Perturbating RSA Public Keys: an Improved Attack

Alexandre Berzati[1,2], Cécile Canovas[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. Since its first introduction by Bellcore researchers [BDL97], fault injections have been considered as a powerful and practical way to attack cryptosystems, especially when they are implemented on embedded devices. Among published attacks, Brier *et al.* followed the work initiated by Seifert to raise the problem of protecting RSA public elements.
We describe here a new fault attack on RSA public elements. Under a very natural fault model, we show that our attack is more efficient than previously published ones. Moreover, the general strategy described here can be applied using multiple transient fault models, increasing the practicability of the attack.
Both the theoretical analysis of the success probability, and the experimental results – obtained with the GMP Library on a PC –, provide evidence that this is a real threat for all RSA implementations, and confirm the need for protection of the public key.

Keywords: RSA, fault attacks, DFA, public key.

1 Introduction

Since the advent of fault attacks, most cryptographic algorithms have been endangered [BECN$^+$04,BS97,CJRR99]. The difficulty of modeling the fault, depending on the attacker abilities, makes it uneasy to define countermeasures [Gir05b]. It is particularly the case for the RSA algorithm, which has been shown vulnerable to many attacks injecting faults on temporary values during the computation, or on value of the private key itself.

Moreover, the vulnerability of the public key elements has been recently proved to be a new security potential threat against various RSA implementations [Sei05,Mui06,BCMCC06]. As the effect of a computation perturbation can take multiple forms, mounting an attack based on the use of an altered public modulus is quite realistic.

In this context we describe here a new efficient attack that exploits a few faults on the modulus and leads to a full recovery of the private exponent in a very reasonable time.

After a brief presentation of RSA, Sect. 3 provides an overview of the previous attacks on standard RSA. We particularly focus on modulus attacks so as to compare them with our new attack. Then we will explain the principle of our method, and give a detailed theoretical analysis of its complexity, based on a detailed computation of the involved probabilities.

2 Background

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 following operations:

RSA Decryption: Decrypting a ciphertext C boils down to compute $\tilde{m} \equiv C^d \bmod N \equiv C^{\sum_{i=0}^{i=n-1} 2^i \cdot d_i} \bmod N$ where d_i stands for the i-th bit of d. 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 $S = \dot{m}^d \bmod N$ where $\dot{m} = \mu(m)$ for some hash and/or deterministic padding function μ. The signature S is validated by checking that $S^e \equiv \dot{m} \bmod N$.

2.1 Modular exponentiation: *"Right-To-Left"* algorithm

In all the paper, we will consider the *"Right-To-Left"* algorithm (see for instance [YKLM02]), which is one of the most used algorithm to perform the modular exponentiation. This algorithm scans the bits of the private exponent d from the least to the most significant ones.

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

```
INPUT: $m$, $N$, $d$
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 :         $A := (A \cdot B) \bmod N$;
6 :     endif
7 :     $B := B^2 \bmod N$;
8 : endfor
9 : return $A$;
```

3 Previous work

3.1 Bellcore's DFA against Standard RSA

Bellcore's researchers not only introduced the concept of Differential Fault Analysis [BDL97] by attacking RSA in CRT mode but they also showed how this new side channel attack can be applied to many public key cryptosystems and their various implementations, such as standard RSA. They explain in [BDL97] and [BDL01] how to advantageously analyse the injection of a fault during the standard RSA signature process to recover the secret exponent. Their attack is described against a particular exponentiation algorithm: the *"Right-To-Left"* one (see Sect. 2.1).

The considered fault model is a transient or permanent bit modification of the memory area containing the current value of the exponentiation algorithm. According to [BDL97,BDL01], recovering d by using windows of length l with a probability greater than $1/2$ requires $(n/l) \cdot \log(2n)$ (message, faulty signature) pairs. In terms of complexity, this attack needs to perform $\mathcal{O}(n^3 \cdot 2^l \cdot \log^2(n)/l^2)$ modular exponentiations. It is worth noticing that the choice of the window length l has an impact on the global complexity of the attack.

This attack principle was later studied and generalized by J. Blömer and M. Otto [Ott04].

Bellcore's attack principle. The attack can be divided into two parts. The first one is *"on-line"* and consists in getting sufficiently many erroneous signatures $\hat{S}_i$ from known plaintexts m_i that are randomly distributed over $\mathbb{Z}/N\mathbb{Z}$. The second part is completely *"off-line"* and consists in analysing the previously obtained faulty signatures. The attack principle is described below in the case of a transient fault model:

1. Getting sufficiently many $(m_i, \hat{S}_i)$ pairs, by injecting a transient fault on the current value during each signature execution.
2. Error analysis. Let S_v be the correct signature and $\varepsilon = \pm 2^b$ the induced error with $0 \leq b < n$. The effects of such a transient error, that has occurred during some unknown iteration j, can be modeled as:

$$\hat{S}_v \equiv \left[\left(\prod_{i=0}^{j-1} \dot{m_v}^{2^i d_i} \right) \pm 2^b \right] \cdot \prod_{i=j}^{n-1} \dot{m_v}^{2^i d_i} \tag{1}$$

$$\equiv S_v \pm \left(2^b \cdot \prod_{i=j}^{n-1} \dot{m_v}^{2^i d_i} \right) \mod N \tag{2}$$

$$\Rightarrow S_v \equiv \hat{S}_v \pm 2^b \cdot \dot{m_v}^{w} \mod N, \text{ where } w = \sum_{i=j}^{n-1} 2^i d_i \tag{3}$$

Using the public exponent e, we finally obtain:

$$\dot{m_v} \equiv (\hat{S}_v \pm 2^b \cdot \dot{m_v}^w)^e \bmod N \tag{4}$$

One can notice from the previous equation that it only depends on the message $\dot{m_v}$ and its faulty signature $\hat{S}_v$.

