Fault attacks for CRT based RSA: new attacks, new results, and new countermeasures

Chong Hee KIM and Jean-Jacques Quisquater

UCL Crypto Group, Université Catholique de Louvain, Belgium
kim@dice.ucl.ac.be, quisquater@dice.ucl.ac.be

Abstract. Nowadays RSA using *Chinese Remainder Theorem* (CRT) is widely used in practical applications. However there is a very powerful attack against it with a fault injection during one of its exponentiations. Many countermeasures were proposed but almost all of them are proven to be insecure. In 2005, two new countermeasures were proposed. However they still have a weakness. The final signature is stored in a memory after CRT combination and there is an error-check routine just after CRT combination. Therefore, if an attacker can do a double-fault attack that gives the first fault during one of the exponentiation and the other to skip the error-checking routine, then he can succeed in breaking RSA. In this paper, we show this can be done with the concrete result employing a glitch attack and propose a simple and almost cost-free method to defeat it.

1 Introduction

After the advent of the concept of *Side Channel Analysis* by Kocher [15], many variants have appeared to attack the embedded systems such as smart cards. Among them, fault attacks introduced by Boneh et al. [5] are the most effective.

There are several kinds of methods to invoke faults such as variations in supply voltage, variations in the external clock, temperature variation, white light, laser, and X-rays and ion beams [3]. The objective of invoking faults is to make an abnormal operation in a target and to compute hidden secret information with the faulty output. In this paper, we use a glitch attack which makes a transient fault with a voltage spike. The target of glitches is to corrupt data transferred between registers and memory or to prevent the execution of the code. The glitch attack is used to attack RSA [1] and recently DSA [16]

The first victim of the fault attack on the cryptographic algorithms was RSA. The straightforward RSA implementation with Chinese Remainder Theorm was shown to be broken by fault attacks [5]. The simplest way to prevent the fault attack is just to compute signatures twice and compare them. However this doubles computation time. Furthermore it cannot avoid permanent errors. Another way is to verify the signature with the public exponent e. That is, the device returns the signature S only when $S^e \equiv m \pmod{N}$. However this method is too costly if e is large. Furthermore in some applications (e.g. javacard), it is not allowed to access the public exponent e during signature generation.

Many countermeasures were proposed [17, 1, 21] but they have been broken [13, 18]. In 2005, two new algorithms were proposed by Ciet and Joye [8] and Giraud [10] separately. However, their schemes also missed one important point. In the next section, we review the RSA-CRT and the existing countermeasures against fault attacks. Section 3 shows the problem of the existing countermeasures and experimental results. Section 4 present the new approach to avoid the new attack and finally we conclude in Section 5.

2 Previous countermeasures

In this section, we briefly review the RSA-CRT (CRT based RSA) signature and countermeasures against fault attacks .

2.1 RSA-CRT signature algorithm and fault attacks on it

Let $N = p \cdot q$ be the RSA modulus, where p and q are two large primes. Let e be the RSA public exponent and d be the RSA private exponent satisfying that $e \cdot d = 1 \bmod (p-1)(q-1)$. We also let d_p (resp. d_q) be the CRT exponent such that $d_p = d \bmod (p-1)$ (resp. $d_q = d \bmod (q-1)$). We denote by Iq the inverse of q modulo p. Then the signature S of the message m is computed as

1. $S_p = m^{d_p} \bmod p$
$S_q = m^{d_q} \bmod q$

2. $S = \mathrm{CRT}(S_p, S_q) = S_q + q \cdot ((S_p - S_q) \cdot Iq \bmod p)$.

Bellcore researchers showed that if an error occurs in only one of the exponentiations (that is, during a computation of S_p or S_q, but not in both), then the factorization of N is possible with the faulty signature $\tilde{S}$ [5]. For example, suppose that an error occurs during computation of S_p. Then a faulty $\tilde{S_p}$ will be used in a CRT combination and $\tilde{S} = \mathrm{CRT}\ (\tilde{S_p}, S_q)$ will be returned. With the correct signature S and the faulty one $\tilde{S}$, the secret prime q can be computed by computing GCD(S-$\tilde{S}$, N).

This attack is further improved with only one execution of algorithm [12]. Secret prime number q can be found by computing $\mathrm{GCD}(\tilde{S}^e - m, N)$.

