Public Key Perturbation of Randomized RSA Implementations

Alexandre Berzati[1,2], Cécile Canovas-Dumas[1], Louis Goubin[2]

[1] CEA-LETI/MINATEC, 17 rue des Martyrs, 38054 Grenoble Cedex 9, France,
{alexandre.berzati,cecile.canovas}@cea.fr
[2] Versailles Saint-Quentin-en-Yvelines University,
45 Avenue des Etats-Unis, 78035 Versailles Cedex, France
Louis.Goubin@prism.uvsq.fr

Abstract. Among all countermeasures that have been proposed to thwart side-channel attacks against RSA implementations, the exponent randomization method – also known as exponent blinding – has been very early suggested by P. Kocher in 1996, and formalized by J.-S. Coron at CHES 1999. Although it has been used for a long time, some authors pointed out the fact that it does not intrinsically remove all sources of leakage. At CHES 2003, P.-A. Fouque and F. Valette devised the so-called "*Doubling Attack*" that can recover the blinded secret exponent from an SPA analysis. In this paper, we consider the case of fault injections. Although it was conjectured by A. Berzati *et al.* at CT-RSA 2009 that exponent randomization avoids fault attacks, we describe here how to recover the RSA private key under a practical fault model. Our attack belongs to the family of public key perturbations and is the first fault attack against RSA implementations with the exponent randomization countermeasure. In practice, for a 1024-bit RSA signature algorithms, the attack succeeds from about 1000 faulty signatures.
Keywords: RSA, fault attacks, exponent randomization/blinding, public modulus.

1 Introduction

The exponent randomization method – also referred as exponent blinding – has been first suggested by P. Kocher [11]. This method inspired J.-S. Coron [7] that later formalizes it to defeat side channel attacks, such as DPA, that gain the information leaked during the exponentiation. This method is widely used because it is easy to implement and the induced overhead is reasonable. However any implementation may still be a potential source of leakage.

The first attack published against this countermeasure is due to P.-A. Fouque and F. Valette [10]. The so-called *"Doubling Attack"* allows an attacker to recover a blinded secret exponent from an SPA analysis. This attack only works for *"Left-To-Right"*-based implementations of the modular exponentiation. Moreover, the attacker is assumed to be able to send many times the same known message and that no message randomization is performed before the modular

exponentiation. At CHES 2006 [8], P.-A. Fouque *et al.* show that if Coron's countermeasure is used with some windowing exponentiation algorithms and a small public key, then a simple SPA combined with a tricky analysis makes it possible to recover both secret key and factorize the public modulus. It is worthwhile to notice that this attack exploits the *non-uniformity* of the exponent randomization countermeasure (see Sect. 3.1). Instead of exploiting the physical leakage due to the execution of a modular exponentiation, like in previous attacks, P.-A. Fouque *et al.* proposed at CHES 2008 [9] to focus on the leakage induced by the computation of the random exponent itself. Since the secret exponent and the blinding part are cut into words, spying on the carries of the adder may reveal information that is used to guess the most significant bits of each word of the secret key. When the number of missing bits is small enough, the attacker can use classical methods, such as Shanks' Baby-Step Giant-Step algorithm, to obtain the whole secret key.

The exponent randomization may also be used to protect implementations against some fault attacks. Namely, this countermeasure is useful to defeat attacks that require multiple faulty signatures to recover the private exponent since each signature is computed with a different exponent. Although the device embedding this countermeasure still remains vulnerable to perturbation, it does not exist, as far as we know, any method for exploiting such faulty outputs. This paper bridges the gap by providing a new fault attack that defeats the exponent randomization. This attack belongs in the recent family of public key perturbations.

Exploiting the perturbation of public elements has been first addressed by I. Biehl *et al.* with several applications to elliptic curves [3]. But it took a half decade before seeing a successful application to RSA [12]. The first exploitation of the RSA public modulus perturbation leading to a full secret key recovery is due to E. Brier *et al.* [5] (see also [6] for further optimizations). In the case of the last attack, the use of the blinded exponent is an efficient countermeasure. A new fault attack based on the public modulus corruption has been proposed lately by A. Berzati *et al.* against both *"Right-To-Left"* and *"Left-To-Right"* implementations of the core RSA modular exponentiation [2,1]. Unlike previous works, the attack takes advantage of a perturbation of the modulus that occurs while the device is performing a signature. Such a fault injection splits the signature into a correct and a faulty part and so, isolates a part of the secret exponent. Then, from a correct/faulty signature pair, the attacker can *guess-and-determine* both faulty modulus and the part of secret exponent. The whole exponent is obtained by cascading the attack on signatures corrupted at different moments of the execution. At CT-RSA'09, authors claimed that the exponent randomization may be used to defeat their fault attack [1].

