# A Survey of Differential Fault Analysis against Classical RSA Implementations

Alexandre Berzati, Cécile Canovas-Dumas, and Louis Goubin

## 1 Introduction

Since the advent of side channel attacks, classical cryptanalysis is no longer sufficient to ensure the security of cryptographic algorithms. In practice, the implementation of algorithms on electronic devices is a potential source of leakage that an attacker can use to completely break a system [29, 15, 21]. The injection of faults during the execution of cryptographic algorithms is considered as an intrusive side channel method because secret information may leak from malicious modifications of a device's behavior [13, 11]. In this context, the security of public key cryptosystems [13] and symmetric ciphers in both block [11] and stream modes [23] has been challenged. Recently, some interesting results have been obtained by attacking public key cryptosystems. More precisely, several papers demonstrated that the perturbation of public elements may induce critical flaws in implementations of public key cryptosystems [10, 14, 25].

We propose here a survey of the applications of fault attacks against different RSA implementations, classical or sophisticated. After a presentation of the RSA cryptosystem and its classical implementation, we review chronologically the attacks and some of the proposed countermeasures. Although first attackers focused their efforts to exploit perturbations of secret elements or related operations (see Sect. 3), recent works adressed the security of public elements (see Sect. 4). This new trend is all the more interesting since public elements are usually handled in a less secure way than private ones. Furthermore it may lead to powerful attacks regardless to the kind of induced perturbations.

Alexandre Berzati · Cécile Canovas-Dumas
CEA Leti, Grenoble, France

Louis Goubin
Versailles Saint-Quentin-en-Yvelines University, France

# 2 RSA Implementations

The celebrated *RSA* has been invented in 1977 by Ronald Rivest, Adi Shamir and Leonard Adleman [36]. This method has been the first one instantiating the principle of public key cryptography, introduced earlier, as a concept, by Whitfield Diffie and Martin Hellman [20]. Nowadays, RSA is certainly the most widely used algorithm to ensure the security of communications or transactions.

The RSA security relies on the Integer Factorization Problem (IFP). The best known method for solving an instance of this problem is the Number Field Sieve which complexity is sub-exponential with respect to the size of the RSA field [30]. To the best of our knowledge, the larger RSA modulus ever factored by this method is a 768-bit one [26]. As a consequence, this method can not be used today to factor practical sized moduli (i.e. 1024 or 2048 bits).

In practice, the RSA can be declined in both standard and CRT modes. The standard mode is the straightforward way for implementing RSA. Its principle is recalled below. The CRT mode of RSA, where CRT stands for *Chinese Remainder Theorem*, is usually used to perform efficient signatures in embedded systems. Further details about CRT-RSA implementations can be found at [35, 31].

## *2.1 Standard RSA*

### 2.1.1 Key Generation.

Let $N$, the public modulus, be the product of two large prime numbers $p$ and $q$. The length of $N$, denoted by $n$, also stands for the RSA length. Let $e$ be the *public exponent*, coprime to $\varphi(N) = (p-1)\cdot(q-1)$, where $\varphi(\cdot)$ denotes Euler's totient function. Then the pair $K_p = (N, e)$ is the RSA public key, that is spread over a network. The public key exponent $e$ is linked to the *private exponent* $d$ by the following equation:

$$e \cdot d \equiv 1 \mod \varphi(N) \tag{1}$$

This exponent is also called private key $K_s$, that is kept secret by its owner.

### 2.1.2 RSA Encryption.

Let $m$ be the plaintext that will be ciphered with the RSA algorithm. Then, the cipher operation relies on the following modular exponentiation involving the public exponent $e$:

$$C \equiv m^e \mod N \tag{2}$$

Here $C$ denotes the ciphertext. After sending this cyphertext throughout a network, the expected receiver of the message may want to recover the original message by decrypting $C$. Then, this operation boils down to compute:

$$\tilde{m} \equiv C^d \mod N \tag{3}$$

If no error occurs during computation, transmission or decryption of $C$, then we expect to get:

$$\tilde{m} \equiv m \mod N \tag{4}$$

One can notice that performing a decryption implies the knowledge of the private key. This ensures that only its owner, and so the expected receiver, is able to recover $m$.

#### 2.1.3 RSA Signature Scheme.

As for the decryption, the RSA signature $S$ of a message $m$ consists in a modular exponentiation to the power $d$:

$$S \equiv m^d \mod N \tag{5}$$

To check the validity of the received signature $(S,m)$, the associated public key is used to ascertain that:

$$m \stackrel{?}{\equiv} S^e \mod N \tag{6}$$

In general, this is not really the plaintext $m$ that is signed but a hash $\dot{m}$. This operation is performed by using a hash function $\mathscr{H}$ such that $\dot{m} = \mathscr{H}(m)$. Then, the sent signature is $(S,\dot{m})$ where $S \equiv \dot{m}^d \mod N$. Furthermore, the usage of a hash function is generally combined with a padding scheme (*e.g.* RSA-OAEP [3] for the RSA encryption and RSA-PSS [4] for the signature). For the different fault attacks presented in this paper, we assume that all RSA decryptions or signatures are performed with some hash and/or deterministic padding function.

