Fault Resistant RSA Signatures: Chinese Remaindering in Both Directions

Arnaud Boscher
Irdeto
Jupiterstraat 42
2132 HD Hoofddorp
The Netherlands
Arnaud.Boscher@irdeto.com

Helena Handschuh
K.U.Leuven, ESAT/SCD
Kasteelpark Arenberg 10
B-3001 Leuven-Heverlee
Belgium
HelenaHandschuh@yahoo.fr

Elena Trichina
STMicroelectronics
190 Avenue Clestin Coq
13106 Rousset Cedex
France
Elena.Trichina@st.com

January 25, 2010

Abstract

Fault attacks are one of the most severe attacks against secure embedded cryptographic implementations. Block ciphers such as AES, DES or public key algorithms such as RSA can be broken with as few as a single or a handful of erroneous computation results. Many countermeasures have been proposed both at the algorithmic level and using ad-hoc methods. In this paper, we address the problem of finding efficient countermeasures for RSA signature computations based on the Chinese Remainder Theorem for which one uses the inverse operation (verification) in order to secure the algorithm against fault attacks. We propose new efficient methods with associated security proofs in two different models; our methods protect against run-time errors, computation errors, and most permanent errors in the key parameters as well. We also extend our methods with infective computation strategies to secure the algorithm against double-faults.

Keywords: Differential Fault Analysis, RSA, Chinese Remainder Theorem, Smart Cards.

1 Introduction

Fault attacks on public key algorithms such as RSA [21] go all the way back to the original 1997 paper by Boneh et al. [5]. It is shown that a single faulty RSA signature computation using the Chinese Remaindering Theorem (CRT) [9] and a correct one typically allow to retrieve the secret signature exponent from a secure embedded device. In follow-up papers such as [14], a single incorrect signature and its original message is sufficient to recover the secret primes of the public modulus thanks to a simple GCD algorithm. Following this trend of research, many new ideas have been proposed to efficiently secure CRT based RSA signature computations against fault attacks. These ideas of detecting errors can be categorized in three main lines of thought.

First of all, there is Shamir's method [22] and its variants [15, 26, 3, 4, 18, 24]. Instead of computing the results modulo the private primes p and q, one computes the 2 exponentiations modulo the product of each modulus by a small random value r. In the comparison step, both results are reduced modulo r and compared to be equal. Matching results guarantee that no fault has occurred during the exponentiations. However, this method does not allow to protect the final recombination step of the CRT RSA computation.

The second line of thought consists in self-secure exponentiation [12, 11, 7, 6, 20]. The principle is to use internal variables from the exponentiation algorithm to ensure they satisfy some relation between themselves. For example, this can be achieved with the Montgomery powering ladder [19, 16] for exponentiation in which two related values are available at the end of the computation, namely $M^d \bmod N$ and $M^{d-1} \bmod N$, where M is the message, d the private exponent and N the modulus. If one then multiplies the second result by the message $M \bmod N$ one more time, one obtains the first result; this serves as a basis for verification that no error occurred during each exponentiation. Those kinds of additional values are then also used to protect the recombination step of the CRT RSA. However, those methods impose the exponentiation algorithms which are always slower and more memory demanding than the classical way of performing exponentiation with the help of the left-to-right binary exponentiation algorithm.

Finally, the third line of research simply follows the idea that the inverse operation can be applied to check the validity of a result, i.e. one can verify the signature before outputting it. This seems the most natural way to protect implementations against fault attacks but suffers from some drawbacks. In practise, when looking at smart card implementations, the public exponent e which allows to verify a signature is rarely available, and recomputing it as well as computing a full-length verification can take a lot of time. Other methods such as the embedding method [13] in which the public exponent is visually embedded into the public modulus N necessarily modify the key generation procedure and do not always comply with existing standards [2].

Our contribution. Our solutions in this paper focus on the third line of research and propose more efficient methods for retrieving the public exponent

and for computing the verification process using the CRT method. We thus present ways of accelerating this computation method which guarantee that computations are made only modulo p and modulo q, which is about four times faster than a direct computation and uses much smaller operands. This is particularly interesting for crypto-coprocessors which have a limited operand size or for which computation times become prohibitive when operands get huge (such as 4096-bit RSA signature operations). Our methods, compared to others, are also able to detect most errors in the stored key parameters themselves. This means that not only computation errors, but also faults targeted at (non-volatile) memory locations can be taken into account. In addition, our solution is oblivious to the exponentiation algorithm itself; any exponentiation algorithm will work just fine. It complies with existing standards and does not require any modification in the on-board key generation algorithms. Finally, performance-wise, our solutions do not require any major overhead in performance, execution time, RAM memory or ROM code storage as compared to unprotected solutions.