The analysis consists in recovering the whole private exponent d by scanning l-bit long windows from the most to the least significant bits. To reach this goal, the values of b and w satisfying (4) are simultaneously searched. The value of w contains a known part of d and at most l unknown bits. These bits are recovered by testing values in $[\![0; 2^l - 1]\!]$ until one of them satisfies (4). *A priori*, from a given pair $(m_i, \hat{S}_i)$, an attacker can not guess when the fault occurs during the signature's computation. So, for each searched value of w, he has to test all the obtained couples $(m_i, \hat{S}_i)$.

In [BDL01], Boneh *et al.* proved that this method allows an attacker to recover the whole private exponent d with a probability greater than 1/2.

3.2 Fault Attack on RSA private exponent

This attack was published by F. Bao *et al.* in [BDJ+96,BDJ+98] and then in [BECN+04]. The principle is to induce a transient error during the decryption, that produces the same effect as a bit modification of the private exponent. In practice this fault will be a *shunt* of the conditional test on the private bit value during the binary exponentiation algorithm.[1] Note that this attack is suitable for multiple errors [BDJ+98]. Moreover the principle can be adapted to attack cryptosystems based on discrete logarithm (DSA, El-Gamal, ...). The following paragraph only describes the attack for a bit error on the exponent d.

Attack principle. In case of a faulty computation, the deciphered text $\hat{m}$ is:

$$\hat{m} \equiv C^{\hat{d}} \bmod N$$

The fault is exploited by dividing the erroneous result by a correct one: $\frac{\hat{m}}{m}$. The induced error can be modeled as a *bit-flip* of the j-th bit of d. We thus have:

$$\hat{m} \equiv C^{\sum_{i=0, i\neq j}^{i=n-1} 2^i \cdot d_i + 2^j \bar{d_j}} \bmod N$$

That implies, either $\frac{\hat{m}}{m} \equiv \frac{1}{C^{2^j}} \bmod N \Rightarrow d_j = 1$,
or $\frac{\hat{m}}{m} \equiv C^{2^j} \bmod N \Rightarrow d_j = 0$.

This method can be repeated until we obtain enough information on the private exponent. This attack strategy was later extended and generalized by M. Joye *et al.* [JQBD97], who describes an improved attack relying on the mere knowledge of the faulty deciphered text.

[1] Algorithms *"Right-To-Left"* or *"Left-to-Right"*.

3.3 J-P. Seifert and J. Muir's attacks

Seifert's attack on RSA signature [Sei05] was the first one using a modification of some public parameter (*i.e.* the modulus N). Unlike the previously described attacks, the objective does not consist in retrieving the secret key, but in compromising the signature verification mechanism.

Attack principle. Seifert's attack is composed of two different phases.

The first – *"off-line"* – phase consists in finding an altered modulus $\hat{N}$, that satisfy some interesting properties, and generating the corresponding signature. In practice the attacker modifies the modulus $\hat{N}$ so that e is coprime to $\varphi(\hat{N})$ [2] and $\hat{N}$ is a possible altered value of N. Then the attacker has to choose an adequate model for the fault that will disrupt the signature verification mechanism. Seifert proposes to require $\hat{N}$ to be prime, so that the previous condition is satisfied and $\hat{d} \equiv e^{-1} \bmod \varphi(\hat{N})$ is easily computable. Muir generalizes the condition and imposes that $\hat{N}$ should be easily factorized [Mui06]. The attacker signs a message m with the computed $\hat{d}$ value and saves the signature with its corresponding message $(m,\hat{S})$.

In the second – *"on-line"* – phase, the attacker inputs $(m,\hat{S})$ into the signature's verification mechanism and tries to inject a fault during the loading of the N value, so that all computations are performed with this altered modulus. The generated fault has to correspond to the chosen fault model (*i.e.* the altered N value equals to previously computed $\hat{N}$ value). In that case, the signature $\hat{S}$ will be accepted by the algorithm. Otherwise, the attacker performs the *"on-line"* phase until its faulty signature is accepted.

The success rate of this attack and the average number of faults depends on the suitability of the chosen fault model. Moreover the attacker must be able to induce a fault corresponding to $\hat{N}$ with a reasonable probability. The resulting implementation of this attack and a further optimization are proposed in [Mui06].

3.4 E. Brier *et al.*'s attack

Whether it is necessary or not to protect RSA public elements was an open question until Brier *et al.* attack proposal for recovering the whole private key. This attack, inspired from Seifert's one [Sei05], was published in [BCMCC06] and reviewed in [Cla07]. It makes it possible to extract the private key using a modulus perturbation. Moreover, in its simplest version, it does not require any hypothesis on the type of induced fault during the signature process. This represents a significant advantage, compared to Seifert's attack.

