Fault Attacks on RSA Public Keys: *Left-To-Right* Implementations are also Vulnerable

Alexandre Berzati[1,3], Cécile Canovas[1], Jean-Guillaume Dumas[2], and Louis Goubin[3]

[1] CEA-LETI/MINATEC, 17 rue des Martyrs, 38054 Grenoble Cedex 9, France
{alexandre.berzati,cecile.canovas}@cea.fr
[2] Université de Grenoble, Laboratoire Jean Kuntzmann, umr CNRS 5224, BP 53X, 51 rue des mathématiques, 38041 Grenoble, France
Jean-Guillaume.Dumas@imag.fr
[3] Versailles Saint-Quentin University, 45 Avenue des Etats-Unis, 78035 Versailles Cedex, France
Louis.Goubin@prism.uvsq.fr

Abstract. After attacking the RSA by injecting fault and corresponding countermeasures, works appear now about the need for protecting RSA public elements against fault attacks. We provide here an extension of a recent attack [BCG08] based on the public modulus corruption. The difficulty to decompose the *"Left-To-Right"* exponentiation into partial multiplications is overcome by modifying the public modulus to a number with known factorization. This fault model is justified here by a complete study of faulty prime numbers with a fixed size. The good success rate of this attack combined with its practicability raises the question of using faults for changing algebraic properties of finite field based cryptosystems.

Keywords: RSA, fault attacks, *"Left-To-Right"* exponentiation, number theory.

1 Introduction

Injecting faults during the execution of cryptographic algorithms is a powerful way to recover secret information. Such a principle was first published by Bellcore researchers [BDL97,BDL01] against multiple public key cryptosystems. Indeed, these papers provide successful applications including RSA in both standard and CRT modes. This work was completed, and named Differential Fault Analysis (DFA), by E. Biham and A. Shamir with applications to secret key cryptosystems [BS97]. The growing popularity of this kind of attack, in the last decade, was based on the ease for modifying the behavior of an execution [BECN$^+$04] and the difficulty for elaborating efficient countermeasures [BOS03,Wag04,Gir05b].

Many applications against the RSA cryptosystem, based on fault injection, have been published. The first ones dealt with the perturbation of the private key or temporary values during the computation [BDL97,BDJ+98,BDL01]. The perturbation of public elements was considered as a real threat when J-P. Seifert published an attack on the RSA signature check mechanism [Sei05,Mui06]. This paper first mentions the possibility of modifying the public modulus N such that the faulty one is prime or easy to factor. Then, E. Brier *et al.* extended this work to the full recovery of the private exponent d for various RSA implementations [BCMCC06]. Both works are based on the assumption that the fault occurs before performing the RSA modular exponentiation. A. Berzati *et al.* first address the issue of modifying the modulus during the exponentiation [BCG08].Still this work was limited to an application against *"Right-To-Left"* type exponentiation algorithms.

In this paper we aim to generalize the previous attack to *"Left-To-Right"* type exponentiations. Under the fault assumption that the modulus can become a number with a known factorization, we prove that it is possible to recover the whole private exponent. We provide a detailed study of this fault model, based on number theory, to show its consistency and its practicability for various kinds of perturbation. Finally, we propose an algorithm to recover the whole private exponent that is efficient either in terms of fault number or in computational time.

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 following operations.

RSA Decryption: Decrypting a ciphertext C boils down to compute $\tilde{m} \equiv C^d \bmod N \equiv C^{\sum_{i=0}^{n-1} 2^i \cdot d_i} \bmod N$ 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 $S \equiv \dot{m}^d \bmod N$ 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

Binary exponentiation algorithms are often used for computing the RSA modular exponentiation $\dot{m}^d \bmod N$ where the exponent d is expressed in a binary form as $d = \sum_{i=0}^{n-1} 2^i \cdot d_i$. Their polynomial complexity with respect to the input length make them very interesting to perform modular exponentiation.

The Algorithm 1 describes a way to compute modular exponentiations by scanning bits of d from least significant bits (LSB) to most significant bits

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 :         $A := (A \cdot B) \bmod N$;
6 :     end if
7 :     $B := B^2 \bmod N$;
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 :     $A := A^2 \bmod N$;
4 :     if $(d_i == 1)$
5 :         $A := (A \cdot m) \bmod N$;
6 :     end if
7 : end for
8 : return $A$;
```

(MSB). That is why it is usually referred to as the *"Right-To-Left"* modular exponentiation algorithm. This is that specific implementation that is attacked in [BCG08] by corrupting the public modulus of RSA.

