Why One Should Also Secure RSA Public Key Elements

[Published in L. Goubin and M. Matsui, Eds., *Cryptographic Hardware and Embedded Systems – CHES 2006*, vol. 4249 of *Lecture Notes in Computer Science*, pp. 324–338, Springer-Verlag, 2006.]

Eric Brier[1], Benoît Chevallier-Mames[1,2],
Mathieu Ciet[1], and Christophe Clavier[1]

[1] Gemalto, Security Labs,
La Vigie, Avenue du Jujubier, ZI Athélia IV,
F-13705 La Ciotat Cedex, France
firstname.familyname@gemalto.com
[2] École Normale Supérieure, Département d'Informatique,
45 rue d'Ulm,
F-75230 Paris 05, France

Abstract. It is well known that a malicious adversary can try to retrieve secret information by inducing a fault during cryptographic operations. Following the work of Seifert on fault inductions during RSA signature verification, we consider in this paper the signature counterpart.
Our article introduces the first fault attack applied on RSA in standard mode. By only corrupting one public key element, one can recover the private exponent. Indeed, similarly to Seifert's attack, our attack is done by modifying the modulus.
One of the strong points of our attack is that the assumptions on the induced faults' effects are relaxed. In one mode, *absolutely no knowledge* of the fault's behavior is needed to achieve the full recovery of the private exponent. In another mode, based on a fault model defining what is called *dictionary*, the attack's efficiency is improved and the number of faults is dramatically reduced. All our attacks are very practical.
Note that those attacks do work even against implementations with deterministic (*e.g.,* RSA-FDH) or random (*e.g.,* RSA-PFDH) paddings, except for cases where we have signatures with randomness recovery (such as RSA-PSS).
The results finally presented on this paper lead us to conclude that it is also mandatory to protect RSA's public parameters against fault attacks.

Keywords: RSA, Standard Mode, Fault Cryptanalysis, Seifert's Attack.

1 Introduction

1.1 Basics

RSA [16] is today the most widely used public key cryptosystem. Let $n = pq$ be the product of two large primes typically of 512 to 1024 bits. Let e be the public exponent, coprime with $\varphi(n) = (p-1)(q-1)$, where $\varphi(\cdot)$ is the Euler totient function. The public exponent e is linked to the so-called private exponent d by equation $ed \equiv 1 \pmod{\varphi(n)}$.

Basically, in RSA cryptosystem [3, 4, 14], public operations (*i.e.,* signature verification or encryption) are done by computing an e-th power, while private operations (*i.e.,* signature generation or decryption) are done by computing a d-th power. To speed up private operations, an efficient technique based on the Chinese Reminder Theorem was proposed [15]: this is referred to the CRT mode, by opposition to the standard mode.

RSA and physical attacks. The security of the RSA public key cryptosystem is linked to the hardness of the factorization. In addition, when implementing cryptosystems, one needs to be very careful about information leakage, which else would allow so-called *side-channel analysis* [11].

In 1996, another type of attacks, called *fault attacks*, has been introduced against the RSA CRT implementation [6]. This attack is known as the *Bellcore attack*: only one fault induction on one half of the computation suffices to recover the modulus factorization from one correct and one faulty signature, by just computing a greatest common divisor. However, in case of the use of random padding, the Bellcore attack cannot be applied.

Nowadays, in case of the standard RSA, there is only one known fault induction attack in order to recover the private exponent. This attack is based on flipping bits of the private exponent one per one.[1]

Type of faulted parameters. All the previous methods are based on fault induction against private parameters.[2] An exception is presented in a recently published article by Seifert [17], where he proposes for the first time to attack the public part of RSA signature scheme, *i.e.,* signature verification. The RSA scheme itself is not endangered, *i.e.,* the attacker is not able to forge new valid signatures, but Seifert's attack allows the attacker to pass — with a certain probability — the signature verification step, for a message of her choice, by corrupting the public modulus: all in all, the attacker's goal is fulfilled, but the attack is "one shot", in the sense that it needs to be launched again to produce another wrong acceptance.

1.2 Our contribution

In this paper, we propose the first fault attack that can be used against RSA in standard mode, to recover the private exponent by corrupting only *public key elements*. This point is very critical, as other existing attacks already target the private exponent, which should in essence be protected against faults. On the

[1] This attack can also be generalized to modify small sets of bits, typically bytes.

[2] Inducing fault against public method has also been considered in the case of elliptic curves [5, 8].

contrary, prior to our paper, it was unclear whether it was necessary or not to protect public elements: our paper clarifies this point by concluding that *RSA public key elements also have to be protected against fault attacks.*

Our attack has the same starting point as Seifert's one: it consists in corrupting the public modulus. However, Seifert's attack allows the attacker to pass a signature verification (with a certain probability), while our attack allows a *full key recovery.* Once the key is recovered, the adversary gets all power, while Seifert's attack allows just a single false acceptance.