2.2 Shamir's countermeasure and its generalizations

Shamir used a redundant way to compute S_p and S_q and checked the correctness of S_p and S_q before RSA combination[17]. Let r be a random k-bit integer (typically, k=32). Then the signature is computed as

1. $S_p^* = m^d \bmod (p \cdot r)$
   $S_q^* = m^d \bmod (q \cdot r)$

2. $\begin{cases} S = \mathrm{CRT}(S_p^*, S_q^*) \bmod N \text{ if } S_p^* \equiv S_q^* \pmod{r}, \\ \mathit{error} \qquad\qquad\qquad\qquad\qquad \text{otherwise.} \end{cases}$

Joye et al. pointed out one drawback of Shamir's method [13]. It requires d which is not known in CRT. Only $d_p = d \bmod (p-1)$ and $d_q = d \bmod (q-1)$ are known. They proposed an improved algorithm which verifies the two half exponentiations separately. Let r_1 and r_2 be two random k-bit integers. Then device computes

1. $S_p^* = m^{d_p} \bmod (p \cdot r_1)$, $s_1 = m^{d_p \bmod \varphi(r_1)} \bmod r_1$
   $S_q^* = m^{d_q} \bmod (q \cdot r_2)$, $s_2 = m^{d_q \bmod \varphi(r_2)} \bmod r_2$

2. $\begin{cases} S = \mathrm{CRT}(S_p^*, S_q^*) \bmod N \text{ if } S_p^* \equiv s_1 \bmod r_1 \text{ and } S_q^* \equiv s_2 \bmod r_2 \\ \mathit{error} \qquad\qquad\qquad\qquad\qquad \text{otherwise.} \end{cases}$

The previous algorithms cannot detect errors occurred during RSA combination. In [1], Aumüller et al. checked the correctness of the result of RSA combination. Let r be a short prime number, e.g., 16 bits. Then device computes

```
1. $p' = p \cdot r$
   $d'_p = d_p + random_1 \cdot (p-1)$
   $S'_p = m^{d'_p} \bmod p'$
   if $\neg(p' \equiv 0 \pmod{p} \land d'_p \equiv d_p \pmod{(p-1)})$ then return $error$

   $q' = q \cdot r$
   $d'_q = d_q + random_2 \cdot (q-1)$
   $S'_q = m^{d'_q} \bmod q'$
   if $\neg(q' \equiv 0 \pmod{q} \land d'_q \equiv d_q \pmod{(q-1)})$ then return $error$

2. $S_p = S'_p \bmod p$
   $S_q = S'_q \bmod q$
   $S = \mathrm{CRT}(S_p, S_q)$
```

```
3.  if $\neg(S \equiv S_p \pmod{p} \wedge S \equiv S_q \pmod{q})$ then return $error$

    $S_{pr} = S'_p \bmod r$
    $d_{pr} = d'_p \bmod (r-1)$
    $S_{qr} = S'_q \bmod r$
    $d_{qr} = d'_q \bmod (r-1)$
    if $(S_{pr}^{d_{qr}} \equiv S_{qr}^{d_{pr}})$ then
        return $S$,
    else
        return $error$.
```

2.3 Infective computations

Yen et al. proposed a different kind of approach, *fault infective computation* [21]. They noted that an error detection based on decisional tests should be avoided. From the viewpoint of low-level implementation of this decision procedure, it often totally relies on the status of the *zero flag* of a processor. The zero flag is a bit of the status register in a processor. So, if an attack can induce a random fault into the status register, then conditional jump instruction may perform falsely. In their method, if an error occurs in one of the exponentiations (S_p or S_q), then it makes both $\tilde{S} \not\equiv S \pmod{p}$ and $\tilde{S} \not\equiv S \pmod{q}$. Unfortunately, both their countermeasures were shown to be insecure by Yen and Kim [19]

Blömer et al. suggested another countermeasure based on Shamir's method and on fault infective computation [7]. Given a security parameter k, for two appropriately chosen k-bit integers r_1 and r_2 (stored in memory), the following quantities are pre-computed and stored in memory:

$$r_1 p, \quad r_2 q, \quad r_1 r_2 N,$$

$$\begin{aligned} d_1 &= d \bmod \varphi(r_1 p), \quad e_1 = {d_1}^{-1} \bmod \varphi(r_1), \\ d_2 &= d \bmod \varphi(r_2 q), \quad e_2 = {d_2}^{-1} \bmod \varphi(r_2). \end{aligned}$$

The device then computes

1. $S_p^* = m^{d_1} \bmod (r_1 p)$,
   $S_q^* = m^{d_2} \bmod (r_2 q)$,

2. $S^* = \mathrm{CRT}(S_p^*, S_q^*) \bmod (r_1 r_2 N)$,

3. $c_1 = (m - {S^*}^{e_1} + 1) \bmod r_1$,
   $c_2 = (m - {S^*}^{e_2} + 1) \bmod r_2$,
   $S = (S^*)^{c_1 c_2} \bmod N$.

