On the Use of RSA Public Exponent to Improve Implementation Efficiency and Side-Channel Resistance

Christophe Giraud(✉)

Cryptography and Security Group, Oberthur Technologies,
4, Allée du Doyen Georges Brus, 33600 Pessac, France
c.giraud@oberthur.com

Abstract. Since the end of the nineties, cryptographic developers must not only provide fast implementations but they must also take Side-Channel Analysis and Fault Injection into account. From that time, many side-channel and fault countermeasures have been proposed to reach a double goal: provide a high level of security while having the smallest impact on performance and memory consumption. In the particular case of RSA, the knowledge of the public exponent has been used to propose the most efficient fault countermeasure in terms of security and performance. However so far no study has been published which exploits such a variable to improve RSA efficiency and side-channel resistance.

In this paper, we fill this gap by proposing an original CRT-RSA implementation which makes use of the knowledge of the public exponent. In particular, we investigate an efficient method using only 4 private key parameters out of 5 and we also propose a free message blinding method to reinforce side-channel resistance.

Keywords: CRT-RSA · Efficient implementation · Side-channel countermeasure

1 Introduction

1996 was one of the most amazing years for the Crypto community. Indeed in a few months, two revolutionary attacks called *Side-Channel Analysis* (SCA) [21] and *Fault Injection* (FI) [5] were published. These two attacks definitely affected *practitioners* by changing the way of implementing cryptographic algorithms and they also challenged *theoreticians* to design new cryptosystems meant to resist such threats. The first kind of attack takes advantage of physical interactions between the embedded device and its environment during the execution of the cryptosystem to recover information on the corresponding secret key [24]. Indeed, it was noticed that these interactions, such as the device power consumption [22] or its electromagnetic radiation [15], contain information on the operations and on the variables manipulated by the device. The second kind of attack aims to disturb the correct execution of the algorithm and uses the corresponding faulty output to obtain information on the secret key [19]. Of course, numerous countermeasures have been published since 1996 to efficiently counteract these attacks and the fields of SCA and FI are now the most active fields of research in cryptography.

As well as being the first practical public-key cryptosystem, RSA [29] has also been the most widely used for many years, especially in electronic signature schemes. It has thus been a privileged target for cryptologists to mount effective SCA and FI and to propose efficient countermeasures. Concerning FI-resistant RSA implementation, the most efficient method consists in using the public exponent to verify the signature before outputting it. Whereas such an approach has been published more than 15 years ago [6], no publication deals with the exploitation of the public exponent to improve RSA implementation efficiency and side-channel resistance. This article addresses such an open topic.

The rest of this paper is organised as follows. Section 2 briefly presents the state-of-the-art of secure CRT-RSA implementation on embedded devices. In Sect. 3, we present our new approach to implement CRT-RSA by taking advantage of the knowledge of the public exponent. After presenting a functional version of our implementation, we improve its side-channel resistance by proposing a free message blinding method. This new approach is then compared with the state-of-the-art implementation. Finally, we conclude in Sect. 4.

2 State-of-the-Art Secure CRT-RSA Implementation

In this section, we firstly describe RSA before presenting the main SCA and FI countermeasures used nowadays to obtain a secure implementation.

2.1 RSA Presentation

In the following we briefly recall how to compute the RSA signature in both standard and CRT modes.

Let N denote the public modulus being the product of two secret large prime integers p and q. Let d refer to the private exponent and e refer to the public exponent satisfying $d \cdot e \equiv 1 \bmod \varphi(N)$, where φ denotes Euler's totient function. The RSA signature of a message $m \in \mathbb{Z}_N$ is then obtained by computing $S = m^d \bmod N$. To verify the signature, one computes $S^e \bmod N$ and checks if the corresponding result is equal to m.