### *2.2 Modular exponentiation methods*

Modular exponentiation is one of the core operations to implement asymmetric cryptography. Indeed for both RSA decryption and RSA signature schemes, one has to compute a modular exponentiation of the input message as efficiently as possible. For a naive implementation of this operation, we may write:

$$m^d \mod N \equiv \underbrace{m \cdot m \cdot \ldots \cdot m}_{\text{d times}} \mod N \tag{7}$$

The computational complexity of the algorithm above is exponential with respect to the exponent length, which is not acceptable to implement the modular exponentiation. However, a method as intuitive as the previous one, but much smarter, consists in expressing the exponent in a binary basis: $d = \sum_{i=0}^{n-1} 2^i \cdot d_i$. Then, we can perform a modular exponentiation by scanning the bits of the exponent:

$$m^d \mod N \equiv \prod_{i=0}^{n-1} m^{2^i d_i} \mod N \tag{8}$$

As a consequence, the power is no longer performed by multiplying $d$ times a message $m$ by itself but by multiplying, at most $log_2(d)$ times, square powers of $m$. Hence the cost of the exponentiation methods declined from this principle, also called *binary exponentiations*, is linear with respect to the exponent length which is much more efficient than the naive approach. The next section briefly introduces common implementations of binary exponentiations

### 2.2.1 *"Right-To-Left"* Exponentiation.

The first method to perform a modular exponentiation is to scan the exponent bits from least to most significant ones. That is why it is also referred as the *"Right-To-Left"* method. In practice, this method is the most intuitive since it consists in computing consecutive square powers of the input message and, depending on the current exponent bit value, multiplying these powers to the accumulation register. As a consequence, this algorithm exactly transcripts (8). The complete algorithm is detailed below.

**Algorithm 1:** *"Right-To-Left"* Modular Exponentiation

**Input**: $m, d = \sum_{i=0}^{n-1} 2^i \cdot d_i, N$
**Output**: $S = m^d \mod N$
$A \leftarrow 1$;
$B \leftarrow m$;
**for** $i = 0$ **to** $n-1$ **do**
  **if** $d_i = 1$ **then**
    $A \leftarrow A \cdot B \mod N$;
  **end**
  $B \leftarrow B^2 \mod N$;
**end**
**return** $A$;

### 2.2.2 *"Left-To-Right"* Exponentiation.

It is also possible to scan the exponent bits from most to least significant bits. The associated method is not surprisingly called *"Left-To-Right"* exponentiation. This other method differs from the dual one by an accumulation register withdrawal and a different execution flow. Indeed, each iteration begins with the execution of a square that can be followed, depending on the current exponent bit value, by a multiplication. This method is also detailed in Algorithm 2.

**Algorithm 2:** *"Left-To-Right"* Modular Exponentiation

**Input**: $m, d = \sum_{i=0}^{n-1} 2^i \cdot d_i, N$
**Output**: $S = m^d \bmod N$
$A \leftarrow 1$;
**for** $i = (n-1)$ **to** $0$ **do**
  $A \leftarrow A^2 \bmod N$ **if** $d_i = 1$ **then**
    $A \leftarrow A \cdot m \bmod N$;
  **end**
**end**
**return** $A$;

#### 2.2.3 Other variants.

Although both previously presented algorithms are quite efficient to compute modular exponentiations, their implementation may leak information on the exponent. Indeed, the execution of a multiplication directly depends on the current value of the exponent bits. This makes smart card implementation of modular exponentiation algorithm potentially vulnerable to side channel attacks, such as DPA. A basic solution to thwart this flaw is to make the execution independent from the exponent value. Such a variant has been already proposed by J.-S. Coron at CHES 1999 [19] and is also known as the *Square & Multiply Always* algorithm. In this case, one square and one multiplication are sequentially executed at each iteration, whatever the value that the current exponent bit may take.

The performance of the exponentiation algorithms can also be improved. For example, we can derive from the previous algorithms the *Sliding-Window Exponentiation* used in the `OpenSSL` library. The principle of this variant is to scan the exponent by, at most, $k$-bit windows. Hence, after precomputing some odd powers of the input message $m$, this method significantly reduces the number of iterations for a modular exponentiation, and so the performance.

Finally, some variants allow to improve both security and performance. This is the case of the Montgomery exponentiation algorithm [32]. Obviously, it exists other efficient implementations of the modular exponentiation, but we advise an interested reader to take a look at [27] for more details.

## 3 Classical Fault Analysis of Standard RSA Implementations

Since the introduction of fault attacks at the end of the nineties, the security against perturbation of CRT-based implementation of RSA has not been the sole implementation mode targeted [13]. Indeed, the security of standard RSA implementations have been also challenged, leading to various attack methodologies. Among these fault attacks, we have chosen to distinguish to main categories. The first one deals

with the perturbation of intermediate computations. The fault attacks that belong in this category take advantages of the perturbation of intermediate values or of the execution flow. The second category we have distinguished gather attacks based on exploiting modifications of RSA public elements. This trend is quite recent but has ever led to successful applications against various standard RSA implementations.