If there is no error, then c_1 and c_2 become 1 and the device returns a correct signature S.

Unfortunately, this countermeasure is also shown to be insecure by Wagner [18]. Let us suppose a random transient fault that modifies the value of m as it is being read from memory in the computation of S_p^* while leaving the value stored in memory unaffected, then $c_1 \neq 1$ but $c_2 = 1$. Then the attacker can mount a Bellcore-like attack by computing $\mathrm{GCD}(m^{c_1} - S^e, N)$ with the guess of c_1. In the scheme, $c_1 = (m - S^{e_1} + 1) \bmod r_1$. Since $S^{e_1} = \tilde{m}$ can be guessed in his fault attack model, the attack was possible. Recently Blömer et al. proposed a variant that overcome the weakness by randomizing the computation of c_i [6]

2.4 Ciet and Joye's countermeasure

In 2005, Ciet and Joye generalized Shamir's countermeasure [13] and adapted fault infective computation [21] to avoid decisional tests [8]. For two co-prime k-bit integers r_1 and r_2 and l-bit integer r_3, we define

$$\begin{aligned} p^* &= r_1 p, \\ q^* &= r_2 q, \\ I_q^* &= (q^*)^{-1} \bmod p^*. \end{aligned}$$

Then device computes

1. $S_p^* = m^{d_p} \bmod p^*$ and $s_2 = m^{d_q \bmod \varphi(r_2)} \bmod r_2$,
   $S_q^* = m^{d_q} \bmod q^*$ and $s_1 = m^{d_p \bmod \varphi(r_1)} \bmod r_1$,

2. $S^* = S_q^* + q^* \cdot I_q^* \cdot (S_p^* - S_q^*) \bmod p^*$,

3. $c_1 = (S^* - s_1 + 1) \bmod r_1$
   $c_2 = (S^* - s_2 + 1) \bmod r_2$
   $\gamma = \lfloor (r_3 c_1 + (2^l - r_3) c_2)/2^l \rfloor$
   $S = (S^*)^\gamma \bmod N$

2.5 Giraud's countermeasure

In 2005, Giraud [10] used the fact that the temporary variables (a_0, a_1) are of the form $(m^\alpha, m^{\alpha+1})$ in Joye and Yen's SPA-countermeasure [14]. Let $(d_{n-1}, \ldots, d_0)$ be the binary representation of d. Then a safe-error resistant exponentiation based on Montgomery Ladder of [14] is computed as following:

```
$a_0 \leftarrow 1$
$a_1 \leftarrow m$
for $i$ from $n-1$ to 0 do
    $a_{\bar{d_i}} \leftarrow a_{\bar{d_i}} \cdot a_{d_i} \bmod N$
    $a_{d_i} \leftarrow a_{d_i}^2 \bmod N$
return $a_0$.
```

To construct a SPA-FA(fault attack)-resistant CRT-RSA, he first proposed SPA-FA-resistant modular exponentiation (d is supposed to be odd):

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

Giraud used $k \cdot N$ instead of N, where k is a 32-bit random number in [11]. Here, we denote above algorithm as $(A, B) \leftarrow$ SPA-FA-EXP(m, d, N). Then the output (A, B) is ($m^{d-1} \bmod N$, $m^d \bmod N$). Finally SPA and FA-resistant CRT-RSA algorithm is as follows:

```
1.  $(S_p^*, S_p) \leftarrow$ SPA-FA-EXP$(m, d_p, p)$
    $(S_q^*, S_q) \leftarrow$ SPA-FA-EXP$(m, d_q, q)$

2.  $S^* = \mathrm{CRT}(S_p^*, S_q^*)$
    $S = \mathrm{CRT}(S_p, S_q)$
    $S^* = m \cdot S^* \bmod (p \cdot q)$

3   if $S^* = S$ & (Parameters $p$ and $q$ not modified) then
        return $S$
    else
        return *error*
```

