CRT RSA Algorithm Protected Against Fault Attacks

Arnaud Boscher[1]*, Robert Naciri[2] and Emmanuel Prouff[2]

[1] Spansion,
105 rue Anatole France,
92684 Levallois-Perret Cedex, France
[2] Oberthur Card Systems,
71-73 rue des Hautes Pâtures,
92726 Nanterre Cedex, France
arnaud.boscher@spansion.com,{r.naciri,e.prouff}@oberthurcs.com

Abstract. Embedded devices performing RSA signatures are subject to Fault Attacks, particularly when the Chinese Remainder Theorem is used. In most cases, the modular exponentiation and the Garner recombination algorithms are targeted. To thwart Fault Attacks, we propose a new generic method of computing modular exponentiation and we prove its security in a realistic fault model. By construction, our proposal is also protected against Simple Power Analysis. Based on our new resistant exponentiation algorithm, we present two different ways of computing CRT RSA signatures in a secure way. We show that those methods do not increase execution time and can be easily implemented on low-resource devices.

Keywords: RSA, Chinese Remainder Theorem, Modular Exponentiation, Fault Attacks, Simple Power Analysis, Smart Card.

1 Introduction

In 1997, Boneh, DeMillo and Lipton [1] introduced a new type of cryptanalysis based on error computations: Fault Attacks (FA). Various public-key cryptosystems were concerned but the RSA algorithm was especially targeted. Indeed, Fault Attacks are particularly effective when the Chinese Remainder Theorem (CRT) is applied. Using these techniques, an RSA modulus of arbitrary length can be factorized practically instantly on a PC.

Fault Attacks can be directed at cryptographic embedded devices, like smart cards, as shown in [2]. Straightforward protection mechanisms compute the signature twice, or verify it by performing the inverse operation. Nevertheless, this can be time consuming and further complicated if the corresponding public key is unknown to the device. So, alternative counter-measures, inside the algorithm itself, have been proposed to protect RSA signatures computations against Fault

* This work was done while the author was with Oberthur Card Systems.

Attacks [3–7]. Unfortunately, many of them have been broken since their publication [2, 8, 9].

Counteracting FA is not sufficient to ensure the security of an embedded cryptosystem. Indeed, another threat comes from physical leakage during cryptographic computations. A category of attacks, called Side Channel Analysis (SCA), exploits this leakage to retrieve information about sensitive data manipulated by the algorithm. Among these attacks, Simple Power Analysis (SPA) is the easiest to mount in practice and an implementation of a cryptosystem in mobile devices must thwart it. Counteracting FA and SPA attacks at the same time is an issue. Indeed, some counter-measures against SPA can been exploited by elaborate Fault Attacks such as Safe Error Attacks. In fact, it appears that Simple Power Analysis and Fault Attacks (classical and Safe Error) must be simultaneously taken into account when implementing cryptographic algorithms.

The paper is organized as follows. In the next section we briefly recall the RSA cryptosystem, the use of the Chinese Remainder Theorem to speed up generation of RSA signatures and the description of Fault Attacks directed against it. Then, in Sect. 3, we present our method for computing a modular exponentiation protected against Fault Attacks, proving its security in a practical fault model whose relevance to (real life) scenarios is discussed. The new algorithm is used in Sect. 4 to design two CRT RSA implementations resistant to FA and SPA.

2 RSA and Physical Attacks

2.1 RSA Cryptosystem

The public-key cryptosystem RSA [10] involves a public modulus N, which is the product of two large secret primes p and q. The public exponent e is co-prime with $(p-1)\cdot(q-1)$ and the private exponent d is the modular inverse of e modulo $(p-1)\cdot(q-1)$.

An RSA signature S of a message M is computed with the following formula:

$$S = M^d \bmod N.$$

To speed-up the exponentiation on low-resource devices, like smart cards, one usually applies the Chinese Remainder Theorem [11]. The resulting *CRT RSA signature* algorithm is four times faster compared to the classical method. It involves two modular exponentiations and a recombination step using Garner's Algorithm [12]. It needs 5 parameters: the two large primes p and q, the values $d_p = d \bmod p-1$, $d_q = d \bmod q-1$, and the pre-computed value $A = p^{-1} \bmod q$.

Algorithm 2.1 RSA Signature using CRT
INPUT: M, p, q, d_p, d_q, A
OUTPUT: S

1. $S_p \leftarrow M^{d_p} \bmod p$ //First Exponentiation

2. $S_q \leftarrow M^{d_q} \bmod q$ //Second Exponentiation
3. $S \leftarrow ((S_q - S_p) \cdot A \bmod q) \cdot p + S_p$ //Garner's Algorithm
4. return(S)

2.2 Simple Power Analysis on RSA Algorithm

By exploiting physical leakage of a device, secret parameters can be retrieved [13, 14] depending on the implementation of the algorithm. Among those Side-Channels Attacks, Simple Power Analysis retrieves information by measuring the power consumption of one execution of the algorithm, whereas Differential Power Analysis (DPA) uses many samples of power consumption and applies statistical techniques to get information. In the following, we particularly focus on SPA attacks. More details about DPA attacks on modular exponentiations, and counter-measures against these attacks, can be found in [15].