The next paragraph details the different attacks that we have identified from our state-of-the-art study.

## *3.1 Perturbation of intermediate computations*

### 3.1.1 Register faults.

Bellcore researchers not only introduced the concept of fault attacks [5] but they also showed how this new concept can be applied to many public key cryptosystems, including standard RSA, and their various implementations. In more details, they explained in [13] how to exploit fault injections during the execution of a standard RSA signature to recover the private exponent. The fault model they considered, the so-called *register fault*, is a transient or permanent bit-flip induced in the memory area containing the current value of the exponentiation algorithm. Under this model, they showed that the perturbation of standard RSA signature, implemented with the *"Right-To-Left"* exponentiation, may leak some secret information. The principle of the attack is described in the next paragraph

General Methodology.

The fault attack against standard RSA signature proposed by Bellcore researchers can be split in two parts. The first one is *"on-line"* and consists in gathering sufficiently many message/faulty signature pairs $(m_i, \hat{S}_i)$ by inducing permanent faults, one per faulty signature, on the register that contains a intermediate values. Then, in the second part, the attackers tries to analyze the collected faulty signatures to recover the whole secret key. Hence, this part of the attack is completely *"off-line"*. The principle of the analysis is recalled below. Let $S_i$ be the correct signature and let $\varepsilon(m_i) = \pm 2^b$ be the mathematical representation of a bit-flip with $0 \leqslant b < n$. Since this fault is supposed to be permanent, then corresponding faulty signature can be expressed as:

$$\hat{S}_i \equiv \left( \left( \prod_{j=0}^{t-1} m_i^{2^j d_j} \right) \pm 2^b \right) \cdot \prod_{j=t}^{n-1} m_i^{2^j d_j} \tag{9}$$

$$\equiv S_i \pm 2^b \cdot \prod_{j=t}^{n-1} m_i^{2^j d_j} \mod N \tag{10}$$

$$\text{or } S_i \equiv \hat{S}_i \pm 2^b \cdot m_i^{d_{[t]}} \mod N \tag{11}$$

with $d_{[t]} = \sum_{j=t}^{n-1} 2^j d_j$. If we use the signature's verification operations, the previous equation becomes:

$$m_i \equiv (\hat{S}_i \pm 2^b . m_i^{d_{[t]}})^e \mod N \tag{12}$$

One can notice that this equation is interesting because it only depends on the message $m_i$ and the corresponding faulty signature $\hat{S}_i$ (*i.e.* the knowledge of the correct signature is no longer required). Moreover, the fault injection has isolated a part $d_{[t]}$ of the private exponent $d$. This is precisely this part of exponent that the attacker has to determine with a *guess-and-determine* approach.

The whole exponent is gradually recovered, from most to least significant bits, by repeating the previous analysis on different faulty signatures. In each analysis, $w$ bits of exponent are retrieved. In more details, for each analysis, the attacker has to simultaneously guess the values of the part of exponent isolated $d_{[t]}$ and the fault induced $\pm 2^b$ such that (12) is satisfied. According to [13], the whole private exponent $d$ can be determined by this way, with a probability greater than $\frac{1}{2}$, from $(n/w)\ log\ (2n)$ message-signature pairs. In this case, the attack complexity is about $O((2^w n^3 log^2(n))/w^3)$ exponentiations. We can additionally remark that this fault attack was later generalized to *"Left-To-Right"*-based exponentiations by J. Blömer and M. Otto [34].

### 3.1.2 Faults on the private exponent.

This attack was published by F. Bao *et al.* in [1] and then in [2]. 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, such an effect can be obtained by flipping a bit of the private exponent, or by corrupting the evaluation made just before the conditional branch in the classical implementations of the modular exponentiation (see Sect. 2.2). The following paragraph only describes the attack for a bit error on the exponent $d$.

General methodology.

Let $m$ be a plaintext and $C$ the corresponding ciphertext obtained from a RSA encryption (see Sect. 2). 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: $\hat{m} \cdot m^{-1}$. The induced error can be modeled as a *bit-flip* of the $t$-th bit of $d$. Therefore, we have:

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

That implies, either $\hat{m} \cdot m^{-1} \equiv C^{-2^t} \bmod N \Rightarrow d_t = 1$,
or $\hat{m} \cdot m^{-1} \equiv C^{2^t} \bmod N \Rightarrow d_t = 0$. This method can be repeated until we obtain enough information on the private exponent. Note that this attack is also suitable in case of a multiple error model [1]. Moreover the principle can be adapted to attack cryptosystems based on discrete logarithm (DSA, El-Gamal, ...). Finally, this attack strategy has been later extended and generalized by M. Joye *et al.* [24], who describe an improved attack relying on the mere knowledge of the faulty deciphered text.