In embedded systems, most RSA implementations use the Chinese Remainder Theorem (CRT) which yields a speed-up factor of four [13]. By using the CRT, the signature generation is composed of two exponentiations $S_p = m^{d_p} \bmod p$ and $S_q = m^{d_q} \bmod q$, where $d_p = d \bmod (p-1)$ and $d_q = d \bmod (q-1)$. The signature is then obtained by recombining S_p and S_q, which is usually done by using Garner's formula [16]:

$$S = S_q + q \cdot (i_q \cdot (S_p - S_q) \bmod p), \tag{1}$$

where $i_q = q^{-1} \bmod p$.

We depict in Algorithm 1 the algorithmic of a standard CRT-RSA implementation as described above.

Algorithm 1. Standard CRT-RSA signature

Inputs: The message m and the private key (p, q, d_p, d_q, i_q)
Output: The signature S of the message m

```
// First exponentiation
1. $S_p \leftarrow m^{d_p} \bmod p$
// Second exponentiation
2. $S_q \leftarrow m^{d_q} \bmod q$
// Recombination
3. $S \leftarrow S_q + q \cdot (i_q \cdot (S_p - S_q) \bmod p)$
4. **return** $S$
```

Although RSA cryptosystem using signature protocol PSS [27] is proved secure against theoretical cryptanalysis, it can be broken if straightforwardly implemented on embedded devices by using Side-Channel Analysis or Fault Injection. In the next sections, we present the main countermeasures which are generally implemented to counteract SCA and FI.

2.2 SCA Countermeasures

When published, SCA was divided into two groups: *Simple Side-Channel Analysis* (SSCA) and *Differential Side-Channel Analysis* (DSCA). The first kind aims at recovering information on the secret key by using the side-channel leakage of only one execution of the algorithm whereas DSCA uses several executions of the algorithm and applies statistical analysis to the corresponding measurements to exhibit information on the secret key.

In the particular case of RSA, the most common countermeasure to prevent SSCA consists in using exponentiation methods where the sequence of modular operations does not depend on the corresponding secret exponent. Example of such exponentiations are the Montgomery Ladder [20] or the Atomic exponentiation [8]. Concerning DSCA countermeasures, most techniques aim at randomizing the message and the exponents. This can be done for instance by applying *additive masking* to these variables [9]. In such a case, one can pick four 64-bit random values k_i, $i \in \{0, \cdots, 3\}$, and compute $S'_p = (m + k_0 \cdot p)^{d_p + k_1 \cdot (p-1)} \bmod 2^{64} \cdot p$ and $S'_q = (m + k_2 \cdot q)^{d_p + k_3 \cdot (q-1)} \bmod 2^{64} \cdot q$ before combining them using the CRT-recombination. The expected signature is finally obtained by performing a final reduction modulo $N = p \cdot q$.

Instead of using additive masking to blind the message, one can apply *multiplicative masking* [24] which consists generally in multiplying the message with $r^e \bmod N$ where r is a non null random value. The blinding is then removed at the end of the computation by multiplying the final result with $r^{-1} \bmod N$. However, the inverse computation to obtain $r^{-1} \bmod N$ is costly in terms of both performance and memory consumption. Such an approach is therefore generally avoided in favor of the traditional additive masking.

When combining both SSCA and DSCA countermeasures, RSA implementations resist most kind of side-channel attacks. However, a third class of SCA called *Horizontal Analysis* (HA) has been published recently and could defeat such implementations by using only one execution of the algorithm [4,10,11,31]. This kind of attack aims generally at distinguishing if each modular operation is a multiplication with the input message or not. To counteract such powerful attacks, one must randomize the order of the single-precision multiplications [4,11] or randomize the blinding of each operand before each long integer multiplication [10].

2.3 FI Countermeasures

RSA has been the first cryptosystem to be analysed versus Fault Injection [6]. In the case of CRT-RSA, only one fault injected during one of the two exponentiations provides a faulty signature which allows the attacker to recover one of the two secret primes p or q. For instance, if a fault is injected during the computation of S_p leading to a faulty signature $\widetilde{S}$ then one can notice that $\widetilde{S} \equiv S \bmod q$ but $\widetilde{S} \not\equiv S \bmod p$. Therefore, the secret parameter q can be easily recovered by computing the gcd of $S - \widetilde{S}$ and N. The other private key parameters can then be straightforwardly deduced.