Attack principle without dictionary. The attacker proceeds in two distinct phases. The feature of this method (without dictionary) is that it does not require

[2] This is equivalent to e being invertible in $\mathbb{Z}/\varphi(\hat{N})\mathbb{Z}$.

any fault model.

In the first *"on-line"* phase, the attacker conducts a perturbation campaign in order to obtain a large enough number of (message, faulty signature) pairs of the form $(m_i, \hat{S}_i)_{1 \leq i \leq K}$, corresponding to computations with unknown (modified) moduli $\hat{N}_i \neq N$. As in Seifert's attack, the N value is modified during its loading, so that each pair satisfies the following relation:

$$\forall i \in [\![1; K]\!],\ \hat{S}_i \equiv \dot{m_i}^d \bmod \hat{N}_i$$

The *"off-line"* phase consists in analysing the gathered data in order to retrieve the secret key, by an application of the Chinese Remainder Theorem. The value $d \bmod r_k$ is gradually determined for small power of some small prime numbers r_k. When $R = \prod_k r_k$ is greater than N (and so than $\varphi(N)$), the Chinese Remainder Theorem is applied for finding d. The way the values $d_k \equiv d \bmod r_k$ are found, is based on a probabilistic approach that is described in [BCMCC06,Cla07]. We note that the method does not require to model the induced fault.

Implementing the attack shows that approximately 25000 faults are necessary to recover 512 bits of exponent d. In comparison, approximately 60000 faults (more than twice) are necessary to extract 1024 bits.

Attack principle with a dictionary. The attack with dictionary requires the choice of a fault model. From this model and the correct modulus N, the attacker builds a list of possible modified moduli, called modulus dictionary. As in the previous case, this attack is divided into two phases.

The first phase is *"on-line"*. As before, the attacker conducts a fault campaign in order to obtain sufficiently many (message, faulty signature) pairs, denoted by $(m_i, \hat{S}_i)_{1 \leq i \leq K}$.

The attacker begins the *"off-line"* phase by building the dictionary. To do so, the attacker experiments and validates an adequate fault model. Then he lists all the possible values for a modified public modulus. Next, for each dictionary entry v_j, he identifies the pairs $(m_i, \hat{S}_i)$ satisfying $v_j = \hat{N}_i$. Each pair that matches a value in the dictionary, a so-called *"touch"*, brings some information about d as shown in [BCMCC06,Cla07].

In terms of performance the use of a dictionary is advantageous, because 1100 faults and 28 *"touches"* are necessary to retrieve 1024 key bits. On the other hand, it requires a relevant fault model.

A third attack proposed in [BCMCC06] and revisited in [Cla07] explains how to optimally exploit fault injections. Authors claims that, in good conditions, this allows to reduce the number of fault injections from 1100 to a dozen.

3.5 Summary

The RSA standard algorithm is not immune to fault attacks. The previously presented attacks show that the protection of the public modulus during the decryption or signature processing has to be considered. Now we will present a brand new attack on the public modulus that have some advantages over the previous methods because of the use of a realistic fault model and greater performance.

4 Principle of our Attack

4.1 Fault model

Our attack is based on modifying the public modulus during the computation of the exponentiation corresponding to the signature scheme. The injected fault affects a byte of the modulus by modifying it in a random way, namely:

$$\hat{N} = N \oplus \varepsilon \tag{5}$$

where $\varepsilon = R_8 \cdot 2^{8i}, i \in [\![0; \frac{n}{8} - 1]\!]$ and R_8 is a non-zero random byte value. These two values are supposed unknown by the attacker because they depend on the fault injection itself. The fault is supposed to be transient, and the modified value $\hat{N}$ is used until the end of the exponentiation. The consistency of this model was already checked in the smart card context, leading to successful applications [BECN$^+$04,Gir05a,BO06].

To make our description easier, we assume that the *"Right-To-Left"* algorithm is used for the exponentiation and the attack will be presented in that specific context. Moreover, whereas the transient fault can first occur during the computation of a square or a multiplication, we will focus on the effect of the perturbation on the square. Perturbating the multiplication will be treated in Appendix A.

4.2 Faulty computation

Let $d = \sum_{i=0}^{n-1} 2^i \cdot d_i$ be the binary representation of d. Then an RSA signature can be written as:

$$S \equiv \dot{m}^{\sum_{i=0}^{n-1} 2^i \cdot d_i} \bmod N \tag{6}$$

If a fault occurs j steps before the end of the exponentiation, then this step will begin with a faulty square, whatever the value of d_{n-j} may be:

$$\hat{B} \equiv \left(\dot{m}^{2^{(n-j-1)}} \bmod N\right)^2 \bmod \hat{N} \tag{7}$$

