On Second-Order Fault Analysis Resistance for CRT-RSA Implementations

Emmanuelle Dottax[1], Christophe Giraud[2], Matthieu Rivain[1,3], and Yannick Sierra[1]

[1] Oberthur Technologies,
71-73, rue des Hautes Pâtures, 92 726, Nanterre, France.
[2] Oberthur Technologies,
4, allée du doyen Georges Brus, 33 600, Pessac, France.
[3] University of Luxembourg,
Faculty of Sciences, Technology and Communication
6, rue Richard Coudenhove-Kalergi, L-1359 Luxembourg.
{e.dottax,c.giraud,m.rivain,y.sierra}@oberthur.com

Abstract. Since their publication in 1996, Fault Attacks have been widely studied from both theoretical and practical points of view and most of cryptographic systems have been shown vulnerable to this kind of attacks. Until recently, most of the theoretical fault attacks and countermeasures used a fault model which assumes that the attacker is able to disturb the execution of a cryptographic algorithm only once. However, this approach seems too restrictive since the publication in 2007 of the successful experiment of an attack based on the injection of two faults, namely a second-order fault attack. Amongst the few papers dealing with second-order fault analysis, three countermeasures were published at WISTP'07 and FDTC'07 to protect the RSA cryptosystem using the CRT mode. In this paper, we analyse the security of these countermeasures with respect to the second-order fault model considered by their authors. We show that these countermeasures are not intrinsically resistant and we propose a new method allowing us to implement a CRT-RSA that resists to this kind of second-order fault attack.

Keywords: Fault Attacks, Second Order, CRT-RSA, Countermeasure.

1 Introduction

When attackers have access to physical implementations, cryptographic algorithms need to be secured against threats beyond classical cryptanalysis. Among these, faults attacks (FA) aim at disturbing cryptographic computations, and exploit erroneous results to recover information on secret data. They were introduced in 1996 [5] and they have been further studied since. Techniques have been improved and they have found applications on a wide variety of cryptographic algorithms. A non exhaustive list comprises RSA [1, 5, 16, 17, 19, 25], DSA [22], DES [3, 14], AES [23] and stream ciphers [15].

A fault attack description must specify the *fault model* it assumes [13]. This model clarifies the capabilities of the attacker such as the kind of error (*e.g.* bit flip in data, program execution modification), the timing precision or the number of errors. The latter characteristic is called the *order* of the attack: *first-order* attacks assume an attacker who can induce only one error per execution of the target algorithm. Similarly, *second-order* attacks assume an attacker who can induce two errors per execution, and so forth. The practicability of the model is of importance to assess the feasibility of an attack.

The seminal work [5] introduces several first-order attacks among which one targets an RSA implementation using the Chinese Remainder Theorem (CRT for short). Indeed, most RSA implementations in embedded systems use CRT because of its performance benefits. Let N denote the public modulus composed of two secret prime numbers p and q such that $N = p \cdot q$. Let e refer to the public exponent and d refer to the private exponent. Whereas a straightforward implementation computes the signature of a message m by performing $S = m^d \bmod N$, a CRT-based implementation is composed of two exponentiations $S_p = m^{d_p} \bmod p$ and $S_q = m^{d_q} \bmod q$, where $d_p = d \bmod (p-1)$ and $d_q = d \bmod (q-1)$. As the signature S satisfies $S \equiv S_p \bmod p$ and $S \equiv S_q \bmod q$, it can be computed from S_p and S_q by using the CRT [9]. This additional computation is called the *recombination* step. The principle of the so-called Bellcore attack [5] is to disturb one of the exponentiations, say S_q, so that the recombination step results in a faulty signature $\widetilde{S}$ satisfying $\widetilde{S} \equiv S \bmod p$ and $\widetilde{S} \not\equiv S \bmod q$. The secret parameter p can then be recovered by computing $\gcd(S - \widetilde{S}, N)$. The fault model of this attack is very weak because the attacker only needs to disturb one exponentiation to succeed. The fault can be introduced at any time during the computation, either in code or in data.

Due to both the performance advantages of the CRT-RSA and its high vulnerability to fault attacks, securing its implementation is an important and challenging task. Several countermeasures have been introduced of which common goal is to prevent the output of a hazardous result. To do so, a simple solution is to verify the signature before returning it. However, this verification may be costly if the public exponent is not small or/and if it is not provided by the API (*e.g.* in Javacard). Therefore, more sophisticated methods have been designed which do not rely on additional parameters. We can distinguish two main approaches. The first one is the introduction of internal coherence checks that aim at verifying the validity of the result before returning it [6, 10, 11]. The second one is to make use of so-called *infective procedures* that render the erroneous signature harmless in case of fault injection [8].

At WISTP'07, Kim and Quisquater [19] introduced a second-order fault model in which they were able to practically break the (first-order) countermeasures of [8] and [10]. Their work also includes improvements of the first-order countermeasures to achieve a secure implementation in their second-order fault model. Later, at FDTC'07, they proposed in [20] an implementation also meant to resist in this model. Unfortunately, we show in this paper that the proposed

countermeasures are actually not intrinsically resistant in this model and that they may be successfully attacked.