To protect RSA against such a threat, dozens of countermeasures have been proposed over the last decade. These methods can be divided into four different groups. The first group is based on Shamir's method proposed in [30]. The idea is to perform the two exponentiations over $\mathrm{GF}(p \cdot t)$ and $\mathrm{GF}(q \cdot t)$ respectively where t is a small random value and then compare both results modulo t. Amongst the numerous variants of Shamir's method, only the improved version of Vigilant's proposal is considered secure against fault injection [12]. The second methodology has been proposed by Giraud in which the fault detection comes from the exponentiation algorithm itself [17]. He pointed out that by using the Montgomery powering ladder [20], both values $m^{d-1} \bmod N$ and $m^d \bmod N$ are available at the end of the computation. These values can then be used to verify the integrity of the exponentiation by testing if m times the first value is equal to the second one. This method has then been extended in [7,28]. The third group corresponds to the *infective computation* method which has been introduced by Yen et al. in [32]. The idea of the countermeasure consists in modifying the signature if a fault is detected such that it provides no information to the attacker. Despite several proposals, each and every infective method has been broken [3]. The fourth and last kind of countermeasure consists in verifying the signature by using the public exponent before outputting it [6].

In the rest of this paper we assume that the public exponent is small, typically less than $2^{16}+1$, which is nearly always the case in practice[1]. Therefore the fourth approach presented above is the most efficient way to counteract fault attacks on CRT-RSA in terms of both security and performances.

2.4 Summary

To sum up Sect. 2, we depict in Algorithm 2 the skeleton of a state-of-the-art secure CRT-RSA [2, Sect. 6.1].

Algorithm 2. Secure CRT-RSA signature

INPUTS: A message m, the public exponent e and the private key (p, q, d_p, d_q, i_q)
OUTPUT: The signature S of the message m

```
1. Generate three 64-bit random values $k_0$, $k_1$ and $k_2$
// Message blinding
2. $m' \leftarrow m + k_0 \cdot p \cdot q$
// First secure exponentiation
3. $d'_p \leftarrow d_p + k_1 \cdot (p-1)$
4. $S'_p \leftarrow m'^{d'_p} \bmod 2^{64} \cdot p$          [Using an SSCA-HA-resistant expo.]
// Second secure exponentiation
5. $d'_q \leftarrow d_q + k_2 \cdot (q-1)$
6. $S'_q \leftarrow m'^{d'_q} \bmod 2^{64} \cdot q$          [Using an SSCA-HA-resistant expo.]
// Secure recombination
7. $S' \leftarrow S'_q + q \cdot (i_q \cdot (S'_p - S'_q) \bmod (2^{64} \cdot p))$
// Signature verification
8. $N \leftarrow p \cdot q$
9. if $S'^e \bmod N = m$ then
10.     return $S' \bmod N$
11. else
12.     Security action
```

3 A New Approach

Whereas the public exponent has been used for more than 15 years to counteract Fault Injection, no study has been done to investigate how such a value can be used to improve RSA performance and side-channel resistance. This is unfortunate since when setting an RSA private key, the corresponding public exponent e is often known. For example in the case of EMV banking applications [14],

[1] According to [23, Table 1], 99.95 % of the RSA public keys which are used nowadays use one of the 15 following values as public exponent: 3, 5, 7, 11, 13, 17, 19, 21, 23, 35, 41, 47, 2^8+1, $2^{16}-1$ and $2^{16}+1$. In particular, more than 95 % of the public exponents are equal to $2^{16}+1$.

there are only 2 different public exponents possible (3 or $2^{16}+1$) and the correct one can be recovered from the private key by using 2 multiplications [18]. It is therefore interesting to investigate an alternative implementation of CRT-RSA taking advantage of the knowledge of the public exponent value.