To explain how SPA allows one to get information on the secret exponent, let us consider the following basic implementation of a modular exponentiation, known as the Square and Multiply Algorithm, in which the exponent bits are scanned from right to left [16]:

Algorithm 2.2 Right-to-Left Modular Exponentiation
INPUT: $M, d = (d_{n-1}, \ldots, d_0)_2, N$
OUTPUT: $M^d \bmod N$

1. $S \leftarrow 1$
2. $A \leftarrow M$
3. for i from 0 to $n-1$ do
4. if $d_i = 1$ then $S \leftarrow S \cdot A \bmod N$
5. $A \leftarrow A^2 \bmod N$
6. return(S)

If the executions of a modular square and a modular multiplication have different power consumptions, it has been shown in [13, 14] that information on the value of the secret exponent d can be retrieved. Indeed, if two consecutive modular squares are identified, this means that the exponent bit processed was 0. On the contrary, if a modular multiplication is interleaved between two modular squares, the exponent bit was equal to 1.

To get around this problem, the following algorithm, called Square and Multiply Always, was proposed in [17]:

Algorithm 2.3 SPA Resistant Right-to-Left Modular Exponentiation
INPUT: $M, d = (d_{n-1}, \ldots, d_0)_2, N$
OUTPUT: $M^d \bmod N$

1. $S[0] \leftarrow 1$
2. $S[1] \leftarrow 1$
3. $A \leftarrow M$
4. for i from 0 to $n-1$ do
5. $\quad S[\overline{d_i}] \leftarrow S[\overline{d_i}] \cdot A \bmod N$
6. $\quad A \leftarrow A^2 \bmod N$
7. return($S[0]$)

In Algorithm 2.3, each iteration of the loop involves a modular multiplication whatever the bit-value of the exponent d. Since the sequence of successive operations performed are independent of the key-bits, attacks such as SPA become impossible.

2.3 Fault Attacks on CRT RSA Algorithm

Fault Attacks have been suggested by Boneh *et al.* [1]. They observed that if a device outputs an erroneous CRT RSA signature, an attacker can deduce the private key from this information and the correct signature.

Indeed, let us assume than an error occurs during one of the modular exponentiations of Algorithm 2.1. This results in an incorrect intermediate result, e.g. $\tilde{S}_p$, which will generate an erroneous signature $\tilde{S}$. The faulty signature $\tilde{S}$ and the correct signature S are likely to satisfy $\tilde{S} \not\equiv S \bmod p$ and $\tilde{S} \equiv S \bmod q$. Consequently, if $S - \tilde{S}$ is not divisible by p, the prime number q is revealed by a *gcd* computation : $q = gcd(S - \tilde{S}, N)$.

Remark 1. As noticed in [18], the attack can also be performed without the knowledge of the correct signature: computing $gcd(\tilde{S}^e - M, N)$ will also discover q.

The classical protection against this attack is to verify the computed signature with the public exponent e before sending the signature. The erroneous signature being not returned, the *gcd* computation can no more be computed. However, this can be costly in time (depending on the value of e) and sometimes impossible, if the public key is unknown to the device[3].

In Sections 2.2 and 2.3, we recalled two simple ways of thwarting SPA and FA separately. In the next section, we show that this approach is not enough to obtain a secure implementation of a modular exponentiation.

2.4 Fault Attacks on SPA-resistant RSA Algorithm

Algorithm 2.3 ensures protection against SPA, but introduces a weakness with respect to another type of Fault Attacks, known as Safe Error, as described in [19].

[3] Computing the public exponent with the knowledge of the 5 CRT parameters is possible but time-consuming on low-resource devices.

When an exponent bit equals 0, the result of the *dummy* computation of the modular multiplication is not used any more in the algorithm. Consequently, if a fault is induced on this modular multiplication, an attacker can determine the value of the bit, depending on the correctness or the incorrectness of the modular exponentiation. If the result is correct, the modified modular multiplication was a dummy operation, and so the bit of the exponent was 0. On the contrary, if the result is erroneous, the modified modular multiplication was used in the rest of the algorithm, meaning that the exponent bit was 1.

This kind of attack can be applied to an RSA signature to recover the private key, irrespective of whether it uses the CRT mode. This kind of cryptanalysis requires more work on the part of the attacker than the analysis discussed in Sect. 2.3 where only one fault was sufficient to obtain the private key. However, it is much more powerful since it thwarts the classical counter-measure consisting in checking the signature before sending it.

The attack described above illustrates the difficulty of thwarting SPA and FA simultaneously. In the following section, we present a new method to compute modular exponentiation resistant against Simple Power Analysis, Fault Attacks and Safe Error Attacks.

3 Exponentiation Resistant to Fault Attacks

3.1 Our Proposal

Our idea consists essentially in modifying an SPA-resistant algorithm by introducing some *coherence test* at the end. This test aims at ensuring that no fault has been induced during the execution of the algorithm. In fact, our reasoning is very close to that proposed in [5] and [20].

