Practical Fault Countermeasures for Chinese Remaindering Based RSA
(Extended Abstract)

Mathieu Ciet[1] and Marc Joye[2,⋆]

[1] Gemplus R&D, Advanced Research and Security Centre
La Vigie, ZI Athélia IV, Av. du Jujubier, 13705 La Ciotat Cedex, France
mathieu.ciet@gemplus.com
[2] CIM-PACA, Centre de Micro-électronique de Provence − George Charpak
Avenue des Anénomes, Quartier Saint Pierre, 13120 Gardanne, France
marc.joye@gemplus.com

Abstract. Most implementations of the widely-used RSA cryptosystem rely on Chinese remaindering (CRT) as this greatly improves the performances in both running times and memory requirements. Unfortunately, CRT-based implementations are also known to be more sensitive to fault attacks: a single fault in an RSA exponentiation may reveal the secret prime factors trough a GCD computation, that is, a total breaking.

This paper reviews known countermeasures against fault attacks and explain why there are not fully satisfactory or secure. It also presents *practical* countermeasures which feature the following advantages:

1. only CRT input elements are needed (in particular, the value of exponents e and/or d is not required),
2. the resulting performances (running times and memory requirements) are not too much affected,
3. no pre-computations or modifications in the personalisation process are needed,
4. the fault detection does not rely on decisional tests as this can be bypassed,
5. all previously known fault attacks are covered.

As a result, our countermeasures enjoy *at the same time* all best known properties to protect against fault attacks in CRT-based implementations of RSA.

Keywords. RSA, Chinese remaindering, CRT, fault attacks, countermeasures.

1 CRT-RSA in the Presence of Faults

The RSA cryptosystem [13] is undoubtedly the most widely-used public-key cryptosystem.

⋆ Seconded from Gemplus.

Let $N = pq$ be the product of two large primes p and q. Let also a public exponent e coprime to $\varphi(N) = (p-1)(q-1)$ where φ denotes Euler's totient function, and the corresponding secret exponent $d = e^{-1} \bmod \varphi(N)$. The signature on message m is given by

$$S = \dot{m}^d \bmod N\,,$$

where $\dot{m} = \mu(m)$ for some *deterministic* padding function μ (e.g., FDH [3] or PKCS#1 v 1.5 [11]). The validity of signature S can then be publicly verified by checking whether $S^e \equiv \mu(m) \pmod N$, using exponent e.

For efficiency reasons, Chinese remaindering (CRT) [12] is used to speed up the generation of signature S. In CRT mode, the secret parameters are $d_p = d \bmod (p-1)$, $d_q = d \bmod (q-1)$, and $i_q = q^{-1} \bmod p$. Signature S is then computed as

$$S = \mathrm{CRT}(s_p, s_q) = s_q + q\big(i_q(s_p - s_q) \bmod p\big) \quad \text{with} \begin{cases} s_p = \dot{m}^{d_p} \bmod p\,, \\ s_q = \dot{m}^{d_q} \bmod q\,. \end{cases} \tag{1}$$

During 1996, Bellcore researchers [5] showed that if an error occurs in a half exponentiation (namely in the computation of s_p or of s_q — but not in both) then the factorization of N can be recovered from the resulting faulty signature and the correct one. This was later improved using only the faulty signature [7].

Assume for example that the computation of s_p is corrupted. Let $\hat{s}_p$ denote the corresponding value. Then the resulting signature, $\hat{S} = \mathrm{CRT}(\hat{s}_p, s_q)$, will be faulty. Hence, from $\hat{S}$, the opponent finds

$$\gcd(\hat{S}^e - \dot{m} \pmod N, N) = q \tag{2}$$

and $p = N/q$, and so recovers the secret key.

In the next section, we review the countermeasures proposed so far to prevent the recovery of the secret factorization of N, in the presence of faults.

2 Preventing Fault Attacks

2.1 General countermeasures

A general countermeasure consists in computing the signatures twice. This however does not detect permanent errors and, from an efficiency viewpoint, also doubles the running time.

Another general countermeasure consists in verifying the validity of the signatures before outputting them: the device outputs a signature S on message m only if $S^e \equiv \dot{m} \pmod N$ [9]. This countermeasure is efficient as public verification exponent e is typically small in practice (e.g., $e = 2^{16} + 1$). It is however not *practical* as the value of e, although public, is usually not available to the cryptographic device.

