known that **addresses of manipulated registers** can affect the power dissipation or the EM radiation. Address-bit attacks have been developed to take advantage of such properties [12]. Suppose we use EM as the physical source of information and that our probe is physically closer to register A than register B. If a group of experiments all read the register A, their EM signature will be significantly different from a group of experiments reading register B.

This idea has been used to break the basic "square-and-multiply-always" algorithm : since there is no extra randomization, the address-bit is always d_i at step i. Therefore the EM signature at this stage of the computation depends on d_i. However this attack no longer works for the exponent splitting method, since the address-bits are randomized at each execution.

3.5 Impact on Exponent Splitting

We observed that, taken separately, none of this well-known attack techniques allows to break the Exponent Splitting protection. Indeed, each exponentiation uses a random exponent, so attacks requiring some degree of randomization are not possible.

In addition, provided at least one of the exponentiation is SPA-protected, SPA does not work against the Exponent Splitting. However, in practice, it is better to **protect both exponentiations against SPA**. Otherwise, an attacker could mount a **combined attack** : apply SPA to the unprotected exponentiation to learn (for instance) r, then attack the remaining SPA-protected exponentiation by other means.

For instance, one could think of an address-bit attack on M^{r^*}, assuming prior knowledge of r : the attacker repeats several time the computation for a fixed given M. He makes an assumption about the i Least Significant Bits (LSB) of the secret d. From this guess and from r, he gets one candidate for the i LSB's of r^* of each computation. Then he looks at the i-th step of M^{r^*} and performs an address-bit attack, as described in Section 3.4. If the guess is right, he will observe two groups with significantly different EM signatures. Otherwise, he will just observe some random data. So he learns the i LSB's of d and it is straightforward to repeat the process to learn more bits of d.

To summarize, it is recommended to **protect both exponentiations against SPA**, in order to thwart combined attacks. With this assumption, no attack is known against the exponent splitting protection. In particular no additional countermeasure (like exponent randomization for instance) is needed.

4 New Attacks Against the Exponent Splitting

We focus on some statistical properties of the exponent splitting, at the bit level. Since r is randomly drawn, it does not leak information about d when considered alone[3]. However, the pair (r, r^*) is not uniformly distributed, since it always satisfies

$$r + r^* = d$$

We explore the statistical impact of this relation at the bit-level. Later, we use the observed imbalance to mount side-channel attacks.

4.1 Statistical properties of the exponent splitting

Although each bit of r and r^* takes the values 0 or 1 with probability 0.5, there is a bias in the distribution of the i-th bits of the pair (r, r^*). We denote respectively by r_i, r_i^* and d_i the i-th bits of r, r^* and d. Besides, the i-th carry bit in the addition $r + r^* = d$ is noted c_i. At the bit level, the following relation is satisfied :

$$c_i + r_i + r_i^* = d_i + 2.c_{i+1} \tag{1}$$

In particular, we have

$$c_i \oplus r_i \oplus r_i^* = d_i \tag{2}$$

Let also p_i denote the probability that $c_i = 0$. The probability is taken over all the possible choices of r and r^*. Initially, $p_0 = 1$.

Suppose that $d_i = 0$. Since r_i is drawn at random, when $c_i = 0$, we get from (2) that :

$$(r_i, r_i^*) = (0, 0)$$

with probability 0.5 and

$$(r_i, r_i^*) = (1, 1)$$

with probability 0.5. In the first case, we obtain from (1) that $c_{i+1} = 0$, while in the second case $c_{i+1} = 1$.

Similarily, when $c_i = 1$, it is equaly likely that (r_i, r_i^*) is equal to $(0, 1)$ or $(1, 0)$. In both cases, we get $c_{i+1} = 1$. Therefore, the transition rule for these probabilities can be summarized as in Table 1. In the case $d_i = 1$, we obtain another rule for probability transitions which is given in Table 2.

From these equations, we can observe that the two bits (r_i, r_i^*) are not uniformly distributed (unless $p_i = 0.5$) and that the imbalance depends on the value of the bits from 0 to i of the secret exponent. Acutally, what we have is a **Markov chain** where the bit-level probabilities of step i can be derived from those of step $i - 1$ using one of the two previous probability transition rules.