### 3.1.3 Exploiting safe-errors.

Classical fault attacks often exploit the difference between a correct and a faulty output to deduce some secret information. However, M. Joye and S.-M. Yen noticed that some secret information may leak even if a fault causes no effect on the final result of the computation [38]. This is why this particular kind of fault attacks is also called *"safe-error"* attack. The attacker must be able to perform some perturbation of which he knows the probably effect with a good repeatability. Among these attacks, two categories are usually distinguished: *C-safe-error attacks* that target dummy operations [39] and *M-safe-error attacks* that target registers allocations [38]. In order to illustrate the principle of safe-error attacks in the context of RSA, we have chosen to detail the *C-safe-error attacks* against the *Square & Multiply Always* exponentiation [19].

General methodology.

The purpose of the *Square & Multiply Always* exponentiation is to make its execution independent from the exponent value. Hence, the conditional branch is withdrawn (see Sect. 2.2) and an extra dummy multiplication is added such that, regardingless of the value of the exponent, a square and a multiplication are always executed at each iteration. The principle of the *C-safe-error attacks* proposed by S.-M. Yen *et al.* [39] is to exploit faults induced while dummy multiplications are performed. To achieve this, the attacker has to inject a fault during a RSA signature and, if the signature remains *"error-free"*, it means that a dummy multiplication has been infected. Therefore, the attacker can deduce that the exponent bit value handled while the perturbation was provoked is zero. Otherwise, the result would be incorrect and the exponent bit value equals to one. The attacker has to repeat the attack at each iteration to gradually recover the whole private exponent. One can notice

that this attack does not apply upon classical exponentiation algorithm but upon a variant expected to defeat *Simple Power Analysis*. As a consequence, checking only the output may not be enough to protect and implementation of a cryptographic algorithm against faults. Furthermore, designers of secure solution may be careful not to add new vulnerability while trying to defeat other ones.

## 4 Exploiting Perturbations of RSA public modulus

Although the issue of exploiting malicious modifications of public elements was addressed in the context of Elliptic Curve based cryptosystems [16], it took a half decade before seeing the first application to RSA. Indeed, the first fault attack against public key elements is due to J.-P. Seifert with a method for corrupting RSA signature's verification [37, 33]. This fault attack aims to corrupt signature verification mechanism by modifying the value of the public modulus $N$. Nevertheless, no information about the private exponent $d$ is revealed with this fault 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 [37, 33], was published in [14] and reviewed in [17]. It makes it possible to extract the private key using a modulus perturbation. As in Seifert's attack, the fault on the modulus is induced before executing the exponentiation. Hence, if the faulty modulus has a small divisor $r$, the attacker will be able to solve an instance of the Discrete Logarithm Problem from the corresponding faulty signature and obtain $d \mod r$.

A new fault attack against *"Right-To-Left"*-based implementations of the core RSA exponentiation [8], completed by the attack of the dual implementation [7] has been presented lately. Contrary to previous attacks, authors assumed that the fault is injected during the execution of an RSA signature. Then, from the knowledge of a correct and a corresponding faulty signature, the attacker guesses-and-determines simultaneously the faulty modulus and the part of the private exponent that has been isolated by the fault injection. To recover the whole exponent, the attacker has to repeat the analysis for sufficiently many signatures perturbated at different moments of the execution.

In the next paragraphs, we will detail the different fault attacks published against RSA public elements.

### *4.1 Modifying $N$ before a signature to solve small DLP.*

Although J.-P. Seifert – with its attack proposal to corrupt a RSA signature verification mechanism [37, 33] – first addressed the issue of exploiting RSA signatures performed under a faulty public modulus $N$, the first analysis leading to a complete secret key recovery is due to E. Brier *et al.* [6]. The main idea of their attack is to an-

alyze faulty signatures performed under a faulty modulus $N$ to recover small parts of the private exponent $d$ (*i.e.* 20 to 30 bits from each faulty signatures). All the parts of exponent extracted from different faulty RSA signatures are finally combined with the Chinese Remainder Theorem to build the full private exponent.

General Methodology.

This fault attack can be split into two distinct phases. The first one is *"on-line"* and consists in gathering $K$ pairs message/faulty signatures $(m_i, S_i)_{1 \leqslant i \leqslant K}$ computed with faulty moduli $\hat{N}_i \neq N$ such that:

$$S_i = \dot{m}_i{}^d \bmod \hat{N}_i \tag{13}$$

The authors assumed that the values of the different faulty $(\hat{N}_i)_{1 \leqslant i \leqslant K}$ are not known by the attacker. However, they have also supposed that theses values are uniformly distributed over the $n$-bit long integers [14, 17]. From this set of faulty signatures, the attacker performs an *"off-line"* phase which consists in recovering the private exponent by parts. The proposed analysis is declined in different variants depending the choice of the attacker to generate, or not, a table of possible values for faulty moduli. But, generating such a table, also referred as the *dictionary of moduli* [14], requires to choose a fault model. Finally, one can notice that the number of signatures to gather (*i.e.* the parameter $K$) depends on the method chosen to perform the analysis. Both methods are detailed below.

Analysis without dictionary.