Before explaining the core idea of our proposal, let us recall the content of the loop of Algorithm 2.3:

4. for i from 0 to $n-1$ do
5. $\quad S[\overline{d_i}] \leftarrow S[\overline{d_i}] \cdot A \bmod N$
6. $\quad A \leftarrow A^2 \bmod N$

Our idea is based on the three following observations:

- The value A is independent of d. At the end of the algorithm, A satisfies:

$$A = M^{2^n} \bmod N.$$

- The value $S[1]$ is the result of the modular exponentiation of M by the binary complement of d, denoted $\overline{d}$ (and satisfying $\overline{d} = 2^n - d - 1$):

$$S[1] = M^{2^n - d - 1} \bmod N.$$

– Since $S[0]$ equals $M^d \bmod N$, the following relation holds for the content of A after the loop:

$$M \cdot S[0] \cdot S[1] = M \cdot M^d \cdot M^{2^n-d-1} = M^{2^n} = A \bmod N \ . \qquad (1)$$

Equation (1) establishes a relationship between the contents of $S[0]$, $S[1]$, A and M after each loop iteration. So, to ensure that none of the modular multiplications was interfered with (thus counteracting Fault Attacks and Safe Error Attacks), we perform a check between the four values involved in the algorithm. We verify that M, $S[0]$, $S[1]$ and A satisfy Equality (1) before returning $S[0]$:

Algorithm 3.1 SPA/FA Resistant Right-to-Left Modular Exponentiation

INPUT: $M \neq 0, d = (d_{n-1}, \ldots, d_0)_2, N$
OUTPUT: $M^d \bmod N$ or "Error"

1. $S[0] \leftarrow 1$
2. $S[1] \leftarrow 1$
3. $A \leftarrow M$
4. for i from 0 to $n-1$ do
5. $\quad S[\overline{d_i}] \leftarrow S[\overline{d_i}] \cdot A \bmod N$
6. $\quad A \leftarrow A^2 \bmod N$
7. if $(M \cdot S[0] \cdot S[1] = A \bmod N)$ and $(A \neq 0)$ then
8. $\quad$ return$(S[0])$
9. else
10. $\quad$ return("Error")

As it can be easily checked, our algorithm is still resistant to SPA: a modular square always follows a modular multiplication, independently of the value of the exponent. We have added two modular multiplications to the original version (Algorithm 2.3). One modular multiplication can be avoided if, at the beginning of the algorithm, $S[1]$ is initialized with the message M. But as we will argue in Sect. 4, the re-use of the message at the end of the algorithm is useful when it comes to protect a CRT RSA that performs exponentiations with Algorithm 3.1. We shall prove in Sect. 3.3 that the coherence check $M \cdot S[0] \cdot S[1] = A \bmod N$ avoids realistic fault attacks when A is not set to zero by the adversary. If A is set to zero, then $S[0]$ and/or $S[1]$ and A shall be null and the coherence check will fail in detecting the fault induction. To prevent such an attack, the verification $A \neq 0$ has been added.

To prove the resistance of our proposal to Fault Attacks, we first have to clarify the capabilities of an attacker. In the following, we define the model in which our algorithm will be proved.

3.2 Attacker Model

As argued in [3] and [8], sensitive applications (e.g. Banking, GSM or Identity Card) cannot make use of countermeasures with ad hoc security but need countermeasures which are provably secure against a precisely modeled adversary. Blömer *et al.* [3], Wagner [8] and, more recently, Lemke-Rust and Paar [21] have introduced adversarial models for Fault Analysis. They consider various natures of faults and attack scenarios with a focus on pervasive computing on low-cost cryptographic devices. The attacker model presented hereafter follows the outlines of those described in [3] and [8]. It is divided into three parts which respectively aim at specifying how the attacker interacts with the device, the kind of variable targeted during the attack and the type of fault.

We shall assume that the attacker is only able to induce one fault per execution of the algorithm (this assumption is discussed in [2]). In [3], Blömer *et al.* identified three different ways to induce faults on an algorithm.

1. Modification of the input parameters [22].
2. Modification of the algorithm execution [23].
3. Modification of the local variables [3].

A powerful adversary is able to induce a fault in the three different manners listed above and nowadays devices are usually provided with hardware mechanisms that render the task of such an adversary as difficult as possible. The adding of redundancy by hardware functions (*e.g.* based on error correcting codes or on hash functions) is often sufficiently effective to protect an implementation against permanent modification of input parameters (first model). Hardware mechanisms can also be successfully involved to guarantee the correctness of an algorithm execution (second model) and they give confidence that the algorithm does not end before all the exponent bits are processed [23]. Even if they are effective and efficient to counteract fault inductions of types 1 and 2, hardware mechanisms are rarely able to thwart attacks based on the perturbation of local variables. Defeating such attacks is usually the main role of software countermeasures. In the rest of the paper, we shall consider an adversary that modifies local variables, assuming that the security against the two other kinds of fault inductions is carried out by the Hardware.

Remark 2. In Appendix A, we propose a slightly modified version of Algorithm 4.1 in which a simple mechanism has been added to counteract some fault injections belonging to the first and the second categories of faults. This version may be used when the effectiveness of some hardware countermeasures is in doubt. It allows to check that the loop has been entirely executed and that the exponent d used during the calculation (and temporarily stored in RAM) has not been modified and equals the exponent d stored in the non-volatile memory of the device.