[3] Actually, it is not true : r is generally drawn at random in the interval $[0, d]$ which is not exactly equivalent to drawing at random an n-bit integer. This results in (small) imbalances that have already been used for cryptanalysis purpose [1]

$$
\begin{aligned}
\Pr[(r_i, r_i^*) = (0,0)] &= 0.5 \times p_i \\
\Pr[(r_i, r_i^*) = (0,1)] &= 0.5 \times (1 - p_i) \\
\Pr[(r_i, r_i^*) = (1,0)] &= 0.5 \times (1 - p_i) \\
\Pr[(r_i, r_i^*) = (1,1)] &= 0.5 \times p_i \\
p_{i+1} &= 0.5 \times p_i
\end{aligned}
$$

Table 1. Probability transition when $d_i = 0$

$$
\begin{aligned}
\Pr[(r_i, r_i^*) = (0,0)] &= 0.5 \times (1 - p_i) \\
\Pr[(r_i, r_i^*) = (0,1)] &= 0.5 \times p_i \\
\Pr[(r_i, r_i^*) = (1,0)] &= 0.5 \times p_i \\
\Pr[(r_i, r_i^*) = (1,1)] &= 0.5 \times (1 - p_i) \\
p_{i+1} = (0.5 \times (1 - p_i)) + p_i &= 0.5 \times (1 + p_i)
\end{aligned}
$$

Table 2. Probability transition when $d_i = 1$

4.2 An example

To illustrate our ideas, we have drawn at random an exponent d of length 24 bits and repeated the splitting method a large number of times. We computed experimentally the probability distribution of (r_i, r_i^*) for all steps i. Table 3 summarizes these results.

	d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}	d_{13}	d_{14}	d_{15}	d_{16}	d_{17}	d_{18}	d_{19}	d_{20}	d_{21}	d_{22}	d_{23}
(r_i, r_i^*)	0	1	0	1	0	1	0	0	0	0	0	0	1	0	0	1	0	0	0	1	1	1	1	1
$(0,0)$	50	25	38	31	35	33	34	16	8	4	2	1	50	25	13	45	28	14	8	47	23	11	5	2
$(1,0)$	0	25	12	19	15	17	16	34	41	46	48	49	0	25	37	5	22	36	42	3	27	39	45	48
$(0,1)$	0	25	13	18	15	17	16	33	42	46	49	49	0	25	36	6	22	36	43	4	28	40	46	49
$(1,1)$	50	25	37	32	35	33	34	17	9	4	1	1	50	25	14	44	28	14	7	46	22	10	4	1

Table 3. An example of bit-level imbalance for a 24-bit secret d

Intuitively, when the secret exponent has a long run of bits equal to 0 or 1, it is very likely that the bits of r and r^* are different. In the case of a run of 0's, we can see that p_i becomes very close to 0, so there is generally no carry bit. However, after a long run of 1's, p_i gets close to 1, so a carry bit is likely to propagate. In the next section, we show applications of these bit-level observations to mount safe-error attacks, fault attacks and address-bit attacks. We assume that an **attacker can infer the value of the bits of d from the probability distribution**

of (r_i, r_i^*). This problem (called the Hidden Markov Problem) has already been handled by Karlof and Wagner in [13]. We also mention that the paper [8] deals with a similar problem to break the Ha-Moon countermeasure [11].

4.3 Application to safe error attack

In this section, we assume that an attacker is able to create faults with enough precision to target a specific step in the "square-and-multiply always" algorithm. We consider a **second-order safe-error attack**, *i.e.* the attacker injects two faults during the same computation and observes if the result remains valid or not.

Suppose the attacker injects a fault during the multiplication at step i of the exponentiation M^r, and a fault during the multiplication at step i of the exponentiation M^{r^*}. These two faults have no effect on the final computation, as long as $(r_i, r_i^*) = (0, 0)$.

By repeating the process, the attacker obtains an estimation of the probability $\Pr[(r_i, r_i^*) = (0, 0)]$ for the positions i of his choice. We have seen in Section 4.1 that, this probability is strongly correlated with the value of the bits d_i (see Table 3 for a concrete example). This observation allows the attacker to learn the secret exponent (refer to Section 5 for further analysis).