Then the algorithm continues its execution by computing faulty operations. Denoting by $A \equiv \dot{m}^{\sum_{i=0}^{(n-j-1)} 2^i \cdot d_i} \bmod N$ as the correct beginning of the computation, we finally obtain:

$$\begin{aligned}
\hat{S} &\equiv ((A \cdot \hat{B})...)\hat{B}^{2^{j-1}} \bmod \hat{N} && (8)\\
&\equiv A \cdot \hat{B}^{\sum_{i=(n-j)}^{n-1} 2^{[i-(n-j)]} \cdot d_i} \bmod \hat{N} && (9)\\
&\equiv [(\dot{m}^{\sum_{i=0}^{(n-j-1)} 2^i \cdot d_i} \bmod N) && (10)\\
&\quad \cdot (\dot{m}^{2^{(n-j-1)}} \bmod N)^{\sum_{i=(n-j)}^{n-1} 2^{[i-(n-j)+1]} \cdot d_i}] \bmod \hat{N}
\end{aligned}$$

As a consequence, the fault injection splits the computation into a correct part and a faulty one. For a given faulty signature $\hat{S}$, the value of j is supposed to be known by the attacker. This assumption comes from the fact that an attacker can trigger its fault injection using a Simple Power Analysis. Hence, he can know which step of the computation was first infected by the fault and – as a consequence – the number of bits of d that are handled with the wrong modulus.

4.3 Cryptanalysis

The attack consists in recovering a part of the private exponent using the effects of the fault. It is a pure differential analysis because it requires the knowledge of both the correct signature S and the faulty one $\hat{S}$. Indeed, the difference between these two computations resides in the end of the exponentiation (which is faulty in the case of $\hat{S}$). Therefore if the attacker chooses a candidate value for the faulty modulus $\hat{N}'$, and another candidate for the first part of d: $d'_{(1)} = \sum_{i=n-j}^{n-1} 2^i \cdot {d_i}'$, he can then compute:

$$S'_{(d'_{(1)},\hat{N}')} \equiv [(S \cdot \dot{m}^{-d'_{(1)}}) \bmod N \cdot (\dot{m}^{2^{(n-j-1)}} \bmod N)^{2^{[1-(n-j)]} \cdot d'_{(1)}}] \bmod \hat{N} \quad (11)$$

The idea of the attack consists in simulating a faulty computation from a correct one. The first multiplication in $\mathbb{Z}/N\mathbb{Z}$ is done to go backwards to the perturbated step of the computation, whereas the second multiplication simulates the effects of the induced fault. Then he checks whether the following equation is satisfied or not:

$$S'_{(d'_{(1)},\hat{N}')} \equiv \hat{S} \bmod \hat{N} \quad (12)$$

If it is the case, this means that the chosen candidates are the searched values with high probability. If no solution is found among the candidate pairs, this means that the attack occurs during a multiplication and the attacker has to perform the cryptanalysis described in Appendix A.

The attacker can optimize the search of candidates for $\hat{N}'$ by noticing that the faulty modulus, has to be greater than $\hat{S}$. Indeed, $\hat{S}$ is a result of a modular reduction by the faulty modulus. This simple property can dramatically reduce the search space for a suitable $\hat{N}$ candidate.

The subsequent secret bits will be found by repeating this attack using the knowledge of the (already found) most significant bits of d and a signature faulted earlier in the process. As a consequence, the attacker has to gather a set of faulty signatures $\hat{S_k}$ obtained by injecting faults at different steps j_k before the end of the exponentiation. Then, the collected information $(\hat{S_k}, j_k)_{1 \leq k \leq n/l}$ are sorted in descending fault location.

The number of bits recovered each time corresponds to the window length denoted by l. Hence, the k-th l-bit part of d recovered is $d_k = \sum_{i=n-j_k}^{n-j_{(k-1)}-1} 2^i \cdot d_i$. For the sake of clarity, we assume that $j_0 = 0$ and $\forall k \in [\![0; \frac{n}{l}]\!],\ j_{k+1} - j_k = l$. But, this assumption can be easily extended to a more general case where faults are not injected regularly: $\forall k \in [\![0; \frac{n}{l}]\!],\ j_{k+1} - j_k < l_{max}$.

4.4 Attack algorithm

In this section, we detail the implementation of our new Differential Fault Analysis, described above. It generalizes the analysis to recover the whole private exponent by taking advantages of faults injected during squaring operations of the *"Right-To-Left"* algorithm. The following attack algorithm has been successfully implemented on PC using the GMP Library. We assume that, in input, the set of pairs (faulty signature, fault location) is sorted in descending fault location.

Algorithm 2: DFA against RSA in Standard mode

```
INPUT: $N$, $\dot{m}$, the correct signature $S$, the set of pairs $(\hat{S}_k, j_k)_{1 \leq k \leq n/l}$
OUTPUT: the private exponent $d$
1: //Initialisation
2: $d := 0$;
3: for $k$ from 1 upto $\lfloor n/l \rfloor$
4:    //We want to recover the next $l$-bit window $d_k$ of $d$
5:    for $d'_k$ from 0 upto $(2^l - 1)$
6:       $d' := [d'_k << (n - j_k)] + d$;
7:       //We search a suitable value for $\hat{N}$
8:       for $R_8$ from 1 upto $(2^8 - 1)$
9:          for $pos$ from 0 upto $(\frac{n}{8} - 1)$
10:            $\hat{N}' := N \oplus (R_8 << 8.pos)$;
11:            $S'_{(d',\hat{N}')} := [(S \cdot \dot{m}^{-d'}) \bmod N$
12:                       $\cdot(\dot{m}^{2^{(n-j_k-1)}} \bmod N)^{2^{[1-(n-j_k)]} \cdot d'}] \bmod \hat{N}$;
13:            //We test if the rebuilt value equals the faulty one
14:            if $(S'_{(d',\hat{N}')} == \hat{S} \bmod \hat{N})$
15:               //If the condition is satisfied, the current value of $d'$ suits
16:               $d := d'$;
17:               //So, we can search the next $l$-bit of $d$
18:               goto line_3;
19:            endif
20:          endfor
21:       endfor
22:    endfor
23: endfor
24: //Don't forget the purpose of our attack ...
25: return $d$;
```