The rest of this paper is organized as follows. Section 2 recalls the fault model introduced by Kim and Quisquater. Sections 3 and 4 present the countermeasures proposed in [19] and [20] respectively and exhibit their vulnerabilities. Section 5 proposes a generic countermeasure that enables to render any method intrinsically resistant in the Kim and Quisquater's fault model. Finally, Section 6 concludes the paper.

2 Kim and Quisquater's Second-Order Fault Model

The literature contains several practical examples of fault attacks. However, to the best of our knowledge, the paper [19] is the first one that reports a successful experiment of a second-order fault attack. In this paper, Kim and Quisquater explain how they practically mounted a second-order attack against a first-order resistant CRT-RSA implementation. We shall refer to this attack as the *KQ-attack*.

An algorithm is said to be first-order resistant when it contains some countermeasure which ensures that any single error occurring in the execution of the algorithm is not exploitable by an attacker. In the case of the CRT-RSA, some redundant computations are usually added in order to check the coherence of the computation (*e.g.* [10]) or to infect the faulty signature (*e.g.* [7]). When a single fault is injected and corrupts the RSA computation, it is systematically detected (or at least with high probability) and the faulty signature is not revealed (or is *infected*). However, an attacker may defeat such a countermeasure by using a second-order fault analysis, namely by injecting two faults. In that case, one of these faults must be dedicated to the corruption of the RSA computation in order to produce an exploitable faulty signature. The second fault is then used to render the countermeasure ineffective. For such a purpose, two approaches are possible:

- In the first approach, the attacker tries to fool the coherence check. Namely, the second fault aims at covering the effects of the first fault in such a way that the coherence check does not detect it while the result of the RSA computation remains faulty. To do so, the attacker needs to precisely control the two fault injections effects. This implies a very strong and yet not practical adversary model.
- The second approach consists in directly skipping the coherence verification (or the infection procedure). Since several experiments have demonstrated the practicability of skipping the execution of one or more operations (see for instance [2,19,21]), this approach corresponds to a weaker model of adversary and is then more natural from a practical point of view.

The Kim and Quisquater's second-order fault model (referred in the sequel as *KQ-model*) corresponds to this latter approach. We formalize it hereafter.

Fault Model (KQ-model). *Two faults are injected. The first fault can be of any type (instructions skip, memory modification, ...), provided it disturbs the RSA computation and produces an exploitable faulty signature. The second fault allows the attacker to skip any operation or set of contiguous operations in order to circumvent the coherence check (or the infective procedure).*

In the next section, we show that the countermeasures proposed by Kim and Quisquater in [19] and [20] do not intrinsically resist all fault attacks in their model.

3 Analysis of WITSP'07 Countermeasures

In a paper published at WISTP'07, Kim and Quisquater have reported some practical experiments of second-order fault analysis [19]. They succeeded in breaking two first-order FA-countermeasures published at FDTC'05: the one by Ciet and Joye [8] and the one by Giraud [10]. Finally, they proposed to modify both countermeasures in order to resist to their second-order fault attack. In this section, we present the two countermeasures improved by Kim and Quisquater in [19] which are meant to resist fault analysis in the KQ-model and we show that they are weak with regard to this fault model.

3.1 Analysis of the First WISTP Countermeasure

Description. This first countermeasure of [19] is an improvement of Ciet and Joye's countermeasure [8] which is a generalization of Shamir's trick [24]. The basic principle is to multiply the modulus p (resp. q) by a small integer r_1 (resp. r_2). The exponentiation is then carried out modulo $r_1 \cdot p$ (resp. $r_2 \cdot q$) which allows to verify the result modulo r_1 (resp. r_2) afterward. If an error is detected, the resulting signature is infected using a random value r_3.

For two security parameters k and l, let r_1 and r_2 as two co-prime k-bit integers and r_3 as an l-bit integer. And let p^*, q^* and I_q^* be defined as $p^* = r_1 \cdot p$, $q^* = r_2 \cdot q$ and $I_q^* = (q^*)^{-1} \bmod p^*$. Algorithm 1 describes the modified Ciet and Joye's scheme.

Algorithm 1 Modified Ciet and Joye's scheme [19]

INPUTS: m, p^*, q^*, d_p, d_q, I_q^*, N, r_1, r_2, r_3
OUTPUT: $S = m^d \bmod N$

1. Choose a random integer a in $\mathbb{Z}^*_{r_1 r_2 N}$
2. Initialize γ with a random number
3. $S_p^* \leftarrow (a + m^{d_p}) \bmod p^*$
4. $s_2 \leftarrow (a + m^{d_q \bmod \varphi(r_2)}) \bmod r_2$
5. $S_q^* \leftarrow (a + m^{d_q}) \bmod q^*$
6. $s_1 \leftarrow (a + m^{d_p \bmod \varphi(r_1)}) \bmod r_1$
7. $S^* \leftarrow S_q^* + q^* \cdot I_q^* \cdot (S_p^* - S_q^*) \bmod p^*$

8. $c_1 \leftarrow (S^* - s_1 + 1) \bmod r_1$
9. $c_2 \leftarrow (S^* - s_2 + 1) \bmod r_2$
10. $\gamma \leftarrow \lfloor (r_3 c_1 + (2^l - r_3) c_2) / 2^l \rfloor$
11. **return** $(S^* - a^\gamma) \bmod N$

If no fault is injected, the result of Step 7 is $S + a \bmod N$ and the result of Step 10 is $\gamma = 1$ (since $s_1 = s_2 = 1$). This way, Step 11 returns the correct signature. On the other hand, if an error is detected, Step 10 returns $\gamma \neq 1$ and Step 11 infects the returned signature with the random value a (see [19] for further details).