4.4 Application to fault attacks

In this section, we use an idea similar to the previous attack, although we are not specifically focusing on safe-errors, *i.e.* faults which will keep the output of the exponentiation unchanged. We suppose that an attacker is able to "flip" the value of the bit r_i by fault injection, like in the usual fault attack (see Section 3.2).

Depending on the value of the bit r_i, the output can be modified to $S \cdot M^{2^i}$ (if $r_i = 0$) or $S \cdot M^{-2^i}$ (if $r_i = 1$). Since r_i is random, this provides no information about d. However, suppose that we consider a **second-order fault attack** where we simultaneously flip the bit r_i and the bit r_i^*. Depending on the value of these two bits, the observed result can take 4 values :

$$\begin{array}{ll}
\text{if } (r_i, r_i^*) = (0, 0) \text{ then we get} & S \cdot M^{2^{i+1}} \\
\text{if } (r_i, r_i^*) = (0, 1) \text{ then we get} & S \\
\text{if } (r_i, r_i^*) = (1, 0) \text{ then we get} & S \\
\text{if } (r_i, r_i^*) = (1, 1) \text{ then we get} & S \cdot M^{-2^{i+1}}
\end{array}$$

Like in the previous section, we obtain a safe-error in 2 of the 4 cases. This allows us to tell when $r_i \neq r_i^*$, although we cannot tell between the two cases $(0, 1)$ and $(1, 0)$. In addition, we can also detect here the case $(0, 0)$ and $(1, 1)$ because we obtained specific faults in the output of the modular exponentiation.

By repeating the process over several experiments, we get much more information than in the previous section, since we learn estimates for

- $\Pr[(r_i, r_i^*) = (0, 0)]$

- $\Pr[(r_i, r_i^*) = (1, 1)]$
- $\Pr[(r_i, r_i^*) = (0, 1)$ or $(1, 0)]$

This information makes the analysis of the Hidden Markov Model [13] easier, however the injected faults need to specifically flip one bit of the exponent. This is more difficult to obtain in practice than an arbitrary fault on the multiplication. Therefore it is not clear that our second-order fault attack will require less messages than the second-order safe-error attack, although it provides a better statistical information.

4.5 Application to address-bit attack

In this section, we suggest to mount a **statistical address-bit attack**. The idea is similar to the usual address-bit attacks, although here we only know probabistically if the two address-bits are equal or not.

Suppose that we already know the $i-1$ less significant bits of d. Therefore we know that the carry bit c_i is equal to 0 with a probability p_i, that can be computed using the rules given in Section 4.1.

If $d_i = 0$ then $r_i = r_i^*$ with probability p_i.
If $d_i = 1$ then $r_i = r_i^*$ with probability $1 - p_i$.

This observation allows us to learn the bit d_i (using EM radiations for instance, as described in Section 3.4), by comparing the addresses manipulated at stage i of both modular exponentiations. Suppose, for example, that $p_i > 0.5$. Then the address-bits should be equal more often if and only if $d_i = 0$. Clearly, the "bad case" here is when $p_i = 0.5$, since we are unable to determine the value of d_i. However, such bad cases remain unlikely, as illustrated in Table 3. The advantage of this address-bit attack is that it is a passive attack. See Section 5 for more details about implementation of this attack.

4.6 Combining safe-error and address-bit attacks

The efficiency of the previous safe-error attack can be improved, by combining it with fault attacks. We want to improve the prediction of the carry bit c_i, so we inject one fault at step $i-1$, while we simultaneously monitor the EM radiations of step i. This might look complicated, but it is not necessarily more difficult than injecting two faults during the same cryptographic computation.

In order to predict the carry bit c_i, we need some information about the step $i-1$. So, we inject an arbitrary fault during the multiplication at round $i-1$, for any one of the two modular exponentiations. If the result remains valid, we learn that $r_{i-1} = 0$. Otherwise $r_{i-1} = 1$.

- In the case $d_{i-1} = 1$. We inject a fault at the $(i-1)$-th step until we find an exponentiation where $r_{i-1} = 0$. Then from relation (1) we see that necessarily, $c_i = 0$. Therefore the address bits r_i and r_i^* are equal if and only if $d_i = 0$. This allows to apply the usual address-bit attack, as described in Section 3.4.