From our presented algorithm, one can notice that correct and faulty signatures are obtained from the same plaintext m. But, the attacker can recover parts of d from different plaintexts and their associated correct and faulty signatures. To perform this he has to replace the algorithm's input by the quadruplets $(\dot{m}_k, S_k, \hat{S}_k, j_k)_{1 \leq k \leq n/l}$.

4.5 Complexity

Computational complexity. To perform our attack, we need to recover both the induced fault and the part of d affected by the perturbation. For each possible candidate pair, a modular exponentiation is performed. Therefore, according to the previously presented algorithm, the complexity C_{attack} of our attack is :

$$C_{attack} \sim \mathcal{O}(\frac{n^2 \cdot 2^l \cdot (2^8 - 1)}{8.l}) \text{ exponentiations} \tag{13}$$

Observing the algorithm, one can notice that the computation of candidates for the faulty modulus can be replaced by a precomputed dictionary of candidates $\hat{N}'$. But, such a time optimisation has to be done according to the chosen fault model.

As a comparison, the attack presented by Bellcore researchers against a standard RSA [BDL97,BDL01] requires $\mathcal{O}(n^3 \cdot 2^l \cdot \log^2(n)/l^2)$ full modular exponentiations (*i.e.* mod N), which is more complex. Concerning the attack of Brier *et al.* [BCMCC06], it needs to resolve $\mathcal{O}(n)$ discrete logarithm problems of reasonable sizes (*i.e.* less than 2^{30} bits). If the Shank's Baby-Step Giant-Step algorithm ($\sim \mathcal{O}(\sqrt{N} \cdot \log(N))$ is used, the associated complexity is in $\mathcal{O}(n \cdot 2^{30} \cdot 30 \cdot \log(2))$. If the chosen window length l is small enough (*i.e.* $l \leq 20$ bits for a 1024-bit RSA) this computational complexity is bigger than our attack's one.

Number of required faults. The principle of our algorithm is based on recovering the secret exponent by using windows of bits. Each faulty signature is used to recover a different window of d. Therefore, if l is the length of the window, recovering the whole secret exponent requires:

$$\text{Number of faults} \sim \mathcal{O}(n/l) \tag{14}$$

For Bellcore's attack against a plain RSA [BDL97,BDL01], the number of required faults is $\mathcal{O}((n/l) \cdot \log(2n))$ and for Brier *et al.* [BCMCC06] it is $\mathcal{O}(n)$.

4.6 Performance

The performance of our proposed attack are evaluated according to our detailed cryptanalysis (see Sect. 4.3). So, to determine the real $(d_{(1)}, \hat{N})$ pair among all possible candidates, the attacker tests if (12) is satisfied by its rebuilt value $S'_{(d'_{(1)}, \hat{N}')}$ (see (11)). However this equation is checked in $\mathbb{Z}/\hat{N}\mathbb{Z}$, so that some wrong candidates for the searched values $d_{(1)}$ and $\hat{N}$ could be accepted by mistake. This is the problem of false-acceptance. We thus have to evaluate the probability for a given (accepted) pair to be a false one. This phenomenon can be modeled as the probability to pass the test knowing that the values are incorrect:

$$\Pr[\text{Equation (12) is satisfied} \mid (d'_{(1)}, \hat{N}') \neq (d_{(1)}, \hat{N})] \tag{15}$$

$$\iff \Pr[\text{Equation (12) is satisfied} \mid (d'_{(1)} \neq d_{(1)} \text{ or } \hat{N}' \neq \hat{N})] \tag{16}$$

Since, this probability is quite difficult to evaluate, we propose in next section a method to maximize it. First, we use the well-known property of conditional probability. If A and B are two dependent events, then, the probability of the event A to occur knowing that B has occurred is:

$$\Pr[A|B] = \frac{\Pr[A \cup B]}{\Pr[B]} \tag{17}$$

This property can be applied to evaluate our probability of false-acceptance by substituting:

- A by the event: "Equation (12) is satisfied";
- B by the event: "$d'_{(1)}$ or $\hat{N}'$ is a false candidate value respectively for $d_{(1)}$ and $\hat{N}$".