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.

3 Modification of the modulus and extension attempt

3.1 Previous work

J-P. Seifert first addressed the issue of corrupting RSA public key elements [Sei05,Mui06]. This fault attack aims to make a signature verification mechanism accept false signatures by modifying the value of the public modulus N. No information about the private exponent d is revealed with this fault attack. Its efficiency is linked to the attacker's ability to reproduce the fault model chosen for the modification of the modulus.

Seifert's work inspired the authors of [BCMCC06] who first used the public modulus perturbation to recover the whole private key d. The attacker has to perform a perturbation campaign to gather a large enough number of (message, faulty signature) pairs. As in Seifert's attack, the fault on the modulus is induced before executing the exponentiation. Three methods based on the use of Chinese Remainder Theorem and the resolution of quite small discrete logarithms are proposed in [BCMCC06] and [Cla07] to recover the private exponent from the set of gathered pairs.

A new fault attack against *"Right-To-Left"* exponentiation has been presented lately [BCG08]. This work completes the state-of-the-art by allowing the attacker to use other fault models for recovering the private exponent. The details of this work are presented below.

3.2 Public key perturbation during RSA execution: case of the *"Right-To-Left"* algorithm

Fault model. In J.P Seifert and E. Brier *et al.*'s proposals [Sei05,BCMCC06] the fault is provoked before the exponentiation so that the whole execution is executed with the faulty modulus, $\hat{N}$.
The attack presented by A. Berzati *et al.* [BCG08] extends the fault model by allowing the attacker to inject the fault during the execution of the *"Right-To-Left"* exponentiation. The modification of N is supposed to be a transient random byte fault modification. It means that only one byte of N is set to a random value. The value of the faulty modulus $\hat{N}$ is not known by the attacker. However, the time location of the fault is a parameter known by the attacker and used to perform the cryptanalysis. This fault model has been chosen for its simplicity and practicability in smart card context [Gir05a,BO06]. Furthermore, it can be easily adapted to 16-bit or 32-bit architectures.

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

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

We consider that a fault has occurred j steps before the end of the exponentiation, during the computation of a square. According to the fault model described, all subsequent operations are performed with a faulty modulus $\hat{N}$. We denote by $A \equiv \dot{m}^{\sum_{i=0}^{(n-j-1)} 2^i \cdot d_i} \mod N$ the internal register value and by $\hat{B}$ the result of the faulty square:

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

Hence, the faulty signature $\hat{S}$ can be written as:

$$\begin{aligned} \hat{S} &\equiv A \cdot \hat{B}^{\sum_{i=(n-j)}^{n-1} 2^{[i-(n-j)]} \cdot d_i} \mod \hat{N} & (3) \\ &\equiv [(\dot{m}^{\sum_{i=0}^{(n-j-1)} 2^i \cdot d_i} \mod N) & (4) \\ &\quad \cdot (\dot{m}^{2^{(n-j-1)}} \mod N)^{\sum_{i=(n-j)}^{n-1} 2^{[i-(n-j)+1]} \cdot d_i}] \mod \hat{N} \end{aligned}$$

From the previous expression of $\hat{S}$, one can first notice that the fault injection splits the computation into a correct (computed with N) and a faulty part (computed with $\hat{N}$). A part of d is used during the faulty computation. This is exactly the secret exponent part that will be recovered in the following analysis.

Attack principle. From both correct signature S and faulty one $\hat{S}$ (obtained from the same message m), the attacker can recover the isolated part of the private key $d_{(1)} = \sum_{i=n-j}^{n-1} 2^i \cdot d_i$. Indeed, he tries to find simultaneously candidate values for the faulty modulus $\hat{N}'$ (according to the random byte fault assumption) and for the part of the exponent $d'_{(1)}$ that satisfies:

$$\hat{S} \equiv \left(S \cdot \dot{m}^{-d'_{(1)}} \bmod N\right) \cdot \left(\dot{m}^{2^{(n-j-1)}} \bmod N\right)^{2^{[1-(n-j)]} \cdot d'_{(1)}} \bmod \hat{N}' \qquad (5)$$

According to [BCG08], the pair $(d'_{(1)}, \hat{N}')$ that satisfies (5) is the right one with a probability very close to 1. Then, 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. In terms of fault number, the whole private key recovery requires an average of (n/l) faulty signatures, where l is the average number of bits recovered each time. As a consequence, this few number of required faults makes the attack both efficient and practicable.