Let X denote the value of a n-bit local variable and let $\widetilde{X}$ denote the corresponding faulty value. From X and $\widetilde{X}$ one can deduce an *error vector* ε such

that $\tilde{X} = X + \varepsilon$. The nature of the error vectors ε essentially depends on the adversary type: a strong adversary shall be able to disturb the value of a local variable at a very precise position (*e.g.* a bit modification at a given position), whereas a weak adversary could induce a fault but could not determine its position or its value. Blömer *et al.* exhibited in [3] four different kinds of fault. We recall their classification hereafter.

1. *Precise Bit Errors.* In the strongest scenario, an attacker can change the value of one bit: $\widetilde{X} = X \pm 2^k$ for $0 \leq k \leq n-1$
2. *Precise Byte Errors.* One selected byte is affected by the attack: $\widetilde{X} = X \pm b \cdot 2^k$ for a known $0 \leq k \leq n-8$ and an unknown $0 \leq b \leq 255$
3. *Unknown Byte Errors.* One random byte is affected by the attack: $\widetilde{X} = X \pm b \cdot 2^k$ for a unknown $0 \leq k \leq n-8$ and an unknown $0 \leq b \leq 255$
4. *Random Errors.* An attacker has no knowledge of the modification: $\widetilde{X} = X \pm f(X)$ for $0 \leq f(X) \leq 2^{n-1}$

In our security proof exhibited in the next section, we shall not need to focus on a type of fault in particular and we will prove that our proposal is secure whatever the nature of the fault ε induced by the adversary.

3.3 Security Proof

The message M being assumed to be not null, it can be easily checked that A cannot equal 0 if no fault is introduced. An attack consisting in setting A to zero during the execution of the loop is thwarted by the second test at Step 7. In the rest of this section, we argue that the first test at Step 7 allows to detect any other kind of fault induction in the model described in Sect.3.2.

Wagner proposed in [8] a framework to prove the resistance of an algorithm against Fault Attacks. He suggests that the algorithm be divided into a succession of finite states that correspond to single step computations and to study how faults propagate throughout the algorithm. Such an analysis allows to establish that the fault is either detected by the algorithm or cannot be exploited by the attacker.

The algorithm is split up in such a way that the initial state corresponds to the input of the algorithm and the final state corresponds to the output. All normal transitions between intermediate states are represented by $\rightsquigarrow$. A Fault Attack between intermediate states is symbolized by $\twoheadrightarrow$.

Algorithm 3.1 involves the three variables $S[0]$, $S[1]$ and A. In Wagner's framework, the algorithm execution can be represented by the three following schemes above:

$$\begin{aligned} 1 &\rightsquigarrow M^{d_0} \rightsquigarrow M^{d_1 \cdot 2 + d_0} \rightsquigarrow \ldots \rightsquigarrow M^{d} \\ 1 &\rightsquigarrow M^{\overline{d_0}} \rightsquigarrow M^{\overline{d_1} \cdot 2 + \overline{d_0}} \rightsquigarrow \ldots \rightsquigarrow M^{\overline{d}} \\ M &\rightsquigarrow M^2 \rightsquigarrow M^4 \rightsquigarrow \ldots \rightsquigarrow M^{2^n} \end{aligned}$$

To prove that the coherence test at the end of our algorithm detects any error during the computation of the three variables, we simulate a fault in a random state $i+1$ for the three schemes above:

1. Attack changing the content of $S[0]$:

$$1 \rightsquigarrow M^{d_0} \rightsquigarrow \ldots \rightsquigarrow M^{\sum_{j=0}^{i-1} d_j \cdot 2^j} \twoheadrightarrow M^{\sum_{j=0}^{i} d_j \cdot 2^j} + \varepsilon \rightsquigarrow \ldots \rightsquigarrow \widetilde{M^d}$$

The wrong state $M^{\sum_{j=0}^{i} d_j \cdot 2^j} + \varepsilon$ implies a final state $\widetilde{M^d}$ satisfying:

$$\widetilde{M^d} = (M^{\sum_{j=0}^{i} d_j \cdot 2^j} + \varepsilon) \cdot (M^{\sum_{j=i+1}^{n-1} d_j \cdot 2^j}),$$

that is $\widetilde{M^d} = M^d + \varepsilon \cdot (M^{\sum_{j=i+1}^{n-1} d_j \cdot 2^j})$ which differs from M^d if ε and M are not equal to 0 modulo N.

2. Attack changing the content of $S[1]$. In a similar way, a disturbance of $S[1]$ at any moment results in the following state:

$$\widetilde{M^{\overline{d}}} = M^{\overline{d}} + \varepsilon \cdot (M^{\sum_{j=i+1}^{n-1} \overline{d_j} \cdot 2^j})$$

which differs from $M^{\overline{d}}$ if ε and M are not equal to 0 modulo N.