2 Differential Fault Analysis of CRT RSA

2.1 Description

First of all, let us recall the basic procedure for a signature computation using the Chinese Remainder Theorem (CRT).

In the classical definition of the RSA public key cryptosystem, the public key is composed of the public modulus N, product of two large secret primes p and q, and the public exponent e. The corresponding private key is made of the same modulus N and the private exponent d, defined as $d = e^{-1} \bmod (p-1) \cdot (q-1)$. The signing operation of a message M is the computation of the value $S = M^d \bmod N$, whereas the verification of the signature checks that the value $S^e \bmod N$ is indeed equal to the original message M.

In order to speed-up the RSA signature computation S of a message M by almost a factor 4, the Chinese Remainder Theorem is employed. Instead of one exponentiation modulo N with a $\log_2(N)$-bit exponent, two exponentiations modulo p and q each with a $\frac{1}{2} \cdot \log_2(N)$-bit exponent are computed:

$$S_p = M^{d \bmod (p-1)} \bmod p, S_q = M^{d \bmod (q-1)} \bmod q.$$

Those two values are then recombined to give the correct signature S:

$$S = S_p \cdot q \cdot (q^{-1} \bmod p) + S_q \cdot p \cdot (p^{-1} \bmod q) \bmod N$$

An alternative recombination step is depicted in Algorithm 1. This solution is preferred for practical implementations as only one value needs to be pre-computed and no computation modulo N is required. With this version, the RSA private key representation is the following five-tuple $(p, q, d_p = d \bmod (p-1), d_q = d \bmod (q-1), i_q = q^{-1} \bmod p)$.

Then a simple way to defeat a CRT RSA implementation using a fault attack is to induce an error in one of the two modular exponentiations. If the

Algorithm 1 RSA signature computations using the CRT

Inputs: Message M, RSA private key in CRT representation
Outputs: Signature S on message M

1: $S_p \leftarrow M^{d_p} \bmod p$
2: $S_q \leftarrow M^{d_q} \bmod q$
3: $S \leftarrow ((S_p - S_q) \cdot i_q \bmod p) \cdot q + S_q$

first exponentiation is targeted, the computation of S_p becomes corrupted and results in a wrong value $\widetilde{S_p} = S_p + \varepsilon$ and therefore a wrong signature value $\widetilde{S}$. Now subtracting the erroneous signature value from the correct one, one is left with $\widetilde{S} - S = \varepsilon \cdot q \cdot (q^{-1} \bmod p)$ which is a multiple of q. Therefore a simple GCD algorithm with $N = p \cdot q$ will reveal the secret key.

And one can show that the same holds for a computation error on S_q, $\widetilde{S} - S$ will be a multiple value of p this time.

2.2 Verification Using the Public Exponent

Now as a countermeasure, the most simple solution to check that a CRT RSA computation was not disturbed is to apply the inverse operation on the output and control that the original value is retrieved. This method has two main advantages. Usually, the public exponent is a small value, making this verification computation very fast. It also protects the implementation against some attacks which can permanently modify the private parameters of the RSA key [28].

A potential drawback of this method is the need to make computations on integers of the size of the public modulus, whereas the goal of using the Chinese Remainder Theorem is indeed to avoid such large computations. Hence, with a hardware public key co-processor operating on inputs of n bits, it is possible to compute RSA signatures of $2 \cdot n$ bits with the help of the Chinese Remainder Theorem. But the verification operation requires much more complex solutions involving size-doubling techniques [8].

Also, it is not always possible to apply this countermeasure in a simple way. Indeed, the public exponent is not always directly available, see for instance [23], since, from a functional point of view, there is absolutely no reason to use it to compute a CRT RSA signature value. Obviously, the public exponent can be recomputed from the private RSA key in its CRT representation, but this operation is quite time consuming, especially on embedded devices with limited capabilities. The RSA public exponent can be computed using the generalized Chinese Remainder Theorem presented in Algorithm 2, see [10] for explanations.

In addition to making computations in the range of the public modulus, this algorithm requires the use of a division operation which is not often available on hardware public key accelerators. Hence, this solution is not very suitable for low-cost devices such as smart cards.