In this article, we show that even if this countermeasure is used, it is possible to recover the private exponent under a practical fault model. To the best of our knowledge, this is the first fault attack that aims to threaten RSA implementations with the exponent randomization countermeasure. The analysis takes advantage of the *non-uniformity* of the exponent randomization. As a con-

sequence, this work completes the state-of-the-art of the side channel analysis of the exponent randomization.

The remainder of this paper is organized as follows: Section 2 describes classical implementations of RSA and the random exponent countermeasure. Our fault analysis is detailed in Sect. 3 and summarized as an algorithm in Sect. 4. Finally, we conclude in Sect. 5 about the vulnerability of random exponent countermeasure implementations in the context of fault attacks.

2 Background

2.1 Notations

Let N, the public modulus, be the product of two large prime numbers p and q. The length of N is denoted by n. Let e be the public exponent, coprime to $\varphi(N) = (p-1)\cdot(q-1)$, where $\varphi(\cdot)$ denotes Euler's totient function. The public key exponent e is linked to the private exponent d by the equation $e \cdot d \equiv 1 \bmod \varphi(N)$. The private exponent d is used to perform the operations below.

RSA Decryption: Decrypting a ciphertext C boils down to compute $\tilde{m} \equiv C^d \bmod N$. If no error occurs during computation, transmission or decryption of C, then $\tilde{m}$ equals m.

RSA Signature: The signature of a message m is given by $S \equiv \dot{m}^d \bmod N$ where $\dot{m} = \mu(m)$ for some hash and/or deterministic padding function μ. The signature S is validated by checking that $S^e \equiv \dot{m} \bmod N$.

2.2 Modular exponentiation algorithms

Algorithm 1: *"Right-To-Left"* modular exponentiation

INPUT: m, d, N
OUTPUT: $A \equiv m^d \bmod N$

```
1 : A:=1;
2 : B:=m;
3 : for i from 0 upto (n − 1)
4 :     if (d_i == 1)
5 :         A := (A · B) mod N;
6 :     end if
7 :     B := B^2 mod N;
8 : end for
9 : return A;
```

Algorithm 2: *"Left-To-Right"* modular exponentiation

INPUT: m, d, N
OUTPUT: $A \equiv m^d \bmod N$

```
1 : A:=1;
2 : for i from (n − 1) downto 0
3 :     A := A^2 mod N;
4 :     if (d_i == 1)
5 :         A := (A · m) mod N;
6 :     end if
7 : end for
8 : return A;
```

Binary exponentiation algorithms are often used to compute the RSA modular exponentiation $\dot{m}^d \bmod N$ where the exponent d is expressed in a binary

form as $d = \sum_{i=0}^{n-1} 2^i \cdot d_i$, where d_i stands for the i-th bit of d. Their polynomial complexity with respect to the input length make them very interesting to perform the core RSA operation. Algorithm 1 describes a way to compute modular exponentiations by scanning bits of d from least significant bits (LSB) to most significant bits (MSB). That is why it is usually referred to as the *"Right-To-Left"* modular exponentiation algorithm.

The dual algorithm that implements the binary modular exponentiation is the *"Left-To-Right"* exponentiation described in Algorithm 2. This algorithm scans bits of the exponent from MSB to LSB and is lighter than *"Right-To-Left"* one in terms of memory consumption.

It exists multiple implementations derived from these dual algorithms, such as `OpenSSL` *fixed/sliding window* implementations or the *Square-and-Multiply-always* variant [7]. For the sake of clarity, we will only focus our presentation on the binary version of the *"Right-To-Left"* method. But the principle of our analysis can be easily adapted to attack its variants.