This first method is used when the attacker is not able to identify a fault model from the set of gathered faulty signatures or, if the identified model induce a dictionary too large to be handled (*i.e.* more than $2^{32}$ entries). In this case, it is not possible to retrieve faulty moduli used to perform the faulty signatures. To overcome this difficulty, the authors give a mean to find some factors $p^a$ of $\hat{N}_i$ thanks to the equation (14) that it satisfied under some conditions [1] with probability 1 if $p^a \mid \hat{N}_i$ and $\frac{1}{r}$ otherwise (where $r$ is a small prime that divides the multiplicative order of $\dot{m}_i$).

$$S_i \equiv \dot{m}_i{}^{d \mod \varphi(p^a)} \mod p^a \tag{14}$$

Then, for such a $p^a$, if $\varphi(p^a)$ is divisible by a small enough prime $r_k$ (*i.e.* 20 to 30-bit long), the attacker can take advantage of the bias (see [14, Proposition 1] for details) with a counting method that enables to determine parts of the private exponent $d_k = d \bmod r_k$ by solving small discrete logarithms on the faulty signatures gathered. Hence, when $R = \prod_k r_k$ is bigger than $N$ (and obviously bigger than $\varphi(N)$), it is possible to use the Chinese Remainder Theorem to build the whole pri-

[1] $p$ is a prime number such that $p \nmid \dot{m}_i$ and $p \nmid S_i$

vate exponent $d$ from the recovered parts.

One can notice that the advantage of this method is that it may lead to a full key recovery regardingless of the faults injected on the different moduli. According to [14, 17], the implementation of this methodology enables to recover 512-bit private exponents (1024-bit in case of small public exponent $e$[2]) by gathering about 25000 faulty signatures. On the other hand, 60000 faulty signatures are enough to recover 1024-bit secret exponents (2048-bit in case of small public exponent $e$).

Analysis with a dictionary.

The attack performance can be improved if the attacker is able to identify and validate a fault model from the faulty signatures collected during the *"on-line"* phase. From this fault model and the knowledge of $N$, the attacker can establish a list of possible values for the faulty moduli. This list is also called *dictionary of moduli*. Once the dictionary is generated, the attacker tries to guess the pairs $(m_i, \hat{S}_i)$ that result from a computation with one of the dictionary entry. Each successful guess, or *"hit"*, brings a certain amount of information on the private exponent $d$. In terms of performance, this approach is very interesting since, according to [14, 17], 28 *"hits"* and only 1100 faulty signatures are enough for recovering a 1024-bit RSA private exponent. Moreover, by finding these *"hits"* with a statistical approach, it is possible to extract the private exponent with a number of faults equal to the number of *"hits"* (which is proved to be optimal). In this case 28 faulty signatures may suffice to recover a 1024-bit RSA private exponent.

Although fault attacks has been considered as a powerful way to attack implementations of cryptographic algorithms, the presented attacks highlighted that even non-critical elements, such as public keys, have to be protected against perturbations. Whereas public elements are not supposed to reveal secret information, their perturbation may be a source of leakage leading to the corruption of a signature verification mechanism [37, 33] or worse, to a full private key recovery [14, 17]. However, the use of an exponent randomization technique may be an effective way for defeating such attacks. Furthermore, these attacks only apply for perturbations that occur before performing the core exponentiation of the RSA signature. The attack presented in the next section proved that this claim is no longer true since the perturbation of public elements during the signature can also be exploited.

[2] When $e$ is small, the authors take advantage of the RSA equation $e \cdot d \equiv 1 \mod \varphi N$ to determine the most significant part of $d$. Indeed, knowing that $\varphi N = N + 1 - p - q \approx N$ for its most significant part, then $d \approx \frac{1+k\cdot(N+1)}{e}$ with $k < e$. Hence, if $e$ is small (*e.g.* $e = 2^16+1$), the most significant part of $d$ can be directly deduced from the previous relation completed by an exhaustive search on $k$.

### *4.2 Exploiting faults on $N$ during a RSA signature.*

In J.P Seifert and E. Brier *et al.*'s proposals [37, 14], authors exploited perturbations of the public modulus provoked before the core exponentiation so that the whole signature is performed with a faulty modulus $\hat{N}$. The attack presented by A. Berzati *et al.* [8] extended the fault model by enabling an attacker to exploit faults injected of a *"Right-To-Left"* based exponentiation. The modification of $N$ was supposed to be a transient random byte fault modification. It means that only one byte of $N$ is changed to a random value. Moreover, they also assumed that 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 [22, 12]. Furthermore, it can be easily adapted to 16-bit or 32-bit architectures. This attack was later generalized to *"Left-To-Right"* based implementations including Montgomery or *Sliding-Window* exponentiations. Finally, the authors proved at CHES 2010 that the exponent randomization method – also referred as exponent blinding – suggested by P. Kocher [28] may not be efficient for protecting RSA implementations against faults on public key elements.

General Methodology.