3.3 Application to the *"Left-To-Right"* modular exponentiation

In this section, we try to apply the previously explained fault attack to the *"Left-To-Right"* implementation of RSA. Under the same fault model, we wanted to know what does prevent an attacker from reproducing the attack against the dual implementation.

We denote by A the internal register value just before the modification of the modulus N:

$$A \equiv \dot{m}^{\sum_{i=j}^{n-1} 2^{i-j} \cdot d_i} \bmod N \qquad (6)$$

Hence, knowing that the first perturbed operation is a square, the faulty signature $\hat{S}$ can be written as:

$$\begin{aligned} \hat{S} &\equiv \left(\left(\left(A^2 \cdot \dot{m}^{d_{j-1}}\right)^2 \cdot \dot{m}^{d_{j-2}}\right)^2 \ldots\right)^2 \cdot \dot{m}^{d_0} \bmod \hat{N} \qquad (7) \\ &\equiv A^{2^j} \cdot \dot{m}^{\sum_{i=0}^{j-1} 2^i \cdot d_i} \bmod \hat{N} \end{aligned}$$

By observing (7), one can notice that the perturbation has two consequences on the faulty signature $\hat{S}$. First, it splits the computation into a correct part ($i.e$: the internal register value A) and a faulty one, like for the perturbation of the*"Right-To-Left"* exponentiation [BCG08]. The other one is the addition of j cascaded squares of the local variable A, computed modulo $\hat{N}$. This added operation defeats the previous attack on the *"Right-To-Left"* exponentiation [BCG08] because of the difficulty to compute square roots in RSA rings.

Our idea for generalizing the previous attack to *"Left-To-Right"* exponentiation is to take advantage of the modulus modification to change the algebraic properties of the RSA ring. In other words, if $\hat{N}$ is a prime number, then it is possible to compute square roots in polynomial time. Moreover, it is actually sufficient that $\hat{N}$ is B-smooth with B small enough to enable an easy factorization of $\hat{N}$, then the Chinese Remainder Theorem enables also to compute square roots in polynomial time. We show next anyway that the number of primes $\hat{N}$ is sufficient to provide a realistic fault model.

4 Fault model

According to the previous section, the square root problem can be overcome by perturbing the modulus N such that $\hat{N}$ is prime. In this section we will study the consistency and the practicability of such a fault model. Even though this model has already been adopted in Seifert's attack [Mui06,Sei05], we propose next further experimental evidences of the practicability of this model.

4.1 Theoretical estimations

Let us first estimate the number of primes with a fixed number of bits. From [Dus98, Theorem 1.10], we have the following bounds for the number of primes π below a certain integer x:

$$\pi(x) \geq \frac{x}{\ln(x)}\left(1+\frac{1}{\ln(x)}+\frac{1.8}{\ln^2(x)}\right), \text{ for } x \geq 32299. \tag{8}$$
$$\pi(x) \leq \frac{x}{\ln(x)}\left(1+\frac{1}{\ln(x)}+\frac{2.51}{\ln^2(x)}\right), \text{ for } x \geq 355991.$$

Then, for numbers of exactly t bits such that $t \geq 19$ bits, the number of primes is $\pi_t = \pi(2^t) - \pi(2^{t-1})$. By using the previous bounds (8), the probability that a t-bit number is prime, $pr_t = \frac{\pi_t}{2^{t-1}}$, satisfies:

$$pr_t > \mathit{Inf}(t) = \frac{0.480t^5 - 1.229t^4 + 0.0265t^3 - 7.602t^2 + 9.414t - 3.600}{t^3(t-1)^3\ln^3(2)} \tag{9}$$
$$pr_t < \mathit{Sup}(t) = \frac{0.480t^5 - 1.229t^4 + 2.157t^3 - 11.862t^2 + 13.674t - 5.02}{t^3(t-1)^3\ln^3(2)}$$

For instance, if $t = 1024$ bits:

$$\mathit{Inf}(1024) = \frac{1}{709.477} \text{ and } \mathit{Sup}(1024) = \frac{1}{709.474}$$

Therefore around one 1024-bit number out of 709 is prime; and among the 2048-bit numbers, more than one out of 1419 is prime.