2.3 Exponent Randomization

The exponent randomization method has been proposed by P. Kocher [11] to defeat side channel attacks, such as DPA, that gain information leaked during the exponentiation. The principle of this countermeasure is based on Fermat's theorem. Indeed, for all $m \in (\mathbb{Z}/N\mathbb{Z})^*$ and $\lambda \in \mathbb{Z}$, $m^{\lambda \cdot \varphi(N)} \equiv 1 \bmod N$. The exponent randomization algorithm derived from this result is detailed below. The complexity of the modular exponentiation algorithm is polynomial with

Algorithm 3: RSA exponent randomization algorithm

INPUT: $\dot{m}$, N, $\varphi(N)$, d and l
OUTPUT: S $= \dot{m}^d \bmod N$

1: *//Randomize the private exponent*
2: Pick a random $\lambda \in [\![0; 2^l - 1]\!]$;
3: $\bar{d} = d + \lambda\varphi(N)$;
4: *//Perform the exponentiation*
5: $S =$ `PowMod`$(\dot{m}, \bar{d}, N)$;
6: **return** S;

respect to the exponent length. Thus, to guarantee a reasonable overhead, the l value as to be small compared to the RSA length n. Typically, for a 1024-bit RSA, $l = 20$ or $l = 32$.

3 Description of our attack

3.1 Bit analysis of a randomized exponent

In this section, we aim to analyze the influence of the different variables that are involved in the computation of a randomized exponent. By definition, the

blinded exponent $\bar{d}$ is built by adding a random multiple of $\varphi(N)$ to the secret exponent d. Using a different expression of $\varphi(N)$, the expression of $\bar{d}$ can also be written as:

$$\begin{aligned} \bar{d} &= d + \lambda\varphi(N) \\ &= d + \lambda\,(p-1)\,(q-1) \\ &= d + \lambda N - \lambda\,(p+q-1) \end{aligned} \tag{1}$$

From the previous expression, one can notice that the randomized exponent $\bar{d}$ is built by adding 3 terms of different sizes. As a consequence, the bits of d are not homogeneously masked by this method. Figure 1 illustrates this statement for a n-bit RSA and a l-bit random value λ. This figure highlights that despite

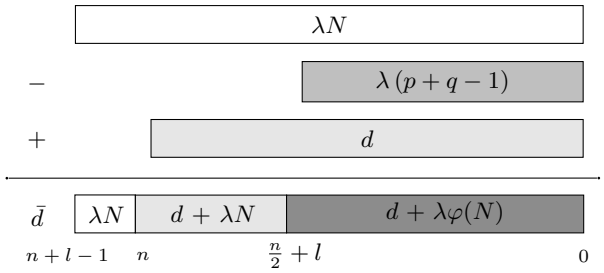

Fig. 1. Bit analysis of a random exponent

being properly masked by a multiple of $\varphi(N)$, the most significant bits of d are masked by a random multiple of N. But the blinding method seems to be more efficient for the least significant bits. In the next section, we will explain how to use the heterogeneity of the exponent randomization method for exploiting faults.

3.2 Fault Model

Description. The model we have chosen to perform the attack is derived from the ones previously used by A. Berzati *et al.* to successfully attack both *"Right-To-Left"* [2] and *"Left-To-Right"* [1] implementations of standard RSA. We suppose that the attacker is able to inject a transient fault that modifies the public modulus N during the execution of a signature with a randomized exponent $\bar{d}$ for a known input message m (see Sect 2.3). The injected fault affects a byte of the modulus by modifying it in a random way, namely:

$$\hat{N} = N \oplus \varepsilon \tag{2}$$

where $\varepsilon = R_8 \cdot 2^{8i}$, $i \in [\![0; \frac{n}{8} - 1]\!]$ and R_8 is a non-zero random byte value. In our assumption, the value of the faulty modulus $\hat{N}$ is not known, *a priori*, by the

attacker. In this article, we consider that the exponentiation is implemented with the *"Right-To-Left"* method or a variant. The fault is injected during a square, at the t-th step of the exponentiation and such that the end of the execution is performed with the faulty modulus $\hat{N}$. It is also assumed that the time location of the injection is controlled by the attacker, and so, the parameter t may be set (or known) by the attacker depending on the exponent part he aims to recover.

Discussion. This fault model has been chosen because of its practicability in the smartcard context. Although the effect of a fault injection is highly dependent of the component attacked, it seems that a random modification of the value of a memory word can be easily produced by a laser. This model has been already used in the literature leading to successful applications [13,4,1]. Furthermore, the timing control of the fault injection is not a restrictive assumption since the attacker can trigger the laser shots using a Simple Power Analysis.