In order to detail the general methodology, we assume that the implementation of the modular exponentiation is *"Right-To-Left"*-based (see Sect. 2.2.1) and that an attacker is able to transiently modified one byte of the public modulus $N$ at the $t$-step of the exponentiation. If we denote by $A_t$ and $B_t$ the internal register values, then for a fault occurring while $B_t$ is computed, we have:

$$\hat{S}_t \equiv A_t \cdot \hat{B}_t^{d_t} \cdot \ldots \cdot \hat{B}_t^{2^{(n-1)-t} \cdot d_{n-1}} \bmod \hat{N} \tag{15}$$

$$\equiv A_t \cdot \hat{B}_t^{\frac{d_{[t]}}{2^t}} \bmod \hat{N} \tag{16}$$

where $d_{[t]} = \sum_{i=t}^{n-1} 2^i \cdot d_i$ Hence, the fault injection splits the computation into a correct and a faulty part. The main consequence is that a part of the private exponent, namely $d_{[t]}$ is isolated by the fault injection. In practice this part of exponent is composed of a known (already determined) part and part to guess. So, if the part to guess 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, the attacker has to choose a candidate value $\hat{N}'$ (built according to the fault model) and another candidate value for the part of $d_{[t]}$ to guess. Then he computes from the correct signature $A'_t \equiv S_t \cdot \dot{m}^{-d'_{[t]}} \bmod N$ and:

$$S'_{(d'_{[t]}, \hat{N}')} \equiv A'_t \cdot \left( \dot{m}^{2^{t-1}} \bmod N \right)^{2 \cdot \frac{d'_{[t]}}{2^t}} \bmod \hat{N}' \tag{17}$$

Then, if this re-built faulty signature satisfies (18) then the pair of candidate values $(\hat{N}', d'_{[t]})$ is the expected one with high probability. Otherwise, the attacker has to choose another candidate pair and perform this test again.

$$S'_{(d'_{[t]};\hat{N}')} \equiv \hat{S}_t \bmod \hat{N}' \tag{18}$$

The whole exponent is gradually recovered by cascading the previous analysis with signatures faulted at different moments of the execution and using the knowledge of already found parts. The performance of this fault attack mainly depends on the number of exponent bits $w$, that the attacker can extract from correct/signature pair. This parameter is a trade-off between fault number and performance and so, has to be carefully chosen. Authors estimated that for $w = 4$ (which ensure a reasonable execution time), the number of faulty signatures to gather for recovering a 1024-bit private exponent is about 250 [8]. Finally, one can notice that a similar analysis can be performed when the first operation infected by the fault is a multiplication (*i.e.* the computation of $A_t$ is infected first). The details of this variant are provided in [8].

Application to "Left-To-Right" based exponentiations.

After exploiting public key perturbation during execution of *"Right-To-Left"* implementations of RSA signatures, A. Berzati *et al.* later generalized their attack to "Left-To-Right" based exponentiations. Although both dual algorithms are very similar, the generalization of the previous fault attack was not so easy. To illustrate the difficulties induced by this generalization they have considered a classical *"Left-To-Right"* exponentiation (see Sect.2.2.2) and an attacker able to perturbate the modulus $N$ during its execution as in the previous fault model but, $t$ steps before the end of the execution. Denoting by $A_{n-t}$ the internal register before the perturbation of $N$, the faulty signature $\hat{S}_t$ can be expressed as below:

$$\hat{S}_t = {A_{n-t}}^{2^t} \cdot \dot{m}^{d_{[t]}} \bmod \hat{N} \tag{19}$$

and, this time $d_{[t]} = \sum_{i=0}^{t-1} 2^i \cdot d_i$ denotes the $t$-bit least significant part of $d$. By observing (19), one can notice that, contrary to the *"Right-To-Left"* case, $t$ cascaded squares are induced by the fault injection. Hence, extending the previous analysis supposes that the attacker is able to compute modular square roots (which is a difficult problem in RSA rings). Fortunately, since these additional squares are computed modulo $\hat{N}$, it is possible to efficiently compute square roots when $\hat{N}$ is prime (or $B$-smooth) using the probabilistic Tonelli & Shanks algorithm [18]. To validate the consistency of considering only prime faulty moduli, the authors provided a complete theoretical analysis based on number theory. Hence they estimated that, according to the fault model and for a 1024-bit RSA, 1 faulty modulus over 305 will be prime [7]. Moreover, they observed that, although two square roots may be computed from a given quadratic residue, the number of $t$-th square root is not $2^t$ but is bounded by the bigger power of two dividing $\hat{N} - 1$. Since this power is usually quite

small (*i.e.* smaller than 5 in their experiments), they concluded that, in practice, the computation of square roots does not prevent an attacker from using a guess-and-determine approach declined from *"Right-To-Left"* analysis and recovering parts of $d$ when *"Left-To-Right"*-based exponentiations are targeted.

In the previous paragraph, we have detailed a proposal for attacking both dual implementations of the modular exponentiation [8, 7]. In both cases, the attacker takes advantage of the fault that occurs during the execution of an RSA signature. However, the previous results show that *"Right-To-Left"*-based exponentiations seem to be easier to attack than *"Left-To-Right"* ones. Moreover, this attack methodology has been later reused to successfully defeat the randomized exponent countermeasure [9]. Although this countermeasure seems to be efficient against perturbations of public elements that occur before the computation of signatures [14], A. Berzati *et al.* showed at CHES 2010 that signatures partially infected by a faulty public modulus are still exploitable when the private exponent is blinded [9]. However, the authors suggest to use the Probabilistic Signature Scheme with RSA (RSA-PSS) [4] for defeating their attacks. Eventually, this work completes the state-of-the-art and highlights the need for protecting RSA public elements against perturbations, even during the computation of signatures.