Algorithm 2 Compute the RSA public exponent from a CRT RSA private key

Inputs: RSA private key in CRT representation
Outputs: RSA public exponent

1: $(x, y, g) \leftarrow$ Extended Euclidian Algorithm$(p-1, q-1)$
2: $d \leftarrow d_p + (\frac{x \cdot (d_q - d_p)}{g} \bmod \frac{q-1}{g}) \cdot (p-1)$
3: $e \leftarrow d^{-1} \bmod (p-1) \cdot (q-1)$

3 New Solution

3.1 Algorithm

In this section, we present a solution that does not require any division nor any modular computation in the range of the public modulus.

Algorithm 3 Fault Resistant CRT RSA

Inputs: CRT key and message M
Outputs: CRT RSA computation or error detection

1: $S_p \leftarrow M^{d_p} \bmod p$
2: $S_q \leftarrow M^{d_q} \bmod q$
3: $S \leftarrow ((S_p - S_q) \cdot i_q \bmod p) \cdot q + S_q$
4: $e_p \leftarrow d_p^{-1} \bmod (p-1)$
5: $e_q \leftarrow d_q^{-1} \bmod (q-1)$
6: $M_p \leftarrow S^{e_p} \bmod p$
7: $M_q \leftarrow S^{e_q} \bmod q$
8: $M' \leftarrow ((M_p - M_q) \cdot i_q \bmod p) \cdot q + M_q$
9: **if** $M' = M$ **then**
10: **return** S
11: **else**
12: **return** Error Message
13: **end if**

The cost of this method is two modular inverse operations and an additional CRT computation. As the public exponent is expected to be a small number, the CRT computation is almost for free. The overhead comes from the modular inverse which is a much slower operation, but remains faster than a modular exponentiation.

Note that in specific cases where the public exponent is large and in the range of the public modulus, such as in rebalanced CRT RSA [25], this method is almost four times faster compared to the classical verification due to the benefit of the Chinese Remainder Theorem.

Regarding code storage, the solution only needs to embed the modular inverse computation as the remaining additional operations are already available.

3.2 Optimization

In most cases, the public exponent e is a small value, this means that the three values e, $e_p = e \bmod (p-1)$ and $e_q = e \bmod (q-1)$ are in fact all equal. Hence, after the first modular inversion which results in the value e_p, one can simply perform a single modular multiplication, $e_p \cdot d_q \bmod (q-1)$, and check if the result is equal to 1. This allows to replace one costly modular inverse computation with only one negligible modular multiplication.

Alternatively, if the public exponent is known, the two values e_p and e_q can be stored in the non-volatile memory of the device as two additional key parameters. In this case, our solution is even faster than the classical verification thanks to the Chinese Remainder Theorem.

In order to further reduce memory overhead, the final CRT at step 8 and final check $M' = M$ can be replaced with a set of two checks, namely $M_p = M \bmod p$ and $M_q = M \bmod q$. This allows to avoid the previous full-size operations.

4 Security Analysis

In this section, we discuss the security of our solution in different fault models.

4.1 Possible Fault Models

In order to evaluate the security of the solution against fault attacks, an accurate fault model must first be defined. In the original paper on fault analysis [5], three different kinds of faults were described:

- Latent faults, coming from hardware or software bugs on rare occasions.
- Transient faults, generated by external perturbation on the cryptographic device (power glitch, high temperature ...).
- Induced faults, by physically accessing the device in order to make hardware faults.

Those faults can be classified in two different sets: transient errors (latent and transient faults) and permanent errors (induced faults). In the following, we consider the resistance of our algorithm in those two different fault models.

4.2 Transient Error Fault Model

The transient error fault model is defined by an attacker applying physical perturbation during one computation. This means that the device will still be correctly running after the attack.

We consider a very powerful attacker which is able to make one error consisting in any kind of modification on any intermediate variable (bit flip, byte flips, chosen error on different chosen bytes ...) or even skip some part of the algorithm. Only the three first steps of the algorithm are of interest for an

attacker as the other computations are not part of the result.

We recall that, by defintion of the Chinese Remainder Theorem, a correct RSA signature S can be expressed as follows:

$$S = a \cdot S_p + b \cdot S_q \bmod N$$

with $a = 1 \bmod p, a = 0 \bmod q$ and $b = 0 \bmod p, b = 1 \bmod q$.