3.3 Result of a Faulty Computation

Let $\bar{d} = \sum_{i=0}^{n+l-1} 2^i \cdot \bar{d}_i$ be the binary representation of a randomized exponent $\bar{d}$. According to the fault model described above, the fault occurs during a square at the t-th step of the execution. Hence, if B_{t-1} denotes the internal register value that contains the result of the consecutive squares before the fault injection:

$$\begin{aligned} \hat{B}_t &\equiv {B_{t-1}}^2 \bmod \hat{N} \\ &\equiv \left(\dot{m}^{2^{t-1}} \bmod N\right)^2 \bmod \hat{N} \end{aligned} \tag{3}$$

The subsequent operations of the exponentiation are also performed with the faulty modulus. If we denote by $A_t \equiv \dot{m}^{\sum_{i=0}^{t} 2^i \cdot \bar{d}_i}$ the internal state value before the fault injection. The result of the faulty RSA signature $\hat{S}_t$ can be written as:

$$\hat{S}_t \equiv A_t \cdot \hat{B}_t^{\bar{d}_t} \cdot \ldots \cdot \hat{B}_t^{2^{(n+l-1)-t} \cdot \bar{d}_{n+l-1}} \bmod \hat{N} \tag{4}$$

$$\equiv A_t \cdot \hat{B}_t^{\frac{\bar{d}_{[t]}}{2^t}} \bmod \hat{N} \tag{5}$$

where $\bar{d}_{[t]} = \sum_{i=t}^{n+l-1} 2^i \cdot \bar{d}_i$. The previous equation highlights that the fault has isolated the most significant part of the blinded exponent $\bar{d}$. In other words the perturbation gives the opportunity for the attacker to focus on the recovery of a part of the exponent. The next section reminds the general methodology used to exploit faults on the public modulus during the execution of the exponentiation (see also [2,1] for further details).

3.4 Analysis

In the following sections we will detail the effects of faults that have been injected according to the model described above. Then we will propose different ways for exploiting perturbations, depending on their timing location t.

General Methodology. The general principle of the analysis consists in making use of the isolation of a part of exponent by the fault injection. Indeed, if the isolated part of exponent is small enough, it is possible to *guess-and-determine* it from a faulty/correct signature pair $(\hat{S}_t, S_t)$. Therefore, since the faulty modulus is also unknown by the attacker, he chooses a candidate value $\hat{N}'$ and another candidate value $\bar{d}'_{[t]}$ for the most significant part of the randomized exponent he has to determine. Then he computes from the correct signature:

$$A'_t \equiv S_t \cdot \dot{m}^{-\bar{d}'_{[t]}} \bmod N \tag{6}$$

This computation aims to retrieve the value of the internal register A_t when the fault occurred. The next step consists in using the candidate values to simulate a faulty end of exponentiation. To do so, the attacker computes:

$$S'_{(\bar{d}'_{[t]}, \hat{N}')} \equiv A'_t \cdot \left(\dot{m}^{2^{t-1}} \bmod N\right)^{2 \cdot \frac{\bar{d}'_{[t]}}{2^t}} \bmod \hat{N}' \tag{7}$$

Finally, he checks if the following equation is satisfied:

$$S'_{(\bar{d}'_{[t]}, \hat{N}')} \equiv \hat{S}_t \bmod \hat{N}' \tag{8}$$

In the case of satisfaction, it means that the chosen candidate pair is the correct one with high probability. Otherwise, the attacker has to choose another candidate pair and perform this test again. One can notice that a similar analysis can be performed when the first operation infected by the fault is a multiplication The details of this variant are provided in [2].

Contrary to the attack presented in [2], the subsequent bits of exponent can not be obtained by repeating the analysis on a signature faulted earlier. Indeed, the exponent randomization countermeasure implies that a fresh exponent is used for each execution of the signature. Hence, it is not possible to repeat the attack by using the knowledge of already found bits of blinded exponent $\bar{d}_{[t]}$ as in [2]. As a consequence we have to adapt this general methodology to extract real bits of the private exponent d from bits of $\bar{d}$ recovered by the analysis detailed above. We show in the following parts how to make use of the *non-homogeneity* of the exponent randomization to do so.

Case of unexploitable faults. This case corresponds to fault that have been injected at final steps of the exponentiation, namely for $n \leq t \leq (n + l - 1)$. The few amount of information about the secret key that belongs in this range of data is due to the carry propagation of the addition of d and a random multiple of $\varphi(N)$ (see Fig. 1). So, the analysis of signatures faulted in this timing range is not relevant for extracting information about the secret key d since. As a consequence, it is worthwhile focusing on faults injected earlier in the computation.