Consider now a set of k randomly selected numbers of exactly t bits and let PN be the random variable expressing the expected number of primes in this set. This variable follows a binomial law $\mathcal{B}(k, pr_t)$. Then we can give the following confidence interval of primes (with a and b integer bounds):

$$\Pr[a \leq PN \leq b] = \sum_{i=a}^{b} \binom{k}{i} pr_t^i (1-pr_t)^{k-i} \tag{10}$$

For example, we construct the following set $\mathcal{N}$ according to a random byte fault model. In other words, if $\oplus$ is the bit by bit exclusive OR, then[1]:

$$\mathcal{N} = \{N \oplus R_8 \cdot 2^{8i},\ R_8 = 0 \,..\, 255,\ i = 0 \,..\, (\frac{n}{8} - 1)\}$$

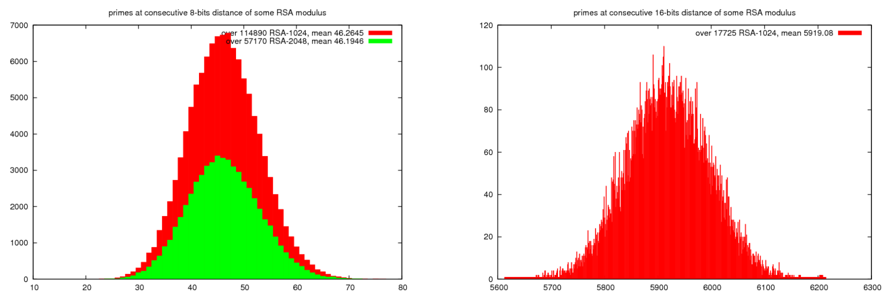
Then the cardinality of $\mathcal{N}$ is

$$|\mathcal{N}| = 256 \cdot \frac{n}{8} = 32 \cdot n$$

Would the set $\mathcal{N}$ be composed of randomly selected values, then the proportion of primes in $\mathcal{N}$ would follow (9). Hence, we can set $k = |\mathcal{N}|$ and compute the corresponding average and bounds with an approximation of pr_t. For $n = 1024$, according to (9), we can estimate pr_{1024} and thus the average number of faulty primes is $32 \cdot 1024/709.47 \approx 46.186$. Equation (10) combined with the estimation of pr_{1024} shows also that the number of primes in a set is comprised between $[18, 80]$ in 99.999% of the cases. For $n = 2048$, the average number of primes is 46.176 and comprised between 18 and 80 in 99.999% of the cases. Obviously $\mathcal{N}$ is not a set of randomly chosen elements; howbeit, empirical evidence shows that such sets behave quite like random sets of elements, as shown below.

4.2 Experimental results

We have computed such sets for randomly selected RSA moduli and counted the number of primes in those sets. The repartition seems to follow a binomial rule (as expected) and we have the following experimental data to support our belief (see Figure 1).



(a) Primes at consecutive 8-bit distance of some RSA modulus

(b) Primes at consecutive 16-bit distance of some RSA modulus

Fig. 1. Experimental distribution of primes among faulty RSA moduli

[1] For the sake of clarity we assume that a byte fault can take 2^8 values. In fact, it can take only $2^8 - 1$. Indeed, the error can not be null otherwise the value of N is unchanged and the fault can not be exploited.

As shown in Table 1 it was anyway *never* the case that no prime was found in a set $\mathcal{N}$ (more than that we always found more than 18 primes in such a set). This experimental lower-bound equals to the one obtained by considering a random set. The same observation can be done for the upper-bound. Hence, our obtained results confirm our theoretical analysis.

Table 1. Experimental counts of primes in $\mathcal{N}$.

Architecture	n bits	$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{N}\rvert \cdot pr_n$	# of exp.	# of primes		
					Min.	Avg.	Max.
8-bit	1024	2^{15}	46.186	114890	18	46.26	79
8-bit	2048	2^{16}	46.176	57170	22	46.19	80
16-bit	1024	2^{22}	5911.83	17725	5621	5919.08	6212
32-bit	1024	2^{37}	$\approx 1,94 \cdot 10^8$				

The presented results can be extended to other fault models. The Table 1 presents also theoretical expected results when 16-bit or 32-bit architectures are targeted. For $t = 1024$ with 16-bit architecture the average number of primes is 5911.83 and is between $[5520, 6320]$ in 99.999% of the cases.

4.3 Consequences

This study strengthen J-P. Seifert's assumption [Sei05,Mui06] of considering only prime modification of the modulus. We have showed that our fault model can be considered as a random modification of the public modulus. Then, an average of 709 faults on N will be required to obtain a prime $\hat{N}$ in the case of a 1024-bit RSA.

Additional remark. By carefully studying the experimental results, one can notice that, for a given modulus N, the byte location of the fault influences the number of prime found in the subset. Thus, if the attacker has the ability of setting the byte location of the fault, he can increase his chances to get a prime faulty modulus and therefore, dramatically reduce the number of faulty signatures required to perform the attack.