Analysis. By noticing that $S^* \equiv S_p + a \bmod p$ and $S^* \equiv S_q + a \bmod q$, we observe that the Bellcore attack can be successfully applied to the output of Algorithm 1 if a is always subtracted to S^*, whatever the fault induced on either S_p^* or S_q^*. Indeed, if an attacker disturbs the computation of, say, S_p^* in Step 3 then a faulty value $\widetilde{S^*}$ is obtained at the beginning of Step 11 and it satisfies: $\widetilde{S^*} \not\equiv S_p + a \bmod p$ and $\widetilde{S^*} \equiv S_q + a \bmod q$. Secondly, if he corrupts the exponentiation $(a, \gamma) \mapsto a^\gamma$ in such a way that it outputs a, Algorithm 1 returns $\widetilde{S}^* - a \bmod N$. The Bellcore attack can then be successfully applied.

The feasibility of this attack depends on implementation details of Step 11 of Algorithm 1. However, we show that a straightforward implementation is vulnerable. We denote by R_0 the register containing S^* and by R_1 the register containing a. A straightforward implementation of Step 11 performs the following operations:

11.1. $R_1 \leftarrow (R_1)^\gamma \bmod N$	$[R_1 = a^\gamma \bmod N]$
11.2. $R_0 \leftarrow R_0 - R_1 \bmod N$	$[R_0 = S^* - a^\gamma \bmod N]$
11.3. **return** R_0	

Such an implementation makes Algorithm 1 insecure in the KQ-model. Indeed, the attacker can skip the operation $R_1 \leftarrow (R_1)^\gamma \bmod N$, which results in the faulty output $\widetilde{S} = \widetilde{S^*} - a \bmod N$.

This example demonstrates that the proposed improvement of the Ciet and Joye countermeasure is not intrinsically secure in the intended KQ-model. Furthermore, this example is particurlarly relevant, as the same one is used by Kim and Quisquater in [19] to break the original countermeasure.

3.2 Analysis of the Second WISTP Countermeasure

Description. The second countermeasure is an improvement of Giraud's scheme [10, 11] that is based on the Montgomery powering ladder. This exponentiation algorithm works on a pair of intermediate results of the form $(m^{\delta-1}, m^\delta)$ which allows to check the coherence of the result.

In the following, $\mathrm{CRT}(S_p, S_q)$ denotes Garner's recombination: $((S_p - S_q) \cdot (q^{-1} \bmod p) \bmod p) \cdot q + S_q$. The improvement of Giraud's scheme uses a modified

exponentiation algorithm that is described in Algorithm 2. The whole modified Giraud's scheme is described in Algorithm 3.

Algorithm 2 Modified SPA-FA-resistant modular exponentiation [19]

INPUTS: m, $d = (1, d_{n-2}, \cdots, d_0)_2$, N, a
OUTPUT: $(a + m^{d-1} \bmod N, a + m^d \bmod N)$

1. $a_0 \leftarrow m$
2. $a_1 \leftarrow m^2 \bmod N$
3. **for** i **from** $n-2$ **to** 1 **do**
   $a_{\overline{d_i}} \leftarrow a_{\overline{d_i}} \cdot a_{d_i} \bmod N$
   $a_{d_i} \leftarrow {a_{d_i}}^2 \bmod N$
4. $a_1 \leftarrow (a + a_1 \cdot a_0) \bmod N$
5. $a_0 \leftarrow (a + {a_0}^2) \bmod N$
6. **if** (Loop Counter i not modified) & (Exponent d not modified) **then**
   return (a_0, a_1),
   else
   return error.

Algorithm 3 Modified Giraud's scheme [19]

INPUTS: m, p, q, d_p, d_q
OUTPUT: $S = m^d \bmod N$

1. Initialize a with a random number in $\mathbb{Z}_N^*$
2. $(S_p^*, S_p) \leftarrow$ Algo. $2(m, d_p, p, a)$
3. $(S_q^*, S_q) \leftarrow$ Algo. $2(m, d_q, q, a)$
4. $S^* \leftarrow \mathrm{CRT}(S_p^*, S_q^*)$
5. $S \leftarrow \mathrm{CRT}(S_p, S_q)$
6. $S^* \leftarrow m \cdot S^* + a \bmod (p \cdot q)$
7. $S \leftarrow S + a \cdot m \bmod (p \cdot q)$
8. **if** $(S^* = S)$ & (Parameters p and q not modified) **then**
   return $(S - a - a \cdot m) \bmod (p \cdot q)$
   else
   return error.

If a fault is injected, then it breaks the coherence between S and S^*, and the test in Step 8 returns an error. Otherwise, the correct signature is returned (see [19] for further details).

Analysis. We observe that if the test $(S^* = S)$ is skipped then the algorithm execution carries on as if the test had been successfully performed. Therefore, if a first fault induces a faulty signature $\widetilde{S}$ and a second one skips the test $(S^* = \widetilde{S})$, an attacker is able to make Algorithm 3 return a faulty signature

$(\widetilde{S} - a - a \cdot m) \bmod N$ which enables the Bellcore Attack. The faulty signature $\widetilde{S}$ can be obtained by disturbing either Step 2 or Step 3 of Algorithm 3 without modifying the values that are checked (*e.g.* by disturbing the value a_1 at Step 4 of Algorithm 2), or by disturbing Step 5 of Algorithm 3.