2.2 Shamir's trick and generalizations

In [15], Shamir proposes an elegant method for defeating fault attacks. The idea consists in computing the two half exponentiations, s_p and s_q in Eq. (1), in a redundant way.

Let r denote a random κ-bit integer for some security parameter κ (typically, $\kappa = 32$). Then the device computes

$$s_p^* = \dot{m}^d \bmod rp \quad \text{and} \quad s_q^* = \dot{m}^d \bmod rq\,,$$

and returns

$$\begin{cases} S = \mathrm{CRT}(s_p^*, s_q^*) \bmod N & \text{if } s_p^* \equiv s_q^* \pmod r\,, \\ \texttt{error} & \text{otherwise}\,. \end{cases} \tag{3}$$

A more efficient variant is achieved by choosing a random prime r and by reducing $d \bmod (p-1)(r-1)$ (resp. $d \bmod (q-1)(r-1)$) in the computation of s_p^* (resp. s_q^*). Unfortunately, this variant (as well as the basic method) requires the value of secret exponent d whereas only the values of d_p and d_q are available as inputs to the cryptographic device. See also [6].

So, the authors of [8] propose to verify the two half exponentiations separately. For two random κ-bit integers r_1 and r_2, the device computes

$$\begin{aligned} s_p^* &= \dot{m}^{d_p} \bmod r_1 p\,, & s_1 &= \dot{m}^{d_p \bmod \varphi(r_1)} \bmod r_1\,, \\ s_q^* &= \dot{m}^{d_q} \bmod r_2 q\,, & s_2 &= \dot{m}^{d_q \bmod \varphi(r_2)} \bmod r_2\,, \end{aligned}$$

and returns

$$\begin{cases} S = \mathrm{CRT}(s_p^*, s_q^*) \bmod N & \text{if } s_p^* \equiv s_1 \pmod{r_1} \text{ and } s_q^* \equiv s_2 \pmod{r_2}\,, \\ \texttt{error} & \text{otherwise}\,. \end{cases} \tag{4}$$

The previous algorithms cannot detect an error occuring during the CRT combination. If, for example, the value of i_q is faulty (we let $\hat{i_q}$ denote the faulty value) then the resulting faulty signature

$$\hat{S} = \widehat{\mathrm{CRT}}(s_p^*, s_q^*) \bmod N = s_q^* + q\big(\hat{i_q}(s_p^* - s_q^*) \bmod r_1 p\big) \bmod N$$

yields as before the factorization of N by computing $\gcd(\hat{S}^e - \dot{m} \pmod N, N) = q$ since $q \cdot \hat{i_q} \not\equiv 1 \pmod p$. A careful implementation checking the CRT combination is detailed in [1, Fig. 6].

2.3 Infective computation

In [19], Yen *et al.* note that error detection based on decisional tests (as the countermeasures presented in §§ 2.1–2.2) should be avoided. Indeed, inducing a random fault in the status register flips the value of the zero flag bit with a probability of 50% and so bypasses the error detection in the case of a faulty computation.

Starting from this observation, they introduce the concept of infective computation. The attack described in Section 1 requires that only a half exponentiation is faulty but not both. The idea behind infective computation is to ensure that both half exponentiations are faulty whenever an error is induced: if $\hat{S} \not\equiv S \pmod{p}$ then $\hat{S} \not\equiv S \pmod{q}$, and conversely.

According to [2], the two infective countermeasures presented in [19] can be broken. Therefore, Blömer, Otto and Seifert [2] suggested an improved countermeasure. Given a security parameter κ, for two appropriately chosen[3] κ-bit integers t_1 and t_2 (stored in memory), the quantities

$$\begin{gathered} t_1 p\,, \quad t_2 q\,, \quad t_1 t_2 N\,, \\ d_1 = d \bmod \varphi(t_1 p)\,, \quad e_1 = {d_1}^{-1} \bmod \varphi(t_1)\,, \\ d_2 = d \bmod \varphi(t_2 q)\,, \quad e_2 = {d_2}^{-1} \bmod \varphi(t_2)\,, \end{gathered}$$

are pre-computed and stored in memory. The cryptographic device computes

$$S^* = \mathrm{CRT}(s_p^*, s_q^*) \bmod (t_1 t_2 N) \quad \text{where} \begin{cases} s_p^* = \dot{m}^{d_1} \bmod (t_1 p)\,, \\ s_q^* = \dot{m}^{d_2} \bmod (t_2 q)\,, \end{cases}$$

and returns the signature

$$S = (S^*)^{c_1 c_2} \bmod N \quad \text{where} \begin{cases} c_1 = (\dot{m} - S^{e_1} + 1) \bmod t_1\,, \\ c_2 = (\dot{m} - S^{e_2} + 1) \bmod t_2\,. \end{cases} \tag{5}$$

Observe that an error-free computation implies $c_1 = c_2 = 1$ and the so-generated signature is correct.