Our probability will be given by computing $\Pr[A \cup B]$ and $\Pr[B]$. For the sake of clarity, both computations are detailed in Appendix B. The obtained results are summarized below:

- $\Pr[A \cup B]$: This represents the probability that (12) is satisfied if at least one candidate is not equal to its expected value. Hence:

$$0 < \Pr[A \cup B] < min\left(\frac{\hat{N}-1}{\hat{N}}, \frac{2^l \cdot n \cdot (2^8-1)-1}{8 \cdot \hat{N}}\right) \tag{18}$$

- $\Pr[B]$: Applying the B's above definition, this is the probability that at least one candidate is not equal to its expected value.

$$\Pr[(d'_{(1)}, \hat{N}') \neq (d_{1)}, \hat{N})] = \frac{n \cdot (2^8-1) \cdot 2^l - 8}{n \cdot (2^8-1) \cdot 2^l} \tag{19}$$

The false-acceptance probability. Using the two partial results, established in Appendix B, and the property of conditional probabilities, the false acceptance probability can be approximated by:

$$0 < \Pr[A|B] < min\left(\frac{n\cdot(\hat{N}-1)\cdot(2^8-1)\cdot 2^l}{n\cdot\hat{N}\cdot(2^8-1)\cdot 2^l - 8}, \frac{(n\cdot(2^8-1)\cdot 2^l-1)\cdot n\cdot(2^8-1)\cdot 2^l}{\hat{N}\cdot(n\cdot(2^8-1)\cdot 2^l - 8))}\right) \tag{20}$$

Even though the false-acceptance probability is bounded by a value close to 1 when $n < 16$ bits, it is interesting to notice that this probability decreases with $\hat{N}$ (and so exponentially with n). As a consequence, the false-acceptance probability rapidly becomes negligible. These theoretical results have been confirmed by our GMP implementation of the attack.

5 Extension of the attack model

5.1 Extension of our fault model

The fault model we have chosen to present our attack principle can be extended to another transient fault model. Such a fault can be induced by the perturbation of a read-access to the public modulus N before computing a square or a multiplication. The perturbation has to influence only the current operation. Indeed, subsequent accesses to N must remain *error-free* [Wag04]. As previously described, the fault still modifies a byte of N by adding a random byte value. With this new assumption, the attacker has to face different cases, depending on the value of d and on the targeted operation. These cases are described below for a fault injected j steps before the end of the exponentiation:

1. $d_{n-j} = 0$ or 1 and the square is perturbated. Whatever the value of d_{n-j} may be, A keeps the same expression: $A \equiv \dot{m}^{\sum_{i=0}^{(n-j-1)} 2^i \cdot d_i} \bmod N$. Moreover, the fault injection modifies the value of B such that the square is computed with a faulty modular reduction $\hat{B} \equiv (\dot{m}^{2^{(n-j-1)}} \bmod N)^2 \bmod \hat{N}$. This faulty computation then spreads in the exponentiation:

$$\begin{aligned}\hat{S} &\equiv ((A \cdot \hat{B})...) \cdot \hat{B}^{2^{j-1}} \bmod N && (21)\\ &\equiv A \cdot \hat{B}^{\sum_{i=(n-j)}^{n-1} 2^{[i-(n-j)]} \cdot d_i} \bmod N && (22)\\ &\equiv (\dot{m}^{\sum_{i=0}^{(n-j-1)} 2^i \cdot d_i} \bmod N) && (23)\\ &\quad \cdot [(\dot{m}^{2^{(n-j-1)}} \bmod N)^2 \bmod \hat{N}]^{\sum_{i=(n-j)}^{n-1} 2^{[i-(n-j)]} \cdot d_i} \bmod N\end{aligned}$$

   This case differs from the previously described one by the modular reduction by $\hat{N}$ applied to the second part of the expression. Moreover, the main product is done here in the finite field $\mathbb{Z}/N\mathbb{Z}$ instead of $\mathbb{Z}/\hat{N}\mathbb{Z}$. Hence, under this transient fault model, the attacker has to apply small changes on the cryptanalysis described in Sect. 4.3 to recover l bits of d.
2. $d_{n-j} = 1$ and the multiplication is perturbated. This second case deals with a perturbation of the multiplication performed j steps before the end of the exponentiation. If $\hat{A}$ stands for the faulty result, then:

$$\begin{aligned}\hat{A} &\equiv [(\dot{m}^{\sum_{i=0}^{[(n-j-2)]} 2^i \cdot d_i} \bmod N) \cdot (\dot{m}^{2^{(n-j-1)}} \bmod N)] \bmod \hat{N} && (24)\\ &\equiv (\dot{m}^{\sum_{i=0}^{(n-j-1)} 2^i \cdot d_i} \bmod N) \bmod \hat{N} && (25)\end{aligned}$$

   No more error occurs during the end of the computation. As a consequence, $B \equiv \dot{m}^{2^{n-j}} \bmod N$ and the faulty signature $\hat{S}$ can be explained as:

$$\begin{aligned}\hat{S} &\equiv ((\hat{A} \cdot B)...)B^{2^{j-1}} \bmod N && (26)\\ &\equiv \hat{A} \cdot B^{\sum_{i=(n-j)}^{n-1} 2^{[i-(n-j)]} \cdot d_i} \bmod N && (27)\\ &\equiv [(\dot{m}^{\sum_{i=0}^{(n-j-1)} 2^i \cdot d_i} \bmod N) \bmod \hat{N} && (28)\\ &\quad .(\dot{m}^{\sum_{i=(n-j)}^{n-1} 2^i \cdot d_i} \bmod N)] \bmod N\end{aligned}$$

This expression can also be cryptanalyzed as described in Sect. 4.3 to obtain l bits of d by noticing that a modular reduction by $\hat{N}$ is applied to A whereas B is not infected by the fault. Moreover the main multiplication is, in this case, computed in $\mathbb{Z}/N\mathbb{Z}$.
Finally, one can notice that the case "$d_j = 0$ and the multiplication is perturbated" is missing. In fact, this case can not occur if we consider the previously presented *"Right-To-Left"* algorithm.