3. Attack changing the content of A:

$$M \rightsquigarrow M^2 \rightsquigarrow \ldots \rightsquigarrow M^{2^{i-1}} \twoheadrightarrow M^{2^i} + \varepsilon \rightsquigarrow \ldots \rightsquigarrow \widetilde{M^{2^n}}$$

Contrary to the two previous cases, attacking A at a random state $i+1$ impacts the content of the two others registers $S[0]$ and $S[1]$ at state $i+2$. To better analyze this error propagation, let us rewrite the error in a multiplicative way:

- If M^{2^i} is co-prime with N, we deduce from the additive error ε the multiplicative error β such that:

$$M^{2^i} + \varepsilon = M^{2^i} \cdot (1 + \varepsilon \cdot M^{-2^i}) = M^{2^i} \cdot \beta$$

- If M^{2^i} is not co-prime with N, we denote by z the least common multiple of M and N. The error β is such that:

$$M^{2^i} + \varepsilon = M^{2^i} \cdot (1 + \varepsilon \cdot z \cdot \left(\frac{M}{z}\right)^{-2^i}) = M^{2^i} \cdot \beta$$

So, the different states of the three variables are the following

$$\begin{aligned}
1 &\rightsquigarrow M^{d_0} \rightsquigarrow \ldots \rightsquigarrow M^{\sum_{j=0}^{i} d_j \cdot 2^j} \rightsquigarrow M^{\sum_{j=0}^{i+1} d_j \cdot 2^j} \cdot \beta^{d_{i+1}} \rightsquigarrow \ldots \\
1 &\rightsquigarrow M^{\overline{d_0}} \rightsquigarrow \ldots \rightsquigarrow M^{\sum_{j=0}^{i} \overline{d_j} \cdot 2^j} \rightsquigarrow M^{\sum_{j=0}^{i+1} \overline{d_j} \cdot 2^j} \cdot \beta^{\overline{d_{i+1}}} \rightsquigarrow \ldots \\
M &\rightsquigarrow M^2 \rightsquigarrow \ldots \rightsquigarrow M^{2^{i-1}} \twoheadrightarrow M^{2^i} \cdot \beta \rightsquigarrow M^{2^{i+1}} \cdot \beta^2 \rightsquigarrow \ldots
\end{aligned}$$

and the contents of $S[0]$, $S[1]$ and A finally equal $M^d \cdot \beta^{\sum_{j=i+1}^{n-1} d_j \cdot 2^{j-(i+1)}}$, $M^{\overline{d}} \cdot \beta^{\sum_{j=i+1}^{n-1} \overline{d_j} \cdot 2^{j-(i+1)}}$ and $M^{2^n} \cdot \beta^{2^{n-i}}$ respectively.

When applying our verification formula, we get:

$$M \cdot S[0] \cdot S[1] = M \cdot M^d \cdot \beta^{\sum_{j=i+1}^{n-1} d_j \cdot 2^{j-(i+1)}} \cdot M^{\overline{d}} \cdot \beta^{\sum_{j=i+1}^{n-1} \overline{d_j} \cdot 2^{j-(i+1)}}$$
$$= M^{2^n} \cdot \beta^{2^{n-i-1}},$$

which is different from the value $M^{2^n} \cdot \beta^{2^{n-i}}$ if the original error ε was not equal to 0.

Remark 3. The error β may have the undesired property that there exists some value k (lower than $n-i-1$) such that $\beta^{2^k} \equiv 1 \mod N$. However, it has been shown in [24] that those values are extremely rare. For instance if N is a RSA modulus equal to the product of two primes p and q, then we have $\beta^{2^k} \equiv 1 \mod N$ iff $2^{n-i-1} \equiv 0 \mod \text{lcm}(f_p, f_q)$, where f_p and f_q are the orders of β modulo p and q respectively. If p and q are such that $p-1$ and $q-1$ are not divisible by large powers of 2, then the probability that this equality holds is comparable to the probability of factoring N by randomly picking one of its prime factors.

Consequently, any error in an intermediate state of the three variables will result in an erroneous result. Thus, we prove that the final check of our algorithm detects any disturbance of any variable during any step of the computation.

4 CRT RSA Resistant to Fault Attacks

In the previous section, we introduced an exponentiation algorithm and proved its security in a realistic fault model. However, even if the two modular exponentiations in the CRT RSA algorithm have not yet been compromised, the correctness of the whole algorithm is not guaranteed. Indeed, it has been shown in [2] that Garner's recombination can be successfully attacked using FA techniques.

The following algorithms use the same principle as the method described in [5] and [20]. A secure modular exponentiation algorithm (Algorithm 3.1) is used to prevent faults during the two exponentiations in the CRT RSA algorithm. Then, additional information given by this secure modular exponentiation is employed to check that the recombination step was not disturbed.

4.1 First Method

Algorithm 3.1 can be used to strengthen the security of a CRT RSA implementation but it has to be slightly modified. Instead of always returning the result of the exponentiation, it returns the three variables if they satisfy Equality (1). Garner's Algorithm is then applied three times, and finally a check is performed to verify that those results satisfy an equality we exhibit below. The goal of this coherence verification is to protect the recombination step.

