RSA with CRT: A new cost-effective solution to thwart fault attacks

David Vigilant

Cryptography Engineering, Gemalto Security Labs
david.vigilant@gemalto.com

Abstract. Fault attacks as introduced by Bellcore in 1996 are still a major threat toward cryptographic products supporting RSA signatures. Most often on embedded devices, the public exponent is unknown, turning resistance to fault attacks into an intricate problem. Over the past few years, several techniques for secure implementations have been published, all of which suffering from inadequacy with the constraints faced by embedded platforms. In this paper, we introduce a novel countermeasure mechanism against fault attacks in RSA signature generation. In the restricted context of security devices where execution time, memory consumption, personalization management and code size are strong constraints, our countermeasure is simply applicable with a low computational complexity. Our method extends to all cryptosystems based on modular exponentiation.

Key words: Bellcore attack, Chinese Remainder Theorem, Fault attacks, RSA, Software countermeasure, Modular exponentiation.

1 Introduction

1.1 Restricted context

Throughout the paper, we will be considering constrained embedded architectures on which one seeks to simultaneously optimize the following:

Execution time The secure RSA-CRT signature computation has to be performed in reasonable time. Without giving concrete bounds, the time overhead added by the countermeasure must remain negligible compared to the whole RSA signature calculation. This is of prime importance for micro-controllers running under a clock frequency of only a few megahertz.

Memory consumption Countermeasures require extra RAM memory buffers to store security parameters. 2K RSA is now supported as a standard functionality and we impose that the whole memory consumption remains comprised between 1 and 2K bytes.

Personalization management The availability of input key parameters is very strict. Only the input message m, as well as the key elements p, q, d_p, d_q, i_q are known while performing the signature and no extra variable parameter can be stored in non-volatile memory. This constraint stems from mass-production requirements where the personalization of unusually formatted keys in the device is costly and no customizable key container is available in EEPROM nor Flash to store anything different from the classical RSA-CRT key sets [1].

Code Size On micro-controllers that have little ROM, the code size will be of a great concern. The extra code size added by the countermeasure must remain negligible compared to the whole code size of the signature. To minimize the code, it is preferable to design a simple countermeasure based on already existing arithmetic bricks.

1.2 The Bellcore attack and related countermeasures

Invasive attacks on a hardware device consist in disturbing its expected behavior and making it work abnormally in order to infer sensitive data. They were introduced in the late nineties. As the technological response of hardware manufacturers evolves, new hardware countermeasures are being added regularly. However it is widely believed that those can only be effective if combined with efficient software countermeasures. Embedded devices are especially exposed to this category of attacks since the attacker has the hardware fully available in hands. A typical example is the original Bellcore attack [2] which allows an attacker to retrieve the RSA private key given one faulty signature.

Since the discovery of the Bellcore attack, countermeasures have been proposed by the research community. In 1997, Shamir proposed an elegant countermeasure [3] assuming that the private exponent d is known when running an RSA signature generation in CRT mode. In practice, however, this parameter is hardly available. Aumüller et al. [4] in 2002, Blömer et al. [5] in 2003, Joye and Ciet [6] and Giraud [7] in 2005, and Kim and Quisquater [8] in 2007 also proposed CRT secure implementations of RSA. All these countermeasures have a dramatic impact either on execution time, memory consumption or personalization management constraints. As an example, Aumüller et al. set out an efficient countermeasure [4] in 2002 using a small prime on which evaluating Euler's totient function is trivial. We will see in the sequel that, on the one hand, this countermeasure gives good performances. On the other hand, the selection of a random prime constitutes a real disadvantage.

This paper presents a simple alternative countermeasure thwarting fault attacks on RSA with CRT. Compared to prior techniques, our countermeasure is cost-effective regarding all considered constraints.

In Section 2, we make a brief review of the Bellcore attack and we show the disadvantages of previous propositions in the defined context. Our secure exponentiation algorithm and its application to RSA in the CRT mode is shown in Section 3. We then analyze its security under a fault model described in Section 4, where brief estimates in terms of time execution, memory consumption, personalization management and code size are undertaken. Finally Section 5 concludes this paper.

2 Related Work

2.1 RSA-CRT system

RSA was introduced in 1977 by Rivest, Shamir and Adleman [9]. In the so-called straightforward mode, (N, e) is the RSA public key and (N, d) the RSA private key such that $N = pq$, where p and q are large prime integers, $\gcd((p-1), e) = \gcd((q-1), e) = 1$ and $d = e^{-1} \bmod (p-1)(q-1)$. The RSA signature of a message $m < N$ is given by $S = m^d \bmod N$.

As the computing power of crypto-enabled architectures increases, RSA key sizes inflate overtime. 2K RSA is now a standard functionality. It is a strong constraint on embedded devices as processors have little RAM memory and run under a clock frequency of a few megahertz. RSA is more efficient in Chinese Remainder Theorem mode than in straightforward mode. The RSA-CRT domain is composed of an RSA public key (N, e) and an RSA private key (p, q, d_p, d_q, i_q) where $N = pq$, p and q are large prime integers, $\gcd((p-1), e) = \gcd((q-1), e) = 1$, $d_p = e^{-1} \bmod (p-1)$, $d_q = e^{-1} \bmod (q-1)$ and $i_q = q^{-1} \bmod p$. As it handles data with half the RSA modulus size, RSA with CRT is theoretically about four times faster and is therefore better suited to embedded devices. The RSA signature in CRT mode is described in Figure 1.

```
**Input:** message $m$, key $(p, q, d_p, d_q, i_q)$
**Output:** signature $m^d \in \mathbb{Z}_N$
$S_p = m^{d_p} \bmod p$
$S_q = m^{d_q} \bmod q$
$S = S_q + q \cdot (i_q \cdot (S_p - S_q) \bmod p)$
**return** $(S)$
```