Faults on MSB. This section aims to provide a method for analyzing RSA signatures that have been faulted while the t-th bit of the blinded exponent is treated, namely if $\left(\frac{n}{2}+l\right) \leq t < n$. As we said in the previous section, our goal is to extract some bits of the real exponent from the recovered part of randomized exponent. In fact, performing the attack by this way seems to be difficult. First, $\bar{d}$ depends on d but also on a random multiple of $\varphi(N)$ and all these values are unknown by the attacker. This difficulty can be overcome thanks to the non-homogeneity of the exponent randomization (see Fig. 1). Indeed, in the bound $\left(\frac{n}{2}+l\right) \leq t < n$, we have:

$$\bar{d} \approx d + \lambda N \tag{9}$$

As a consequence, the most significant part of $\bar{d}$ only depends on d and λ. Instead of searching $\bar{d}_{[t]}$ and extracting bits of d from it, we have decided to directly guess both d and λ by building "good" candidate values for $\bar{d}_{[t]}$:

$$\bar{d}_{[t]} = \sum_{i=t}^{n+l-1} 2^i \cdot \bar{d}_i \tag{10}$$

$$\approx \sum_{i=t}^{n+l-1} 2^i \cdot (d + \lambda N)_i \tag{11}$$

$$\approx \sum_{i=t}^{n-1} 2^i \cdot d_i + \sum_{i=t}^{n+l-1} 2^i \cdot (\lambda N)_i + carry_t\,(d, \lambda N) \cdot 2^t \tag{12}$$

where $carry_t\,(a, b)$ denotes the carry bit resulting from the bit-wise addition of the t first bits of a and b. Equation (12) shows that guessing $\bar{d}_{[t]}$ boils down to simultaneously guess the random value λ and the $(n-t)$ most significant bits of d. In the general case, this part of d splits into a known (already recovered) part d_{MSB} and w missing bits denoted by d_w. The carry bit is considered as an uncertainty on the parity of d_w. As a consequence, if an attacker builds candidate values for $\bar{d}_{[t]}$ that satisfy (8), then he can directly deduce $w-1$ new bits of d. Thus, the known part d_{MSB} grows up of $w-1$ bits, and it can be used again to analyze signatures faulted earlier in the execution and cascade the resolution of almost half of the secret key. That way, even if the isolated part of random exponent grows up, the part of exponent to be determined d_w remains constant. Hence, instead of guessing a pair of candidate values to satisfy (8) as described in the general methodology, we guess the triplet of values $(d_w, \lambda, \hat{N})$ and thus deduce a part of the secret key.

By carefully studying our improvement, one can wonder if using the sole relation (8) is enough for determining with high probability a triplet of values. This remark is all the more relevant since our implementation of the attack showed us that multiple candidate triplets may satisfy (8). The authors of [2,1] previously proved that the order false-acceptance probability is about $\frac{1}{N}$. Thus, it is highly negligible for common RSA length. But, we also noticed that for all false-accepted triplets, the candidate values accepted for λ are always smaller than the correct one. Hence, the triplet that contains the biggest candidate value

for λ is always the correct one. This heuristic was successfully adopted to improve our attack algorithm by reducing the number of candidate triplets that satisfy (8) to the correct one only. We also formalized this heuristic in the following theorem. The proof of the theorem is given in Appendix A.

Theorem 1. *Let $\hat{S}_t$ be a faulty signature performed under an exponent randomized by λ, and S the corresponding correct signature. For all candidate pairs $(d'_w, \lambda') \in [\![0; 2^w]\!] \times [\![0; 2^l]\!]$, if $\lambda' > \lambda$, then (8) can not be satisfied.*

As a consequence, by combining the approximation of the randomization (see Eq. (12)) for building candidate values for $\bar{d}_{[t]}$, and the theorem above, an attacker will be able to recover a part of the real private exponent d with high probability from only one correct/faulty signature pair. Moreover, our method enables the attacker to use the already found bits of d and cascade the analysis for signatures faulted earlier in their execution. By this way, it is possible to recover almost all the most significant bits of d.

Faults on LSB. We will focus here in the recovery of the least significant bits of d. From Fig. 1, if $0 \leq t < \left(\frac{n}{2} + l\right)$, then $\varphi(N)$ is, this time, fully involved in the randomization of the secret exponent d. Since this value is private, contrary to the modulus N, the attacker can not run the analysis described for the recovery of the most significant part of d. But, we will describe in this section how we have used the previous analysis on multiple faulty signatures to overcome this difficulty.