3.1 Generic Description

In practice, the CRT-recombination is implemented by using Garner's formula as presented in (1) since it is the most efficient formula published so far. However, the CRT-recombination can also be performed by using the Gauss recombination:

$$S = p \cdot i_p \cdot S_q + q \cdot i_q \cdot S_p \bmod N \tag{2}$$

where $S_p = m^{d_p} \bmod p$, $S_q = m^{d_q} \bmod q$, $i_p = p^{-1} \bmod q$ and $i_q = q^{-1} \bmod p$. Of course, such a method requires either to consume extra memory to add the extra private parameter i_p or to perform a costly inverse computation to obtain such a value on-the-fly. However, we explain in the following that CRT-RSA using such a recombination can be more efficient than using Garner's method if the public exponent is known.

Our new method is based on Relation (3):

$$(m \cdot q^e)^{d_p - 1} \cdot m \cdot q^{e-2} \equiv i_q \cdot S_p \bmod p \tag{3}$$

Proof. When expanding the first term of left part of (3), we obtain:

$$\begin{aligned} (m \cdot q^e)^{d_p-1} &\equiv m^{d_p-1} \cdot q^{e\cdot(d_p-1)} \bmod p & (4)\\ &\equiv m^{d_p-1} \cdot q^{e\cdot d_p - e} \bmod p & (5)\\ &\equiv m^{d_p-1} \cdot q^{1-e} \bmod p & (6)\end{aligned}$$

Therefore

$$\begin{aligned} (m \cdot q^e)^{d_p-1} \cdot m \cdot q^{e-2} &\equiv m^{d_p-1} \cdot q^{1-e} \cdot m \cdot q^{e-2} \bmod p & (7)\\ &\equiv m^{d_p} \cdot q^{1-e+e-2} \bmod p & (8)\end{aligned}$$

This straightforwardly leads to (3). □

Obviously, a similar relation is obtained modulo q:

$$(m \cdot p^e)^{d_q - 1} \cdot m \cdot p^{e-2} \equiv i_p \cdot S_q \bmod q \tag{9}$$

Finally, one may note that (2) is equivalent to the following relation:

$$S = p \cdot (i_p \cdot S_q \bmod q) + q \cdot (i_q \cdot S_p \bmod p) \bmod N \tag{10}$$

Therefore, by combining Relations (3) and (9) with Relation (10), the signature $S = m^d \bmod N$ of a message m can be computed by using the following relation:

$$S = p \cdot S1_q + q \cdot S1_p \bmod N \tag{11}$$

where

$$S1_p = (m \cdot q^e)^{d_p - 1} \cdot m \cdot q^{e-2} \bmod p,$$
$$S1_q = (m \cdot p^e)^{d_q - 1} \cdot m \cdot p^{e-2} \bmod q.$$

We depict in Algorithm 3 the algorithmic of our new method.

Algorithm 3. Our new CRT-RSA signature implementation with e known

INPUTS: A message m, the public key e and a subpart of the private key (p, q, d_p, d_q)
OUTPUT: The signature S of the message m

```
// First exponentiation
1. $q_1 \leftarrow m \cdot q^{e-2} \bmod p$
2. $q_2 \leftarrow q_1 \cdot q^2 \bmod p$                    $[q_2 = m \cdot q^e \bmod p]$
3. $S1_p \leftarrow q_2^{d_p - 1} \cdot q_1 \bmod p$         $[S1_p = i_q \cdot S_p \bmod p]$
// Second exponentiation
4. $p_1 \leftarrow m \cdot p^{e-2} \bmod q$
5. $p_2 \leftarrow p_1 \cdot p^2 \bmod q$                    $[p_2 = m \cdot p^e \bmod q]$
6. $S1_q \leftarrow p_2^{d_q - 1} \cdot p_1 \bmod q$         $[S1_q = i_p \cdot S_q \bmod q]$
// Recombination
7. $S \leftarrow p \cdot S1_q + q \cdot S1_p \bmod (p \cdot q)$
8. return $S$
```

Comparison with the Standard Method. The main advantage of Algorithm 3 over Algorithm 1 consists in a much smaller key since it does not require the private parameter i_q. This leads to a gain of $\log_2(N)/2 - \log_2(e)$ bits of memory to store the key. When using a 2048-bit RSA for instance, we gain 125 bytes when $e = 2^{16} + 1$. Such an improvement is of uttermost importance on embedded devices where the memory space is very limited.