Fig. 1. Naive CRT implementation of RSA

2.2 The Bellcore attack against RSA with CRT

In 1996, the Bellcore Institute introduced a differential fault attack [2] which is still weakening the RSA-CRT signature security today. On embedded platforms, this attack is usually considered as “easy” since the attacker has full access to the device. Disturbing the calculation of either $S_p = m^{d_p} \bmod p$ or $S_q = m^{d_q} \bmod q$ can be achieved in ways such as voltage glitches, laser or temperature variation. Once the precise disturbance is obtained the attack succeeds, and allows an attacker to retrieve the RSA prime factors with a single gcd calculation. By construction, $S = S_q + q \cdot (i_q \cdot (S_p - S_q) \bmod p) = S_p + p \cdot (i_p \cdot (S_q - S_p) \bmod q)$. Noting S the correct signature and $\tilde{S}$ the faulty signature where either S_p or S_q (but not both) is incorrect for the same input message, $\gcd(S - \tilde{S}, N)$ is either q or p. A standard improvement of the Bellcore attack [10] leads to retrieving the factorization of N without the genuine signature by calculating $\gcd((\tilde{S}^e - m) \bmod N, N) \in \{p, q\}$. Thus, the RSA private elements p and q are recovered and, as a consequence, the whole RSA-CRT private key is recovered.

2.3 Previous countermeasures

Shamir’s method and generalizations One year after the discovery of the Bellcore attack, Shamir proposed an elegant countermeasure [3] where the method consists in computing $S_p^* = m^d \bmod pr$ and $S_q^* = m^d \bmod qr$ separately and in checking the consistency of S_p^* and S_q^* by testing whether $S_p^* = S_q^* \bmod r$. A more efficient variant suggests to choose r prime and reduce d modulo $(p-1)(r-1)$ and $(q-1)(r-1)$. However, requiring the RSA straightforward-mode private exponent d, while performing an RSA signature generation in CRT mode, is unpractical since the key material is given in CRT format [1]. This parameter is most often not known and it would be unacceptable in our context to personalize d for each device. d could be computed from p, q, d_p and d_q, but as no key container would be available to store it, the computation of d would be mandatory at each RSA signature. As described in [11], this would lead to an unreasonable execution time overhead since we need to invert $(p-1)$ modulo $(q-1)$. Moreover, the CRT recombination is not protected at all since injecting a fault in i_q during the recombination allows the gcd attack.

Other improvements of Shamir’s method which include the protection of the recombination were proposed later. As an example, Aumüller et al. [4] in 2002 proposed a careful implementation that also protects the CRT recombination. As opposed to Shamir’s method, only d_p and d_q (and not d) are required. The algorithm is fully described in Figure 2. The proposal uses the efficient variant of the method where the parameter t is prime. Therefore the solution gives good performances. Compared to the naive CRT implementation of RSA, only two extra exponentiations modulo t and a few modular reductions are required. This solution presents a big disadvantage: the way the random prime is selected. Is it fixed or picked at random in a fixed table? (If this prime is recovered, does it make new flaws appear?). Is it different on each device? (This would impact personalization management). Is it generated at random for each signature? (This would lead to an unacceptable slowdown).

```
Input: message $m$, key $(p, q, d_p, d_q, i_q)$
32-bit prime integer t
Output: signature $m^d \in \mathbb{Z}_N$
$p' = pt$
$d'_p = d_p + random_1 \cdot (p-1)$
$S'_p = m^{d'_p} \bmod p'$
if $(p' \bmod p \neq 0)$ or $(d'_p \bmod (p-1) \neq d_p)$ then
  return (error)
end if

$q' = qt$
$d'_q = d_q + random_2 \cdot (q-1)$
$S'_q = m^{d'_q} \bmod q'$
if $(q' \bmod q \neq 0)$ or $(d'_q \bmod (q-1) \neq d_q)$ then
  return (error)
end if

$S_p = S'_p \bmod p$
$S_q = S'_q \bmod q$
$S = S_q + q \cdot (iq \cdot (S_p - S_q) \bmod p)$
if $(S - S'_p \neq 0 \bmod p)$ or $(S - S'_q \neq 0 \bmod q)$ then
  return (error)
end if

$S_{pt} = S'_p \bmod t$
$S_{qt} = S'_q \bmod t$
$d_{pt} = d'_p \bmod (t-1)$
$d_{qt} = d'_q \bmod (t-1)$
if $S_{pt}^{d_{qt}} \equiv S_{qt}^{d_{pt}} \bmod t$ then
  return (S)
else
  return (error)
end if
```

Fig. 2. Aumüller et al.'s secure CRT implementation of RSA

Interestingly, other solutions combining generalizations of Shamir's method and infective computation were proposed. The main idea of this combination consists in infecting the signature S whenever a fault is induced, such that the gcd attack is no more feasible on the faulty signature S', i.e. $S' \neq S \bmod p$ and $S' \neq S \bmod q$. This concept was introduced in 2001 by Yen, Kim, Lim and Moon [12]. Later, Blömer, Otto and Seifert suggested a countermeasure [5] based on infective computation in 2003. Unfortunately, as for Shamir's original method, it requires the availability of d. Moreover, some parameters t_1 and t_2 required by the countermeasure have to satisfy quite strong properties: amongst the required properties, it is needed that: $\gcd(t_1, t_2) = \gcd(d, \varphi(t_1)) = \gcd(d, \varphi(t_2)) = 1$, where φ represents the Euler's totient function. t_1 and t_2 should normally be generated one time with the RSA key and the same values used throughout the lifetime of the key, but t_1 and t_2 cannot be stored in this strong personalization context. Therefore the generation of t_1 and t_2 at each signature is not negligible. Compared to Aumüller et al.'s countermeasure, the BOS algorithm requires the generation of t_1 and t_2, two evaluations of the totient function φ on t_1 and t_2 and two inversions. This constitutes a real disadvantage in terms of simplicity and execution time.