An additional key property of our attack is that, in one of its mode, the attacker needs *absolutely no knowledge* of the fault effect. No matter what the fault's effect is, she might recover the private exponent. This clearly improves upon Seifert's attack (where the attacker must *guess* the faulty modulus), or upon flipping bit attack (where the fault attack must be unrealistically precise).

In another mode, our attack can be improved. With the help of a *fault model*, we are able to dramatically reduce the number of faults needed to fully recover the private key. As explained later, the attacker is not assumed to be so powerful, as her knowledge of the fault she produced may be probabilistic or unprecise: some of the off-line phases of the attack are proposed to deal with uncertainty.

The new fault attacks presented in this article apply to standard RSA and not to the CRT mode. Moreover, fixed paddings (*e.g.,* RSA-FDH [3]) or random paddings with joint randomness (*e.g.,* RSA-PFDH [9]) do not influence the attack. The only limitation is in case of the signature with randomness recovery (*e.g.,* RSA-PSS [4]) where the problem remains open.

1.3 Organization of the paper

This article is organized as follows. In Section 2, we remind the background regarding fault attacks and the novelty introduced by Seifert. The core of our paper begins at Section 3 where we define the general framework of our attack. Then, in Section 4, we introduce the first mode of our attack, where the adversary needs no particular knowledge about the fault induced on the device. Later, in Section 5, we refine our attack to the case where a model of the fault attack is accessible to the adversary. Finally, we conclude in Section 6.

2 Preliminaries

In the paper, the notation $DL(\mu, s, n)$ is used to express the discrete logarithm of s with respect to the basis μ modulo n, which either is an integer defined modulo the multiplicative order of $\mu \bmod n$ or does not exist mean that s is not a power of $\mu \bmod n$. Clearly, it can be generalized to any prime power p^a dividing n, and any integer r dividing the multiplicative order of $\mu \bmod p^a$ as $DL(\mu, s, p^a) \bmod r$ (denoted $DL(\mu, s, p^a, r)$ in the sequel), which is an integer defined modulo r or does not exist.

We remind that for relatively small value of r — say from 15 to 20 digits —, the discrete logarithm $DL(\mu, s, n, r)$ can be computed efficiently by square root methods such as *baby-step giant-step* or *Pollard's rho* [12].

2.1 Fault models

Fault based attacks can be realized in practice by various ways. In the past, it was possible on certain components to induce faults using VCC glitches [1]. Nowadays, chips are designed to resist such fault induction means.

The best tools today to inject fault is certainly using a laser [2]. The effects of the fault may vary according to the component, to the type of laser used, to the various smart mechanisms implemented by the hardware designers *etc.* Various fault models are commonly considered according to the "hypothetical" capabilities of the attacker, in terms of location and timing precision of her faults.

From a practical point of view, the fault effect is highly dependent on the component. The most simple fault to induce is to change a word (whose size depends on the architecture) in an undetermined way. This can simply be obtained by inducing a fault on address decoders for example, when parameters stored in EEPROM or in Flash are transferred to RAM. If this transfer includes a random ordering, then the location, in terms of word index is also unknown.

For some component the effect of the fault can be known, eventually with some probability. In the literature, single bit flip models are sometimes considered. However, this is not so easy to make in practice whereas faulty word models are very realistic. Moreover, a distinction is also done between permanent (sticky bits) and transient faults: in the following we mainly consider values changed from the beginning to the end of their use in a processus.

In this paper, we make less assumptions on the attacker's injection capabilities and stick a more realistic model

2.2 About the attack of Seifert and Muir

Before going further, let us first give a brief description of the Seifert's paper that motivated this article [17] and its generalization by Muir [13]. For the sake of simplicity, the attack is called *Seifert's attack* in the rest of this article. We refer the interested reader to the original papers for further details.

The basic principle of Seifert's attack is the following: the attacker tries to find (off-line) a faulty modulus n' such that the public exponent e and $\varphi(n')$ are coprime, and such that n' is a *possible* or even *plausible* faulty value of modulus n. To this aim, the adversary should use a fault model.

Furthermore, the attacker needs to compute efficiently the inverse of $e \bmod \varphi(n')$. This is possible when the factorization of n' is known. Once d', the inverse of $e \bmod \varphi(n')$ is computed, the attacker constructs a signature $s' = \mu^{d'} \bmod n'$.

This first operation, that consists in trying to find a n' satisfying an useful property and constructing an associated "faulty" signature, is done before the attack. Then, an *on-line* procedure is carried out: the attacker executes the signature verification algorithm with (s', μ) as input, and tries to inject a fault during this procedure in order to proceed computations modulo targeted n' instead of modulo n. Clearly, the probability of success, and so the average required number of faults, is dependent on the accuracy of the fault model and the capability for an attacker to produce an enough precise fault to be able to obtain the faulty modulus n' with non negligible probability.

3 Framework of our Extensions to Seifert's Attack

Seifert's attack succeeds in forging a signature that is accepted as valid, but does not reveal any information about the private key elements. Some unauthorized access can be granted but the RSA key itself is not broken.