4.4 The Algorithm of Tonelli and Shanks

The algorithm of Tonelli and Shanks [Coh93] is a probabilistic and quite efficient algorithm used to compute square roots modulo P, where P is a prime number. The principle of the algorithm is based on the isomorphism between the multiplicative group $(\mathbb{Z}/P\mathbb{Z})^*$ and the additive group $\mathbb{Z}/(P-1)\,\mathbb{Z}$. Suppose $P-1$ is written as:

$$P - 1 = 2^e \cdot r, \qquad \text{with } r \text{ odd.} \tag{11}$$

Then, the cyclic group G of order 2^e is a subgroup of $\mathbb{Z}/\left(P-1\right)\mathbb{Z}$. Let z be a generator of G, if a is a quadratic residue modulo N, then:

$$a^{(P-1)/2} \equiv \left(a^r\right)^{2^{e-1}} \equiv 1 \bmod P \tag{12}$$

Noticing that $a^r \bmod P$ is a square in G, then it exists an integer $k \in [\![0 : 2^e - 1]\!]$ such that

$$a^r \cdot z^k = 1 \text{ in } G \tag{13}$$

And so, $a^{r+1} \cdot z^k = a$ in G. Hence, the square root of a, is given by

$$a^{1/2} \equiv a^{(r+1)/2} \cdot z^{k/2} \bmod P \tag{14}$$

Both main operations of this algorithm are:

- Finding the generator z of the subgroup G,
- Computing the exponent k.

The whole complexity of this algorithm is that of finding k, $\mathcal{O}\left(\ln^4 P\right)$ binary operations or $\mathcal{O}\left(\ln P\right)$ exponentiations. The details of the above algorithm are described in [Coh93]. In practice, on a Pentium IV 3.2GHz, the GIVARO[2] implementation of this algorithm takes on average 7/1000 of a second to find a square root for a 1024-bit prime modulus.

4.5 Smooth modulus

As in [Mui06], what we really need for the faulty modulus is only to be easily factorable. Indeed, one can compute square roots modulo non prime modulus as long as the factorization is known. The idea is first to find square roots modulo each prime factors of $\hat{N}$. Then to lift them independently to get square roots modulo each prime power. And finally to combine them using the Chinese Remainder Theorem (see e.g. [Sho05, §13.3.3] for more details). The number of square roots increases but since they are computed on comparatively smaller primes, the overall complexity thus remains $\mathcal{O}\left(\ln^4 \hat{N}\right)$ binary operations. In the following we thus consider only prime faulty moduli.

5 Cryptanalysis

The purpose of our fault attack against the *"Left-To-Right"* exponentiation is similar to the attack against the *"Right-To-Left"* one [BCG08]. The modulus N is transiently modified to a prime value during a squaring, j_k steps before the end of the exponentiation. Then, from a correct/faulty signature pair $(S, \hat{S}_k)$, the attack aims to recover the part of private exponent $d_{(k)} = \sum_{i=0}^{j_k-1} 2^i \cdot d_i$ isolated by the fault. By referring to [BCG08], the following analysis can be easily adapted for faults that first occurs during a multiplication.

[2] GIVARO is an open source C++ library over the GNU Multi-Precision Library. It is available on `http://packages.debian.org/fr/sid/libgivaro-dev`

Dictionary of prime modulus. The first step consists in computing a dictionary of prime faulty modulus candidates ($\hat{N}_i$). The attacker tests all possible values obtained by modifying N according to a chosen fault model. Then, candidate values for $\hat{N}$ are tested using the probabilistic Miller-Rabin algorithm [Rab80]. According to our study (see Sect. 4.1), for a random byte fault assumption, the faulty modulus dictionary will contain 46 entries in average either for a 1024-bit or a 2048-bit RSA. The size of the dictionary depends on the fault model (see Table 1).