Joye and Ciet also set out an elegant countermeasure based on infective computation [6]. Their generalization of Shamir's method is more efficient than BOS since, compared to Aumüller et al.'s countermeasure, one only needs to compute $\varphi(t_1)$ and $\varphi(t_2)$ for two random numbers t_1 and t_2. However, evaluations are not negligible as they imply a full factorization of t_1 and t_2. As a consequence, Joye and Ciet's countermeasure is not satisfactory in terms of execution time.

Last year, Kim and Quisquater proposed a CRT implementation of RSA defeating fault attacks and all known side-channel attacks [8], based on combination of Shamir's method and infective computation too. However, their proposed scheme requires either one inversion modulo N, or to update and store three unusually formatted parameters of size $|N|$, at each signature. As defined in Section 1.1, no key container is available in non-volatile memory and therefore, this solution becomes hardly acceptable in terms of execution time.

Giraud's method In 2005, Giraud proposed an efficient way [7] to protect RSA with CRT against fault attacks. His countermeasure is based on the properties of the Montgomery-ladder exponentiation algorithm [13]. Using this exponentiation algorithm, we compute successively (m^{d_p}, m^{d_p-1}) and (m^{d_q}, m^{d_q-1}). The Montgomery-Ladder algorithm infects both results whenever a fault is induced. The two recombined values S and $S' = m^{d_q-1} + q \cdot (i_q \cdot (m^{d_p-1} - m^{d_q-1}) \bmod p)$ are computed and the final verification $S = mS'$ is made. This solution is also SPA-safe. Unfortunately, the memory consumption is clearly prohibitive since it requires the storage of m, S_p, S_q, S'_p and S'_q in RAM during the calculation of S. For large RSA key sizes, this countermeasure seems hardly feasible in portable devices.

This shows that devising a CRT implementation of RSA that thwarts the Bellcore attack and meets the strong requirements of embedded systems remains a hard problem.

3 Our secure RSA with CRT

3.1 Mathematical view

We consider a generic exponentiation of a message m to the exponent d modulo N. We perform the exponentiation modulo NR where R is for example a 64-bit random integer. We impose that N and R are coprime, i.e. $\gcd(N, R) = 1$.

$$\text{Let } \alpha \text{ be such that} \begin{cases} \alpha \equiv 0 \bmod R \\ \alpha \equiv 1 \bmod N \end{cases} \qquad \text{and } \beta \text{ be such that} \begin{cases} \beta \equiv 1 \bmod R \\ \beta \equiv 0 \bmod N \end{cases}$$

Applying the Chinese Remainder Theorem, we get the existence and the uniqueness of α and β in $\mathbb{Z}_{NR}$. We build these integers using Garner's algorithm:

$$\alpha = R \cdot (R^{-1} \bmod N) = 1 - [N \cdot (N^{-1} \bmod R)] \bmod NR$$

$$\beta = N \cdot (N^{-1} \bmod R) = 1 - [R \cdot (R^{-1} \bmod N)] \bmod NR$$

Considering R now such that $R = r^2$, where r is for example a 32-bit random number, we get the following result:

Theorem 1 (Exponentiation Identity in $\mathbb{Z}_{Nr^2}$). *Let N and r be integers such that $\gcd(N, r) = 1$, let $\beta = N \cdot (N^{-1} \bmod r^2)$ and $\alpha = 1 - \beta \bmod Nr^2$. For any $m \in \mathbb{Z}_{Nr^2}$ and for any $d \in \mathbb{N}^*$,*

$$(\alpha m + \beta \cdot (1 + r))^d = \alpha m^d + \beta \cdot (1 + dr) \bmod Nr^2$$

We refer to Appendix A for a proof and related mathematical details. Theorem 1 provides a way to perform a secure exponentiation in any ring $(\mathbb{Z}_N, +, \cdot)$, $N \in \mathbb{N}^*$.

3.2 A Secure exponentiation algorithm

We want to perform an exponentiation m^d of an integer $m < N$ over $\mathbb{Z}_N$. Pick a random integer r coprime with N and compute $\beta = N \cdot (N^{-1} \bmod r^2)$ and $\alpha = 1 - \beta \bmod Nr^2$. Applying Theorem 1, in order to exponentiate the element m and verify that no disturbance occurred, proceed as follows:

1. Compute $\hat{m} = \alpha m + \beta \cdot (1 + r) \bmod Nr^2$
2. Verify that $\hat{m} = m \bmod N$ and in case of inequality return "error detected"
3. Compute $S_r = \hat{m}^d \bmod Nr^2$ and $S = S_r \bmod N$ $(= m^d \bmod N)$
4. Verify that $S_r = \alpha S + \beta \cdot (1 + dr) \bmod Nr^2$ and in case of inequality return "error detected"