Proposal We denote by l the bit-length of the secret moduli. Our CRT-RSA algorithm protected against FA is:

Algorithm 4.1 FA-Resistant RSA Signature using CRT
INPUT: $M \neq 0, p, q, d_p, d_q, A$, and l the bit-length of p and q
OUTPUT: S or "Error"

1. $(S_p, S'_p, T_p) \leftarrow (M^{d_p} \bmod p, M^{2^l - d_p - 1} \bmod p, M^{2^l} \bmod p)$
2. $(S_q, S'_q, T_q) \leftarrow (M^{d_q} \bmod q, M^{2^l - d_q - 1} \bmod q, M^{2^l} \bmod q)$
3. $S \leftarrow ((S_q - S_p) \cdot A \bmod q) \cdot p + S_p$
4. $S' \leftarrow ((S'_q - S'_p) \cdot A \bmod q) \cdot p + S'_p$
5. $T \leftarrow ((T_q - T_p) \cdot A \bmod q) \cdot p + T_p$
6. if $M \cdot S \cdot S' = T \bmod N$ then
7. return(S)
8. else
9. return("Error")

Correctness We now consider the relevance of the coherence test in Step 6. First, let us denote by $\overline{d}, \overline{d_p}$ and $\overline{d_q}$ the binary complements of the values d, d_p and d_q respectively. They all satisfy:

$$d + \overline{d} = 2^{2l} - 1, \;\; d_p + \overline{d_p} = 2^l - 1, \;\; d_q + \overline{d_q} = 2^l - 1.$$

Moreover, by definition of d_p, there exists an integer k such that $d = d_p + k \cdot (p-1)$. Thus, we have:

$$\begin{aligned} d &= d_p + k \cdot (p-1), \\ 2^{2l} - 1 - \overline{d} &= 2^l - 1 - \overline{d_p} + k \cdot (p-1), \\ \overline{d} - 2^{2l} + 2^l &= \overline{d_p} - k \cdot (p-1) \;\; . \end{aligned}$$

In a same manner, for an integer k' we have:

$$\overline{d} - 2^{2l} + 2^l = \overline{d_q} - k' \cdot (q-1) \;\; .$$

Due to the Chinese Remainder Theorem, the result of the Garner's recombination of $S'_p = M^{\overline{d_p}} \bmod p$ with $S'_q = M^{\overline{d_q}} \bmod q$ is $S' = M^{\overline{d} - 2^{2l} + 2^l} \bmod N$. Thus, since $\overline{d}$ equals $2^{2l} - d - 1$, the value S' satisfies $S' = M^{2^l - d - 1} \bmod N$.

After multiplying S' by the signature S and the message M (Step 6), we get $M \cdot S \cdot S' = M \cdot M^d \cdot M^{2^l - d - 1} \bmod N$ that is $M \cdot S \cdot S' = M^{2^l} \bmod N$.

The fifth step of Algorithm 4.1 computes the value $T = M^{2^l} \bmod N$. Consequently, if no error occurs during the execution of the CRT RSA algorithm, then the four values M, S, S' and T must satisfy the equality:

$$M \cdot S \cdot S' = T \bmod N.$$

Security The security of Algorithm 4.1 with respect to FA is straightforwardly deduced from the coherence test and the analysis done in Sect. 3 (it thwarts in particular the recent attack [25]). The Square and Multiply Always structure of the algorithm makes it resistant against known-plaintext SPA attacks. SPA-Attacks assuming that the messages can be chosen by the adversary (*e.g.* [26,27]) are out of the scope of this paper. Classical countermeasures such as the randomization of M (see for instance [28]) can be used together with our SPA/FA countermeasure to counteract such attacks by rendering the value of M unpredictable. The use of the message at the end of Algorithm 4.1 (during the last check) protects against modification of the message before one of the two exponentiations and thwarts the attack described in [8]. To insure the validity of the other input parameters of Algorithm 4.1, hardware mechanisms may be used (for instance in order to check the CRC value of each parameter).

Complexity This method requires adding only two Garner's recombinations and two modular multiplications to the classical CRT RSA algorithm. However, memory consumption is larger. Four l-bit values and two additional $2l$-bit values are required compared to non-protected implementations.

As an alternative, we propose the following algorithm which detects an error with some probability.

4.2 Second Method

Our second proposal uses less memory than the previous one, but the coherence verification is made with a probability error, depending on the bit-length b of a *security parameter* r. This means that an error can remain undetected with a probability equal to $\frac{1}{2^b}$. In the following, we decided to choose a 32-bit parameter b, which is a good compromise between security and efficiency:

Our memory-optimized version of Algorithm 4.1 is:

Algorithm 4.2 FA-Resistant RSA Signature using CRT
INPUT: $M \neq 0, p, q, d_p, d_q, A$, and l the bit-length of p and q
OUTPUT: S or "Error"

1. $(S_p, S'_p, T_p) \leftarrow (M^{d_p} \bmod p, M^{2^l - d_p - 1} \bmod p, M^{2^l} \bmod p)$
2. $(S_q, S'_q, T_q) \leftarrow (M^{d_q} \bmod q, M^{2^l - d_q - 1} \bmod p, M^{2^l} \bmod q)$
3. $r \leftarrow$ 32-bit random number
4. $R_p \leftarrow T_p \bmod r$
5. $R_q \leftarrow T_q \bmod r$
6. $S \leftarrow ((S_q - S_p) \cdot A \bmod q) \cdot p + S_p$
7. if $(M \cdot S \cdot S'_p \bmod p) \neq R_p \bmod r$ then
8. return("Error")
9. if $(M \cdot S \cdot S'_q \bmod q) \neq R_q \bmod r$ then
10. return("Error")
11. return(S)