3 Problem of previous countermeasures

The previously known countermeasures to defeat fault attacks on RSA-CRT mostly consist of three parts. Firstly the device computes two exponentiation S_p^* and S_q^*. The computation of S_p^* (resp. S_q^*) is done by either straightforward computation like $S_p = m^{d_p} \bmod p$ (resp. $S_q = m^{d_q} \bmod q$) or inclusion of a kind of redundancy which will be used later to check errors. Secondly it combines two exponentiations to compute signature S^*.

The final step can be divided into two categories. The first one uses conditional check routine in which if an error does not occur then it outputs correct signature and if error occurs it gives predefined signal like “error has been detected” (e.g. Aumüller et al.[1], Giraud’s [10], etc.). In the other method, instead of using the conditional check routine it gives a random value instead of a signature if error occurs (e.g. Infective computations[21, 6], Ciet and Joye's [8], etc.).

```
Step 1. Computation of two exponentiation
        - Compute $S_p^*$ and $S_q^*$

Step 2. CRT combination
        - Compute $S^* \leftarrow \mathrm{CRT}(S_p^*, S_q^*)$

Step 3. Fault detection
        - Return $\begin{cases} S \leftarrow f(S^*) & \text{if there is no error,} \\ \perp & \text{otherwise.} \end{cases}$
```

Suppose that the attacker tries to skip "fault detection" routine (Step 3 in the above model) after CRT combination (Step 2). Then the attacker can get S^*. Furthermore S can be computed easily since $S = S^*$ in the conditional check routine approach and $S = S^* \bmod N$ in the other approach. Therefore we can consider the following attack scenario.

Our fault attack model. The attacker tries to do a double-fault attack. He gives the fist fault during only one of the two exponentiations to corrupt its value. Then he gives the other fault during fault detection routine to skip some operations. If he succeed in doing a double-fault attack, then he can get the output of the CRT combination. That is, he gives a fault during Step 1. and gets the output of Step 2. by skipping some operations of Step 3. by faults in the above model.

Unfortunately all previous known countermeasures can be vulnerable to our fault attack scenario. Because all of them check the occurrence of errors after computation of a final signature.

3.1 Experiments of our attack

General description As seen in the previous section, RSA-CRT with a countermeasure against fault attacks is composed of three parts. The attacker tries to give two times faults. In the first trial, he tries to make errors during only one of the two exponentiations during Step 1. If the attacker can get the faulty signature as the output of Step 2, then he can compute the private keys. Therefore, he gives his second faults during Step 3 to skip some operations.

In the next section, as an example, we chose Ciet and Joye's countermeasure and implemented it. We gave a fault during the computation of S_p^*. Then we gave the next fault during the computation of $S = (S^*)^\gamma \bmod N$ and tried to skip it.

Results We implemented 128-bit RSA-CRT with Ciet and Joye's countermeasure (We note it as *RSA-CRT with CJ*) [8] in an Atmel 8-bit AVR microcontroller, ATMega168 [2]. The program is implemented as follows:

```
Main() {
        ...
        Set I/O pin low
        Call subroutine RSA-CRT with CJ (as in 2.4)
        Set I/O pin high
        ...
}
```

The tools used to create the glitches and the target board can be seen in Fig.1. The chip is communicating with a computer via serial communication and the power consumption is monitored by an oscilloscope even though they are not shown in the figure. Fig.2 shows the I/O pin and power profiles. The x-axis represents time and y-axis represents voltage (for I/O profile) and the consumption of power (for power profile). The upper line represents the profile of I/O behavior. The lower profile shows the power profile. The *RSA-CRT with CJ* starts at the time block 0.4 and ends at the time block 6.8. In the figure the blocks are numbered from 0 to 10. In our case the first exponentiation S_p^* lies in the time frame 0.4 to 3.2. and the second one S_q^* in 3.2 to 6.0.