By virtue of equalities $\beta = \beta^2$ and $\alpha\beta = 0$ in $\mathbb{Z}_{Nr^2}$ (by construction of α and β), the consistency of S_r can also be verified by any one of the following checks:

1. $\beta S_r = \beta \cdot (1 + dr) \bmod Nr^2$
2. $N \cdot (S_r - \beta \cdot (1 + dr)) = 0 \bmod Nr^2$
3. $S_r = 1 + dr \bmod r^2$

The optimal choice will depend on the hardware architecture and the algorithmic context. This countermeasure may be applied to any cryptographic scheme based on exponentiation in $(\mathbb{Z}_N, +, \cdot)$, $N \in \mathbb{N}^*$ (RSA [9], Diffie-Hellman key exchange [14], ElGamal [15], ...). Here we underline its application to the CRT implementation of RSA, where it appears to be particularly relevant.

3.3 Application to RSA with CRT

As p and q are prime, r is automatically coprime with p and q ,we define: $\beta_p = p \cdot (p^{-1} \bmod r^2)$, $\alpha_p = 1 - \beta_p \bmod pr^2$, $\beta_q = q \cdot (q^{-1} \bmod r^2)$ and $\alpha_q = 1 - \beta_q \bmod qr^2$. Figure 3 shows a possible application of our countermeasure to RSA with CRT. Exponentiations S_{pr} and S_{qr} are performed over $\mathbb{Z}_{pr^2}$ and $\mathbb{Z}_{qr^2}$. We verify that each exponentiation has not been disturbed by checking:

$$\beta_p S_{pr} = \beta_p \cdot (1 + d'_p r) \bmod pr^2 \quad \text{and} \quad \beta_q S_{qr} = \beta_q \cdot (1 + d'_q r) \bmod qr^2.$$

We pick up two 64-bit random integers R_3 and R_4. We then transform:

$$S_{pr} \text{ into } S'_p \text{ s.t.} \begin{cases} S'_p \equiv S_p \bmod p \\ S'_p \equiv R_3 \bmod r^2 \end{cases} \quad \text{and } S_{qr} \text{ into } S'_q \text{ s.t.} \begin{cases} S'_q \equiv S_q \bmod q \\ S'_q \equiv R_4 \bmod r^2 \end{cases}$$

Next, the resulting signature is recombined over $\mathbb{Z}_{Nr^2}$:

$$S = S'_q + q \cdot \left[i_q \cdot (S'_p - S'_q) \bmod pr^2 \right] \ ,$$

and, we perform the final consistency check:

$$S = R_4 + q i_q \cdot (R_3 - R_4) \bmod r^2 \ .$$

If all verifications are positive, we return the result $S \bmod N$.

3.4 Recommendations

The quality of the random number generator must be verified. We recommend to choose r such that $i_q \neq 0 \bmod r$. Indeed if $r \mid i_q$, the fault detection probability is reduced since the verification $N \cdot [S - R_4 - q i_q \cdot (R_3 - R_4)] \equiv 0 \bmod Nr^2$ is true even though the result of $S_p - S_q \bmod pr^2$ or q has been modified. So we recommend to renew the generation of the random r while r divides i_q. r must be as large as possible within the limits of the hardware architecture. Since we can see r as a security parameter, the larger it is, the higher the fault detection probability. Indeed, the highest success probability of an attack is $2^{-(|r|-1)} \ln 2$ (see Section 4.1 and Appendix B for more details). So we suggest that r should be at least a 32-bit random integer. Finally, we choose r with most significant bit equal to one, in order to optimize the security level. We also choose r odd in order to optimize the efficiency of the inversion.

```
Input: message $m$, key $(p, q, d_p, d_q, i_q)$
32-bit random integer $r$
64-bit random integers $R_1, R_2$, $R_3$ and $R_4$
Output: signature $m^d \in \mathbb{Z}_N$

$p' = pr^2$, $m_p = m \bmod p'$
$i_{pr} = p^{-1} \bmod r^2$, $\beta_p = pi_{pr}$ and $\alpha_p = 1 - \beta_p \bmod p'$
$\hat{m}_p = \alpha_p m_p + \beta_p \cdot (1 + r) \bmod p'$
if $(\hat{m}_p \neq m \bmod p)$ then
  return (error)
end if
$d'_p = d_p + [R_1 \cdot (p - 1)]$
$S_{pr} = \hat{m}_p^{d'_p} \bmod p'$
if $(\beta_p S_{pr} \neq \beta_p \cdot (1 + d'_p r) \bmod p')$ or $(d'_p \neq d_p \bmod (p - 1))$ then
  return (error)
end if
$S'_p = S_{pr} - \beta_p \cdot (1 + d'_p r - R_3)$

$q' = qr^2$, $m_q = m \bmod q'$
$i_{qr} = q^{-1} \bmod r^2$, $\beta_q = qi_{qr}$ and $\alpha_q = 1 - \beta_q \bmod q'$
$\hat{m}_q = \alpha_q m_q + \beta_q \cdot (1 + r) \bmod q'$
if $(\hat{m}_q \neq m \bmod q)$ or $(m_p \bmod r^2 \neq m_q \bmod r^2)$ then
  return (error)
end if
$d'_q = d_q + [R_2 \cdot (q - 1)]$
$S_{qr} = \hat{m}_q^{d'_q} \bmod q'$
if $(\beta_q S_{qr} \neq \beta_q \cdot (1 + d'_q r) \bmod q')$ or $(d'_q \neq d_q \bmod (q - 1))$ then
  return (error)
end if
$S'_q = S_{qr} - \beta_q \cdot (1 + d'_q r - R_4)$

$S = S'_q + q \cdot (i_q \cdot (S'_p - S'_q) \bmod p')$
$N = pq$
if $(N \cdot [S - R_4 - qi_q \cdot (R_3 - R_4)] \neq 0 \bmod Nr^2)$ or $(qi_q \neq 1 \bmod p)$
then
  return (error)
end if
return $(S \bmod N)$
```