In the sequel, extensions of Seifert's attack are presented. They let an attacker recover the private exponent d from several faulty computations when the modulus is altered before a standard RSA exponentiation.

3.1 General description and constraints of our attack

General methodology. Similarly to [13, 17], our fault attack consists in modifying the modulus before an RSA exponentiation. The operation $s = \mu^d \bmod n$ is targeted, and several faults are induced to collect faulty signatures from which the attacker learns the private exponent d.

Definition 1 (Fault campaign, Fault couples). *It is said that an attacker processes the* fault attack campaign *if she executes the exponentiation* $s = \mu^d \bmod n$ I *times, and corrupts these executions by changing the modulus* n *into unknown moduli* n'_i*, to obtain* fault couples $(\mu_i, s_i)_{1\leq i\leq I}$.

Paddings. A general constraint comes from the use of RSA in real life: it is folklore knowledge that one needs to use functions (called *paddings*, and denoted Λ) that reduces the malleability of the RSA prior to the exponentiation. Some of the paddings are *deterministic* — *i.e.,* $\mu = \Lambda(m)$—, others are *probabilistic* — *i.e.,* $\mu = \Lambda(m, r)$.[3] In the probabilistic case, the randomness can be either joint or self-recovered.

Because of redundancy checks of the paddings, after the decryption phase (*e.g.,* in RSA-OAEP), exploitation of fault attacks during decryption is generally not possible, and so decryption is out of scope of this paper. For signatures, fault attacks might be possible if μ_i are known to the attacker. It is the case when the padding is deterministic (*e.g.,* RSA-FDH) or if the randomness is joint with the signature (*e.g.,* RSA-PFDH). On the contrary, if the randomness is self-recovered from the signature (*e.g.,* RSA-PSS), then the faulty result does not allow recovering μ_i and our attack cannot be done.

From now, we suppose the attacker can compute bases μ_i used during the faulty exponentiations.

3.2 Dictionary of moduli

The literature is plenty of fault models (*cf.* Section 2.1) that would allow the adversary to guess how she could have modified the modulus n into n'_i during the faulty exponentiations. Once such a choice is made, the adversary is then able to construct a dictionary.

[3] The notations here are obvious: m is the message, and r is the randomness.

Definition 2 (Dictionary). *Depending on a fault model that the attacker might have experimented, the attacker may be able to establish* a priori *a list of possible values for the faulty moduli n'_i. Such a list is called a* dictionary *(of moduli).*

Whether a dictionary is available to the attacker governs which methods she may use to recover the private exponent d. As shown below, if an attacker has access to a dictionary, then the main part of her work is to learn which of the possible moduli of the dictionary was used for a given fault.

A dictionary is not necessarily mandatory and a first general method where no dictionary is needed is presented in the next section. This particularly implies that *no fault model* is required.

4 Recovering the Private Exponent Without Dictionary

This section describes a method to recover the private exponent d when the attacker has no clue about what value a faulty modulus may take. This corresponds to an attacker who is unable to predict or identify any fault model from the experimental setting of the attack. Note also that in the case where the attacker has actually identified a fault model and that the induced dictionary is too large to be practically handled (typically 2^{32} entries) the attacker may ignore this "useless" dictionary and place herself in the context of no dictionary as well.

For the sake of clarity, in the description of the different attacks, we denote by p's the (possible) divisors of n'_i, and by q's the (possible) divisors of the orders of considered subgroups. Of course, these integers are not to be confused with the unknown factorization of targeted modulus n.

4.1 General description of the attack

Once the fault campaign is performed, the attacker knows some fault couples $(\mu_i, s_i)_{1 \leq i \leq I}$, corresponding to unknown moduli $n'_i \neq n$, related by $s_i = \mu_i^d \bmod n'_i$. Input μ_i and output s_i are known to the attacker while n'_i is unknown and modeled as uniformly distributed over the integers less than 2^{ℓ_n}, where ℓ_n is the modulus bitsize.

From the data of the fault campaign, the private exponent d is retrieved off-line, by progressively determining $d \bmod r_k$, for some small prime powers r_k. When the product $R = \prod_k r_k$ exceeds the modulus n (and so unknown $\varphi(n)$), d can be recovered by means of the Chinese Remainder Theorem.