The previous analysis shows that our new attack is not limited to a unique transient fault model. Accordingly, this increases the practicability of the attack on cryptographic devices that implement the *"Right-To-Left"* algorithm.

6 Conclusion

This paper introduces a new fault attack against the *"Right-To-Left"* implementation of RSA. We detail a new way of exploiting faulty RSA public elements (*i.e.* the public modulus N). We show in our theoretical analysis that our attack is more efficient than previously published ones [Sei05,Mui06,BCMCC06]. Moreover its GMP implementation as well as the use of practicable fault models demonstrate that this new attack represents a real threat for RSA public elements. As a consequence, the protection of RSA public key elements against Differential Fault Analysis is more than ever a hot topic.

References

[BCMCC06] 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.

[BDJ$^+$96] F. Bao, R.H. Deng, A. Jeng, A.D. Narasimhalu, and T. Ngair. Another New Attack to RSA on Tamperproof Devices. 1996.

[BDJ$^+$98] F. Bao, R.H. Deng, A. Jeng, A.D. Narasimhalu, and T. Ngair. Breaking Public Key Cryptosystems on Tamper Resistant Devices in the Presence of Transient Faults. In M. Lomas and B. Christianson, editors, *Security Protocols*, volume 1361 of *Lecture Notes in Computer Science*, pages 115–124. Springer-Verlag, 1998.

[BDL97] D. Boneh, R.A. DeMillo, and R.J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In W. Fumy, editor, *EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer-Verlag, 1997.

[BDL01] D. Boneh, R.A. DeMillo, and R.J. Lipton. On the Importance of Eliminating Errors in Cryptographic Computations. *Journal of Cryptology*, 14(2):101–119, 2001.

[BECN$^+$04] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The Sorcerer's Apprentice Guide to Fault Attacks. Cryptology ePrint Archive, Report 2004/100, 2004.

[BO06] 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.

[BS97] E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In *Crypto'97*, 1997.

[CJRR99] S. Chari, C. Jutla, J.R. Rao, and P. Rohatgi. A Cautionary Note Regarding Evaluation of AES Candidates on Smart-Cards. In *Second Advanced Encryption Standard (AES) Candidate Conference*, 1999.

[Cla07] C. Clavier. *De la sécurité physique des crypto-systèmes embarqués.* PhD thesis, Université de Versailles Saint-Quentin, 2007.

[Gir05a] C. Giraud. DFA on AES. In V. Rijmen, H. Dobbertin, and A. Sowa, editors, *Fourth Conference on the Advanced Encryption Standard (AES4)*, volume 3373 of *Lecture Notes in Computer Science*, pages 27–41. Springer-Verlag, 2005.

[Gir05b] C. Giraud. Fault-Resistant RSA Implementation. In L. Breveglieri and I. Koren, editors, *Fault Diagnosis and Tolerance in Cryptography*, pages 142–151, 2005.

[JQBD97] M. Joye, J.J. Quisquater, F. Bao, and R.H. Deng. RSA-types Signatures in the Presence of Transient Faults. In M. Darnell, editor, *Cryptography and Coding, 6th IMA International Conference*, volume 1355 of *Lecture Notes in Computer Science*, pages 155–160. Springer-Verlag, 1997.

[Mui06] J.A. Muir. Seifert's RSA Fault Attack : Simplified Analysis and Generalizations. Cryptology ePrint Archive, Report 2005/458, 2006.

[Ott04] M. Otto. *Fault Attacks and Countermeasures.* PhD thesis, University of Paderborn, December 2004.

[Sei05] 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.

[Wag04] 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.

[YKLM02] S-M. Yen, D. Kim, S. Lim, and S. Moon. A Countermeasure Against One Physical Cryptanalysis May Benefit Another Attack. In K. Kim, editor, *Information Security and Cryptology (ISISC 2001)*, volume 2288 of *Lecture Notes in Computer Science*, pages 414–427. Springer-Verlag, 2002.

A Fault injection before a multiplication

The principle of our attack was described for a permanent fault injected before a square. But, if $d_{n-j} = 1$, then a multiplication is done and can be the first operation modified. So, in this appendix, we present the effects of such a perturbation and how to take advantage of it to recover the bits of the private exponent d.