Note that such an attack has been successfully put into practice by Kim and Quisquater in [19]. This shows that the proposed countermeasure is vulnerable in their model and therefore does not reach its goal.

4 Analysis of the FDTC'07 Countermeasures

At FDTC'07, Kim and Quisquater proposed an implementation of CRT-RSA which is meant to resist both side channel analysis and fault analysis [20]. In particular, they claimed the security of their proposal versus the KQ-model. However, we show in this section that their countermeasure is not intrinsically resistant in this model.

We analyse two variants of this countermeasure. The first one is the original countermeasure which has been published in FDTC'07 proceedings. The second one is a patched version that has been presented at the workshop.

4.1 Analysis of the Original FDTC Countermeasure

Description. Similarly to Ciet and Joye's scheme, the implementation proposed by Kim and Quisquater is based on Shamir's trick. For two co-prime k-bit integers t_1 and t_2, let p^* and q^* be defined as $p^* = p \cdot t_1$ and $q^* = q \cdot t_2$. The following values are then computed and stored in the device: $d_p^* = d \bmod \varphi(p^*)$, $d_q^* = d \bmod \varphi(q^*)$, $e_{t_i} = d^{-1} \bmod \varphi(t_i)$, where $i = 1, 2$. Algorithm 4 describes the exponentiation algorithm which is used to compute the two CRT components S_p^* and S_q^* while Algorithm 5 describes the whole protected CRT-RSA.

Algorithm 4 FA-DPA-resistant modular exponentiation [20]

INPUTS: m, $d = (d_{n-1}, \cdots, d_0)_2$, N, a, $r = a^{-1} \bmod N$
OUTPUT: $C = m^d \bmod N$

1. $C \leftarrow r \bmod N$
2. $a_0 \leftarrow a$
3. $a_1 \leftarrow m \cdot a \bmod N$
4. **for** i **from** $n-1$ **to** 0 **do**
   $C \leftarrow C^2 \bmod N$
   $C \leftarrow C \cdot a_{d_i} \bmod N$
5. **return** C

Algorithm 5 FA-DPA-resistant CRT-RSA [20]

INPUTS: m, p^*, q^*, d_p^*, d_q^*, N, e_{t_1}, e_{t_2}, t_1, t_2
OUTPUT: $S = m^d \bmod N$

1. Select a random r in $\mathbb{Z}^*_{p^*q^*}$ and compute $a = r^{-1} \bmod p^*q^*$
2. Initialize c_1 and c_2 with random values
3. $S_p^* \leftarrow$ Algo. $4(m, d_p^*, p^*, a, r)$
4. $S_q^* \leftarrow$ Algo. $4(m, d_q^*, q^*, a, r)$
5. $S^* \leftarrow \mathrm{CRT}(S_p^*, S_q^*)$
6. $c_1 \leftarrow (m \cdot r^{e_{t_1}} - (S^*)^{e_{t_1}} + 1) \bmod t_1$
7. $c_2 \leftarrow (m \cdot r^{e_{t_2}} - (S^*)^{e_{t_2}} + 1) \bmod t_2$
8. **return** $S = S^* \cdot a^{c_1 \cdot c_2} \bmod N$

If no fault is injected during the signature computation (*i.e.* during Steps 3 to 5) then the result of Step 5 is $S \cdot r \bmod N$ and the results of Steps 6 and 7 are $c_1 = c_2 = 1$. This way Step 8 returns the correct signature. If an error is detected, then we have either $c_1 \neq 1$ or $c_2 \neq 1$ and Step 8 returns a signature infected by the random value a (see [19] for further details).

Analysis. The principle of the attack described below is similar to the one described in Section 3.1. By noticing that we have $S^* \equiv S_p \cdot r \bmod p$ and $S^* \equiv S_q \cdot r \bmod q$, we observe that the Bellcore attack can be successfully applied to the output of Algorithm 5 if S^* is always multiplied by a in Step 8 whatever the fault induced on either S_p^* or S_q^*.

Let us examine the feasibility of this attack according to the implementation details of Step 8 of Algorithm 5. We denote by R_0 the register containing S^* and by R_1 the register containing a. A straightforward implementation of Step 8 performs the following operations:

8.1. $R_1 \leftarrow (R_1)^{c_1 \cdot c_2} \bmod N$	$[R_1 = a^{c_1 \cdot c_2} \bmod N]$
8.2. $R_0 \leftarrow R_0 \cdot R_1 \bmod N$	$[R_0 = S^* \cdot a^{c_1 \cdot c_2} \bmod N]$
8.3. **return** R_0	

If an attacker corrupts one of the two exponentiations (Step 3 or Step 4) and then skips the operation $R_1 \leftarrow (R_1)^{c_1 \cdot c_2} \bmod N$ in Step 8, then Step 8 returns a faulty value of S^* multiplied by a which enables the Bellcore attack.

Here again, this example demonstrates a weakness of the proposed countermeasure with respect to the KQ-model. The authors had been informed of this attack after the printing of the FDTC'07 proceedings, but before the workshop [12]. Therefore, they have patched their countermeasure and presented the following improved version to the workshop.