– In the case $d_{i-1} = 0$, we inject a fault until we find an exponentiation where $r_{i-1} = 1$. Then, we see that necessarily, $c_i = 1$. Therefore the address bits r_i and r_i^* are equal if and only if $d_i = 1$.

The advantage of this combined attack is that it no longer requires any Markov model analysis, so the number of required message is much smaller.

4.7 Summary

We proposed a variety of attacks against the exponent splitting countermeasure, based on statistical properties of the sharing method, at the bit-level. We proposed a passive attack based on **statistical address-bit analysis** and several active attacks, either based on **faults** or **safe-errors**. See Table 4 for a summary of our proposed attacks.

Type of attack	*Needs Markov analysis ?*	*Active ?*	*Number of Faults*
Safe-error	yes	yes	2
Fault	yes	yes	2
Address-bit	yes	no	0
Combined	no	yes	1

Table 4. Summary of our proposed attacks. The number of faults is given per sample.

5 Experimental results

In this Section, we implemented software simulations for the safe-error attack and the statistical address-bit attack. Both are based on a Markov model analysis, where one wants to retrieve d_i from partial information about the distribution of (r_i, r_i^*). We want to obtain a more accurate estimation of the real cost of this analysis. As we have seen previously, some problem will arise. For instance, when $p_i = 0.5$, it is impossible to tell whether d_i is equal to 0 or 1. In particular, such problems happen after a long run of consecutive 0's or 1's. We want to estimate the impact of this "difficult" positions.

Besides, we did not implement the fault attack, since it relies essentially on the same principle than the safe-error attack, although the underlying assumptions are much stronger (we need the ability to inject faults that specifically flip the value of some exponent bits). We did not implement the combined attack either, since it is an improvement of the statistical address-bit attack. Besides, there is no Markov model analysis in this attack (see Table 4), so its complexity depends on the quality of our address-bit observations : in theory, 2 observations per bit of the secret exponent should be sufficient.

5.1 Safe-error attack

Let $q_i = \Pr[(r_i, r_i^*) = (0,0)]$. From the probability transition rules of Table 1 and 2, we obtain :

$$\begin{array}{lll} \text{if } d_{i-1} = d_i = 0 & \text{then} & q_i = 0.5 \cdot q_{i-1} \\ \text{if } d_{i-1} = d_i = 1 & \text{then} & q_i = 0.5 \cdot q_{i-1} \\ \text{if } d_{i-1} = 0 \text{ and } d_i = 1 & \text{then} & q_i = 0.5 \cdot (1 - q_{i-1}) \\ \text{if } d_{i-1} = 1 \text{ and } d_i = 0 & \text{then} & q_i = 0.5 \cdot (1 - q_{i-1}) \end{array}$$

Then, we adopt a recursive approach : we learn d_i from d_{i-1}, by testing whether the probability q_i is closer to $0.5 \cdot q_{i-1}$ or from $0.5 \cdot (1 - q_{i-1})$. Clearly, the only problem arises when $q_{i-1} \simeq 0.5$ where it is very difficult to make a decision. The following Table 5 summarizes our experimental results, where L denotes the exponent length and D the number of experiments, *i.e.* the number of fault injections here. Besides, this process must be repeated for each bit of the scalar, so the number of experiments is about $L \times D$ in total. Actually, the parameter D was chosen such that we could derive the value of the most "difficult" bits. For many positions, the value of d_i is easy to determine with much less than D faults. This was not taken into account in our evaluation of the cost. To average these figures, we repeated the attack several hundred times, for randomly chosen exponents. To conclude, if one wants to remove most errors

L	D	Errors	Unable to decide
40	20	3.93	5.21
40	100	2.22	2.40
160	100	8.48	6.79
160	300	6.06	1.81
160	1000	3.58	1.22
1024	100	52.25	39.90
1024	1000	7.26	37.48

Table 5. Experimental results for the safe-error attacks

and "no decision" cases, one needs about 100 faults for a 160-bit scalar, and 1000 faults for a 1024-bit scalar.