Fig. 3. Our secure CRT implementation of RSA

4 Analysis

4.1 Resistance against fault attacks

The following fault model defines what an attacker is able to do by assumption. By disturbing the device, we mean that an attacker can:

– modify a value in memory obtaining a totally random result uncorrelated to the original value (as known as permanent fault);
– modify a value when it is handled in local registers, without modifying the global value in memory. The value handled obtained is fully random looking to the attacker and uncorrelated to the original value (as known as transient fault);

The design does not address attackers who can:

– modify the code execution. Processor instructions cannot be replaced or removed while executing code. Such an attacker might have the power to dump EEPROM and obtain the secret key;
– inject a permanent fault in the input elements, the message m as well as the key (p, q, d_p, d_q, i_q). We suppose that input elements are given along with an integrity value that can be verified whenever during the signature;
– Change the Boolean result of a conditional check. An expression "if $a = b$" has a result $true$ or $false$ that cannot be modified. We made here a compromise on the level of security. Indeed, contrary to some other methods based on infective computations, our design uses conditional checks. However it would be possible to replace these checks by unconditional infections of the computation.

We consider the CRT implementation of RSA described in Figure 3 and we assume the recommendations discussed in Section 3.4 have been followed. Noting $|a|$ the bit size of a and $\underline{a}$ the faulty value of a, let us review some fault scenarios and identify the associated success probabilities (probabilities are more detailed in Appendix B):

– Modifying p or r in a transient way during the calculation of p' or modifying p' in a permanent way before the check of $\hat{m}_p$ (The same holds for q'):
$\Pr[\underline{\hat{m}_p} = m \bmod p] \approx 2^{-(|p|-1)} \ln 2$
After the check of $\hat{m}_p$, if the permanent fault occurs only during the exponentiation:
$\Pr[\beta_p \underline{S_{pr}} = \beta_p \cdot (1 + d'_p r) \bmod p'] \approx 2^{-(|p'|-1)} \ln 2$
– Modifying m in a transient way during the calculation of $\hat{m}_p$ or modifying $\hat{m}_p$ in a permanent way before the check (The same holds for $\hat{m}_q$):
$\Pr[\underline{\hat{m}_p} = m \bmod p] \approx 2^{-(|p|-1)} \ln 2$
– Modifying m in a permanent way after the first exponentiation (we may also consider that m is associated with an integrity value that is verified):
$\Pr[\underline{m_q} \bmod r^2 = m_p \bmod r^2] \approx 2^{-(2|r|+1)}$
If the permanent fault occurs after the check of $\hat{m}_p$:
$\Pr[\beta_p \underline{S_{pr}} = \beta_p \cdot (1 + d'_p r) \bmod p'] = \Pr[\underline{\hat{m}_p} = 1 + r \bmod r^2] \approx 2^{-2|r|+1}$

- Modifying p or r^2 in a transient way during the calculation of i_{pr}, or modifying i_{pr} in a permanent way (The same holds for i_{qr}):
  $\Pr[(\underline{\alpha_p}m+\underline{\beta_p}\cdot(1+r) = m \bmod p)\cap(\underline{\alpha_p}m+\underline{\beta_p}\cdot(1+r) = (1+r) \bmod r^2)] = 0$
- Modifying p or i_{pr} in a transient way during the calculation of β_p or modifying β_p in a permanent way (The same holds for β_q):
  $\Pr[(\underline{\alpha_p}m+\underline{\beta_p}\cdot(1+r) = m \bmod p)\cap(\underline{\alpha_p}m+\underline{\beta_p}\cdot(1+r) = (1+r) \bmod r^2)] = 0$
- Modifying β_p or p' in a transient way during the calculation of α_p or modifying α_p in a permanent way (The same holds for α_q):
  $\Pr[\beta_p\underline{S_{pr}} = \beta_p \cdot (1 + d'_p r) \bmod p'] = \Pr[\underline{\alpha_p} = 0 \bmod r^2] \approx 2^{-2|r|+1}$
- Modifying $(p-1)$ or d_p in a transient way during the calculation of d'_p or modifying d'_p in a permanent way (The same holds for d'_q):
  $\Pr[\underline{d'_p} = d_p \bmod (p-1)] \approx 2^{-(|p|-1)} \ln 2$
- Modifying d'_p in a transient way during the computation of S_{pr} (The same holds for S_{qr}):
  $\Pr[\beta_p\underline{S_{pr}} = \beta_p \cdot (1 + d'_p r) \bmod p'] = \Pr[\underline{d'_p} = d'_p \bmod r] \approx 2^{-(|r|-1)} \ln 2$
- Modifying $\hat{m}_p$ or p' in a transient way during the computation of S_{pr} (The same holds for S_{qr}):
  $\Pr[\beta_p\underline{S_{pr}} = \beta_p \cdot (1 + d'_p r) \bmod p'] = \Pr[\underline{\hat{m}_p} = 1 + r \bmod r^2] \approx 2^{-2|r|+1}$
- Modifying S_{pr}, $\beta_p \cdot (1 + d'_p r)$, R_3 or p' in a transient way during the computation of S'_p, or modifying S'_p in a permanent way (The same holds for S'_q):
  $\Pr[S - R_4 - qi_q \cdot (R_3 - R_4) = 0 \bmod r^2] \approx 2^{-2|r|+1}$
- Modifying S'_p, S'_q,p', q, i_q or S'_q in a transient way during the recombination:
  $\Pr[N \cdot (S - R_4 - qi_q \cdot (R_3 - R_4)) = 0 \bmod Nr^2] \approx 2^{-2|r|+1}$

4.2 Side-Channel Analysis

Although side-channel analysis is not studied in this paper, the design should be combined with adapted extra countermeasures against side-channel attacks.