By comparing the complexity of the standard method depicted in Algorithm 1 and of our new proposal depicted in Algorithm 3, one can notice that the performances are very similar for public exponents which are generally used. For instance, if $e = 3$ (resp. $e = 2^{16} + 1$) then we add 8 (resp. 68) modular operations to perform the two exponentiations. In the case of a 2048-bit RSA, this corresponds to a tiny overhead of 0.3 % (resp. 2.2 %) on average in terms of modular operations[2]. Moreover, one may note that the modular reduction of Step 7 of Algorithm 3 can be replaced by a conditional subtraction with N since $p \cdot S1_q + q \cdot S1_p$ is always smaller than $2 \cdot N$.

[2] To compute these figures, we assume that a modular exponentiation using d_p, $d_p - 1$, d_q or $d_q - 1$ as exponent requires 1023 squares and 512 multiplications on average, i.e. 1585 modular operations.

One can also notice that the key generation of our method is slightly faster than the traditional one, cf. Algorithms 5 and 6 in Appendix A. Indeed in such a case, the costly inverse computation of $i_q = q^{-1} \bmod p$ is not necessary.

Last but not least, we do not need to change the key structure defined in the Java Card standard [26] to use our method. Indeed, we just need to store the public exponent e instead of the parameter i_q. To do so, the methods setPQ and getPQ, which are meant to set and to output the value of i_q respectively, must be adapted to fit our approach while keeping in line with the Java Card standard functionality. The first method setPQ must compute the public key e from the private key parameters (p, q, d_p, d_q) and store it in the buffer PQ. Most of the time, such a computation can be performed by using the efficient method of [18]. Regarding the method getPQ, it must output $q^{-1} \bmod p$ instead of outputting the content of the buffer PQ. Even if this inverse computation is costly, this is not a problem in practice since this method is almost never used.

3.2 A Free Message Blinding Method

In this section, we take advantage of the new approach previously described to provide a very efficient message blinding method to counteract Side-Channel Analysis.

We notice that by replacing q in Relation (3) by $q' = q \cdot r \bmod p$ where r is a random different from 0 modulo p and modulo q, we obtain:

$$(m \cdot q'^{e})^{d_p - 1} \cdot m \cdot q'^{e-2} \equiv i_q \cdot S_p \cdot r^{-1} \bmod p \tag{12}$$

Similarly, by replacing p with $p' = p \cdot r \bmod q$ in Relation (9), we obtain:

$$(m \cdot p'^{e})^{d_q - 1} \cdot m \cdot p'^{e-2} \equiv i_p \cdot S_q \cdot r^{-1} \bmod q \tag{13}$$

By combining Relations (12) and (13) with Relation (11), we obtain a randomized signature S' which is equal to:

$$\begin{align} S' &= p \cdot S1'_q + q \cdot S1'_p \bmod N \tag{14}\\ &= r^{-1} \cdot S \bmod N \tag{15}\end{align}$$

where

$$\begin{aligned} S1'_p &= (m \cdot q'^{e})^{d_p - 1} \cdot m \cdot q'^{e-2} \bmod p,\\ S1'_q &= (m \cdot p'^{e})^{d_q - 1} \cdot m \cdot p'^{e-2} \bmod q. \end{aligned}$$

The expected signature S is then obtained by multiplying S' with r modulo N.

To reach a fully secure CRT-RSA, one need also to blind the exponents d_p and d_q, use SSCA-HA-resistant exponentiations and to verify the signature by using the public exponent. We depict in Algorithm 4 such an implementation.