4.2 Analysis of the Improved FDTC Countermeasure

Description. Let α be a small random value (less than one byte) and let β be defined as $\beta = \alpha^{-1} \bmod \varphi(N)$. Algorithm 6 describes the patched version of the Kim and Quisquater countermeasure presented at FDTC'07.

Algorithm 6 Improved FA-DPA-resistant CRT-RSA

INPUTS: m, p^*, q^*, d_p^*, d_q^*, N, e_{t_1}, e_{t_2}, t_1, t_2, α, β
OUTPUT: $S = m^d \bmod N$

1. Select a random r in $\mathbb{Z}_N^*$ and compute $a = r^{-1} \bmod N^*$
2. Initialize c_1 and c_2 with random values
3. $b \leftarrow a^\beta \bmod N$
4. $S_p^* \leftarrow$ Algo. 4(m, d_p, p^*, a, r)
5. $S_q^* \leftarrow$ Algo. 4(m, d_q, q^*, a, r)
6. $S^* \leftarrow \mathrm{CRT}(S_p^*, S_q^*)$
7. $c_1 \leftarrow (m \cdot r^{e_{t_1}} - (S^*)^{e_{t_1}} + 1) \bmod t_1$
8. $c_2 \leftarrow (m \cdot r^{e_{t_2}} - (S^*)^{e_{t_2}} + 1) \bmod t_2$
9. **return** $S^* \cdot b^{c_1 \cdot c_2 \cdot \alpha} \bmod N$

Analysis. By noticing that $S^* \equiv S_p \cdot b^\beta \bmod p$ and $S^* \equiv S_q \cdot b^\beta \bmod q$, we observe that the Bellcore attack can be successfully applied to the output of Algorithm 5 if S^* is always multiplied by b^α whatever the fault induced on either S_p^* or S_q^*. Therefore, by disturbing an exponentiation computing either S_p^* or S_q^* and by skipping the multiplication $(c_1, c_2, \alpha) \mapsto c_1 \cdot c_2 \cdot \alpha$ in such a way that it returns α, the attacker obtains a faulty signature $\widetilde{S} = \widetilde{S^*} \cdot b^\alpha \bmod N$ allowing him to recover the secret CRT parameters since $\widetilde{S}$ is congruent to S_q modulo q (resp. to S_p modulo p) but not to S_p modulo p (resp. to S_q modulo q).

The effectiveness of such an attack depends on the implementation of the multiplication. We can distinguish three different cases depending on the result location:

1. $c_1 \leftarrow c_1 \cdot c_2$,
2. $c_2 \leftarrow c_1 \cdot c_2$,
3. $temp \leftarrow c_1 \cdot c_2$.

For the two first cases, if the attacker skips the multiplication $c_1 \cdot c_2$ then the value of c_1 (resp. c_2) remains unchanged. Therefore, if case 1 (resp. case 2) is used to implement the multiplication, the attacker must disturb the computation of S_q (resp. S_p) and skip the multiplication $c_1 \cdot c_2$. As $c_1 = 1$ (resp. $c_2 = 1$) since no error has been induced on S_p (resp. on S_q), then the next multiplication with α will result in outputting α. The attack is thus effective. However, if the third method is used, the skipping of the multiplication $c_1 \cdot c_2$ will result in outputting the previous value of *temp* which is unlikely to be equal to 1. In this case, our attack cannot be applied.

This analysis shows that in the KQ-model most implementations of the improved FDTC'07 countermeasure are vulnerable, in other words this countermeasure is not intrinsically secure.

5 Countermeasure

In this section, we describe a countermeasure which can be used to counteract the attacks presented in Sections 3 and 4. The proposed countermeasure is indeed generic and can make any kind of first-order FA-countermeasure resistant to attacks in the KQ-model, provided it uses a form of redundancy and a checking (or infective) procedure.

5.1 Description

Firstly, we assume that the FA-countermeasure computes some redundant value c which is involved in the checking (or infective) procedure. Thus c is expected to take a given value, denoted $c^\star$, otherwise the checking (resp. infective) procedure returns an error (resp. infects the returned signature).

Our countermeasure is based on a simple mechanism that advantageously replaces the checking (resp. infective) procedure: given c and its expected value $c^\star$, we perform the check $(c = c^\star)$ twice while inserting in between a simple but pivotal statement. The whole procedure is described hereafter.

Procedure 1 Lock - principle

INPUTS: Res, S, c and $c^\star$
PROCESS: $\{Res \leftarrow S\}$ if $c = c^\star$ and $\{Res \leftarrow 0; S \leftarrow 0\}$ otherwise

1. **if** $(c \neq c^\star)$ erase S
2. $Res \leftarrow S$
3. **if** $(c \neq c^\star)$ erase Res
4. **return** Res

Remark 1. Some works have argued that coherence checks using conditional branches must be avoided for FA security [4, 7, 26]. The argument behind this assertion is that such a test can be skipped by corrupting the status register. As we show in Section 5.2, our solution is secure in the KQ-model despite the use of conditional branches. For the sake of completeness, we propose in Appendix A an implementation of the Lock procedure that avoid conditional branches.