Computation of square roots. For each entry $\hat{N}_i$ of the modulus dictionary, the attacker chooses a candidate value for the searched part of the private exponent $d'_{(k)}$. Now he can compute[3]:

$$R_{(d'_{(k)},\hat{N}_i)} \equiv \hat{S}_k \cdot \dot{m}^{-d'_{(k)}} \bmod \hat{N}_i \tag{15}$$

For the right pair $(d_{(k)}, \hat{N})$, $R_{(d_{(k)},\hat{N})}$ is expected to be a multiple quadratic residue (*i.e:* a j_k-th quadratic residue, see Sect. 3.3). As a result, if $R_{(d'_{(k)},\hat{N}_i)}$ is not a quadratic residue, the attacker can directly deduce that the candidate pair $(d'_{(k)}, \hat{N}_i)$ is a wrong one. The quadratic residuosity test can be done in our case because all precomputed candidate values for the faulty modulus are prime numbers. The test is based on Fermat's theorem:

$$\text{If } \left(R_{(d'_{(k)},\hat{N}_i)}\right)^{\left(\hat{N}_i-1\right)/2} \equiv 1 \bmod \hat{N}_i \tag{16}$$

$$\text{then } R_{(d'_{(k)},\hat{N}_i)} \text{is a quadratic residue modulo } \hat{N}_i$$

If the test is satisfied then the attacker can use the Tonelli and Shanks algorithm (see Sect. 4.4) to compute the square roots of $R_{(d'_{(k)},\hat{N}_i)}$. Therefore, to compute the j_k-th square root of $R_{(d'_{(k)},\hat{N}_i)}$, this step is expected to be repeated j_k-times. But, when one of the j_k quadratic residuosity test fails, the current candidate pair is directly $(d'_{(k)}, \hat{N}_i)$ rejected and the square root computation is aborted. The attacker has to choose another candidate pair.

Final modular check. The purpose of the two first steps is to cancel the effects on the faulty signature due to the perturbation. Now, from the j_k-th square root of $R_{(d'_{(k)},\hat{N}_i)}$ the attacker will simulate an error-free end of execution by computing:

$$S' \equiv \left(\left(R_{(d'_{(k)},\hat{N}_i)}\right)^{1/2^{j_k}} \bmod \hat{N}_i\right)^{2^{j_k}} \cdot \dot{m}^{d'_{(k)}} \bmod N \tag{17}$$

[3] This computation is possible only when d'_k is invertible in $\mathbb{Z}/\mathbb{Z}\hat{N}_i$; in our case all the considered N_i are primes and Euclid's algorithm always computes the inverse.

Finally, he checks if the following equation is satisfied:

$$S' \equiv S \bmod N \tag{18}$$

As in the *"Right-To-Left"* attack [BCG08], when this latter condition is satisfied, it means that the candidate pair is very probably the searched one (see Sect. 6.3). Moreover, the knowledge of the already found least significant bits of d is used to reproduce the attack on the subsequent secret bits. As a consequence, the attacker has to collect a set of faulty signatures $\hat{S}_k$ by injecting the fault at different steps j_k before the end of the exponentiation. Moreover, multiple faulty signature $\hat{S}_{k,f}$ have to be gathered for a given step j_k to take into account the probability for having a faulty signature $\hat{S}_k$ computed under a prime $\hat{N}$, that is to say exploitable by the cryptanalysis. This set $(\hat{S}_{k,f}, j_k)_{k,f}$ is sorted in descending fault location. If faults are injected regularly, each sorted pair is used to recover a l-bit part of the exponent such that for the k-th pair $(\hat{S}_{k,f}, j_k)$, the recovered part of d is $d_{(k)} = \sum_{i=0}^{j_k-1} 2^i \cdot d_i = \sum_{i=0}^{k \cdot l-1} 2^i \cdot d_i$. These results can be applied for faults that are not injected regularly ($i.e$: $j_k - j_{k-1} = l_k < l_{max}$). The attack algorithm is given in more details next.

6 Performance

6.1 Fault number

Our fault model is based on the modification of the modulus N such that its corresponding faulty value is prime. In Section 4.1, we have shown that the probability for a t-bit number to be prime, pr_t, can be bounded. Now, let the number of fault to make $\hat{N}$ prime be the random variable F_t. This random variable follows a geometric probability law. Hence the average number of faults to make $\hat{N}$ prime is:

$$\frac{1}{\mathit{Sup}(t)} < \mu\left(F_t\right) = \frac{1}{pr_t} < \frac{1}{\mathit{Inf}(t)} \tag{19}$$

For large values of t ($i.e$: at least 1024 or 2048-bit RSA), we can use the pinching (or sandwich) theorem to approximate this value asymptotically :

$$\mu\left(F_t\right) \sim \frac{t \cdot \ln^3(2)}{0.480} \sim \frac{t}{1.441} \tag{20}$$

From a given faulty signature, the attacker can recover a l-bit part of d. There are at most n/l such parts for an RSA of size n. This shows that the average number of faults required for a whole private key satisfies:

$$\text{Number of faults} = \mathcal{O}\left(\frac{n^2}{1.441 \cdot l}\right) \text{tries} \tag{21}$$

This number can be dramatically reduced if the attacker has the ability to chose the byte location of the fault (see Sect. 4.1) or if the fault model is larger ($i.e$: smooth modulus, different architectures targeted ...).