Improving the fraction of bits of d to know. If e is small (typically $e = 3$ or $e = 2^{16} + 1$), then the equation relating public and private RSA exponents

$$ed = 1 + k\varphi(n) = 1 + k(n + 1 - \alpha)$$

can be used in order to reduce the fraction of d's bits the attacker has to find to recover d. Here α is an unknown value, and k verifies $0 < k < e$. If k is known, or guessed by exhaustive search when e is small, we have

$$d = \frac{1 + k(n+1)}{e} - \frac{k\alpha}{e}$$

where the unknown part $k\alpha/e$ verifies (assuming balanced factorization of n):

$$\frac{k\alpha}{e} < \alpha < 2^{\lceil \frac{\ell_n}{2} + 1 \rceil} \quad .$$

Denoting $u = \lfloor \frac{\ell_n}{2} - 1 \rfloor$, d may be expressed as $d = \overline{d} 2^u + \underline{d}$, where $\overline{d}$ is known, and $0 < \underline{d} < 2^u$ is unknown. Knowledge of $d \bmod R$ implies knowledge about $\underline{d} \bmod R$, so that d may be retrieved as soon as R is $\lceil \frac{\ell_n}{2} + 1 \rceil$ bits long. Hence, in the following, for each attack, the two cases e small or e relatively large are considered. It is thus possible to see how much it reduces the number of faults required.

4.2 A useful proposition

Before detailing the off-line part, we state the following heuristics used hereafter.

Proposition 1. *Let (μ_i, s_i) be a fault couple corresponding to modulus n'_i, and p^a a prime power such that $p \nmid \mu_i$ and $p \nmid s_i$. Let also δ be the multiplicative order of μ_i modulo p^a. Then, for any r dividing δ we have:*

$$d \equiv DL(\mu_i, s_i, p^a) \pmod{r} \tag{1}$$

with probability 1 if $p^a \mid n'_i$, and probability close to $\frac{1}{r}$ otherwise.

Proof. By definition, $s_i = \mu_i^d \bmod n'_i$. Hence, when $p^a \mid n'_i$, we have:

$$\begin{aligned} s_i &\equiv \mu_i^{d \bmod \varphi(p^a)} \pmod{p^a} \\ &\equiv \mu_i^{d \bmod \delta} \pmod{p^a} \end{aligned}$$

so that $d \equiv DL(\mu_i, s_i, p^a) \pmod{\delta}$, from which Equation (1) follows.

On the contrary, when $p^a \nmid n'_i$, we admit that uniform distribution of n'_i over the integers implies quasi uniform distribution of $DL(\mu_i, s_i, p^a)$ over residue classes modulo r, hence the proposition. □

Of course, without knowing n'_i, it is impossible to decide which p^a can be used to determine $d \bmod r$ with certainty, for some divisors r of $\varphi(p^a)$. Nevertheless, Proposition 1 suggests that, even if n'_i is unknown (and so its factorization), one can mount an attack based upon a bias in favor of the true value d_r of the residue class of d modulo r.

4.3 The off-line phase

The basic idea is that determining d_r for some integer r, may be achieved by considering some p^a for which $r \mid \varphi(p^a)$, and by taking the discrete logarithm of s_i in base μ_i modulo p^a. From Proposition 1, and provided that r also divides the multiplicative order of μ_i modulo p^a, the probability distribution of $DL(\mu_i, s_i, p^a, r)$ is:

$$\Pr\left((DL(\mu_i, s_i, p^a, r)) = x)\right) = \begin{cases} \frac{1}{p^a} + \frac{p^a-1}{r\cdot p^a} & \text{if } x = d_r \\ \frac{p^a-1}{r\cdot p^a} & \text{if } x \neq d_r \end{cases}$$

By computing the value $DL(\mu_i, s_i, p^a, r)$ for all the fault couples of the fault campaign, and counting how many times each residue class is suggested, we expect that the correct value d_r emerges from the noise, and is suggested more often than others.

Note that the value of the bias

$$\varepsilon = \frac{\frac{1}{p^a} + \frac{p^a-1}{r\cdot p^a}}{\frac{p^a-1}{r\cdot p^a}} - 1 = \frac{r}{p^a - 1}$$

vanishes proportionally to $p^a - 1$. This means that given r, the smaller p^a, the larger the bias, and the smaller the number of faults needed to determine d_r. This suggests Algorithm 1 which, given r as input, tries to find the residue class d_r. Among all possible values of p^a such that $r \mid \varphi(p^a)$, this algorithm only considers the smallest prime p such that $r \mid p - 1$ as this choice gives the largest possible bias with high probability.

Algorithm 1 Predicting $d \bmod r$ by counting method

INPUT: $r = q^f$, a small power of a small prime
OUTPUT: A prediction for $d_r = d \bmod r$

Initialize an array `count`$[0, \ldots, r-1]$ to zero.

Phase 1: Search for the least prime p so that $r \mid p - 1$

$p \leftarrow 2r + 1$
while p is not prime
  $p \leftarrow p + r$

Phase 2: Compute $d_r = d \bmod r$ via the bias

for each fault couple (μ_i, s_i)
  if $p \nmid \mu_i$ and $p \nmid s_i$
    if $r \mid$ `order of` μ_i `modulo` p **and if** $DL(\mu_i, s_i, p, r)$ `exists`
      `count`$[DL(\mu_i, s_i, p, r)]$++
return d_r such that `count`$[d_r] = \max_i$ `count`$[i]$

Algorithm 1 leads to the knowledge of d_r for individual prime powers $r = q^f$. The attacker may integrate this building block into a higher level procedure which determines d_r for as much r values as needed so that $R = \prod_k r_k$ is large enough to fully recover d (or $\underline{d}$ when e is small).