4.3 Performance analysis

Execution time The most expensive steps are the two inversions. They are performed on parameters with length twice the length of r. Noting $i_{pr0} = p^{-1} \bmod r$ and $i_{qr0} = q^{-1} \bmod r$, we make use of tricks to compute i_{pr} and i_{qr} from i_{pr0} and i_{qr0}. Indeed let $p = p_0 + p_1 r \bmod r^2$ and $i_{pr1} = [-i_0 p_1 - (i_0 p_0 - 1)] \cdot i_0 \bmod r$. Then $i_{pr} = r i_{pr1} + i_{pr0}$ (The same holds for i_{qr}). Thus, only two inversions modulo r are needed to compute i_{pr} and i_{qr}. If r is for example a 32-bit value and implementation is carried out on a 32-bit chip architecture, an SPA-safe extended binary gcd algorithm can be implemented very efficiently since loops of the algorithm would be composed of comparisons, shifts, subtractions and additions on 32-bit single precision data. In this context, the execution time added by our countermeasure would be clearly less costly than Aumüller et al.'s countermeasure [4]. On smaller micro-controllers, execution time will depend on the hardware architecture, but a good approximate being that the two inversions can be considered at most as costly as two exponentiations modulo t (if

$|t| = |r|$). Our proposal is therefore more efficient than Joye and Ciet's solution [6] where two extra totient calculations are needed. We can also consider that our algorithm is about as efficient as Giraud's countermeasure [7], if our exponentiation algorithm only has the property that an attacker cannot distinguish squarings from multiplications. In the case of RSA with CRT where the exponents are masked, the exponentiation algorithm could be unbalanced contrary to Montgomery-Ladder algorithm [13]. If we suppose that the modulus and the exponent are randomized by a 64-bit random integer, we perform about $\left\lfloor \frac{|p|}{2} \right\rfloor - 96$ and $\left\lfloor \frac{|q|}{2} \right\rfloor - 96$ fewer modular multiplications for each exponentiation, but with larger operands. As an example, if the implementation is carried out on a 32-bit architecture, one Montgomery modular multiplication with two operands of length k 32-bit words, theoretically requires $2k(k+1)$ single-precision multiplications. Thus, one Montgomery-Ladder exponentiation requires about $128k^2(k+1)$ single-precision multiplications with clear data, versus $96(k+2)^2(k+3)$ for a classical exponentiation with randomized data. As a consequence, for p and q greater than about 640 bits, our algorithm would be slightly more efficient than Giraud's one. Under this size, it would be the opposite.

Memory consumption Our countermeasure requires about as much memory as Aumüller et al.'s [4] and Joye and Ciet's implementation [6]. Obviously, it requires far less memory than Giraud's proposal [7] where memory consumption is a real disadvantage. We can consider in Figure 3 that β_p, β_q are not kept in RAM during the calculations of S'_p and S'_q since i_{pr} and i_{qr} can be stored on the stack. β_p and β_q can be calculated "on-the-fly" when needed. In the same way for the value m_p, only $m_p \bmod r^2$ can be stored on the stack. The instant when memory consumption is the highest occurs during the recombination (as in a classical RSA-CRT signature), except that S'_p, respectively S'_q, have length $|p| + 2|r|$, and $|q| + 2|r|$. The final result has length $|N| + 2|r|$. Some crypto-processors are not able to perform the final verification $(S - R_4 - qi_q \cdot (R_3 - R_4)) \cdot N \equiv 0 \bmod Nr^2$ if N is a 2K integer, since the co-processor register size may be limited to 2K. In this case, the final verification can be replaced with $S - R_4 - qi_q \cdot (R_3 - R_4) \equiv 0 \bmod r^2$.

Personalization management The proposed implementation only requires the usual parameters needed for the computation, the input message m and the classical RSA-CRT key set (p, q, d_p, d_q, i_q).

Code Size The countermeasure is mainly based on arithmetic operations already developed for the RSA-CRT signature. Only the modular inversion, which is also based on classical arithmetic operations, should be implemented. The code of the modular inversion is often contained in products that supply the RSA signature as they supply the RSA key generation too. Even if the code of modular inversion must be added, this leads to an acceptable code size overhead.

5 Conclusion

This paper presents an original algorithm which computes secure exponentiations in an arbitrary integer ring $(\mathbb{Z}_N, +, \cdot)$ where $N \in \mathbb{N}^*$. Our countermeasure mechanism can be applied to secure any cryptosystem requiring exponentiations in rings or finite fields of integers, such as Diffie-Hellman key exchange [14], El Gamal decryption [15], RSA in straightforward mode [9], Schnorr [16], DSA [17], KCDSA [18] and so forth. However, it is especially relevant in the case of RSA with CRT where it constitutes an efficient defense line against Bellcore attack.