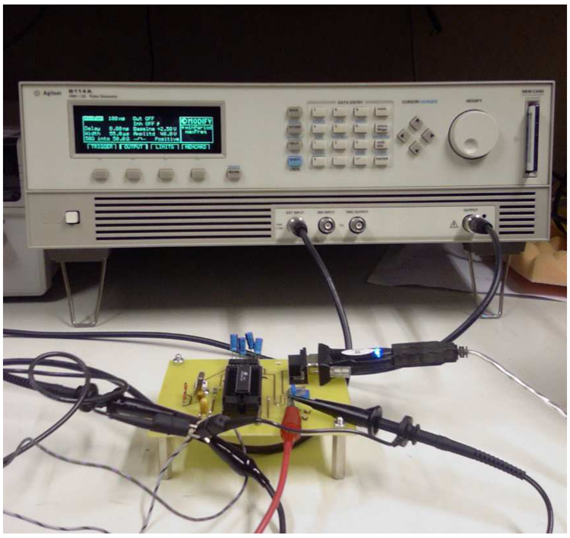

Fig. 1. Experiment setup for the glitch attack

Firstly we gave a glitch into chip's power supply during the first exponentiation. You can see the result in Fig.3. There is a high peak in power profile in

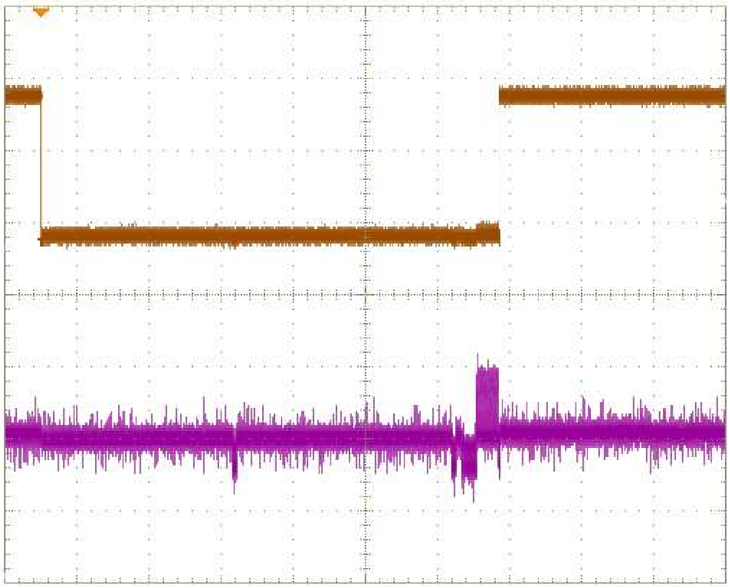

Fig. 2. RSA-CRT with CJ method without faults

the time frame about 0.6. You can also see the increase of total execution time due to the faults. Since γ is no more 1, the computation of $S = (S^*)^\gamma \bmod N$ requires more time. In Fig.3 the time frame from 6.4 to 7.0 corresponds to this.

Then we gave another glitch just before the last computation of $S = (S^*)^\gamma \bmod N$ as in Fig.4. Compared to Fig.3, you can see the total time is reduced and the final computation of $S = (S^*)^\gamma \bmod N$ is disappeared. However you can see the chip is still working and PC(*program counter*) is in *main* function by seeing the I/O pin is *high*. If the chip is dead then the I/O pin will stay at *low* state. Because I/O pin is set to *high* in the main program after calling subroutine *RSA-CRT with CJ*. We confirmed the computation of $S = (S^*)^\gamma$ is skipped by reading the returned value and computing the prime number p and q with this fault signature with *Bellcore-like attack* [4].

In addition, the second fault skipping operations was more difficult than the one making faults during an exponentiation. Sometimes a chip was stunned and it never returned back to main function. Sometimes it showed whole RAM values (there is a command shows specific RAM value, but it seemed that there was a disturbance on the required address of RAM). We could also change the value of γ by giving a glitch during the computation of this.