Algorithm 4. Secure CRT-RSA signature using our new approach

```
INPUTS: The message $m$, the public key $e$ and a subpart of the private key $(p, q, d_p, d_q)$
OUTPUT: The signature $S$ of the message $m$

1. Generate a random value $r$ of size $\log_2(N)/2$ such that $r \not\equiv 0 \bmod p$ and $r \not\equiv 0 \bmod q$
2. Generate two 64-bit random values $k_0$ and $k_1$
// First exponentiation
3. $q' \leftarrow q \cdot r \bmod p$
4. $q_1 \leftarrow m \cdot q'^{e-2} \bmod p$
5. $q_2 \leftarrow q_1 \cdot q'^2 \bmod p$                         $[q_2 = m \cdot r^e \cdot q^e \bmod p]$
6. $d'_p \leftarrow d_p + k_0 \cdot (p-1)$
7. $S1'_p \leftarrow q_2^{d'_p - 1} \cdot q_1 \bmod p$              [Using an SSCA-HA-resistant expo.]
// Second exponentiation
8. $p' \leftarrow p \cdot r \bmod q$
9. $p_1 \leftarrow m \cdot p'^{e-2} \bmod q$
10. $p_2 \leftarrow p_1 \cdot p'^2 \bmod q$                        $[p_2 = m \cdot r^e \cdot p^e \bmod q]$
11. $d'_q \leftarrow d_q + k_1 \cdot (q-1)$
12. $S1'_q \leftarrow p_2^{d'_q - 1} \cdot p_1 \bmod q$             [Using an SSCA-HA-resistant expo.]
// Recombination
13. $S' \leftarrow p \cdot S1'_q + q \cdot S1'_p \bmod (p \cdot q)$    $[S' = r^{-1} \cdot S \bmod N]$
// Signature verification
14. $N \leftarrow p \cdot q$
15. if $(r \cdot S')^e \equiv m \bmod N$ then
16.     return $r \cdot S' \bmod N$
17. else
18.     Security action
```

Comparison with the Standard State-of-the-Art Method. Firstly, Algorithm 4 inherits from the various advantages presented in Sect. 3.1 over the standard Algorithm 2. In particular, it does not require the private key parameter i_q. Since Algorithms 2 and 4 both require the value of the public exponent e, we gain in memory the size of one private key parameter, i.e. $\log_2(N)/2$ bits. Moreover, since Algorithm 2 requires the full private key and the public exponent, the latter must be computed on-the-fly in the context of Java Card environment where the format of the CRT-RSA key is standardized and an extra parameter cannot be added. In such a context, our method simply stores the public exponent e instead of the private parameter i_q.

From a performance point of view, our method keeps the original size of the operands whereas the traditional additive masking used in Algorithm 2 requires to work with 64-bit longer operands and modulus. Since the performance of the crypto-processor is directly linked to the size of the variables which are used, Algorithm 4 is thus expected to be faster than Algorithm 2 since we work with smaller operand length.

Table 1. Performance improvement of Algorithm 4 compared to Algorithm 2 on a smart card providing a 32-bit modular multiplication co-processor.

CRT-RSA key size in bits	Performance improvement of Algorithm 4 compared to Algorithm 2
1024	14.2 %
2048	8.2 %

Table 1 represents our analysis on a smart card providing a 32-bit modular multiplication co-processor. The difference between Algorithms 4 and 2 could be much more significant in some cases, especially with co-processors having a precision of 128 bits.

Moreover, when comparing our method versus the original multiplicative message blinding, one can notice that the costly inverse computation $r^{-1} \bmod N$ is done for free during the exponentiations.

Our approach is not only faster but it also provides various advantages versus Side-Channel Analysis. For instance, since we use Gauss' method to recombine the results of the exponentiations, our method is not vulnerable to specific side-channel attacks on Garner's formula such as the ones presented in [1,25].