4.4 Results

This counting method have been implemented. 512 bits of residue class information about d are easily recovered within 25 000 faults, which is enough for a 1024-bit key with small public exponent. About 60 000 faults allow to recover 1024 bits of information, which is enough for either a 1024-bit key in the general case, or a 2048-bit key with small public exponent.

5 Recovering the Private Exponent With a Dictionary

As already mentioned, no dictionary is needed for applying the method of Section 4. Nevertheless, when a dictionary S is available to the attacker, it is then possible to improve upon this counting method.

5.1 General methodology

The core observation is that, with a dictionary S, it becomes possible to relate a particular modulus $\nu_j \in S$ to some fault couple (μ_i, s_i). Let us thus introduce the following definition.

Definition 3 (Hit). *For any $\nu_j \in S$, we say that an attacker found a* hit *for ν_j if she was able to identify some fault couple (μ_i, s_i) for which $n'_i = \nu_j$.*

Given a hit in hand, a certain amount of information about d may be collected. Indeed, it is then possible to extract information related to each known p^a dividing ν_j as in Equation (1). One may then retrieve $d \bmod q^f$ for each q^f which divides the multiplicative order of μ_i modulo p^a.

We stress that the full factorization of ν_j is not needed since only some known factors of ν_j may be considered and exploited. The attack thus consists in identifying hits for a few moduli, and gathering information relative to known factors for each of them. This raises the question: how many hits provide enough information to recover the private exponent?

Table 1 shows some simulation results where the number of bits of information retrieved about d is given as a function of the number of hits exploited. These hit moduli were factorized by elliptic curve method up to 20–25 digits factors, and information was retrieved with respect to all q^f less than a given limit which took values 10^5, 10^7 and 10^9 respectively. Simulations have been conducted several times, and average over 200 experiments are presented below.

When the discrete logarithm computation limit is taken to 10^9, then 28 hits are enough to recover a 1024-bit RSA key (13 in the case of small public

Table 1. Amount of information (in bits) deduced from exploitation of hits

DL limit	Number of hits									
	1	2	3	4	5	6	7	8	9	10
10^5	33	62	87	111	136	159	182	206	227	248
10^7	41	75	113	150	184	219	251	285	315	346
10^9	47	93	135	177	214	255	296	334	374	412
	11	12	13	14	15	16	17	18	19	20
10^5	267	289	312	331	352	373	396	416	436	455
10^7	374	406	436	465	493	**522**	553	580	609	639
10^9	452	490	**526**	561	599	638	673	709	744	780
	21	22	23	24	25	26	27	28	29	30
10^5	477	496	**516**	537	533	570	587	602	618	635
10^7	667	695	723	751	778	807	833	861	890	916
10^9	815	849	881	917	953	988	1018	**1055**	1088	1124

exponent), and 59 hits allow to recover the private exponent of a 2048-bit RSA key (28 in the case of small public exponent).

Beside knowing how many hits are needed, we now present in the next subsections, two methods aiming at identifying them.

5.2 Finding hits by the collision method

Let r be an integer dividing the multiplicative order of μ_i modulo p^a. Proposition 1 implies that computing $DL(\mu_i, s_i, p^a, r)$ for different couples (μ_i, s_i) always gives the correct value of d_r, as soon as p^a divides n'_i. Otherwise, results are uniformly distributed between 0 and $r-1$.

This suggests a method which detects collisions like:

$$DL(\mu_{i_1}, s_{i_1}, p^a, r) = DL(\mu_{i_2}, s_{i_2}, p^a, r) \ .$$

For suitably chosen p^a and r values, with high probability, such a collision reveals, not only that p^a divides both n'_{i_1} and n'_{i_2}, but also that $n'_{i_1} = n'_{i_2} = \nu_j$ (see Remark 1 below). This is particularly useful to identify one hit for this common modulus.

Definition 4 (Marker). *For a given modulus $\nu \in S$, a couple (p, q) is called a* marker *for ν, if p is a known prime factor of ν, and q is a* not too small[4] *prime dividing $p-1$.*

Preparation phase. For as many moduli $\nu \in S$ as possible, we try to find a specific marker. The set of moduli for which a marker has been identified is denoted S^*.

[4] The fact that q should be *not too small* is required to avoid false positive in the collision search (*cf.* Remark 1).

Collision search phase. For each $\nu_j \in S^*$ with marker (p_j, q_j), we maintain a list $\mathcal{D}_{\nu_j}$ of all $DL(\mu_i, s_i, p_j, q_j)$ for all fault couples exploited so far. As soon as two fault couples have the same modulus value $\nu_j = n'_{i_1} = n'_{i_2}$, a collision is found in $\mathcal{D}_{\nu_j}$. By disregarding possible false positive, we can identify a hit for ν_j.