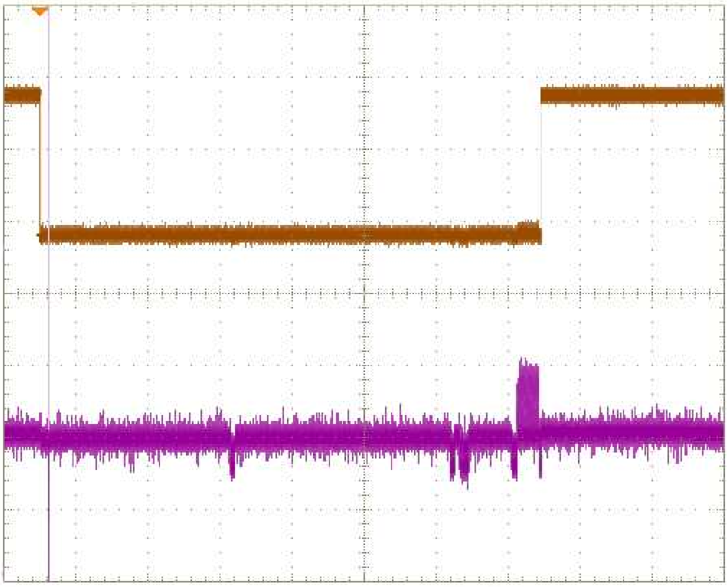

Fig. 3. Glitch attack during S_p^* operation

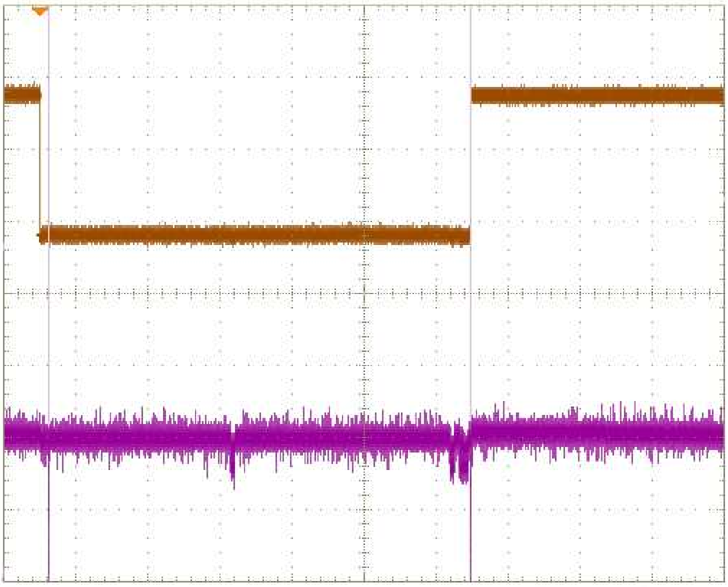

Fig. 4. Glitch attack both S_p^* operation and $(S^*)^\gamma \bmod N$

4 New approach to prevent fault attacks

The simplest method to prevent our attack in Ciet and Joye's scheme is to allocate all intermediate variables in different memory buffers that are not used for the returned signature. (Initialization of γ with a random value was not perfect in our experiment because we skipped only the last exponentiation after computation of γ). It means, for example, 128 bytes in a 1024-bit RSA implementation should not be used during whole RSA operations. It can be a burden for a programmer. Usually there is a specialized RAM, it is called *crypto RAM*, in a smart card only for cryptographic usage and this is not so large. Therefore the programmer should implement a RSA within comparatively small crypto RAM. Furthermore, because there is a possibility to know the value stored in RAM according to our experiments, the best idea is not to store the final signature until the end of checking errors.

Therefore we suggest more general idea to overcome previously mentioned problem. It is that the final signature S to be computed and stored only after Step 3. Then with the result of Step 2 the attacker could not get any useful information about secret keys. We modify Ciet and Joye's scheme and Giraud's scheme as examples. We put a randomness before CRT combination and get rid of it after an error check. Our idea can be applied for other countermeasures.

Modified Ciet and Joye's scheme

```
0. Choose a random integer $\mathbf{a}$ in $Z^*_{r_1 r_2 N}$
   Initialize $\gamma$ with a random number

1. $S^*_p = (\mathbf{a} + m^{d_p}) \bmod p^*$ and $s_2 = (\mathbf{a} + m^{d_q \bmod \varphi(r_2)}) \bmod r_2$,
   $S^*_q = (\mathbf{a} + m^{d_q}) \bmod q^*$ and $s_1 = (\mathbf{a} + m^{d_p \bmod \varphi(r_1)}) \bmod r_1$,

2. $S^* = S^*_q + q^* \cdot I^*_q \cdot (S^*_p - S^*_q) \bmod p^*$,

3. $c_1 = (S^* - s_1 + 1) \bmod r_1$
   $c_2 = (S^* - s_2 + 1) \bmod r_2$
   $\gamma = \lfloor (r_3 c_1 + (2^l - r_3) c_2)/2^l \rfloor$
   $S = (S^* - \mathbf{a}^\gamma) \bmod N$
```

After Step 2, S^* is the form of $S + a$, where a is a random value. Therefore, the attacker cannot get any information on the real signature S even though he succeeded in skipping the subsequent operations. More detail analysis follows.