Error on S_p

We suppose that the faulty signature, denoted $\widetilde{S}$, results from an error modifying S_p (during Steps 1 or 3). It is possible to write the erroneous signature in the following manner:

$$\widetilde{S} = a \cdot S_p \cdot \varepsilon + b \cdot S_q \bmod N$$

where ε (different from 1) represents the error.

The modular inverse operations (Steps 4 and 5) and the exponentiation modulo q (Step 7) are not impacted by the generated error. But at Step 6, the algorithm computes the value:

$$\widetilde{M_p} = (S_p \cdot \varepsilon)^{e_p} = S_p^{e_p} \cdot \varepsilon^{e_p} = M_p \cdot \varepsilon^{e_p} = M_p + M_p \cdot (\varepsilon^{e_p} - 1) \bmod p.$$

Hence the final recombination at Step 8 computes the value:

$$\begin{aligned} \widetilde{M'} &= (\widetilde{M_p} - M_q) \cdot i_q \bmod p) \cdot q + M_q \\ &= (M_p - M_q) \cdot i_q \bmod p) \cdot q + M_q + (M_p \cdot (\varepsilon^{e_p} - 1) \cdot i_q \bmod p) \cdot q \\ &= M + (M_p \cdot (\varepsilon^{e_p} - 1) \cdot i_q \bmod p) \cdot q. \end{aligned}$$

So the second CRT operation outputs a value different from M when ε is different from 1, meaning when there is an error disturbing the S_p value. This is due to the fact that M_p is unlikely to be 0, since otherwise the input message would be a multiple of the private factor p and one could immediately factor the public modulus using a GCD computation with N without having to attempt a fault attack in the first place.

Error on S_q

When the induced error changes the value S_q, the same behavior is observed but the result of the second CRT computation is slightly different. Hence, if we again denote the error by ε (different from 1), the incorrect signature is expressed as:

$$\widetilde{S} = a \cdot S_p + b \cdot S_q \cdot \varepsilon \bmod N$$

Steps 4, 5 and 6 of the algorithm are correct, but the second exponentiation modulo q computes the quantity:

$$\widetilde{M_q} = (S_q \cdot \varepsilon)^{e_q} = S_q^{e_q} \cdot \varepsilon^{e_q} = M_q \cdot \varepsilon^{e_q} = M_q + M_q \cdot (\varepsilon^{e_q} - 1) \bmod q.$$

And the final CRT recombination outputs the following:

$$\begin{aligned}\widetilde{M'} &= (M_p - \widetilde{M_q}) \cdot i_q \bmod p) \cdot q + \widetilde{M_q} \\ &= (M_p - M_q) \cdot i_q \bmod p) \cdot q + \widetilde{M_q} - (M_q \cdot (\varepsilon^{e_q} - 1) \cdot i_q \bmod p) \cdot q \\ &= M + M_q \cdot (\varepsilon^{e_q} - 1) \bmod q - (M_q \cdot (\varepsilon^{e_q} - 1) \cdot i_q \bmod p) \cdot q\end{aligned}$$

Thus, with an error on S_q, the second CRT operation will compute a value different from the original message M and will detect an attempt to perform a fault attack. This is because the second part of the equation cannot be zero since its first term is smaller than q and the second term is a non-zero multiple of q so both terms cannot cancel each other out.

4.3 Permanent Error Fault Model

Permanent errors can be achieved by manipulating the non volatile memory [1] where the keys are stored. As opposed to transient errors, the device will always be computing wrong values.

This attack is dangerous in the sense that even when using a secure algorithm such as [3, 12, 7, 24, 6, 20], differential fault analysis can still be applied since the secure algorithm manipulates wrong parameters leading to an erroneous result useful for the analysis. Indeed, almost all previous methods known to detect permanent errors during a CRT RSA computation are vulnerable to this kind of attacks and are therefore never considered in the security analysis.