Complexity. In the ideal case where a marker has been found for all moduli in S (*i.e.,* $S^* = S$), the number of faults required to obtain such a collision is $\mathcal{O}(\sqrt{|S|})$. For small t, obtaining t hits requires $\mathcal{O}(\sqrt{t|S|})$ faults.

In the more practical case where only a fraction $\alpha = |S^*|/|S|$ of all possible moduli are affiliated with a marker, the number of faults required for obtaining t hits is $\mathcal{O}(\sqrt{\frac{t}{\alpha}|S|})$.

Remark 1 (False positives). For a given ν_j, a true collision appears in $\mathcal{D}_{\nu_j}$ after $2|S|$ faults on average, while a false collision appears after $\mathcal{O}(\sqrt{q_j})$ faults. Therefore, false positive occurrence problem may be neglected as soon as $\min_j \sqrt{q_j} \gg |S|$. This inequality explains the notion of *not too small* introduced in Definition 4.

Application. Concretely, assume an attacker targeting the transfer of the modulus from EEPROM to RAM, able to randomly modify any individual byte of the modulus, but unable to control which particular byte she is modifying. This fault model is very realistic when, as a counter-measure, the modulus bytes are transferred in random order. The corresponding dictionary contains $2^8 \cdot \frac{1024}{8} = 2^{15}$ (resp. 2^{16}) moduli for a 1024-bit (resp. 2048-bit) RSA key. Furthermore, assume that a marker has been found for 80% of the moduli. Referring to Table 1, retrieving a key in the general case requires about 1 100 faults (resp. about 2 200 faults for 2048-bit). When a small public exponent is used, only about 750 faults are needed for 1024-bit (resp. about 1 500 faults for 2048-bit). This demonstrates that even when applied to such a pretty large dictionary, this *square root* method allows to dramatically reduce the number of required faults compared to the case where no fault model is identified.

5.3 Finding hits by optimally exploiting faults

The objective of this method is to guess vectors of hits by optimally exploiting the information brought by fault couples.

We incrementally build lists Σ_t containing information provided by the faults (μ_i, s_i) for $1 \le i \le t$. Σ_{t+1} is built by combining previous Σ_t with next fault (μ_{t+1}, s_{t+1}), and by removing elements that are incompatible. In other words, for a given t, this method considers t faults $(\mu_i, s_i)_{1 \le i \le t}$ acquired during the fault campaign, and exhibits the set Σ_t of data that are compatible with the given t faults.

More precisely, Σ_t is a list of triples $(\boldsymbol{\nu}, \boldsymbol{\rho}, \boldsymbol{\sigma})$, where :

- The t-uple $\boldsymbol{\nu} = (\nu_{j_1}, \ldots, \nu_{j_t})$ represents possible values taken by the faulty moduli corresponding to the considered t faults;
- The residue knowledge about d, $\boldsymbol{\rho}$, is a collection of triples (q, f, α_{q^f}), each meaning that $d \equiv \alpha_{q^f} \pmod{q^f}$, provided that $\boldsymbol{\nu}$ is the correct guess for the vector $(n'_1, \ldots, n'_t)$, *i.e.,* each ν_{j_i} is the correct modulus corresponding to i-th fault;
- The selectivity $\boldsymbol{\sigma}$ associated to $\boldsymbol{\nu}$ and $\boldsymbol{\rho}$ is a scalar allowing to quantify the relative likelihood of this particular $\boldsymbol{\nu}$.

Below, we detail this method.

Initial phase. Given (μ, s), not all $\nu \in S$ are compatible with this fault. Indeed, ν must simultaneously verify several conditions:

1. The signature s must be smaller than the modulus candidate ν.
2. For each p dividing ν, either ($p \mid \mu$ and $p \mid s$) or ($p \nmid \mu$ and $p \nmid s$)
3. For each p^a dividing ν, denoting $\delta(\mu)$ and $\delta(s)$ the multiplicative orders modulo p^a of μ and s respectively, we must have $\delta(s) \mid \delta(\mu)$.
4. If $q^f \mid \varphi(p^a)$ and $q^f \mid \varphi(p'^{a'})$, where both p^a and $p'^{a'}$ divide ν, then if $DL(\mu, s, p^a, q^f)$ and $DL(\mu, s, p'^{a'}, q^f)$ both exist, their respective values must be equal.

This first phase hence consists, for every fault, in reducing, from S to $S_{(\mu,s)} \subseteq S$, the set of all moduli in the dictionary which are compatible with that fault. Note that this reduction is quite selective as — on average in our simulations — only a mere 3% of the moduli verify all four conditions.