Reviewing related work on CRT implementation of RSA and considering simultaneously all practical constraints faced by cryptographic devices, our solution matches all desirable requirements.

Although here side-channel attacks have not been studied, our CRT implementation of RSA can be simply associated with appropriate countermeasures against simple and differential side-channel attacks.

Acknowledgments The author wishes to thank Pascal Paillier, Mathieu Chartier and the CHES2008 reviewers for helpful remarks on the preliminary version of this paper.

References

1. Sun Microsystems Inc.: Javacard 2.2.2 - application programming interface. Technical report (2006)
2. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of checking cryptographic protocols for faults. Lecture Notes in Computer Science **1233** (1997) 37–51
3. Shamir, A.: Method and apparatus for protecting public key schemes from timing and fault attacks, U.S. Patent Number 5,991,415. (November 1999 (also presented at the rump session of EUROCRYPT '97))
4. Aumüller, C., Bier, P., Fischer, W., Hofreiter, P., Seifert, J.P.: Fault attacks on rsa with crt: Concrete results and practical countermeasures. In B.S. Kaliski Jr., c.K., Paar, C., eds.: Cryptographic Hardware and Embedded Systems — CHES 2002. Volume 2523 of Lecture Notes in Computer Science. (2002) 260–275
5. Blömer, J., Otto, M., Seifert, J.P.: A new crt-rsa algorithm secure against bellcore attacks. In: CCS '03: Proceedings of the 10th ACM conference on Computer and communications security, New York, NY, USA, ACM (2003) 311–320
6. Joye, M., Ciet, M.: Practical fault countermeasures for chinese remaindering based rsa. In Breveglieri, L., Koren, I., eds.: 2nd Workshop on Fault Diagnosis and Tolerance in Cryptography - FDTC 2005. (2005)
7. Giraud, C.: Fault resistant rsa implementation. In Breveglieri, L., Koren, I., eds.: 2nd Workshop on Fault Diagnosis and Tolerance in Cryptography — FDTC 2005. (2005) 142–151
8. Kim, C.H., Quisquater, J.J.: How can we overcome both side channel analysis and fault attacks on rsa-crt? In Breveglieri, L., Gueron, S., Koren, I., Naccache, D., Seifert, J.P., eds.: FDTC. (2007) 21–29

9. Rivest, R.L., Shamir, A., Adelman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Technical Report MIT/LCS/TM-82 (1977)
10. Joye, M., Lenstra, A.K., Quisquater, J.J.: Chinese remaindering based cryptosystems in the presence of faults. Journal of Cryptology: the journal of the International Association for Cryptologic Research **12**(4) (1999) 241–245
11. Joye, M., Paillier, P.: Gcd-free algorithms for computing modular inverses. In B.S. Kaliski Jr., c.K., Paar, C., eds.: CHES. (2003) 243–253
12. Yen, S.M., Kim, S., Lim, S., Moon, S.: Rsa speedup with residue number system immune against hardware fault cryptanalysis. In: ICISC '01: Proceedings of the 4th International Conference Seoul on Information Security and Cryptology, London, UK, Springer-Verlag (2002) 397–413
13. Joye, M., Yen, S.: The montgomery powering ladder. In B.S. Kaliski Jr., c.K., Paar, C., eds.: Cryptographic Hardware and Embedded Systems — CHES 2002. Volume 2523 of Lecture Notes in Computer Science. (2002) 291–302
14. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory **IT-22**(6) (1976) 644–654
15. ElGamal, T.: A public-key cryptosystem and a signature scheme based on discrete logarithms. In: CRYPTO. (1984) 10–18
16. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of Cryptology **4(3)** (1991) 161–174
17. National Institute of Standards and Technology: Digital Standard Signature. Federal Information Processing Standards Publications **186** (1994)
18. Lim, Lee: A study on the proposed korean digital signature algorithm. In: ASIACRYPT: Advances in Cryptology – ASIACRYPT: International Conference on the Theory and Application of Cryptology, LNCS, Springer-Verlag (1998) 175–186

A Proof of Theorem 1

Claim. Let N and R be integers such that $\gcd(N, R) = 1$, let $\beta = (N \cdot (N^{-1} \bmod R))$ and $\alpha = 1 - \beta \bmod NR$. Then α and β are non zero elements verifying the following properties:

1. $\alpha^2 = \alpha \bmod NR$
2. $\beta^2 = \beta \bmod NR$
3. $\alpha\beta = 0 \bmod NR$ (α and β are zero divisors in $(\mathbb{Z}_{NR}, +, \cdot)$)

Proof. This trivially comes from the definition of α and β.

Lemma 1. *Let N and r be integers such that $\gcd(N, r) = 1$, let $\beta = N \cdot (N^{-1} \bmod r^2)$ and $\alpha = 1 - \beta \bmod Nr^2$. Then, for any $d \in \mathbb{N}^*$ and any pair $(A, B) \in (\mathbb{Z}_{Nr^2} \times \mathbb{Z}_{Nr^2})$:*

$$(\alpha A + \beta B)^d = \alpha A^d + \beta B^d \bmod Nr^2 \tag{1}$$

Proof. Let us take $R = r^2$. Since $\alpha\beta = 0 \bmod Nr^2$, for any $d \in \mathbb{N}^*$ and for any $(A, B) \in (\mathbb{Z}_{Nr^2})^2$, we get:

$$(\alpha A + \beta B)^d = (\alpha A)^d + (\beta B)^d \bmod Nr^2 = \alpha A^d + \beta B^d \bmod Nr^2 \ ,$$

as $\alpha^d = \alpha$ and $\beta^d = \beta$ modulo Nr^2.