## 5 Conclusion

The study of the fault injection in RSA implementations shows that a large panel of different attacks exist. Of course the popularity of RSA is widely accountable, but the variety of the proposed implementations, even secured ones, leads to different fault exploitations. If first instance of fault attacks has led to very powerful applications, especially for CRT-RSA where one fault may suffice, standard RSA implementations seems to be more difficult to attack. Indeed, for such implementations, the goal of the attacker is not to factor the modulus but gradually recovering the private exponent by bit windows or by residues. The vulnerability of this kind of implementations is the modular exponentiation that scans the private exponent bit by bit. In the both cases, the conditional checks must be avoided or secured and the public elements have to be protected as the others values. One can conclude that implementations that parcel the secret elements for the computation, are by construction vulnerable to fault attacks. The countermeasures as masking techniques lead to confuse the isolated parts, but they may be not enough efficient, as proven by the number of attacks that nevertheless exploit the residual vulnerabilities.

## References


1. Bao, F., Deng, R.H., Han, Y., Jeng, A.B., Narasimhalu, A.D., Ngair, T.H.: Breaking public key cryptosystems on tamper resistant devices in the presence of transient faults. In: B. Christianson, B. Crispo, T.M.A. Lomas, M. Roe (eds.) Security Protocols, *Lecture Notes in Computer Science*, vol. 1361, pp. 115–124. Springer (1998)
2. Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C.: The sorcerer's apprentice guide to fault attacks. Proceedings the IEEE **94**(2), 370–382 (2006)
3. Bellare, M., Rogaway, P.: Optimal asymmetric encryption. In: A. De Santis (ed.) Advances in Cryptology – EUROCRYPT '94, *Lecture Notes in Computer Science*, vol. 950, pp. 92–111. Springer (1995)
4. Bellare, M., Rogaway, P.: The exact security of digital signatures. In: Advances in Cryptology – EUROCRYPT '96, *Lecture Notes in Computer Science*, vol. 1070, pp. 399–416. Springer (1996)
5. Bellcore: New threat model breaks crypto codes. Press Release (1996)
6. Berzati, A.: Attaques par canaux cachés sur les implémentations logicielles d'algorithmes cryptographiques. Master's thesis, Grenoble INP & Université Joseph Fourier Grenoble I (2007)
7. Berzati, A., Canovas, C., Dumas, J.G., Goubin, L.: Fault attacks on rsa public keys: Left-to-right implementations are also vulnerable. In: M. Fischlin (ed.) Topics in Cryptology – CT-RSA 2009, *Lecture Notes in Computer Science*, vol. 5473, pp. 414–428. Springer (2009)
8. Berzati, A., Canovas, C., Goubin, L.: Perturbating RSA public keys: An improved attack. In: E. Oswald, P. Rohatgi (eds.) Cryptographic Hardware and Embedded Systems – CHES 2008, *Lecture Notes in Computer Science*, vol. 5154, pp. 380–395. Springer (2008)
9. Berzati, A., Canovas-Dumas, C., Goubin, L.: Public key perturbation of randomized RSA implementations. In: S. Mangard, F.X. Standaert (eds.) Cryptographic Hardware and Embedded Systems – CHES 2010, *Lecture Notes in Computer Science*, vol. 6225, pp. 306–319. Springer (2010)
10. Biehl, I., Meyer, B., Müller, V.: Differential fault analysis of elliptic curve cryptosystems. In: M. Bellare (ed.) Advances in Cryptology – CRYPTO 2000, *Lecture Notes in Computer Science*, vol. 1880, pp. 131–146. Springer (2000)
11. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: B.S. Kaliski Jr. (ed.) Advances in Cryptology – CRYPTO '97, *Lecture Notes in Computer Science*, vol. 1294, pp. 513–525. Springer (1997)
12. Blömer, J., Otto, M.: Wagner's attack on a secure CRT-RSA algorithm reconsidered. In: L. Breveglieri, I. Koren, D. Naccache, J.P. Seifert (eds.) Fault Diagnosis and Tolerance in Cryptography – FDTC 2006, *Lecture Notes in Computer Science*, vol. 4236, pp. 13–23. Springer (2006)
13. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of eliminating errors in cryptographic computations. Journal of Cryptology **14**(2), 101–119 (2001). Earlier version published in EUROCRYPT '97
14. Brier, É., Chevallier-Mames, B., Ciet, M., Clavier, C.: Why one should also secure RSA public key elements. In: L. Goubin, M. Matsui (eds.) Cryptographic Hardware and Embedded Systems – CHES 2006, *Lecture Notes in Computer Science*, vol. 4249, pp. 324–338. Springer (2006)
15. Brumley, D., Boneh, D.: Remote timing attacks are practical. Computer Networks **48**(5), 701–716 (2005)
16. Ciet, M., Joye, M.: Elliptic curve cryptosystems in the presence of permanent and transient faults. Designs, Codes and Cryptography **36**(1), 33–43 (2005)
17. Clavier, C.: De la sécurité des cryptosystèmes embarqués. Ph.D. thesis, Université de Versailles Saint-Quentin-en-Yvelines (2007)
18. Cohen, H.: A Course in Computational Algebraic Number Theory, *Graduate Texts in Mathematics*, vol. 138. Springer (1993)