In the list $S_{(\mu,s)}$, we associate to each modulus ν, the set ρ of all triples (q, f, α_{q^f}) with $\alpha_{q^f} = DL(\mu, s, p^a, q^f)$, for q of reasonable size, where $p^a \mid \nu$. Such a α_{q^f} value is always uniquely determined since all incompatible moduli (w.r.t. condition 4) have been removed from $S_{(\mu,s)}$. Furthermore, for each modulus, we also compute a selectivity parameter $\sigma = \tilde{\nu}/\delta(\mu)$, where $\tilde{\nu}$ is the factored part of ν, and $\delta(\mu)$ is the multiplicative order of μ modulo $\tilde{\nu}$ (*i.e.,* the product of all q^f used in the DL computations). Doing this for all faults allows to compute I different potential initial sets Σ_1. We then choose one of them for initiating our process.

Combining faults. Once we have extracted as much information as possible from each individual fault, we start a phase of combining these pieces of information. We use an iterative approach where we combine information from the list Σ_t with information brought by the $(t+1)$-th fault to update the data structure into a new Σ_{t+1}.

For this purpose, we exhaust all $(\boldsymbol{\nu}, \boldsymbol{\rho}, \boldsymbol{\sigma})$ of Σ_t, and all (ν, ρ, σ) where moduli ν belong to $S_{(\mu,s)}$. We consider combinations of each $(\boldsymbol{\nu}, \boldsymbol{\rho}, \boldsymbol{\sigma})$ with each (ν, ρ, σ). Each such combination results in a new triple $(f(\boldsymbol{\nu}, \nu), g(\boldsymbol{\rho}, \rho), h(\boldsymbol{\sigma}, \sigma))$, which will be kept and added to Σ_{t+1} only if evaluation of $g(\boldsymbol{\rho}, \rho)$ does not lead to any inconsistency (see below).

The new guess of moduli, $f(\boldsymbol{\nu}, \nu)$ trivially consists in appending ν to $\boldsymbol{\nu}$. That is, $f(\boldsymbol{\nu}, \nu) = (\nu_{j_1}, \ldots, \nu_{j_t}, \nu_{j_{t+1}})$ where $\nu_{j_{t+1}} = \nu$.

The new residue knowledge on d, $g(\boldsymbol{\rho}, \rho)$, consists in the union of $\boldsymbol{\rho}$ with ρ. If two triples $(q, f, d_{q^f}) \in \boldsymbol{\rho}$ and $(q, f', d_{q^{f'}}) \in \rho$ share the same prime q, then only the one with the largest exponent $\max(f, f')$ is kept. Moreover, in this case, the compatibility between both constraints modulo q must be checked. That is (assuming w.l.o.g. that $\max(f, f') = f$), $d_{q^f} \bmod q^{f'}$ must be equal to $d_{q^{f'}}$. If this consistency is not verified, then that particular combination of $(\boldsymbol{\nu}, \boldsymbol{\rho}, \boldsymbol{\sigma})$ with (ν, ρ, σ) is not kept.

The new selectivity $h(\boldsymbol{\sigma}, \sigma)$ takes the value $\boldsymbol{\sigma} \cdot \sigma \cdot \kappa$, where the multiplication by σ accounts for the selectivity of ν, and multiplication by κ accounts for a cross-selectivity between $\boldsymbol{\nu}$ and ν. This cross-selectivity factor is the product of moduli in the intersection of $\boldsymbol{\rho}$ and ρ, that is $\kappa = \prod_q q^{\min(f,f')}$. Of course, in this formula, if q is not in $\boldsymbol{\rho}$ (resp. in ρ), we set the corresponding exponent f (resp. f') to 0.

Final phase. Now that we have combined information from a set of faults, we get a (possibly large) list of modular information about the private exponent d, each associated to a likelihood/selectivity parameter $\boldsymbol{\sigma}$. We can sort that list according to this last parameter, and, for each entry, check the value of d recovered by applying the Chinese Remainder Theorem on $\boldsymbol{\rho}$, until we get the correct one. Note that if the residue knowledge $\boldsymbol{\rho}$ corresponding to some entry does not allow to unambiguously determine d, then one or several more faults must be exploited again, and combined with Σ. As the correct guess $\boldsymbol{\nu}$ about $(n'_1, \ldots, n'_t)$ necessarily belongs to Σ_t, this algorithm must eventually succeed in recovering d for some value t.

Remark 2. According to the size of the dictionary, handling these lists may become intractable. In this case, one can choose to keep only track of a fraction of the list, eliminating triples $(\boldsymbol{\nu}, \boldsymbol{\rho}, \boldsymbol{\sigma})$ with the lowest selectivity. The parameter $\boldsymbol{\sigma}$ is in strong connection with an *a posteriori* probability of the guess $\boldsymbol{\nu}$ about faulty moduli. Practical implementation and tests we performed show undoubtedly that a strategy based on $\boldsymbol{\sigma}$ is efficient.

Of course, a drawback of this idea is that one might remove the correct combination of moduli from the list, and so this could lead to an unsuccessful end of the algorithm. This may be the price to pay for shortening the list to a manageable size.