A solution to strengthen an FA-resistant CRT-RSA based on the *Lock* procedure follows the series of steps hereafter. First, the result buffer Res is initialized at 0. Then the FA-resistant CRT-RSA is executed: from a message m and some parameters $\mathcal{P}$ it outputs a signature $S = m^d \bmod N$ and a couple of values $(c, c^\star)$ which depends on the FA-countermeasure. Afterward, the *Lock* procedure described in Procedure 1 is executed and Res is eventually returned. If $c = c^\star$ then

Lock processes $Res \leftarrow S$ and the computed signature is returned. Otherwise, the *Lock* procedure handles the signature erasure.

The overall RSA implementation resistant is described in Algorithm 7.

Algorithm 7 KQ-attack resistant RSA

INPUTS: m, $\mathcal{P}$
OUTPUT: $m^d \bmod N$

1. $Res \leftarrow 0$
2. $(S, c, c^\star) \leftarrow$ FA-Res-CRT-RSA$(m, \mathcal{P})$
3. Lock$(Res, S, c, c^\star)$

This generic solution can be applied to any FA-countermeasure based on redundant computation and it is resistant to the KQ-attack as shown in the next section.

5.2 Security Analysis

In the KQ-model (see Section 2), a first fault is dedicated to the corruption of the RSA computation and a second fault aims to avoid the erasure of the faulty signature by skipping some operations. According to this model, we assume that an attacker can:

- inject a fault in Step 2 of Algorithm 7 producing a faulty signature $\widetilde{S}$ and a faulty pair of checking values $(\widetilde{c}, \widetilde{c^\star})$ such that $\widetilde{c} \neq \widetilde{c^\star}$ (this results from the soundness of the first-order countermeasure that is used),
- skip a set of contiguous operations of Algorithm 7.

We demonstrate hereafter that the skipping of any set of contiguous operations of Algorithm 7 cannot prevent the *Lock* procedure from erasing the faulty signature $\widetilde{S}$ (or returning an unexploitable result) while the faulty checking values $\widetilde{c}$ and $\widetilde{c^\star}$ are different.

Let us first assume that the adversary skips the entire Step 2 of Procedure 1. One can check that in this case *Res* holds its initialization value and Algorithm 7 then returns an unexploitable result.

On the opposite, if Step 2 of Procedure 1 is not entirely skipped, then either all the previous operations or all the following operations are properly executed since the set of skipped operations is contiguous. As a result, either Step 1 or Step 3 of Procedure 1 is executed which ensures the signature erasure in case of fault detection (*i.e.* if $\widetilde{c}$ and $\widetilde{c^\star}$ are different).

To conclude, any KQ-attack implies the return of an unexploitable output and the attacker gains no sensitive information.

5.3 Application

In this section, we show how to apply the generic countermeasure described is Section 5.1 to Ciet and Joye's scheme [7] and to Giraud's scheme [11][1]. For this purpose, we have to modify these schemes in such a way that they output a checking value c and its expected value $c^\star$. They can then be used in replacement of the function FA-Res-CRT-RSA in Algorithm 7.

Modified Ciet and Joye's Scheme. We describe hereafter the modified Ciet and Joye's scheme that includes the countermeasure proposed in Section 5.1.

Algorithm 8 New Modified Ciet and Joye's scheme

INPUTS: m, p^*, q^*, d_p, d_q, I_q^*, N
OUTPUTS: $S = m^d \bmod N$, $c = (c_1, c_2)$, $c^\star = (c_1^\star, c_2^\star)$, r_1, r_2

1. Initialize (c_1, c_2) and $(c_1^\star, c_2^\star)$ with different arbitrary values
2. $S_p^* \leftarrow m^{d_p} \bmod p^*$
3. $c_2 \leftarrow m^{d_q \bmod \varphi(r_2)} \bmod r_2$
4. $S_q^* \leftarrow m^{d_q} \bmod q^*$
5. $c_1 \leftarrow m^{d_p \bmod \varphi(r_1)} \bmod r_1$
6. $S^* \leftarrow S_q^* + q^* \cdot I_q^* \cdot (S_p^* - S_q^*) \bmod p^*$
7. $c_1^\star \leftarrow S^* \bmod r_1$
8. $c_2^\star \leftarrow S^* \bmod r_2$
9. $S \leftarrow S^* \bmod N$
10. **return** $(S, (c_1, c_2), (c_1^\star, c_2^\star))$

If a fault injection implies $(c_1, c_2) \neq (c_1^\star, c_2^\star)$ (*i.e.* if the signature computation is corrupted), then one can check that it is impossible to force $(c_1, c_2) = (c_1^\star, c_2^\star)$ by skipping a set of contiguous operations in Algorithm 8. Indeed, (c_1, c_2) and $(c_1^\star, c_2^\star)$ are initialized with different values in Step 1 and they remain different until the end of the algorithm whether the different steps are executed or not. The only way one could force $(c_1, c_2) = (c_1^\star, c_2^\star)$ would be by skipping Steps 1 to 8 while assuming that the default values (before Step 1) of the buffers storing (c_1, c_2) and $(c_1^\star, c_2^\star)$ are the same. This would imply that the RSA computation is not performed which would prevent any attack.