A.1 Faulty computation

Our attack is performed against the *"Right-To-Left"* exponentiation algorithm with the fault model previously described (see Sect. 4.1). In this case, the fault first occurs during a multiplication, j steps until the end of the exponentiation, so:

$$\hat{A} \equiv (\dot{m}^{\sum_{i=0}^{n-j-2} 2^i \cdot d_i} \bmod N) \cdot B \bmod \hat{N} \quad (29)$$
$$\equiv [(\dot{m}^{\sum_{i=0}^{n-j-2} 2^i \cdot d_i} \bmod N) \cdot \dot{m}^{2^{(n-j-1)}} \bmod N] \bmod \hat{N} \quad (30)$$

Then, this operation is followed by a square:

$$\hat{B} \equiv (\dot{m}^{2^{(n-j-1)}} \bmod N)^2 \bmod \hat{N} \quad (31)$$

After, the cryptographic device finishes the exponentiation:

$$\hat{S} \equiv ((\hat{A} \cdot \hat{B})...) \cdot \hat{B}^{2^j} \bmod \hat{N} \quad (32)$$
$$\equiv \hat{A} \cdot \hat{B}^{\sum_{i=n-j}^{n-1} 2^{i-(n-j)} \cdot d_i} \bmod \hat{N} \quad (33)$$
$$\equiv [(\dot{m}^{\sum_{i=0}^{n-j-2} 2^i \cdot d_i} \bmod N) \quad (34)$$
$$\cdot (\dot{m}^{2^{(n-j-1)}} \bmod N)^{\sum_{i=n-j-1}^{n-1} 2^{i-(n-j)+1} \cdot d_i}] \bmod \hat{N}$$

where $d_{n-j} = 1$

A.2 Cryptanalysis

As shown in Sect. 4.3, the cryptanalysis consists in guessing possible values for the private exponent's value $d'_{(1)} = \sum_{i=n-j-1}^{n-1} 2^i \cdot d'_i$ and the public modulus one $\hat{N}'$. Then the attacker uses the correct signature to forge a possible faulty one. If this forged signature equals to the real faulty one, this means that the chosen candidates are probably the searched values. As described before, the attacker has to compute:

$$S'_{(d'_{(1)}, \hat{N}')} \equiv [(S \cdot \dot{m}^{-d'_{(1)}}) \bmod N \cdot (\dot{m}^{2^{(n-j-1)}} \bmod N)^{2^{-(n-j)} \cdot d'_{(1)}}] \bmod \hat{N}' \quad (35)$$

And then, he checks if the following equation is satisfied :

$$S'_{(d'_{(1)}, \hat{N}')} \equiv \hat{S} \bmod \hat{N}' \quad (36)$$

In that case, the value $d'_{(1)}$ gives $l - 1$ bits of d (and d_{n-j} is already known).

B Details of performance evaluation

B.1 Evaluation of $\mathbf{Pr}[A \cup B]$

According to A and B's respective definitions (see Sect. 4.6), $\Pr[A \cup B]$ represents the probability that the equation is satisfied if at least one candidate, $d'_{(1)}$ or $\hat{N}'$, is not equal to its expected value. This probability is quite difficult to evaluate since it depends on all the unknown values of (12). However, we can find a maximum and a minimum for this probability. Indeed, this equation is satisfied in the finite field $\mathbb{Z}/\hat{N}\mathbb{Z}$, so that, if the correct value is removed, the probability of this event to occur is at least $(1-1)/\hat{N} = 0$. But $d'_{(1)}$ is a l-bit value and so, can take 2^l possible values. $\hat{N}'$ can take $(2^8-1)\cdot\frac{n}{8}$ values (possible values and position for the error ε). Hence, at most, the equation can be satisfied for $\frac{2^l\cdot n\cdot(2^8-1)-1}{8\cdot\hat{N}}$ different values. As a result, the probability can be upper-bounded:

$$0 < \Pr[A \cup B] < min\left(\frac{\hat{N}-1}{\hat{N}}, \frac{2^l\cdot n\cdot(2^8-1)-1}{8\cdot\hat{N}}\right) \tag{37}$$

B.2 Evaluation of Pr[*B*]

This probability seems easier to evaluate than the last one. Indeed, it is the probability that at least one of the two candidates $d'_{(1)}$ or $\hat{N}'$, is not equal to its expected value. But one can notice that this event can be divided into two independent events. Hence, we have:

$$\Pr[(d'_{(1)}, \hat{N}') \neq (d_{(1)}, \hat{N})] \tag{38}$$

$$= 1 - \Pr[(d'_{(1)} = d_{(1)}) \text{ and } (\hat{N}' = \hat{N})] \tag{39}$$

$$= 1 - \Pr[d'_{(1)} = d_{(1)}]\cdot\Pr[\hat{N}' = \hat{N}] \tag{40}$$

As seen in the previous paragraph, $d'_{(1)}$ and $\hat{N}'$ can take respectively 2^l and $(2^8-1)\cdot\frac{n}{8}$ possible values. In both cases, there is only one correct value to find. As a consequence, we finally obtain:

$$\Pr[(d'_{(1)}, \hat{N}') \neq (d_{(1)}, \hat{N})] = 1 - \frac{8}{n\cdot(2^8-1)\cdot 2^l} \tag{41}$$

$$= \frac{n\cdot(2^8-1)\cdot 2^l - 8}{n\cdot(2^8-1)\cdot 2^l} \tag{42}$$