The accuracy of the proposed algorithm comes from the definition of the modular exponentiation algorithm employed. The value R_p computed before the recombination is equal to:

$$\begin{aligned} R_p &= T_p \bmod r \\ R_p &= (M^{2^l} \bmod p) \bmod r \ . \end{aligned}$$

The first check computes the value:

$$\begin{aligned} M \cdot S \cdot S'_p &= (M \cdot M^{2^l-1} \bmod p) \bmod r \\ &= (M^{2^l} \bmod p) \bmod r \\ &= R_p \ . \end{aligned}$$

In a similar way, the last verification step is coherent:

$$\begin{aligned} M \cdot S \cdot S'_q &= (M^{2^l} \bmod q) \bmod r \\ &= R_q \ . \end{aligned}$$

This method requires two additional comparisons with respect to the previous one. But these comparisons are made on values of the same length as p (or q), whereas the comparison in Algorithm 4.1 involves values of same length as N.

Algorithm 4.2 does not require the storage of the l-bit values $M^{2^l} \bmod p$ and $M^{2^l} \bmod q$, during the Garner recombination. Only three 32-bits values must be stored. Instead of T_p and T_q, we store their remainders modulo the 32-bit random number r. Also, the computation and the storage of the public modulus N is no more required.

This optimized version ensures that no attack occurs during the recombination step with a detection probability, which can be parameterized following the bit-size b of the security parameter r.

5 Conclusion

In this paper, we propose a modular exponentiation algorithm that is resistant to Fault Attacks and Simple Power Analysis. We formally prove that this algorithm thwarts classical Fault Attacks and Safe Error Attacks in a realistic and practical fault model. Moreover the timing/memory overhead incurred by the security add-on is quite reasonable. Compared to the classical modular exponentiation algorithms that are resistant to SPA, it requires only two additional modular multiplications.

We show that this exponentiation algorithm can be used to strengthen both versions of the CRT RSA signature algorithm against Fault Attacks. In the first

one, only two additional Garner's recombinations and two modular multiplications are needed. And in the second one, only two modular reductions and two modular multiplications are required. These additional overheads do not considerably increase execution time, particularly when compared to the overhead of computing the signature twice over, and is well suited for use on low-resource devices.

Further, the method proposed for computing modular exponentiation in a secure way could be used to compute scalar multiplication of points over the group defined by the points of an elliptic curve.

References

1. Boneh, D., DeMillo, R., Lipton, R.: On the Importance of Checking Cryptographic Protocols for Faults. In Fumy, W., ed.: Advances in Cryptology – EUROCRYPT '97. Volume 1233 of Lecture Notes in Computer Science., Springer (1997) 37–51
2. Aumüller, C., Bier, P., Fischer, W., Hofreiter, P., Seifert, J.P.: Fault attacks on RSA with CRT: Concrete Results and Practical Countermeasures. In Kaliski Jr., B., Koç, Ç., Paar, C., eds.: Cryptographic Hardware and Embedded Systems – CHES 2002. Volume 2523 of Lecture Notes in Computer Science., Springer (2002) 260–275
3. Blömer, J., Otto, M., Seifert, J.P.: A New RSA-CRT Algorithm Secure Against Bellcore Attacks. In Jajodia, S., Atluri, V., Jaeger, T., eds.: ACM Conference on Computer and Communications Security – CCS'03, ACM Press (2003) 311–320
4. Ciet, M., Joye, M.: Practical Fault Countermeasures for Chinese Remaindering Based RSA. In Breveglieri, L., Koren, I., eds.: Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC'05. (2005) 124–132
5. Giraud, C.: Fault Resistant RSA Implementation. In Breveglieri, L., Koren, I., eds.: Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC'05. (2005) 142–151
6. Shamir, A.: Improved method and apparatus for protecting public key schemes from timing and fault attacks. International Patent Number : WO 98/52319 (1998) Also presented at the rump session of EUROCRYPT'97.
7. Yen, S.M., Kim, S.J., Lim, S.G., Moon, S.J.: RSA Speedup with Residue Number System Immune against Hardware Fault Cryptanalysis. In Kim, K., ed.: Information Security and Cryptology – ICISC 2001. Volume 2288 of Lecture Notes in Computer Science., Springer (2001) 397–413
8. Wagner, D.: Cryptanalysis of a Provable Secure CRT-RSA Algorithm. In Pfitzmann, B., Liu, P., eds.: ACM Conference on Computer and Communications Security – CCS'04, ACM Press (2004) 82–91
9. Blömer, J., Otto, M.: Wagner's Attack on a Secure CRT-RSA Algorithm Reconsidered. In: Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC'06. (2006)
10. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM **21** (1978) 120–126
11. Couvreur, C., Quisquater, J.J.: Fast decipherment algorithm for RSA public-key cryptosystem. Electronics Letters **18** (1982) 905–907
12. Garner, H.: The residue number system. IRE Transactions on Electronic Computers **8** (1959) 140–147