Modified Giraud's Scheme. We describe hereafter the modified Giraud's scheme that includes the countermeasure proposed in Section 5.1. The Giraud's scheme [11] is based on the Montgomery powering ladder [18] that from m, p and d_p returns the pair $(m^{d_p - 1} \bmod p, m^{d_p} \bmod p)$. The coherence is checked by verifying the relation between the two returned values. Moreover, in order to avoid attacks disturbing the exponent, the modulus or the loop counter some checking

[1] It can also be straightforwardly applied to the implementation described in [19] since the FA-countermeasure is very similar to the one presented in [7].

information is computed. Let us denote by CI this checking information and by $CI^\star$ its expected value. Instead of checking $CI = CI^\star$ at the end of the exponentiation as in [11], $(CI, CI^\star)$ is returned as well as $(m^{d_p-1} \bmod p, m^{d_p} \bmod p)$. We denote by MME the modified Montgomery powering ladder.

Algorithm 9 summarizes the modified Giraud's scheme.

Algorithm 9 New Modified Giraud's scheme

INPUTS: m, p, q, d_p, d_q
OUTPUTS: $S = m^d \bmod N$, $c = (S, CI_p, CI_q)$, $c^\star = (S^*, CI_p^\star, CI_q^\star)$

1. Initialize (CI_p, CI_q) and $(CI_p^\star, CI_q^\star)$ with different arbitrary values
2. $(S_p^*, S_p, CI_p, CI_p^\star) \leftarrow \text{MME}(m, d_p, p)$
3. $(S_q^*, S_q, CI_q, CI_q^\star) \leftarrow \text{MME}(m, d_q, q)$
4. $S^* \leftarrow \text{CRT}(S_p^*, S_q^*)$
5. $S \leftarrow \text{CRT}(S_p, S_q)$
6. $S^* \leftarrow S^* \cdot m \bmod N$
7. **return** $\big(S, (S, CI_p, CI_q), (S^*, CI_p^\star, CI_q^\star)\big)$

In [11], it is not detailed how to compute the checking information CI on the loop counter, on the exponent and on the modulus. In [10], it is suggested to double the loop index and to compute a checksum for the exponent and the modulus. We do not give further details here since it is not our purpose. However, we mention that for our countermeasure to be valid, it is important that if a fault injection disturbs either the loop index or the exponent or the modulus, the attacker cannot force $CI = CI^\star$ by skipping some operations.

On the other hand, if one injects a fault that does not affect the loop counter nor the exponent nor the modulus, then it breaks the relation between the S_p^* and S_p (resp. S_q^* and S_q) in a way that is unpredictable for the attacker [11]. This makes it impossible for the attacker to recreate this relation – and hence to force $c = c^\star$ – by skipping some operations.

6 Conclusion

In this paper, we have analysed the security of the second-order FA-countermeasures published at WISTP'07 and FDTC'07. We have shown that these countermeasures are not intrinsically resistant with regard to the corresponding second-order fault model. We have also proposed a new method to protect CRT-RSA against this particular class of second-order fault attacks which induces a very small overhead compared to the traditional first-order FA-countermeasures. Protecting the CRT-RSA against a wider class of second-order fault attacks is still an open issue. This problem requires all our attention in order to anticipate future evolutions in practical fault induction which could follow the recent publication of the first practical application of a second-order fault attack.

References

1. C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert. Fault Attacks on RSA with CRT: Concrete Results and Practical Countermeasures. In B. Kaliski Jr., Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 260–275. Springer, 2002.
2. H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The Sorcerer's Apprentice Guide to Fault Attacks. *IEEE*, 94(2):370–382, 2006.
3. E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystem. In B. Kalisky Jr., editor, *Advances in Cryptology – CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 1997.
4. J. Blömer, M. Otto, and J.-P. Seifert. A New RSA-CRT Algorithm Secure against Bellcore Attacks. In S. Jajodia, V. Atluri, and T. Jaeger, editors, *ACM Conference on Computer and Communications Security – CCS'03*, pages 311–320. ACM Press, 2003.
5. D. Boneh, R. DeMillo, and R. 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, 1997.
6. A. Boscher, R. Naciri, and E. Prouff. CRT RSA Algorithm Protected against Fault Attacks. In D. Sauveron, K. Markantonakis, A. Bilas, and J.-J. Quisquater, editors, *Information Security Theory and Practices – WISTP 2007*, volume 4462 of *Lecture Notes in Computer Science*, pages 229–243. Springer, 2007.
7. M. Ciet and M. Joye. Elliptic Curve Cryptosystems in the Presence of Permanent and Transient Faults. *Designs, Codes and Cryptography*, 36(1):33–43, July 2005.
8. M. Ciet and M. Joye. Practical Fault Countermeasures for Chinese Remaindering Based RSA. In L. Breveglieri and I. Koren, editors, *Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC'05*, pages 124–132, 2005.
9. H. Garner. The Residue Number System. *IRE Transactions on Electronic Computers*, 8(6):140–147, June 1959.
10. C. Giraud. Fault Resistant RSA Implementation. In L. Breveglieri and I. Koren, editors, *Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC'05*, pages 142–151, 2005.
11. C. Giraud. An RSA Implementation Resistant to Fault Attacks and to Simple Power Analysis. *IEEE Transactions on Computers*, 55(9):1116–1120, Sept. 2006.
12. C. Giraud. Personnal communication, June 29, 2007.
13. C. Giraud and H. Thiebeauld. A Survey on Fault Attacks. In J.-J. Quisquater, P. Paradinas, Y. Deswarte, and A. E. Kalam, editors, *Smart Card Research and Advanced Applications VI – CARDIS 2004*, pages 159–176. Kluwer Academic Publishers, 2004.
14. L. Hemme. A Differential Fault Attack against Early Rounds of (Triple-)DES. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 254–267. Springer, 2004.
15. J. Hoch and A. Shamir. Fault Analysis of Stream Ciphers. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 240–253. Springer, 2004.
16. M. Joye, A. Lenstra, and J.-J. Quisquater. Chinese Remaindering Based Cryptosystems in the Presence of Faults. *Journal of Cryptology*, 12(4):241–245, 1999.