Security analysis

Bellcore-like attack. Suppose that an attacker succeeded in introducing a fault during one of the two exponentiations and also getting rid of the final operation

$S = (S^* - \mathbf{a}^\gamma) \bmod N$. Let $\tilde{S}_p^*$ be the faulty exponentiation and $\tilde{S}^*$ be the faulty output of Step 2. Then the attacker will try to compute $\mathrm{GCD}((\tilde{S}^*)^e - m, N)$. However since $(\tilde{S}^*)^e = (\tilde{S}+a)^e$, neither $((\tilde{S}^*)^e - m) \not\equiv 0 \bmod p$ nor $((\tilde{S}^*)^e - m) \not\equiv 0 \bmod q$. Consequently the attacker cannot factorize N.

This is the same when the attacker tries to compute $\mathrm{GCD}(S^* - \tilde{S}^*, N)$. Let a be the random number used in computing S^* and b be the random number used in computing the second faulty signature $\tilde{S}^*$. Then $S^* \equiv S_p + a_p \pmod p$ and $S^* \equiv S_q + a_q \pmod q$. And $\tilde{S}^* \not\equiv \tilde{S}_p + b_p \pmod p$ and $\tilde{S}^* \equiv S_q + b_q \pmod q$. Therefore, since $(S^* - \tilde{S}^*) = (S_p + a_p - \tilde{S}_p - b_p) \not\equiv 0 \bmod p$ and $(S^* - \tilde{S}^*) = (S_q + a_p - S_q - b_q) \not\equiv 0 \bmod q$, the attacker cannot factorize N.

Consideration on skipping operations. Step 3 is consists of two parts. The one is computing c_1, c_2, and γ (Let's say Step 3.1). The other is computing S (We call it as Step 3.2). Our experiments focused on skipping Step 3.2 . Therefore if an attacker succeeds in skipping this, he gets $(S^* + a)$ and receives no valuable information on secret keys. This is the same when he skips both Step 3.1 and Step 3.2. Then how about skipping only Step 3.1? In our modified scheme, γ is initialized with a random number, therefore he gets $(S^* - a^\gamma)$. The only possibility to attack is to make $\gamma = 1$ which is negligible.

Let us consider Giraud's scheme. We first modify SPA-FA-resistant modular exponentiation in order to make the output as the form of $(a + m^{d-1} \bmod N$, $a + m^d \bmod N)$, where a is a random value. Similar security analysis is possible, but since Giraud's scheme uses a conditional check-routine, if it is skipped the attack is possible. Therefore avoiding a conditional check-routine is much better. The proposed one is a simple example to avoid our attack (skipping Step 3.2 and skipping both Step 3.1 and Step 3.2). To avoid the attack skipping only Step 3.1 (a conditional check) can be prevented by adding a randomness before the start of exponentiation . Because if an error occurs in a one of exponentiations, then it will affect also a random number used.

Modified SPA-FA-resistant modular exponentiation, SPA-FA-EXP*

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

```
Modified Giraud's scheme

0.  Choose a random integer $\mathbf{a}$ in $Z_N^*$

1.  $(S_p^*, S_p) \leftarrow$ SPA-FA-EXP$^*(m, d_p, p, \mathbf{a})$
    $(S_q^*, S_q) \leftarrow$ SPA-FA-EXP$^*(m, d_q, q, \mathbf{a})$

2.  $S^* = \mathrm{CRT}(S_p^*, S_q^*)$
    $S = \mathrm{CRT}(S_p, S_q)$
    $S^* = (m \cdot S^* + \mathbf{a}) \bmod (p \cdot q)$
    $S = (S + \mathbf{a} \cdot m) \bmod (p \cdot q)$

3   if $(S^* = S)$ & (Parameters $p$ and $q$ not modified) then
        return $(S - \mathbf{a} - \mathbf{a} \cdot m) \bmod N$
    else
        return *error*
```

5 Conclusions

In this paper, we pointed out the weakness of previous countermeasures against fault attacks on CRT-RSA. Previous countermeasures are all vulnerable since they are constructed without considering this weakness. Furthermore, to the authors' best knowledge, we showed the first (public reported) physical experiment allowing double faults during one execution of the algorithm. Finally, we proposed a simple and almost cost-free method to defeat this attack.