Algorithm 3: DFA against *"Left-To-Right"* modular exponentiation

INPUT: N, $\dot{m}$, the correct signature S, the size of the dictionary D_{length},
the set of pairs $(\hat{S}_{k,f}, j_k)_{0 \leq k < n/l,\ 1 \leq f \leq \mu(F_n)}$
OUTPUT: the private exponent d

```
1: //Computation of the dictionary of prime faulty modulus candidates
2: Dict = Build_Prime_Dict($N$, $D_{length}$);
3: //Initialization
4: $d := 0$;
5: //All the faulty signatures are tested
6: for $k$ from 0 upto $\lfloor n/l \rfloor$
7:    for $f$ from 1 upto $\mu(F_n)$
8:       for $d_{(k)}$ from 0 upto $2^l - 1$
9:          $d' := d_{(k)} \cdot 2^{j_k} + d$;
10:         for $i$ from 1 upto $D_{length}$
11:            $R := \hat{S}_{k,f} \cdot \dot{m}^{-d'} \bmod Dict[i]$;
12:            //The function computes $j_k$ square roots and returns 0 when a test fails
13:            $R := Test\_And\_Tonelli(R,\ j_k,\ Dict[i])$;
14:            //If a test fails, then we have to test another candidate pair
15:            if $(R == 0)$
16:               break;
17:            else
18:               $S' := R^{2^{j_k}} \cdot \dot{m}^{d'} \bmod N$
19:               //Final check
20:               if $(S' == S \bmod N)$
21:                  //The attack continues for the subsequent $l$-bit part of $d$
22:                  $d := d'$;
23:                  goto line_6;
24:               end if;
25:            end if;
26:         end for;
27:      end for;
28:   end for;
29: end for;
30: return $d$;
```

6.2 Computational complexity

We now give the overall complexity of the attck. The size of the dictionary, D_{length}, is let as an attack parameter since the attacker can fix a limit if the chosen fault model requires more resources than he can get. According to our previous analysis (see Sect. 4.1), $D_{length} = 46$ for a random byte fault assumption.

Theorem 1. *Algorithm 3 is correct and its average complexity for a random byte fault perturbation of the modulus satisfies:*

$$C_{attack} = \mathcal{O}\left(\frac{2^{8+l} \cdot n^3 \cdot (n+l)}{16 \cdot l}\right) \textit{ exponentiations}$$

Proof. Correctness as been shown in section 5. Now for the complexity, the attacker has to test all possible candidate pairs $(d'_{(k)}, \hat{N}_i)$. The number of pairs depends on the size of the dictionary of prime modulus denoted by D_{length} and the window recovery length l:

$$|(d'_{(k)}, \hat{N}_i)| = 2^l \cdot D_{length} \tag{22}$$

For each pair the attacker first computes $R_{(d'_{(k)}, \hat{N}_i)}$ (see (15)) by executing a modular exponentiation of the message and a multiplication.
Then, he performs a series of at most j_k quadratic residuosity tests and, for each success, a square root is computed. By noticing that the probability to fail in the test follows a geometric probability law, the average number of performed tests[4] is $\frac{1}{\Pr[\text{Test fails}]} = 2$. As a consequence, the average complexity of this step is:

$$\begin{aligned} C_{Square\ roots}(k) &= \mathcal{O}\left(2 \cdot C_{Test} + C_{Tonelli\ \&\ Shanks}\right) \\ &= \mathcal{O}\left(j_k \cdot n\right) \text{ exponentiations} \end{aligned} \tag{23}$$

The last step of the attack is the final check (see (17)). It requires to compute j_k modular squares and a modular exponentiation of the message followed by a multiplication. The latter computation is also bounded by $\mathcal{O}(j_k \cdot n)$ exponentiations.

Now in the case of a fixed size dictionary the average number of primes of this dictionary for a byte modification of the modulus is $N_{faults\ per\ blocs} = \frac{2^8 n/8}{D_{length}}$.