4 Conclusion

Despite the fact that the public exponent has been used for a long time to protect RSA implementation against Fault Injection, no study has been done to investigate the benefit we can obtain from a performance and side-channel point of view. In this paper, we present a novel approach to implement CRT-RSA making use of the knowledge of the public exponent. We show that we can shrink the key length and reach the same level of performance. Moreover, we also show that this new approach can be combined with multiplicative message blinding method without any overhead, leading to the most efficient message blinding scheme published so far.

Acknowledgments. The author would like to thank Guillaume Barbu, Alberto Battistello, Emmanuelle Dottax and Gilles Piret for their comments on the preliminary version of this paper.

A CRT-RSA Key Generation Algorithms

Algorithm 5 describes the standard CRT-RSA key generation and Algorithm 6 presents the specific CRT-RSA key generation for our new method. One can observe that the costly inverse computation $q^{-1} \bmod p$ is no more necessary. Moreover, since the public exponent is always provided as input for the key generation, we do not need extra-computation to provide such a value.

Algorithm 5. Standard CRT-RSA key generation

```
INPUTS: The public exponent $e$ and the expected key bit length $n$
OUTPUT: The private key $(p, q, d_p, d_q, i_q)$

1. Generate a $n/2$-bit random prime $p$
2. Generate a $n/2$-bit random prime $q$
3. $d_p \leftarrow e^{-1} \bmod (p-1)$
4. $d_q \leftarrow e^{-1} \bmod (q-1)$
5. $i_q \leftarrow q^{-1} \bmod p$
6. return $(p, q, d_p, d_q, i_q)$
```

Algorithm 6. CRT-RSA key generation for our new method

```
INPUTS: The public exponent $e$ and the expected key bit length $n$
OUTPUT: The private key $(p, q, d_p, d_q, e)$

1. Generate a $n/2$-bit random prime $p$
2. Generate a $n/2$-bit random prime $q$
3. $d_p \leftarrow e^{-1} \bmod (p-1)$
4. $d_q \leftarrow e^{-1} \bmod (q-1)$
5. return $(p, q, d_p, d_q, e)$
```

References

1. Amiel, F., Feix, B., Villegas, K.: Power analysis for secret recovering and reverse engineering of public key algorithms. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 110–125. Springer, Heidelberg (2007)
2. Barbu, G., Battistello, A., Dabosville, G., Giraud, C., Renault, G., Renner, S., Zeitoun, R.: Combined attack on CRT-RSA. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 198–215. Springer, Heidelberg (2013)
3. Battistello, A., Giraud, C.: Fault analysis of infective AES computations. In: Fischer, W., Schmidt, J.-M. (eds.) Fault Diagnosis and Tolerance in Cryptography - FDTC 2014, pp. 101–107. IEEE Computer Society (2014)
4. Bauer, A., Jaulmes, E., Prouff, E., Wild, J.: Horizontal and vertical side-channel attacks against secure RSA implementations. In: Dawson, E. (ed.) CT-RSA 2013. LNCS, vol. 7779, pp. 1–17. Springer, Heidelberg (2013)
5. Bonech, D., DeMillo, R., Lipton, R.: New Threat Model Breaks Crypto Codes. Bellcore Press Release, Morristown (1996)
6. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of checking cryptographic protocols for faults. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 37–51. Springer, Heidelberg (1997)
7. Boscher, A., Naciri, R., Prouff, E.: CRT RSA algorithm protected against fault attacks. In: Sauveron, D., Markantonakis, K., Bilas, A., Quisquater, J.-J. (eds.) WISTP 2007. LNCS, vol. 4462, pp. 229–243. Springer, Heidelberg (2007)