Unfortunately, the above countermeasure is shown to be insecure. The attack exploits a transient single-byte fault that modifies the value of $\dot{m}$ as it is read in memory in the computation of s_p^* but leaves its value stored in memory unaffected [16]. As a result, the device outputs the faulty signature

$$\hat{S} = {\hat{S}^*}^{\hat{c_1}} \bmod N$$

such that $\hat{S}^* \equiv S^* \pmod{q}$ but $\hat{S}^* \not\equiv S^* \pmod{p}$, with $\hat{S}^* = \mathrm{CRT}(\hat{s_p^*}, s_q^*) \bmod t_1 t_2 N$. The opponent then guesses the value of $\hat{c_1}$ (which, for a properly induced fault on $\dot{m}$ is independent of t_1) and recovers the factorization of N from $\gcd(\hat{S}^e - \dot{m}^{\hat{c_1}} \pmod{N}, N)$. For example, for a 1024-bit RSA modulus and $\kappa = 80$ the success probability of this attack is about 4% [16].

3 Our Countermeasures

3.1 Basic algorithm

Our countermeasures start with the generalized Shamir's trick [8] (cf. Eq. (4)) and adapt it to avoid decisional tests using the infective computation methodology of [19]. More precisely, for two security parameters κ and ℓ, the device

[3] Namely, t_1 and t_2 must satisfy (i) $\gcd(t_1, t_2) = 1$, (ii) $\gcd(d, \varphi(t_1)) = \gcd(d, \varphi(t_2) = 1$, (iii) $t_1 \equiv t_2 \equiv 3 \pmod{4}$, (iv) t_1 and t_2 square-free, and (v) $t_2 \nmid (t_1 p) \cdot [(t_1 p)^{-1} \bmod (t_2 q)]$.

computes $i_{q^*} = (r_2 q)^{-1} \bmod (r_1 p)$,

$$\begin{aligned} s_p^* &= \dot{m}^{d_p} \bmod (r_1 p)\,, \quad s_1 = \dot{m}^{d_p \bmod \varphi(r_1)} \bmod r_1\,, \\ s_q^* &= \dot{m}^{d_q} \bmod (r_2 q)\,, \quad s_2 = \dot{m}^{d_q \bmod \varphi(r_2)} \bmod r_2\,, \end{aligned}$$

where r_1 and r_2 are two co-prime random κ-bit integers, and returns the signature

$$S = (S^*)^\gamma \bmod N \text{ where } \begin{cases} S^* = s_{q^*} + (r_2 q)\big(i_{q^*}(s_{p^*} - s_{q^*}) \bmod (r_1 p)\big)\,, \\ c_i = (S^* - s_i + 1) \bmod r_i\,, \text{ for } i \in \{1,2\}\,, \\ \gamma = \lfloor (r_3\, c_1 + (2^\ell - r_3)\, c_2)/2^\ell \rfloor\,. \end{cases} \tag{6}$$

We give below a more detailed implementation.

Input: $\dot{m}, \{p, q, d_p, d_q, i_q\}$
Output: $S = \dot{m}^d \bmod N$
Parameters: κ, ℓ

---

1. For two co-prime κ-bit integers r_1 and r_2, define
   (a) $p^* = r_1\, p$,
   (b) $q^* = r_2\, q$,
   (c) $i_{q^*} = (q^*)^{-1} \bmod p^*$,
   (d) $N = p\, q$.
2. Compute
   (a) $s_{p^*} \leftarrow \dot{m}^{d_p} \bmod p^*$ and $s_2 \leftarrow \dot{m}^{d_q \bmod \varphi(r_2)} \bmod r_2$,
   (b) $s_{q^*} \leftarrow \dot{m}^{d_q} \bmod q^*$ and $s_1 \leftarrow \dot{m}^{d_p \bmod \varphi(r_1)} \bmod r_1$.
3. Compute $S^* \leftarrow s_{q^*} + q^*\ i_{q^*}(s_{p^*} - s_{q^*}) \bmod p^*$.
4. Compute
   (a) $c_1 \leftarrow (S^* - s_1 + 1) \bmod r_1$,
   (b) $c_2 \leftarrow (S^* - s_2 + 1) \bmod r_2$.
5. For an ℓ-bit integer r_3, set $\gamma \leftarrow \lfloor (r_3\, c_1 + (2^\ell - r_3)\, c_2)/2^\ell \rfloor$.
6. Return $S = (S^*)^\gamma \bmod N$.