19. Coron, J.S.: Resistance against differential power analysis for elliptic curve cryptosystems. In: Ç.K. Koç, C. Paar (eds.) Cryptographic Hardware and Embedded Systems − CHES '99, *Lecture Notes in Computer Science*, vol. 1717, pp. 292–302. Springer (1999)
20. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory **IT-22**(6), 644–654 (1976)
21. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: Concrete results. In: Ç.K. Koç, D. Naccache, C. Paar (eds.) Cryptographic Hardware and Embedded Systems − CHES 2001, *Lecture Notes in Computer Science*, vol. 2162, pp. 251–261. Springer (2001)
22. Giraud, C.: DFA on AES. In: H. Dobbertin, V. Rijmen, A. Sowa (eds.) Advanced Encryption Standard − AES (AES 2004), *Lecture Notes in Computer Science*, vol. 3373, pp. 27–41. Springer (2005)
23. Hoch, J.J., Shamir, A.: Fault analysis of stream ciphers. In: M. Joye, J.J. Quisquater (eds.) Cryptographic Hardware and Embedded Systems − CHES 2004, *Lecture Notes in Computer Science*, vol. 3156, pp. 240–253. Springer (2004)
24. Joye, M., Quisquater, J.J., Bao, F., Deng, R.H.: RSA-type signatures in the presence of transient faults. In: M. Darnell (ed.) Cryptography and Coding, *Lecture Notes in Computer Science*, vol. 1355, pp. 155–160. Springer (1997)
25. Kim, C.H., Bulens, P., Petit, C., Quisquater, J.J.: Fault attacks on public key elements: Application to DLP-based schemes. In: S.F. Mjølsnes, S. Mauw, S.K. Katsikas (eds.) Public Key Infrastructure (EuroPKI 2008), *Lecture Notes in Computer Science*, vol. 5057, pp. 182–195. Springer (2008)
26. Kleinjung, T., Aoki, K., Franke, J., Lenstra, A.K., Thomé, E., Bos, J.W., Gaudry, P., Kruppa, A., Montgomery, P.L., Osvik, D.A., te Riele, H.J.J., Timofeev, A., Zimmermann, P.: Factorization of a 768-bit RSA modulus. In: T. Rabin (ed.) Advances in Cryptology − CRYPTO 2010, *Lecture Notes in Computer Science*, vol. 6223, pp. 333–350. Springer (2010)
27. Koç, Ç.K.: High-speed RSA implementation. Tech. Rep. TR 201, RSA Laboratories (1994)
28. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: N. Koblitz (ed.) Advances in Cryptology − CRYPTO '96, *Lecture Notes in Computer Science*, vol. 1109, pp. 104–113. Springer (1996)
29. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: M.J. Wiener (ed.) Advances in Cryptology − CRYPTO '99, *Lecture Notes in Computer Science*, vol. 1666, pp. 388–397. Springer (1999)
30. Lenstra, A.K., Lenstra Jr, H.W. (eds.): The Development of the Number Field Sieve, *Lecture Notes in Mathematics*, vol. 1554. Springer (1993)
31. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press (1997)
32. Montgomery, P.L.: Speeding up the Pollard and elliptic curve methods of factorization. Mathematics of Computation **48**(177), 243–264 (1987)
33. Muir, J.A.: Seifert's RSA fault attack: Simplified analysis and generalizations. In: P. Ning, S. Qing, N. Li (eds.) Information and Communications Security (ICICS 2006), *Lecture Notes in Computer Science*, vol. 4307, pp. 420–434. Springer (2006)
34. Otto, M.: Fault attacks and countermeasures. Ph.D. thesis, Institut für Informatik, Universität Paderborn (2004)
35. Quisquater, J.J., Couvreur, C.: Fast decipherment algorithm for RSA public-key cryptosystem. Electronics Letters **18**(21), 905–907 (1982)
36. Rivest, R.L., Shamir, A., Adleman, L.M.: Method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM **21**(2), 120–126 (1978)
37. Seifert, J.P.: On authenticated computing and RSA-based authentication. In: V. Atluri, C. Meadows, A. Juels (eds.) 12th ACM Conference on Computer and Communications Security (CCS 2005), pp. 122–127. ACM Press (2005)
38. Yen, S.M., Joye, M.: Checking before output may not be enough against fault-based cryptanalysis. IEEE Transactions on Computers **49**(9), 967–970 (2000)
39. Yen, S.M., Kim, S., Lim, S., Moon, S.J.: A countermeasure against one physical cryptanalysis may benefit another attack. In: K. Kim (ed.) Information Security and Cryptology − ICISC 2001, *Lecture Notes in Computer Science*, vol. 2288, pp. 269–294. Springer (2002)