As we previously said, we can not approximate the least significant bits of the blinded exponent $\bar{d}$ as the sum of the real exponent d and a random multiple of the public modulus (see Fig. 1). But, let us rewrite the expression of a part of randomized exponent isolated by the fault injection:

$$\bar{d}_{[t]} = \sum_{i=t}^{n+l-1} 2^i \cdot (d + \lambda\varphi(N))_i \tag{13}$$

$$= \sum_{i=t}^{n+l-1} 2^i \cdot (d + \lambda N - \lambda(p + q - 1))_i \tag{14}$$

$$\approx \sum_{i=t}^{n-1} 2^i \cdot \delta_i + \sum_{i=t}^{n+l-1} 2^i \cdot (\lambda N)_i \tag{15}$$

where $\delta_i = (d - \lambda(p + q - 1))_i$. As for the MSB case, for any iteration of the analysis, we assume that the attacker has already determined both most significant parts of d and $(p+q-1)$ respectively denoted by d_{MSB} and $(p+q-1)_{MSB}$. Here the value $\delta = \sum_{i=t}^{n-1} 2^i \cdot \delta_i$ splits into w missing bits denoted by δ_w and a part δ_{MSB} that depends on λ, d_{MSB} and $(p + q - 1)_{MSB}$. This last part is also unknown, but it becomes computable whenever λ is guessed. Equation (15) shows that the attacker can apply the same *guess-and-determine* method to recover the part of randomized exponent $\bar{d}_{[t]}$ that satisfies (8) by building it from

the triplet of candidate values for $(\delta_w, \lambda, \hat{N})$. Hence, such an analysis applied on a correct/faulty signature pair only returns the searched triplet with high probability. But, one can notice that contrary to the MSB case, the analysis does not directly returns a part of d, but a more intricate value δ_w. So, the attacker has to perform a complementary analysis on the variable δ_w he has just recovered to extract both expected parts of d and $\lambda(p+q-1)$. According to Fig. 1, it is relevant to notice that δ_w depends on:

- w unknown bits of the exponent d,
- w unknown bits of the sum of RSA primes $(p+q-1)$,
- the random value λ that has been just recovered,
- some carry bits.

Thus, from this value, the attacker obtains one equation that involves two unknown variables. In order to recover simultaneously w bits of the exponent and w bits of the sum of RSA primes, it is necessary to get additional equations (at least one more). This can be achieved by repeating this analysis on a signature faulted at the same step of its execution to recover another δ_w value for a different λ.

For the sake of clarity, we have voluntary withdrawn the influence of the carry bits in the system. In practice, as for the MSB case, these carry bits add some uncertainty on the low order bits of d_w. In other words, instead of recovering a unique value for d_w, in practice two or three solutions are returned in the worst case. But, the wrong values obtained for d_w will be discarded when subsequent analysis will be performed. Thus, the number of false-accepted candidates does not grow up exponentially in cascading the resolution, but stay bounded. When the part of d recovered by repeating the described analysis is large enough, the attacker may complete the attack by using classical methods such as Shank's Baby-Step Giant-Step or lattice techniques.

4 Attack Algorithm

4.1 Summary of our attack

In this section, we detail the implementation of our Differential Fault Analysis described above. This part completes our previous theoretical approach by providing a more pragmatic description of our attack methodology. This algorithm has been successfully implemented on a standard PC using the GMP Library [3] leading.

Gather faulty signatures. The attacker first chooses a window length w for the recovery of the secret key d. Then he has to gather multiple signatures faulted at different steps of the execution. If we denote by t the time location of the fault injection, the attacker has to gather:

[3] The GNU Multiple Precision Library. Available at `http://gmplib.org/`

- One faulty signature and the corresponding correct one if $\left(\frac{n}{2}+l\right) \leq t < n$
- Two (or more) faulty signatures and the corresponding correct one if $0 \leq t < \left(\frac{n}{2}+l\right)$

where t is decremented by w each time. The collected signatures are sorted in descending fault locations.

Analysis of the MSB. For each correct/faulty signature pairs the attacker *guesses-and-determines* the triplet of values $(d_w, \lambda, \hat{N})$. The part of exponent d_w is composed by the bits obtained from previous analysis and the w bits to guess. Hence, with our method, the analysis of one correct/faulty signature pair reveals each time w bits of d. So, this analysis has to be repeated on all signature pairs $(\hat{S}_t, S_t)$ such that $\left(\frac{n}{2}+l\right) \leq t < n$ to recover almost all the most significant bits of d.

Analysis of the LSB. For all the gathered signatures, the attacker has to perform an analysis split into two parts:

- First, he has to *guess-and-determine* two or more triplets of values $(\delta_w, \lambda, \hat{N})$ from different pairs of faulty/correct signatures
- Then, he extracts both w bits of d and $(p+q-1)$ by solving the obtained system of equations.