Fig. 1. Our basic algorithm.

Again, note that when there are no errors, we have $c_1 = c_2 = \gamma = 1$.

3.2 Security analysis

From a security viewpoint, a step-by-step inspection *in the security model of [2]* shows that the opponent gains no information in inducing a fault during the generation of a signature. Furthermore, the fault attack of Wagner (cf. § 2.3) does not apply.

The order of the computation in Step 2 is very important: s_p^*, s_2, s_q^* and s_1. If the computation is carried out as s_p^*, s_1, s_q^* and s_2 then a long-lived fault on $\dot{m}$ before the computation of s_q^* (and after s_1) would go undetected (i.e., $c_1 = c_2 = \gamma = 1$) and the resulting faulty signature, $\hat{S}$, would be such that $\gcd(\hat{S}^e - \hat{m} \pmod N, N) = p$.

Importantly, we implicitly assume that the values *defined* in Step 1 are error-free. For practicability, we do not want to modify the personalisation process and so do not require, as in [2], that those quantities are available as inputs in memory. As aforementioned, the standard inputs in CRT mode are parameters (p, q, d_p, d_q, i_q) — along with a CRC.

We also implicitly assume that the CRC is checked after that a CRT parameter is used or erased from memory (this is not included in Fig. 1 for clarity). We note that this only protect against long-lived faults.

There are several ways to ensure the correctness of the values defined in Step 1. A first solution is to "reverse" the computations and to use the CRC information for checking the correctness. For example, the value of p^* can be checked for correctness as $\mathrm{CRC}(p^*/r_1) \stackrel{?}{=} \mathrm{CRC}(p)$. Alternatively (or in a combined way), one can also exploit algebraic properties. Again, with the example of p^*, one can check that $p^* \equiv 0 \pmod p$ (cf. [1, Fig. 6]). Yet another possibility is to make use of the relations between the different quantities and to test the coherence; e.g., $N = (p^*q^*)/(r_1r_2)$. We note in this last case that a multiplicative CRC can be useful (for instance defined as a modular reduction).

As previously discussed in § 2.3, all these checks should be carried out without any decisional test. For example, the result of a test of the form $A \stackrel{?}{=} B$ should be replaced with $\gamma \leftarrow \gamma(A \oplus B \oplus 1)$ where γ denotes a κ-bit register (for some security parameter κ) keeping track of the errors ($\gamma = 1$ means no fault detected). Again this is not explicitly included in Fig. 1 for clarity (Step 5 should be modified accordingly).

3.3 Computing i_{q^*}

The computation of $i_{q^*} = (q^*)^{-1} \bmod p^*$ (with $p^* = r_1 p$ and $q^* = r_2 q$) can be carried out using the extended Euclidean algorithm. A more efficient method is to make use of the value of input CRT parameter $i_q = q^{-1} \bmod p$. We obviously have $i_{q^*} \equiv i_q \cdot {r_2}^{-1} \pmod p$. Moreover, the value of ${r_2}^{-1} \bmod p$ can be evaluated from the value of $p^{-1} \bmod r_2$ as

$$ {r_2}^{-1} \bmod p = \frac{1 + p\,(-p^{-1} \bmod r_2)}{r_2} \ . $$

The value of i_{q^*} is then obtained by Chinese remaindering modulo $r_1 p$. So, the computation is fast since only reductions modulo r_i ($i \in \{1, 2\}$) are involved.

3.4 Variants and remarks

There are numerous variants of our basic algorithm. We describe below a few of them.