13. Kocher, P.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Koblitz, N., ed.: Advances in Cryptology – CRYPTO '96. Volume 1109 of Lecture Notes in Computer Science., Springer (1996) 104–113
14. Kocher, P., Jaffe, J., Jun, B.: Differential Power Analysis. In Wiener, M., ed.: Advances in Cryptology – CRYPTO '99. Volume 1666 of Lecture Notes in Computer Science., Springer (1999) 388–397
15. Messerges, T., Dabbish, E., Sloan, R.: Power analysis attacks on modular exponentiation in smartcards. In Koç, Ç., Paar, C., eds.: Cryptographic Hardware and Embedded Systems – CHES '99. Volume 1717 of Lecture Notes in Computer Science., Springer (1999) 144–157
16. Menezes, A., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography. CRC Press (1997) Electronic version available at http://www.cacr.math.uwaterloo.ca/hac/.
17. Coron, J.S.: Resistance Against Differential Power Analysis for Elliptic Curve Cryptosystems. In Koç, Ç., Paar, C., eds.: Cryptographic Hardware and Embedded Systems – CHES '99. Volume 1717 of Lecture Notes in Computer Science., Springer (1999) 292–302
18. Joye, M., Lenstra, A., Quisquater, J.J.: Chinese Remaindering Based Cryptosystems in the Presence of Faults. Journal of Cryptology **12** (1999) 241–246
19. Yen, S.M., Joye, M.: Checking before output may not be enough against fault-based cryptanalysis. IEEE Transactions on Computers **49** (2000) 967–970
20. Joye, M., Yen, S.M.: The Montgomery Powering Ladder. In Kaliski Jr., B., Koç, Ç., Paar, C., eds.: Cryptographic Hardware and Embedded Systems – CHES 2002. Volume 2523 of Lecture Notes in Computer Science., Springer (2002) 291–302
21. Lemke-Rust, K., Paar, C.: An Adversarial Model for Fault Analysis Against Low-Cost Cryptographic Devices. In Breveglieri, L., Koren, I., Naccache, D., Seifert, J.P., eds.: Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC'06. Volume 4236 of Lecture Notes in Computer Science., Springer (2006) 131–143
22. Yen, S.M., Moon, S., Ha, J.C.: Permanent Fault Attack on RSA with CRT. In Safavi-Naini, R., Seberry, J., eds.: Information Security and Privacy - 8th Australasian Conference – ACISP 2003. Volume 2727 of Lecture Notes in Computer Science., Springer (2003) 285–296
23. Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C.: The Sorcerer's Apprentice Guide to Fault Attacks. In Breveglieri, L., Koren, I., eds.: Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC'04, IEEE Computer Society (2004) 330–342
24. Fumaroli, G., Vigilant, D.: Blinded Fault Resistant Exponentiation. In: Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC'06. (2006) Available from http://eprint.iacr.org/.
25. Boreale, M.: Attacking Right-to-Left Modular Exponentiation with Timely Random Faults. In Breveglieri, L., Koren, I., Naccache, D., Seifert, J.P., eds.: Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC'06. Volume 4236 of Lecture Notes in Computer Science., Springer (2006) 24–35
26. Fouque, P.A., Valette, F.: The Doubling Attack: Why Upwards is better than Downwards. In Walter, C., Koç, Ç., Paar, C., eds.: Cryptographic Hardware and Embedded Systems – CHES 2003. Volume 2779 of Lecture Notes in Computer Science., Springer (2003) 269–280
27. Yen, S.M., Lien, W.C., Moon, S.J., Ha, J.C.: Power Analysis by Exploiting Chosen Message and Internal Collisions – Vulnerability of Checking Mechanism for RSA-

Decryption. In Dawson, E., Vaudenay, S., eds.: Progress in Cryptology – Mycrypt 2005. Volume 3715 of Lecture Notes in Computer Science., Springer (2005) 183–195
28. Chaum, D.: Blind signatures for untraceable payments. In Chaum, D., Rivest, R., Sherman, A., eds.: Advances in Cryptology – CRYPTO '82, Plenum Press (1982) 199–204
29. Otto, M.: Fault Attacks and Countermeasures. PhD thesis, Universität Paderborn (2004)

A SPA/FA resistant exponentiation with software checking of the exponent

Algorithm A.1 SPA/FA Resistant Right-to-Left Modular Exponentiation - 2nd version

INPUT: $M \neq 0, d = (d_{n-1}, \ldots, d_0)_2, N$
OUTPUT: $M^d \bmod N$ or "Error"

1. $S[0] \leftarrow 1$
2. $S[1] \leftarrow 1$
3. $A \leftarrow M$
4. ectrl $\leftarrow 0$
5. for i from 0 to $n-1$ do
6. $\quad S[\overline{d_i}] \leftarrow S[\overline{d_i}] \cdot A \bmod N$
7. $\quad$ ectrl $\leftarrow$ ectrl $+ 2^n \cdot d_i$
8. $\quad A \leftarrow A^2 \bmod N$
9. $\quad$ ectrl $\leftarrow$ ectrl$/2$
10. if $(M \cdot S[0] \cdot S[1] = A \bmod N)$ and (ectrl $= d$) and $(A \neq 0)$ then
11. $\quad$ return$(S[0])$
12. else
13. $\quad$ return("Error")