4.3.1 Error on the Modulus p

We suppose that the modulus p is changed such that $\widetilde{p} = p + \varepsilon$. Hence, an erroneous signature $\widetilde{S}$ is computed and can lead to the recovery of the prime factor q. Indeed, the first modular exponentiation calculates a wrong value $\widetilde{S_p}$, and during the recombination phase, the following quantity is involved:

$$((\widetilde{S_p} - S_q) \cdot i_q \bmod \widetilde{p}) = ((S_p - S_q) \cdot i_q \bmod p) + \alpha$$

where α is some unknown value, function of the original error ε. Thus, when the CRT operation is completed, the final result has the expected structure for the GCD calculation: $\widetilde{S} = S + \alpha \cdot q$

To show that this erroneous value is detected by our test, we have to show that the second CRT operation will not recover the original message M, i.e. that the signature and verification operations are not the inverse of each other. However, the exponent e_p used for the verification will also be modified:

$$\widetilde{e_p} = d_p^{-1} \bmod \widetilde{p}$$

In the following we make the assumption that the inverse of d_p modulo $\widetilde{p}$ exists. Otherwise, the computation of the modular inverse through the extended GCD algorithm will recognize that the final GCD value is not 1 and will detect something went wrong and deduce there was a permanent error on p. If such

an inverse exists, it could seem at first sight that the second CRT operation retrieves the original message as the exponents and the moduli are correctly related to each other.

The fact that makes the detection test successful is that the value i_q used in both recombinations is not correct with respect to the two used moduli $\widetilde{p}$ and q: $i_q \neq q^{-1} \bmod \widetilde{p}$. If we consider the two moduli $\widetilde{p}$ and q and the exponents d_p and d_q for the first CRT operation, the algorithm in fact computes a (faulty) signature due to the wrong i_q with respect to the following private key:

- modulus: $\widetilde{p} \cdot q$
- private exponent: $\widetilde{d} = d_p + (\frac{x \cdot (d_q - d_p)}{g} \bmod \frac{q}{g}) \cdot \varphi(\widetilde{p})$, where g is the GCD between $\varphi(\widetilde{p})$ and $q-1$, x one Bezout coefficient and φ the Euler totient function. The exponent $\widetilde{d}$ comes from the generalized Chinese Remainder Theorem (see Algorithm 2)

And the second CRT operation computes a (wrong) verification due to the same i_q value with respect to the following public key:

- modulus: $\widetilde{p} \cdot q$
- public exponent: $\widetilde{e} = \widetilde{e_p} + (\frac{x \cdot (e_q - \widetilde{e_p})}{g} \bmod \frac{q}{g}) \cdot \varphi(\widetilde{p})$

Only when $\widetilde{p}$ is a prime number we have $\widetilde{e} = \widetilde{d}^{-1} \bmod \varphi(\widetilde{p}) \cdot (q-1)$. We are in the same situation as in Section 4.3.3 (see below) where all key parameters are correct except for the modular inverse of one of the moduli.

When $\widetilde{p}$ is a composite number, in addition to having a correct i_q value, the inverse operation would require that $e_p = d_p^{-1} \bmod \varphi(\widetilde{p})$, whereas our algorithm computes and uses the value $d_p^{-1} \bmod \widetilde{p} - 1$.

Remark 1. If $\widetilde{p}$ is in fact a multiple of p, then $i_q = q^{-1} \bmod \widetilde{p}$. However such a faulty modulus will not lead to a wrong signature value useful for the GCD attack. In this case, the final result will be corrupted with a multiple of N.

Remark 2. In specific implementations where the exponents e_p and e_q are pre-computed (and therefore correct), the exponents involved are no longer correctly related (and the value i_q is again not the modular inverse of q modulo $\widetilde{p}$), so the verification process cannot retrieve the original message.

4.3.2 Error on the Modulus q

Interestingly with this specific implementation, a wrong modulus $\widetilde{q} = q + \varepsilon$ used during all the computations will not lead to a wrong result $\widetilde{S}$ useful to reveal one of the secret prime numbers. This is due to the fact that, with such an erroneous modulus, the second exponentiation will be wrong, $\widetilde{S_q} = S_q + \alpha$, but

the recombination step will also be impacted:

$$\begin{aligned}
\widetilde{S} &= ((S_p - \widetilde{S_q}) \cdot i_q \bmod p) \cdot \widetilde{q} + \widetilde{S_q} \\
&= ((S_p - S_q) \cdot i_q \bmod p) \cdot (q + \varepsilon) + S_q + \alpha - (\alpha \cdot i_q \bmod p) \cdot (q + \varepsilon) \\
&= S + \alpha - (\alpha \cdot i_q \bmod p) \cdot (q + \varepsilon) + \varepsilon \cdot ((S_p - S_q) \cdot i_q \bmod p) \\
&= S + \alpha - (\alpha \cdot i_q \bmod p) \cdot q - \varepsilon \cdot ((\alpha \cdot i_q \bmod p) - ((S_p - S_q) \cdot i_q \bmod p))
\end{aligned}$$

And $\widetilde{S} - S$ is not a multiple of one of the secret primes as there is no reason that $\alpha = \varepsilon \cdot ((\alpha \cdot i_q \bmod p) - ((S_p - S_q) \cdot i_q \bmod p))$ or that $\alpha = (S_p - S_q) \bmod p$ and α is a multiple of q.