5.2 Statistical address-bit attack

We implemented the statistical address-bit attack. We suppose that the i LSB's of the scalar are known and try to determine if the next bit is 0 or 1 by looking at the equality of the address-bits r_i and r_i^*. The results are summarized in Table 6. We observe that these figures are slightly better than those of Table 5. Besides,

l	D	Errors	Unable to decide
40	20	2.57	1.83
40	100	1.65	0.40
160	100	5.78	1.64
160	300	3.82	0.49
160	1000	2.35	0.12
1024	100	35.90	9.52
1024	1000	12.49	1.10

Table 6. Experimental results for the statistical address-bit attack

the attack does not need to be repeated for each bit of the scalar, since we can monitor the N steps of the exponentiation algorithm. So this attack is actually much more efficient than the safe-error attack. However, these two attacks rely on different physical assumptions : the safe-error attack is active, although the underlying assumption is much weaker than for fault attacks (where one bit of the exponent needs specifically to be flipped). On the other hand, the address-bit attack is passive.

5.3 Dealing with Errors and the "Unable to decide" case

In the previous statistical attacks, it occured that we were unable to make a decision between $d_i = 0$ and $d_i = 1$. In particular, this occurs after long runs of 0 or 1's, where the statistical behavior is quite special.

Actually, it is not necessarily a problem if some bits of the secret scalar remain unknown at the end of the attack. Indeed, there are some mathematical solutions to deal with it. First, if the number of unknowns is small, we can simply guess these bits. Secondly, there exist some algorithms to take advantage of (even relatively small) partial key exposure for an RSA exponent (see the paper by Blömer and May, for instance [2]). Therefore, we may retrieve the "missing" bits by mathematical means, once several bits have been leaked using side-channels.

5.4 Additional countermeasures

A natural question is to tell whether such attacks can be thwarted by putting more countermeasures (in addition to the SPA countermeasures).

- Countermeasures based on **randomizing the message** will not work here. Indeed, our attacks do not exploit the actual values that are manipulated during the exponentiations. So it does not matter that the message is initially randomized.
- There is **another way to initially split the secret exponent** : draw at random r in the interval $[0, \varphi(N)]$, then compute $r^* = d - r \bmod \varphi(N)$. This

is not likely to be implemented, because $\varphi(N)$ is not always known by the device (although it could be recomputed from d and e). The problem with this alternative splitting is that we could face two possible targets instead of one :

$$r + r^* = d \text{ and } r + r^* = d' = d + \varphi(N)$$

It is better if d is far from $\frac{\varphi(N)}{2}$, because one of the two targets is over-represented, and the other one just acts as noise. So we would recover, either d or $d + \varphi(N)$, but in both cases this is equivalent to the secret key. The tricky case is when $d \simeq \frac{\varphi(N)}{2}$, because d and d' occur equally often. An open problem is to propose a dedicated analysis, in order to recover two such exponents simultaneously.

– However, as soon as the device knows $\varphi(N)$, it is very likely that the **exponent randomization** would be implemented. The idea of this countermeasure is to draw a random x for each exponentiation, and to perform each exponentiation with exponent $d + x \cdot \varphi(N)$ instead of the actual exponent d. Implementing this protection apparently thwarts our attacks.
– Countermeasures based on **randomizing the exponentiation algorithm itself** (see [5] for a nice example) seem also to thwart our attacks.

6 Conclusion

Contrarily to a widespread belief, the exponent splitting countermeasure does not offer, by itself, a satisfying level of security against side-channel attacks. Although analyzing a single modular exponentiation is useless, attacks become possible as soon as one considers both exponentiations together.

We described statistical weaknesses of the countermeasure at the bit-level, which show that the i-th stages of both modular exponentiations are strongly correlated. We showed a variety of attacks that break the exponent splitting countermeasure (safe-error, fault, address-bit, combined attacks). All of them are higher-order attacks, *i.e.* they require to exploit simultaneously both exponentiations. There are some technical difficulties to realize second-order attacks (like injecting two faults in the same cryptographic computations), however it is unreasonable for the security to rely on this difficulty only. Therefore, we recommand not to use the exponent splitting protection alone. One should either combine it with additional countermeasures like the randomization of the exponent, or use some of the recent alternative, like [5], which has the advantage of being much more efficient.