Acknowledgments The author would like to thank for anonymous reviewers for their valuable comments.

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, *Cryptographic Hardware and Embedded Systems – CHES 2002, LNCS V.2523*, pp.260-275, 2002
2. *http://www.atmel.com/product/AVR/*
3. H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, The Sorcerers apprentice guide to fault attacks, *Workshop on Fault Diagnosis and Tolerence in Cryptgraphy in association with DSN 2004 – The International Conference on Dependable Systems and Networks*, pp.330-342, 2004
4. D. Boneh, R.A. DeMillo, and R.J. Lipton, On the importance of checking cryptographic protocols for faults, *Advances in Cryptology – EUROCRYPT'97, LNCS V.1233*, pp.37-51, 1997.

5. D. Boneh, R.A. DeMillo, and R.J. Lipton, On the importance of eliminating errors in cryptographic computations, *Journal of Cryptology 14(2)*, pp.101-119, 2001. An earlier version appears in [4].
6. J. Blömer and M. Otto, Wagner's attack on a secure CRT-RSA algorithm recondiered, *Fault Diagnosis and Tolerance in Cryptography – FDTC'06 LNCS V.4236*, pp.13-23, 2006
7. J. Blömer, M. Otto, and J.-P. Seifert, A new CRT-RSA algorithm secure against Bellcore attacks, *10th ACM Conference on Computer and Communications Security*, pp.311-320, 2003
8. M. Ciet and M. Joye, Practical fault countermeasures for Chinese Remaindering based RSA, *Fault Diagnosis and Tolerance in Cryptography – FDTC'05*, pp.124-131, 2005
9. P.-A. Fouque and F. Valette, The doubling attack - why upward is better than downwards, *Cryptographic Hardware and Embedded Systems – CHES'03, LNCS V.2779*, pp.269-280, 2003
10. C. Giraud, Fault resistant RSA implementation, *Fault Diagnosis and Tolerance in Cryptography - FDTC'05*, pp.142-151, 2005
11. C. Giraud, An RSA implementaiton resistant to fault attacks and to simple power analysis, *IEEE Transactions on computers, VOL. 55, NO. 9,* pp.1116-1120, 2006
12. M. Joye, A.K. Lenstra, and J.-J. Quisquater, Chinese remaindering based cryptosystems in the presence of faults, *Journal of Cryptology 12(4)*, pp.241-245,1999.
13. M. Joye, P. Pailler, S.-M. Yen, Secure evaluation of modular functions, *International Workshop on Cryotpology and Network Security 2001*, pp.227-229, 2001
14. M. Joye and S.-M. Yen, The Montgomery powering Ladder, *Cryptographic Hardware and Embedded Systems – CHES 2002, LNCS V.2523*, pp.291-302, 2002
15. P. Kocher, Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems, *CRYPTO'96, LNCS V.1109*, pp.104-113, 1996
16. D. Naccache, P.Q. Nguyen, M. Tunstall, and C. Whelan, Experimenting with Faults, Lattices and the DSA, *Public Key Cryptography - PKC 2005, LNCS V.3386*, pp.16-28, 2005
17. A. Shamir, Method and apparatus for protecting public key schemes from timing and fault attacks, United States Patent ♯5,991,415, November 23, 1999. Also presented at the rump session of EUROCRYPT'97.
18. D. Wagner, Cryptanalysis of a provably secure CRT-RSA algorithm, *11th ACM Conference on Computers and Communications Security*, pp.92-97, 2004
19. S.-M. Yen and D. Kim, Cryptanalysis of two protocols for RSA with CRT based on fault infection, *Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC'04*, pp.381-385, 2004
20. S.-M. Yen, S. Kim, S. Lim, and S. Moon, RSA speedup with residue number system immune against hardware fault cryptanalysis, *Information Security and Cryptology – ICISC 2001 LNCS V.2288*, pp.397-413, 2001
21. S.-M. Yen, S. Kim, S. Lim, and S. Moon, RSA speedup with Chinese remainder theorem immune against hardware fault cryptanalysis, *IEEE Transactions on Computers 52(4)*, pp.461-472, 2003. An earlier version appears in [20]