Notwithstanding, the same proof as for an error on the prime number p applies, as the i_q value used is not $\widetilde{q}^{-1} \bmod p$, meaning that the check will detect the error as the two CRT computations do not involve related correct CRT keys. And also Step 5 may detect an error if the inverse of exponent d_q modulo $\widetilde{q}$ does not exist.

4.3.3 Error on the Pre-Computed Constant i_q

Let us suppose now that the modular inverse of one of the moduli, $i_q = q^{-1} \bmod p$, is perturbated and becomes $\widetilde{i_q} = i_q + \varepsilon$ for some value ε. The faulty signature computed is equal to:

$$\widetilde{S} = ((S_p - S_q) \cdot (i_q + \varepsilon) \bmod p) \cdot q + S_q = S + ((S_p - S_q) \cdot \varepsilon \bmod p) \cdot q$$

When reducing this value modulo q before one of the two exponentiations, we get a correct value modulo q: $\widetilde{S} = S_q \bmod q$.

So, the exponentiation of Step 7 returns: $M_q = (S_q)^{e_q} = M \bmod q$.

But the other reduction will involve the error ε:

$$\widetilde{S} = S_p + X_p \bmod p, \text{ where } X_p = (S_p - S_q) \cdot \varepsilon \cdot q \bmod p.$$

Or writing this value in a multiplicative way:

$$\widetilde{S} = S_p \cdot Y_p \bmod p, \text{ with } Y_p = 1 + S_p^{-1} \cdot (S_p - S_q) \cdot \varepsilon \cdot q \bmod p.$$

Hence, the exponentiation at Step 6 becomes:

$$\widetilde{M_p} = (S_p \cdot Y_p)^{e_p} = M_p \cdot Y_p^{e_p} = M_p + (M_p \cdot (Y_p^{e_p} - 1)) \bmod p$$

And the final CRT recombination gives:

$$\begin{aligned}
\widetilde{M'} &= ((\widetilde{M_p} - M_q) \cdot (i_q + \varepsilon) \bmod p) \cdot q + M_q \\
&= ((\widetilde{M_p} - M_q) \cdot i_q) \bmod p) \cdot q + M_q + (\varepsilon \cdot (\widetilde{M_p} - M_q) \bmod p) \cdot q \\
&= M + (\varepsilon \cdot (\widetilde{M_p} - M_q) \bmod p) \cdot q + (M_p \cdot (Y_p^{e_p} - 1) \cdot i_q \bmod p) \cdot q \\
&= M + q \cdot (\varepsilon \cdot (M_p \cdot Y_p^{e_p} - M_q) + M_p \cdot (Y_p^{e_p} - 1) \cdot i_q \bmod p) \\
&= M + q \cdot ((\varepsilon + i_q) \cdot M_p \cdot Y_p^{e_p} - \varepsilon \cdot M_q - M_p \cdot i_q \bmod p)
\end{aligned}$$

By definition of the error, we have $\varepsilon \neq 0$ and $Y_p^{e_p} \neq 1$, so $(\varepsilon + i_q) \cdot M_p \cdot Y_p^{e_p} \neq M_p \cdot i_q \bmod p$. And since M_p and M_q are not related to each other, the quantity $(\varepsilon + i_q) \cdot M_p \cdot Y_p^{e_p} - \varepsilon \cdot M_q - M_p \cdot i_q \bmod p$ will be different from 0 with high probability.

4.3.4 Error on the Exponents d_p and d_q

We consider two different cases, depending whether the values e_p and e_q are stored pre-computed in memory or rather recomputed during the execution of the algorithm.

Recomputed Parameters. If one of the private exponents is permanently modified, the method will not detect this error in every case. Only when the erroneous exponent, $\widetilde{d_p}$ or $\widetilde{d_q}$, is not invertible can the error be detected. There are $\varphi(p-1)$ values which admit an inverse. The worst case would be when $p - 1 = 2 \cdot p'$ where p' is a prime number (p' is a Sophie Germain prime). In this case the probability that an error remains undetected is $\frac{p'-1}{2 \cdot p'} = \frac{1}{2} - \frac{1}{2 \cdot p'}$ meaning that it is still more likely to detect an error rather than not to.