References

1. D. Bleichenbacher. On the Generation of DSA One-time Keys. Presented at the *Workshop on Elliptic Curve Cryptography – ECC'02*, 2002.
2. J. Blömer and A. May. New Partial Key Exposure Attacks on RSA. In D. Boneh, editor, *Advances in Cryptology – CRYPTO'03*, volume 2729 of *Lectures Notes in Computer Science*, pages 27–43. Springer, 2003.

3. D. Boneh, R. DeMillo, and R. Lipton. On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract). In W. Fumy, editor, *Advances in Cryptology – Eurocrypt'97*, volume 1233 of *Lectures Notes in Computer Science*, pages 37–51. Springer, 1997.
4. S. Chari, C. Jutla, J. Rao, and P. Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In M. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lectures Notes in Computer Science*, pages 398–412. Springer, 1999.
5. B. Chevallier-Mames. Self-Randomized Exponentiation Algorithms. In T.Okamoto, editor, *CT-RSA 2004*, volume 2964 of *Lectures Notes in Computer Science*, pages 2.3–249. Springer, 2004.
6. C. Clavier and M. Joye. Universal Exponentiation Algorithm. In Ç. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES) – 2001*, volume 2162 of *Lectures Notes in Computer Science*, pages 300–308. Springer, 2001.
7. J-S. Coron. Resistance Against Differential Power Analysis for Elliptic Curve Cryptosystems. In Ç. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES) – 1999*, volume 1717 of *Lectures Notes in Computer Science*, pages 292–302. Springer, 1999.
8. P-A. Fouque, F. Muller, G. Poupard, and F. Valette. Defeating Countermeasures Based on Randomized BSD Representations. In M. Joye and J-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems (CHES) – 2004*, volume 3156 of *Lectures Notes in Computer Science*, pages 312–327. Springer, 2004.
9. K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic Analysis : Concret Results. In Ç. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES) – 2001*, volume 2162 of *Lectures Notes in Computer Science*, pages 251–261. Springer, 2001.
10. L. Goubin and J. Patarin. DES and Differential Power Analysis, The "Duplication" Method. In Ç. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES) – 1999*, volume 1717 of *Lectures Notes in Computer Science*, pages 158–172. Springer, 1999.
11. J. Ha and S. Moon. Randomized signed-scalar Multiplication of ECC to resist Power Attacks. In B. Kaliski, Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES) – 2002*, volume 2523 of *Lectures Notes in Computer Science*, pages 551–563. Springer, 2002.
12. K. Itoh, T. Izu, and M. Takenaka. Address-Bit Differential Power Analysis of Cryptographic Schemes OK-ECDH and OK-ECDSA. In B. Kaliski, Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES) – 2002*, volume 2523 of *Lectures Notes in Computer Science*, pages 129–143. Springer, 2002.
13. C. Karlof and D. Wagner. Hidden Markov Model Cryptanalysis. In C. Walter, Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES) – 2003*, volume 2779 of *Lectures Notes in Computer Science*, pages 17–34. Springer, 2003.
14. P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Others Systems. In N. Koblitz, editor, *Advances in Cryptology – Crypto'96*, volume 1109 of *Lectures Notes in Computer Science*, pages 104–113. Springer, 1996.
15. P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M. Wiener, editor, *Advances in Cryptology – Crypto'99*, volume 1666 of *Lectures Notes in Computer Science*, pages 388–397. Springer, 1999.

16. National Institute of Standards and Technology (NIST). Digital Signature Standard (DSS) FIPS Publication 186-2, February 2000. Available at `http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf`.
17. R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. In *Communications of the ACM 21(2)*, pages 120–126, 1978.
18. A. Shamir. How to Share a Secret. *Communications of the ACM (CACM)*, 22(11):612–613, November 1979.
19. E. Trichina and A. Bellezza. Implementation of Elliptic Curve Cryptography with Built-In Counter Measures against Side Channel Attacks. In B. Kaliski, Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES) – 2002*, volume 2523 of *Lectures Notes in Computer Science*, pages 98–113. Springer, 2002.