As a result, the analysis of the signatures faulted at the same step t of their respective execution allows an attacker to recover both a w-bit part of d and $(p+q-1)$. As for the MSB case, this step has to be repeated (completed by Baby-Step Giant-Step or lattice techniques if necessary) to recover the missing bits of d.

4.2 Performance

Fault Number. Since our attack is based on fault injection, it seems relevant to evaluate the number of faulty signatures an attacker has to collect to recover the secret key. According to the description of our analysis (see Sect. 3.4), the number of faults depends on the part of d the attacker aims to recover. In the case of the MSB, the attacker will be able to recover w bits of d from one correct/faulty signature pair. For the LSB, the attacker has to collect at least two signatures faulted at the same step, since he has to solve a system of equations to extract w bits of d. As a result, the number of faulty signatures to collect $\mathcal{F}$ is:

$$\mathcal{F} = \mathcal{O}\left(\frac{n}{w}\right) \tag{16}$$

In practice, for a 1024-bit RSA and a resolution window length $w = 2$, our attack succeeded from about 1000 faulty signatures which is a little more than expected. This extra cost is due to the LSB analysis that required an average of 3 faulty signatures to correctly recover both parts of d and $(p+q-1)$. But this number of fault is still reasonable and highlights the practicability of our fault attack.

Complexity. The general principle of our attack is based on extracting a w part of d from each correct/faulty signature pair collected according to the model. This can be achieved by guessing and determining simultaneously the w bits of d isolated by the fault, the l-bit random value λ and the faulty modulus $\hat{N}$. Therefore, according to the fault model and the described algorithm, the computational complexity $\mathcal{C}$ of our attack is:

$$\mathcal{C} = \mathcal{O}\left(\frac{2^{(w+l)} \cdot n^2}{w}\right) \text{exponentiations} \tag{17}$$

The extra analysis required for the LSB case does not appear in this expression since it is dominated by the search of a triplet that satisfy (8). Moreover one can notice that the complexity exponentially depends on the random length l. So lengthening λ exponentially hardens our analysis. But, some computational optimizations can be done to bypass this problem.

Computational Optimizations. In this part, we propose to speed up the execution of our fault attack by using some optimizations inspired from particular feature of the attack. First, instead of computing candidate values the faulty modulus "on the fly", the attacker can precompute a dictionary of possible values for the faulty modulus according to the fault model chosen. Moreover, by using the Theorem 1, the attacker may advantageously compute the *guess-and-determine* step by decrementing the candidate values for λ and stopping when a triplet satisfies (8). This optimization is all the more interesting if the exponent is blinded with a λ close to 2^l. The last optimization also concerns the *guess-and-determine* step. Indeed, one can notice that for a given faulty signature, all candidate values can be tested independently. As a consequence, this step can be easily computed in parallel. So, if an attacker can get a cluster of k machines, then he can distribute the *guess-and-determine* step and reduce the global complexity of the attack $\mathcal{C}$ to $\frac{\mathcal{C}}{k}$.

5 Conclusion

This paper presents the first fault attack against implementations of an RSA signature scheme that embedding the exponent randomization countermeasure. Through their *"Doubling Attack"*, P.-A. Fouque and F. Valette first alerted the community that using this countermeasure may introduce a physical leakage if it is combined with a *"Left-To-Right"*-based modular exponentiation. We complete this work by showing in this paper that implementations of RSA based on the dual exponentiation may be vulnerable to fault. Indeed, we demonstrate that the exploitation of a reasonable number of faulty signatures may lead to a full secret key recovery. Moreover the GMP implementation of our method as well as the use of a practicable fault model provide evidences that the perturbation of public elements represents a real threat for RSA implementation, even randomized. Thus, it might be worthwhile to check the effective robustness of the exponent blinding against other fault attacks.