Therefore, this is an improvement compared to previous known methods where any error in one of the exponents remains undetected and leads to the factorization of the public modulus. And when considering the five parameters of the CRT RSA key, our solution is protected against any modification in three out of these five parameters and in most cases in the two remaining ones, so the detection probability of a permanent error in the key is greater than 0.8 where other solutions are totally unprotected.

Stored Parameters. In this case, the error is detected. Indeed, as the stored public exponents are correct, the second CRT recombination computes the correct verification on a corrupted signature: $\widetilde{S}^e \bmod p \cdot q$. Since by definition of RSA, only $S^e = M \bmod p \cdot q$, we necessarily have $\widetilde{S}^e \neq M \bmod p \cdot q$, and the check detects the error on one private exponent.

5 Infective Method

The infective method [27] is a different approach to protect CRT RSA against fault attacks. Instead of trying to detect an error, the idea is to output a faulty signature useless for the GCD computation.

We propose a variant of our method which combines our detection test and an infective computation in order to protect the implementation against two fault injections during the same signature computation [17]. The result is presented in Algorithm 4. With this algorithm, if the conditional check at Step 9 is bypassed, the output value will be random and useless for the GCD computation.

Algorithm 4 Detection and Infective Fault Resistant CRT RSA

Inputs: CRT key and message M
Outputs: CRT RSA computation or error detection

1: $S_p \leftarrow M^{d_p} \bmod p$
2: $S_q \leftarrow M^{d_q} \bmod q$
3: $S \leftarrow ((S_p - S_q) \cdot i_q \bmod p) \cdot q + S_q$
4: $e_p \leftarrow d_p^{-1} \bmod (p-1)$
5: $e_q \leftarrow d_q^{-1} \bmod (q-1)$
6: $M_p \leftarrow S^{e_p} \bmod p$
7: $M_q \leftarrow S^{e_q} \bmod q$
8: $M' \leftarrow ((M_p - M_q) \cdot i_q \bmod p) \cdot q + M_q$
9: **if** $M' = M$ **then**
10: **return** $(S + M_p - (M \bmod p) + M_q - (M \bmod q))$
11: **else**
12: **return** Error Message
13: **end if**

Again, in order to further reduce memory overhead, the final CRT step 8 and final check $M' = M$ can be replaced with two checks, $M_p = M \bmod p$, $M_q = M \bmod q$, and with a single infective computation step in which the algorithm simply returns $S = S + M_p - (M \bmod p) + M_q - (M \bmod q)$.

As previously, a single modular multiplication can be interleaved between the two modular inverse computations to detect if the value e_q is in fact equal to e_p and one can potentially skip the second modular inverse calculation.

6 Conclusion

In this paper we have presented a new solution to counter faults attacks using the full-fledged power of the Chinese Remaindering Theorem in both computation directions: signature and verification when the private parameters of an RSA signature key are available in the implementation but the public exponent is not. Our solution and its variants can accommodate several fault models including those targeting run-time computation errors and the key parameter storage in memory. Our methods are more efficient than previously described techniques and do not require any major overhead in performance or RAM or ROM memory. They are particularly well suited for low-cost smart card implementations.

References

[1] R. Anderson and M. Kuhn. Low Cost Attacks on Tamper Resistant Devices. In B. Christianson, B. Crispo, T. Mark, A. Lomas, and M. Roe, editors, 5^{th} *Security Protocols Workshop*, volume 1361 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 1997.

[2] ANSI X9.31. *Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA)*. American National Standards Institute, 1998.

[3] 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.S. Kaliski Jr., Ç.K. 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.

[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.A. DeMillo, and R.J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In W. Fumy, editor, *Advances in Cryptology – EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997.

[6] A. Boscher, H. Handschuh, and E. Trichina. Blinded Fault Resistant Exponentiation Revisited. In L. Breveglieri, S. Gueron, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC'09*, pages 3–9. IEEE Computer Society, 2009.

[7] A. Boscher, R. Naciri, and E. Prouff. CRT RSA Algorithm Protected Against Fault Attacks. In D. Sauveron, C. Markantonakis, A. Bilas, and J.-J. Quisquater, editors, *WISTP 2007*, volume 4462 of *Lecture Notes in Computer Science*, pages 229–246. Springer, 2007.