17. M. Joye, J.-J. Quisquater, F. Bao, and R. Deng. RSA-type Signatures in the Presence of Transient Faults. In M. Darnell, editor, *Cryptography and Coding*, volume 1355 of *Lecture Notes in Computer Science*, pages 155–160. Springer, 1997.
18. M. Joye and S.-M. Yen. The Montgomery Powering Ladder. In B. Kaliski Jr., Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 291–302. Springer, 2002.
19. C. H. Kim and J.-J. Quisquater. Fault Attacks for CRT Based RSA: New Attacks, New Results, and New Countermeasures. In D. Sauveron, K. Markantonakis, A. Bilas, and J.-J. Quisquater, editors, *Information Security Theory and Practices – WISTP 2007*, volume 4462 of *Lecture Notes in Computer Science*, pages 215–228. Springer, 2007.
20. C. H. Kim and J.-J. Quisquater. How Can We Overcome Both Side Channel Analysis and Fault Attack on RSA-CRT? In L. Breveglieri, S. Gueron, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2007*, pages 21–29. IEEE Computer Society, 2007.
21. O. Kommerling and M. Kuhn. Design Principles for Tamper Resistant Smartcard Processors. In *the USENIX Workshop on Smartcard Technology (Smartcard '99)*, pages 9–20, 1999.
22. D. Naccache, P. Nguyen, M. Tunstall, and C. Whelan. Experimenting with Faults, Lattices and the DSA. In S. Vaudenay, editor, *Public Key Cryptography – PKC 2005*, volume 3386 of *Lecture Notes in Computer Science*, pages 16–28. Springer, 2005.
23. G. Piret and J.-J. Quisquater. A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD. In C. Walter, Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 77–88. Springer, 2003.
24. A. Shamir. How to check modular exponentiation. Eurocrypt'97 rump session, 1997.
25. S.-M. Yen, D. Kim, and S. Moon. Cryptanalysis of Two Protocols for RSA with CRT Based on Fault Infection. 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 53–61. Springer, 2006.
26. S.-M. Yen, S.-J. Kim, S.-G. Lim, and S.-J. Moon. RSA Speedup with Residue Number System Immune against Hardware Fault Cryptanalysis. *IEEE Transactions on Computers*, 52(4):461–472, 2003.

A An Implementation of the Lock Procedure Without Conditional Branches

Let us introduce few notations. The bit-size of the checking value c is denoted by k and the radix bit-size of the microprocessor is denoted by w. The i^{th} w-bit digit of a buffer X is denoted by $X[i]$ and the size of the RSA modulus in radix 2^w is denoted by l. Our solution makes use of a logical function $\mathcal{M} : \mathbb{F}_2^k \rightarrow \mathbb{F}_2^w$ that satisfies:

$$\mathcal{M} : X \mapsto \begin{cases} (1, 1, \cdots, 1) & \text{if } X = 0 \\ (0, 0, \cdots, 0) & \text{if } X \neq 0 \end{cases}, \tag{1}$$

For a fast implementation of $\mathcal{M}$, we use a look-up table LUT of 256 w-bit words storing the $\mathbb{F}_2^8 \rightarrow \mathbb{F}_2^w$ version of $\mathcal{M}$. Namely $LUT[0] = 2^w - 1$ and $LUT[i] = 0$ for every $i \in \{1, \cdots, 255\}$.

The implementation of $\mathcal{M}$ is described in Algorithm 10.

Algorithm 10 An implementation of function $\mathcal{M}$

INPUTS: $X \in \mathbb{F}_2^k$
OUTPUT: $\mathcal{M}(X) \in \mathbb{F}_2^w$

1. $Res \leftarrow 2^w - 1$
2. **for** $i = 0$ **to** $k/w - 1$ **do**
3. **for** $j = 0$ **to** $w/8 - 1$ **do**
4. $Res \leftarrow Res \wedge LUT[(X[i] \gg 8j) \wedge 255]$
5. **return** Res

The implementation of the *Lock* procedure without conditional branches is described hereafter.

Procedure 2 Lock

INPUTS: Res, S, c and $c^\star$
PROCESS: $\{Res \leftarrow S\}$ if $c = c^\star$ and $\{Res \leftarrow 0; S \leftarrow 0\}$ otherwise

1. $mask \leftarrow \mathcal{M}(c \oplus c^\star)$
2. **for** $i = 0$ **to** $l - 1$
3. $S[i] \leftarrow S[i] \wedge mask$
4. $Res \leftarrow S$
5. $mask \leftarrow \mathcal{M}(c \oplus c^\star)$
6. **for** $i = 0$ **to** $l - 1$
7. $Res[i] \leftarrow Res[i] \wedge mask$