8. Chevallier-Mames, B., Ciet, M., Joye, M.: Low-cost solutions for preventing simple side-channel analysis: side-channel atomicity. IEEE Trans. Comput. **53**(6), 760–768 (2004)
9. Clavier, C., Feix, B.: Updated recommendations for blinded exponentiation vs. single trace analysis. In: Prouff, E. (ed.) COSADE 2013. LNCS, vol. 7864, pp. 80–98. Springer, Heidelberg (2013)
10. Clavier, C., Feix, B., Gagnerot, G., Giraud, C., Roussellet, M., Verneuil, V.: ROSETTA for single trace analysis. In: Galbraith, S., Nandi, M. (eds.) INDOCRYPT 2012. LNCS, vol. 7668, pp. 140–155. Springer, Heidelberg (2012)
11. Clavier, C., Feix, B., Gagnerot, G., Roussellet, M., Verneuil, V.: Horizontal correlation analysis on exponentiation. In: Soriano, M., Qing, S., López, J. (eds.) ICICS 2010. LNCS, vol. 6476, pp. 46–61. Springer, Heidelberg (2010)
12. Coron, J.-S., Giraud, C., Morin, N., Piret, G., Vigilant, D.: Fault attacks and countermeasures on vigilant's RSA-CRT algorithm. In: Breveglieri, L., Joye, M., Koren, I., Naccache, D., Verbauwhede, I. (eds.) Fault Diagnosis and Tolerance in Cryptography - FDTC 2010, pp. 89–96. IEEE Computer Society (2010)
13. Couvreur, C., Quisquater, J.-J.: Fast decipherment algorithm for RSA public-key cryptosystem. Electron. Lett. **18**(21), 905–907 (1982)
14. EMV. Integrated Circuit Card Specifications for Payment Systems - Book 2 - Security and Key Management, June 2008
15. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: concrete results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, p. 251. Springer, Heidelberg (2001)
16. Garner, H.: The residue number system. IRE Trans. Electron. Comput. **8**(6), 140–147 (1959)
17. Giraud, C.: An RSA implementation resistant to fault attacks and to simple power analysis. IEEE Trans. Comput. **55**(9), 1116–1120 (2006)
18. Joye, M.: Protecting RSA against fault attacks: the embedding method. In: Breveglieri, L., Gueron, S., Koren, I., Naccache, D., Seifert, J.-P. (eds.) Fault Diagnosis and Tolerance in Cryptography - FDTC 2009, pp. 41–45. IEEE Computer Society (2009)
19. Joye, M., Tunstall, M.: Fault Analysis in Cryptography. Information Security and Cryptography. Springer, Heidelberg (2012)
20. Joye, M., Yen, S.-M.: The Montgomery powering ladder. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 291–302. Springer, Heidelberg (2003)
21. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
22. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
23. Lenstra, A.K., Hughes, J.P., Augier, M., Bos, J.W., Kleinjung, T., Wachter, C.: Ron was wrong, Whit is right. Cryptology ePrint Archive, report 2012/064 (2012). http://eprint.iacr.org/
24. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks - Revealing the Secrets of Smartcards. Springer, New York (2007)
25. Novak, R.: SPA-based adaptive chosen-ciphertext attack on RSA implementation. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 252–262. Springer, Heidelberg (2002)
26. Oracle Corp. Application Programming Interface, Java Card Platform, Version 3.0.4 Classic Edition (2011)

27. PKCS #1. RSA Cryptography Specifications Version 2.1. RSA Laboratories (2003)
28. Rivain, M.: Securing RSA against fault analysis by double addition chain exponentiation. In: Fischlin, M. (ed.) CT-RSA 2009. LNCS, vol. 5473, pp. 459–480. Springer, Heidelberg (2009)
29. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978)
30. Shamir, A.: How to check modular exponentiation. In: Eurocrypt'97 rump session (1997)
31. Walter, C.D.: Sliding windows succumbs to Big Mac attack. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 286–299. Springer, Heidelberg (2001)
32. Yen, S.-M., Kim, S., Lim, S., Moon, S.-J.: RSA speedup with residue number system immune against hardware fault cryptanalysis. In: Kim, K. (ed.) ICISC 2001. LNCS, vol. 2288, pp. 397–413. Springer, Heidelberg (2002)