References

1. A. Berzati, C. Canovas, J-G. Dumas, and L. Goubin. Fault Attacks on RSA Public Keys: Left-To-Right Implementations are also Vulnerable. In M. Fischlin, editor, *RSA Cryptographer's Track (CT-RSA 2009)*, volume 5473 of *Lecture Notes in Computer Science*, pages 414–428, San Francisco (USA), 2009. Springer.
2. A. Berzati, C. Canovas, and L. Goubin. Perturbating RSA Public Keys: an Improved Attack. In *Cryptographic Hardware and Embedded Systems (CHES 2008)*, Lecture Notes in Computer Science, Washington DC (USA), 2008. Springer-Verlag.
3. I. Biehl, B. Meyer, and V. Müller. Differential Fault Attacks on Ellitic Curve Cryptosystems. In M. Bellare, editor, *Advances in Cryptology (CRYPTO 2000)*, volume 1880 of *Lecture Notes in Computer Science*, pages 131–146. Springer-Verlag, 2000.
4. J. Blömer and M. Otto. Wagner's Attack on a secure CRT-RSA Algorithm Reconsidered. In L. Breveglieri, I. Koren, D. Naccache, and J-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography (FDTC 2006)*, volume 4236 of *Lecture Notes in Computer Science*, pages 13–23. Springer-Verlag, 2006.
5. E. Brier, B. Chevallier-Mames, M. Ciet, and C. Clavier. Why One Should Also Secure RSA Public Key Elements. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems (CHES 2006)*, volume 4249 of *Lecture Notes in Computer Science*, pages 324–338. Springer-Verlag, 2006.
6. C. Clavier. *De la sécurité physique des crypto-systèmes embarqués*. PhD thesis, Université de Versailles Saint-Quentin, 2007.
7. J-S. Coron. Resistance Against Differential Power Analysis for Elliptic Curve Cryptosystems. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES 1999)*, volume 1717 of *Lecture Notes in Computer Science*, pages 292–302. Springer, 1999.
8. P-A. Fouque, S. Kunz-Jacques, G. Martinet, F. Muller, and F. Valette. Power Attack on Small RSA Public Exponent. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems (CHES 2006)*, volume 4249 of *Lecture Notes in Computer Science*, pages 339–353. Springer, 2006.
9. P-A. Fouque, D. Réal, F. Valette, and M. Drissi. The Carry Leakage on the Randomized Exponent Countermeasure. In E. Oswald and P. P. Rohatgi, editors, *Cryptographic Hardware and Embedded Systems (CHES 2008)*, volume 5154 of *Lecture notes in Computer Science*, pages 198–213. Springer, 2008.
10. P-A. Fouque and F. Valette. The Doubling Attack – *why Upwards Is Better than Downwards*. In C.D. Walter, Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES 2003)*, volume 2779, pages 269–280. Springer, 2003.
11. P. Kocher. Timing attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, *Advances in Cryptology (CRYPTO 1996)*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.
12. J-P. Seifert. On Authenticated Computing and RSA-Based Authentication. In *ACM Conference on Computer and Communications Security (CCS 2005)*, pages 122–127. ACM Press, 2005.
13. D. Wagner. Cryptanalysis of a provably secure CRT-RSA algorithm. In *Proceedings of the 11th ACM Conference on Computer Security (CCS 2004)*, pages 92–97. ACM, 2004.

A Proof of the Theorem 1

Let us consider a candidate value λ' for the random value λ used in a faulty RSA signature $\hat{S}_t$ such that $\lambda' > \lambda$. Then we can also write $\lambda' \geq \lambda + 1$. Now, let us use this relationship to build a candidate value for $d_{[t]}$:

$$\lambda' \cdot N \geq (\lambda + 1)\, N \tag{18}$$

$$\Leftrightarrow \lambda' \cdot N \geq \lambda N + N \tag{19}$$

But, since the secret key d is computed as the invert of e modulo $\varphi(N)$, we also know that:

$$N > \varphi(N) > d \tag{20}$$

As a consequence, we can deduce from the previous relations that:

$$\lambda' N > \lambda N + d \tag{21}$$

$$\Rightarrow \lfloor \frac{\lambda' N}{2^t} \rfloor > \lfloor \frac{\lambda N + d}{2^t} \rfloor \tag{22}$$

Now let us rewrite the previous equation using the binary representation of the operands:

$$\sum_{i=t}^{n+l-1} 2^{i-t} \cdot (\lambda' N)_i > \sum_{i=t}^{n+l-1} 2^{i-t} \cdot (\lambda N + d)_i \tag{23}$$

$$\Leftrightarrow \sum_{i=t}^{n+l-1} 2^{i} \cdot (\lambda' N)_i > \sum_{i=t}^{n+l-1} 2^{i} \cdot (\lambda N + d)_i \tag{24}$$

$$\Leftrightarrow \sum_{i=t}^{n+l-1} 2^{i} \cdot (\lambda' N)_i > \bar{d}_{[t]} \tag{25}$$

From this inequality, we can conclude that any candidate value computed with λ' will be strictly greater than the searched $\bar{d}_{[t]}$. So, (8) can not be satisfied for such a λ' $\square$