Then, the attack has to test all of the gathered faulty signatures in order to recover the whole exponent. Hence, as j_k is bounded by $k \cdot l$, the overall computational complexity is bounded by:

$$\begin{aligned} C_{attack} &= \sum_{k=0}^{n/l} N_{faults\ per\ blocs} \cdot C_{Square\ roots}(k) \cdot 2^l \cdot D_{length} \\ &= \mathcal{O}\left(\frac{2^{8+l} \cdot n^3 \cdot (n+l)}{16 \cdot l}\right) \end{aligned} \tag{24}$$

[4] The test fails when tested value is not a quadratic residue. But all the $\hat{N}_i$ are prime. Let be z_i a generator in $\mathbb{Z}/\hat{N}_i\mathbb{Z}$, all the elements of the group can be expressed as a power of z_i. Hence one element out of 2 is a power of ${z_i}^2$ and a quadratic residue.

The presented attack is thus longer than the *"Right-To-Left"* one [BCG08], the principal reason being the extra number of faulty pairs to analyze in order to get a prime modulus.

6.3 False-acceptance probability

As defined in [BCG08], the false-acceptance probability is the probability for a wrong pair $(d'_{(k)}, \hat{N}_i)$ to satisfy (18). In our case, the computation of the final check is done in $\mathbb{Z}/N\mathbb{Z}$ and requires extra squares. As a consequence the false-acceptance probability given in [BCG08] has to be adapted by replacing the search space for $\hat{N}$ by the dictionary length D_{length}:

$$0 < \Pr[F.A] < min\left(\frac{(N-1)\cdot 2^l \cdot D_{length}}{N\cdot(2^l \cdot D_{length}-1)}, \frac{2^l \cdot D_{length}}{N}\right) \tag{25}$$

Moreover, because of the quadratic residuosity tests (see Sect. 5), false candidates can be rejected before computing the final check. Hence, the final check will not always be done. The probability that a wrong pair pass all the j_k tests is given by:

$$\begin{aligned} &\Pr\left[R_{(d'_{(k)},\hat{N}_i)}\text{is a } j_k\text{-times quadratic residue}\right] \qquad (26)\\ &= \prod_{i=0}^{j_k-1} \Pr\left[\left(R_{(d'_{(k)},\hat{N}_i)}\right)^{1/2^i} \text{ is a quadratic residue}\right]\\ &= \frac{1}{2^{j_k}} \end{aligned}$$

This probability indicates that, for recovering the k-th part of d, only one out of 2^{j_k} wrong pairs will pass all the quadratic residuosity tests. Eventually, the false-acceptance probability can be upper-bounded:

$$\Pr[F.A] < min\left(\frac{1}{2^{j_k}}, \frac{(N-1)\cdot 2^l \cdot D_{length}}{N\cdot(2^l \cdot D_{length}-1)}, \frac{2^l \cdot D_{length}}{N}\right) \tag{27}$$

This expression first shows that because of the last term $\frac{2^l \cdot D_{length}}{N}$, the false-acceptance probability is highly negligible for commonly used RSA length. Furthermore, one can advantageously notice that the final check can be avoided when the number of consecutive quadratic residuosity tests to pass is large enough ($i.e$: $2^{j_k} > D_{length} \cdot 2^l$).

7 Conclusion

In this paper, we generalize the fault attack presented in [BCG08] to *"Left-To-Right"* implementation of RSA by assuming that the faulty modulus can be prime. Although this model has been already used [Sei05], this paper provides a detailed theoretical analysis in fault attack context. Furthermore this analysis proves that such a fault model is not only practicable but extendible to different architectures. This emphases the need for protecting RSA public elements during the execution.

More generally the use of a faulty prime modulus to compute square roots in polynomial time raises the question of using faults for changing algebraic properties of the underlying finite domain. This paper provides an element of answer that may be completed by future fault exploitations.

References

[BCG08] 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. Springer-Verlag, 2008.

[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$^+$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.

[BOS03] J. Blömer, M. Otto, and J-P. Seifert. A New CRT-RSA Algorithm Secure Against Bellcore Attack. In *ACM Conference on Computer and Communication Security (CCS 2003)*, pages 311–320. ACM Press, 2003.

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

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

[Coh93] H. Cohen. *A Course in Computational Algebraic Number Theory.* Springer-Verlag New York Inc., 1993.

[Dus98] P. Dusart. *Autour de la fonction qui compte le nombre de nombres premiers*. PhD thesis, Université de Limoges, 1998.

[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.

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

[Rab80] Michael O. Rabin. Probabilistic algorithm for testing primality. *Journal of Number Thoery 12*, 1:128–138, 1980.

[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.

[Sho05] V. Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University 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.