Lemma 2. *Let N and r be coprime integers and $\beta = N \cdot (N^{-1} \bmod r^2)$. For any $d \in \mathbb{N}^*$, we have:*

$$\beta \cdot (1+r)^d = \beta \cdot (1+dr) \bmod Nr^2 \tag{2}$$

Proof. Since $\beta = 0 \bmod N$, the equation holds modulo N. It also holds modulo r^2 since $\beta = 1 \bmod r^2$ and for any $d \in \mathbb{N}^*$, $(1+r)^d = 1 + dr \bmod r^2$. By Chinese remaindering, the equation therefore holds modulo Nr^2.

□

Finally, combining Equations (1) and (2), we get the exponentiation identity of Theorem 1, for any $m \in \mathbb{Z}_{Nr^2}$ and for any $d \in \mathbb{N}^*$:

$$(\alpha m + \beta \cdot (1+r))^d = \alpha m^d + \beta \cdot (1+dr) \bmod Nr^2$$

B Details concerning success probabilities of fault attacks

Let us consider the fault model defined in 4.1. Assume that the attacker modifies a value A ($A = B \bmod C$) and obtains a random value $\underline{A}$ uncorrelated to A. We give here a generic expression of a success probability for passing the test $\underline{A} = B \bmod C$ where C is a t-bit integer. We force $2^{t-1} < C < 2^t$, $C = 1 \bmod 2$. According to our recommendations in Section 3.4, r is odd, its most significant bit is one and we can deduce the same property for p. We suppose that C is uniform. We note E the event that the fault is undetected, $\Pr[E]$ the total probability of E, $\Pr[E \mid C]$ the probability of E assuming C, $\Pr[c = C]$ the probability of taking an element c in the considered set S such that $c = C$. Since the random result obtained is uniformly distributed, we know that:

$$\Pr[E \mid C] = \frac{1}{C} \tag{3}$$

We want to compute $\Pr[E]$. Let $S = \big\{C \text{ s.t. } 2^{t-1} < C < 2^t \text{ and } C = 1 \bmod 2\big\}$. From the total probability Theorem, we have:

$$\Pr[E] = \sum_{C \in S} \left(\Pr[E \mid C] \cdot \Pr[c = C]\right) \tag{4}$$

Since C is uniform:

$$\Pr[c = C] = \frac{1}{|S|} \tag{5}$$

Replacing Identities (3) and (5) in Equation (4), we get:

$$\Pr[E] = \frac{1}{|S|} \cdot \sum_{C \in S} \frac{1}{C}$$

Let $\bar{S} = \big\{C \text{ s.t. } 2^{t-1} < C < 2^t \text{ and } C = 0 \bmod 2\big\}$, then:

$$\sum_{C \in S \cup \bar{S}} \frac{1}{C} = [\ln C]_{2^{t-1}}^{2^t} = \ln(2^t) - \ln(2^{t-1}) = t \ln 2 - (t-1) \ln 2 = \ln 2$$

Since, $|S| = |\bar{S}|$, we may approximate:

$$\Pr[E] = \frac{1}{|S|} \cdot \sum_{C \in S} \frac{1}{C} \approx \frac{1}{|S|} \cdot \frac{1}{2} \sum_{C \in \cup \bar{S}} \frac{1}{C} = \frac{1}{|S|} \cdot \frac{\ln 2}{2}$$

Hence:

$$\Pr[E] \approx \frac{1}{|S|} \cdot \frac{ln2}{2} = \frac{1}{2^{t-2}} \cdot \frac{ln2}{2} = 2^{-(t-1)} \ln 2$$

This explains the probability values $2^{-(|p|-1)} \ln 2$, $2^{-(|p'|-1)} \ln 2$ and $2^{-(|r|-1)} \ln 2$.

Given the same C, we now assume that the attacker modifies a value A ($A = B \bmod C^2$) and obtains a random value $\underline{A}$ uncorrelated to A. We apply the same argument, we compute the success probability for passing the test $\underline{A} = B \bmod C^2$. In this case:

$$\Pr[E \mid C] = \frac{1}{C^2} \tag{6}$$

The Identity (5) still applies here. Hence, replacing Identities (5) and (6) in Equation (4):

$$\Pr[E] = \frac{1}{|S|} \cdot \sum_{C \in S} \frac{1}{C^2}$$

$$\sum_{C \in S \cup \bar{S}} \frac{1}{C^2} = \left[-\frac{1}{C} \right]_{2^{t-1}}^{2^t} = -\frac{1}{2^t} + \frac{1}{2^{t-1}} = -\frac{1}{2^t} + \frac{2}{2^t} = 2^{-t}$$

In the same way, we may approximate:

$$\Pr[E] = \frac{1}{|S|} \cdot \sum_{C \in S} \frac{1}{C^2} \approx \frac{1}{|S|} \cdot \frac{1}{2} \sum_{C \in S \cup \bar{S}} \frac{1}{C^2} = \frac{1}{|S|} \cdot \frac{1}{2^{t+1}}$$

And therefore:

$$\Pr[E] \approx \frac{1}{|S|} \cdot 2^{-(t+1)} = 2^{-(t-2)} \cdot 2^{-(t+1)} = 2^{-2t+1}$$

This leads to the probability value $2^{-(2|r|+1)}$.