- The basic algorithm assumes that r_1 and r_2 are co-prime. More generally, if $\gcd(r_1, r_2) = \delta$ then the extended Euclidean algorithm yields the value of

$$\frac{\delta}{q^*} \bmod \frac{p^*}{\delta} \ ,$$

  instead of $i_{q^*} = (q^*)^{-1} \bmod p^*$ [6], which can be evaluated efficiently using the technique of § 3.3. The expression of c_1 then becomes $c_1 = \big(S^* - \delta s_1 + (\delta - 1)s_{q^*} + 1\big) \bmod r_1$.
- The final step (Step 6 in Fig. 1) involves the computation of a (short) exponentiation modulo RSA modulus N. When there is not enough memory, this can advantageously replaced with

$$S = (\gamma\, S^* \oplus (\gamma - 1)\, r) \bmod N$$

  for an $(|N|_2 + 2\kappa)$-bit random r.
- The algorithm remains valid when CRT exponents d_p and d_q are masked as $d_{p^*} = d_p + \rho_1(p-1)$ and $d_{q^*} = d_q + \rho_2(q-1)$ for two random integers ρ_1 and ρ_2. Such blindings are useful to prevent certain side-channel attacks [10] and safe-error attacks [17].

4 Conclusion

This paper presented simple and practical countermeasures against fault attacks for Chinese remaindering based implementations of RSA. Our algorithms are generic and several parts can be tuned according to the processor being used.

As in the previous works, our security analysis is only assessed against known attacks. The next step would be to formally prove the security of our algorithms, as was initiated in [2, 19]. However, as exemplified by [16], this is highly non-trivial.

References

1. C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert. Fault attacks on RSA with CRT: Concrete results and practical countermeasures. In B.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-Verlag, 2002.
2. J. Blömer, M. Otto, and J.-P. Seifert. A new CRT-RSA algorithm secure against Bellcore attacks. In *10th ACM Conference on Computer and Communications Security*, pages 311–320. ACM Press, 2003.

3. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *1st ACM Conference on Computer and Communications Security*, pages 62–73. ACM Press, 1993.
4. D. Boneh, R.A. DeMillo, and R.J. Lipton. On the importance of checking cryptographic protocols for faults. In W. Fumy, editor, *Advances in Cryptology – EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer-Verlag, 1997.
5. D. Boneh, R.A. DeMillo, and R.J. Lipton. On the importance of eliminating errors in cryptographic computations. *Journal of Cryptology* **14**(2):101–119, 2001. An earlier version appears in [4].
6. W. Fischer and J.-P. Seifert. Note on fast computation of secret RSA exponents. In L. Batten and J. Seberry, editors, *Information Security and Privacy (ACISP 2002)*, volume 2384 of *Lecture Notes in Computer Science*, pages 136–143. Springer-Verlag, 2002.
7. M. Joye, A.K. Lenstra, and J.-J. Quisquater. Chinese remaindering based cryptosystems in the presence of faults. *Journal of Cryptology* **12**(4):241–245, 1999.
8. M. Joye, P. Paillier, and S.-M. Yen. Secure evaluation of modular functions. In R.J. Hwang and C.K. Wu, editors, *International Workshop on Cryptology and Network Security*, pages 227–229, 2001.
9. B.S. Kaliski Jr and M.J.B. Robshaw. Comments on a new attack on cryptographic devices. *RSA Laboratories' Bulletin* **5**, July 14, 1997.
10. P.C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer-Verlag, 1999.
11. PKCS #1 v 1.5: RSA Cryptography Standard.
12. J.-J. Quisquater and C. Couvreur. Fast decipherment algorithm for RSA public-key cryptosystem. *Electronics Letters* **18**(21):905–907, 1982.
13. R.L. Rivest, A. Shamir, and L.M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM* **21**(2):120–126, 1978.
14. A. Shamir. How to check modular exponentiation. Presented at EUROCRYPT '97 rump session, Konstanz, May 1997.
15. A. Shamir. Method and apparatus for protecting public key schemes from timing and fault attacks. United States Patent #5,991,415, November 23, 1999. See also [14].
16. D. Wagner, Cryptanalysis of a provably secure CRT-RSA algorithm. In *11th ACM Conference on Computer and Communications Security*, pages 92–97. ACM Press, 2004.
17. S.-M. Yen and M. Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Transactions on Computers* **49**(9):967–970, 2000.
18. S.-M. Yen, S. Kim, S. Lim, and S. Moon. RSA speedup with residue number system immune against hardware fault cryptanalysis. In K. Kim, editor, *Information Security and Cryptology – ICISC 2001*, volume 2288 of *Lecture Notes in Computer Science*, pages 397–413. Springer-Verlag, 2001.
19. 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):461–472, 2003. An earlier version appears in [18].