Results. This method aims at determining the list of moduli vectors $\boldsymbol{\nu}$ compatible with a given set of faults. Necessarily, it always succeeds in proposing the correct guess for $\boldsymbol{\nu}$,[5] leading to the identification of t hits with only t faults, which is obviously *optimal in terms of required number of faults.* Of course, the important question is whether the correct vector appears near the top of the

[5] If the trick discussed in Remark 2 is not used.

sorted list. If so, d is retrieved within only a few trials. Otherwise, the exhaustive search for the correct guess on $(n'_1, \ldots, n'_t)$ may be out of reach, or this vector may have been dropped if the decimation process suggested by Remark 2 was implemented.

With a pretty well factorized dictionary of 1 000 moduli, we experimented that this method allows to recover d with little computational effort in most cases, with as few faults as required according to Table 1. We expect that similar results may be obtained with moderate effort in the case of a dictionary of 10 000 moduli.

6 Conclusion

In this paper, we have proposed the first fault attack that can be realized against RSA in standard mode, to recover the private exponent by corrupting only *public key elements*. Our contribution can, in this sense, be viewed as a generalization of Seifert's and Muir's recent articles on obtaining a false signature acceptance by corrupting the modulus. However, this latter kind of attack only allows to pass a signature verification, while ours allows a *full key recovery*.

Our attack is divided into two modes. In the first one, the attacker needs *absolutely no knowledge* of the fault's behavior to recover the private exponent. This attack is also very attractive from a practical point of view, and represents, to our knowledge, the only known fault attack on RSA in standard mode requiring no fault model. The second mode, based on a fault model, has been proved to be particularly efficient. It dramatically reduces the number of faults needed to fully recover the private key. For this technique to work, the attacker does not need to be particularly powerful in the sense that she does not have to master the fault's exact effect. The fault she produces may be probabilistic or unprecise. Two variants have been proposed, with separate pros and cons and use cases.

There still are so open issues like whether our attacks can be adapted in the case of randomized exponent, or whether one could tackle with a probabilistic padding scheme with randomness recovery such as RSA-PSS.

Nevertheless, this paper teaches us that, as in the case of elliptic curves [5,8], *one should also protect RSA public key elements* against fault attacks.

Acknowledgements

The authors would like to thank the anonymous referees for their useful remarks and Marc Joye and Jacques Fournier for their careful reading of this paper.

The work described in this document has been financially supported by the European Commission through the IST Program under Contract IST-2002-507932 ECRYPT.

References

1. C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, J.-P. Seifert. Fault attacks on RSA with CRT: Concrete results and practical countermeasures. In *CHES 2002*, volume 2523 of *LNCS*, pages 260-275
2. H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The sorcerer's apprentice guide to fault attacks. In *Workshop on Fault Detection and Tolerance in Cryptography*, 2004
3. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *1st ACM Conference on Computer and Communications Security*, pages 62–73. ACM Press, 1993.
4. M. Bellare and P. Rogaway. The exact security of digital signatures - How to sign with RSA and Rabin. In *Advances in Cryptology – EUROCRYPT '96*, volume 1070 of *LNCS*, pages 399–416. Springer, 1996.
5. I. Biehl, B. Meyer, and V. Müller. Differential fault analysis on elliptic curve cryptosystems. In *Advances in Cryptology – CRYPTO 2000*, vol. 1880 of *LNCS*, pages 131–146. Springer, 2000.
6. D. Boneh, R.A. DeMillo, and R.J. Lipton. On the importance of checking cryptographic protocols for faults. In W. Fumy, editor, *Advances in Cryptology − EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer-Verlag, 1997.
7. 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. An earlier version appears in [6].
8. M. Ciet and M. Joye. Elliptic curve cryptosystem in presence of permanent and transient faults. *Designs Codes and Cryptography* **36**(1), 2005.
9. J.-S. Coron. Optimal security proofs for PSS and other signature schemes. In *Advances in Cryptology – EUROCRYPT '02*, volume 2332 of *LNCS*, pages 272–287. Springer, 2002.
10. M. Joye, A.K. Lenstra, and J.-J. Quisquater. Chinese remaindering based cryptosystems in the presence of faults. *Journal of Cryptology* **12**(4):241–245, 1999.
11. P.C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. Wiener, editor, *Advances in Cryptology − CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer-Verlag, 1999.
12. A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. Handbook of applied cryptography. CRC Press, 1997.
13. J.A. Muir. Seiferts RSA fault attack: Simplified analysis and generalizations. IACR Eprint archive 2005.
14. PKCS #1 v 1.5: RSA Cryptography Standard.
15. J.-J. Quisquater and C. Couvreur. Fast decipherment algorithm for RSA public-key cryptosystem. *Electronics Letters* **18**(21):905–907, 1982.
16. R.L. Rivest, A. Shamir, and L.M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM* **21**(2):120–126, 1978.
17. J.-P. Seifert. On authenticated computing and RSA-based authentication. *ACM Conference on Computer and Communications Security 2005*: pages 122–127, 2005.