[8] B. Chevallier-Mames, M. Joye, and P. Paillier. Faster Double-Size Modular Multiplication from Euclidean Multipliers. In C.D. Walter, Ç.K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 214–227. Springer, 2003.

[9] C. Couvreur and J.-J. Quisquater. Fast Decipherment Algorithm for RSA Public-Key Cryptosystem. *Electronics Letters*, 18(21):905–907, 1982.

[10] W. Fischer and J.-P. Seifert. Note on Fast Computation of Secret RSA Exponents. In L. Battern and J. Seberry, editors, *Information Security and Privacy - 7th Australasian Conference – ACISP 2002*, volume 2384 of *Lecture Notes in Computer Science*, pages 136–143. Springer, 2002.

[11] G. Fumaroli and D. Vigilant. Blinded Fault Resistant Exponentiation. In L. Breveglieri, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC'06*, volume 4236 of *Lecture Notes in Computer Science*. Springer, 2006. Available from `http://eprint.iacr.org/2006/143.pdf`.

[12] C. Giraud. An RSA Implementation Resistant to Fault Attacks and to Simple Power Analysis. *IEEE Transactions on Computers*, pages 1116–1120, June 2006.

[13] M. Joye. Protecting RSA Against Fault Attacks: The Embedding Method. In L. Breveglieri, S. Gueron, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC'09*, pages 41–45. IEEE Computer Society, 2009.

[14] M. Joye, A.K. Lenstra, and J.-J. Quisquater. Chinese Remaindering Based Cryptosystems in the Presence of Faults. *Journal of Cryptology*, 12(4):241–246, 1999.

[15] M. Joye, P. Paillier, and S.-M. Yen. Secure Evaluation of Modular Functions. In R.J. Hwang and C.K. Wu, editors, *2001 International Workshop on Cryptology and Network Security – CNS 2001*, pages 227–229, 2001.

[16] M. Joye and S.-M. Yen. The Montgomery Powering Ladder. In B.S. Kaliski Jr., Ç.K. 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.

[17] C. H. Kim and J.-J. Quisquater. Fault Attacks for CRT Based RSA: New Attacks, New Results, and New Countermeasures. In D. Sauveron, C. Markantonakis, A. Bilas, and J.-J. Quisquater, editors, *WISTP 2007*, volume 4462 of *Lecture Notes in Computer Science*, pages 215–228. Springer, 2007.

[18] C. H. Kim and J.-J. Quisquater. How can we Overcome both Side Channel Analysis and Fault Attacks on RSA-CRT? In L. Breveglieri, S. Gueron, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC'07*, pages 21–29. IEEE Computer Society, 2007.

[19] P.L. Montgomery. Speeding the Pollard and Ellitpic Curve Methods of Factorization. *Mathematics of Computation*, 48:243–264, 1987.

[20] M. Rivain. Securing RSA against Fault Analysis by Double Addition Chain Exponentiation. In M. Fischlin, editor, *Topics in Cryptology – CT-RSA 2009*, volume 5473 of *Lecture Notes in Computer Science*, pages 459–480. Springer, 2009. Available from `http://eprint.iacr.org/2009/165.pdf`.

[21] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[22] A. Shamir. Improved method and apparatus for protecting public key schemes from timing and fault attacks. International Patent Number : WO 98/52319, November 1998. Also presented at the rump session of EUROCRYPT'97.

[23] Sun Microsystems. Java Card^{TM} Version 3.0 Platform Specification, March 2008. `http://java.sun.com/javacard/3.0`.

[24] D. Vigilant. RSA with CRT: A New Cost-Effective Solution to Thwart Fault Attacks. In E. Oswald and P. Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2008.

[25] M.J. Wiener. Cryptanalysis of short RSA secret exponents. *IEEE Transaction on Information Theory*, 36(3):553–558, May 1990.

[26] S.-M. Yen, S.-J. Kim, S.-G. Lim, and S.-J. Moon. A Countermeasure against one Physical Cryptanalysis May Benefit Another Attack. In K. Kim, editor, *Information Security and Cryptology – ICISC 2001*, volume 2288 of *Lecture Notes in Computer Science*, pages 414–427. Springer, 2001.

[27] 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.

[28] S.-M. Yen, S.J. Moon, and J.-C. Ha. Hardware Fault Attack on RSA with CRT Revisited. In P.J. Lee and C.H. Lim, editors, *Information Security and Cryptology – ICISC 2002*, volume 2587 of *Lecture Notes in Computer Science*, pages 374–388. Springer, 2002.