Practical Analysis of RSA Countermeasures Against Side-Channel Electromagnetic Attacks

Guilherme Perin, Laurent Imbert, Lionel Torres, and Philippe Maurine

LIRMM/UM2 - 161, Rue Ada
34095 Montpellier

Abstract. This paper analyzes the robustness of RSA countermeasures against electromagnetic analysis and collision attacks. The proposed RSA cryptosystem uses residue number systems (RNS) for fast executions of the modular calculi with large numbers. The parallel architecture is protected at arithmetic and algorithmic levels by using the Montgomery Ladder and the Leak Resistant Arithmetic countermeasures. Because the architecture can leak information through control and memory executions, the hardware RNS-RSA also relies on the randomization of RAM accesses. Experimental results, obtained with and without randomization of the RNS moduli sets, suggest that the RNS-based RSA with bases randomization and secured RAM accesses is protected.
Keywords: RSA, RNS, Montgomery Exponentiation, Countermeasures, Electromagnetic Analysis.

1 Introduction

Side-Channel Attacks (SCA) are a serious threat for public-key cryptosystems and notably for the RSA [1]. These attacks aim at recovering a secret manipulated by cryptographic algorithms, by analyzing various sources of side-channel leakages (time, power consumption, electromagnetic (EM) radiations, etc) during their execution on a hardware device.

Countermeasures to prevent simple (SPA) and differential (DPA) power analysis on RSA can be categorized in algorithmic and hardware countermeasures. The Square-and-Multiply Always [2] and the Montgomery Ladder [3] ensure that all operations in the binary method run in a constant sequence of operations in order to prevent SPA like attacks. To deal with DPA attacks, the idea of algorithmic countermeasures is to randomize the message or the exponent (private key) that are processed during the execution of a modular exponentiation. However, most of these countermeasures do not provide sufficient protection against high-order DPA attacks or sophisticated SPA-attacks [4][21].

Residue Number System (RNS), coupled together with SPA-protected methods, is an interesting alternative to increase the robustness at the arithmetic level. RNS provides a natural way of masking the data and the internal computations because all intermediate values can be represented in different RNS bases. However, differential, correlation and collision EM attacks [5–8] remains fully efficient if no randomization of the RNS bases are used to effectively mask sensitive computations. This idea is the foundation of the Leak Resistant Arithmetic (LRA) concept proposed in [9].

The RSA hardware approach proposed in this work implements different countermeasures. To provide protection against correlation analyses and collision attacks, the design offers protection at arithmetic level by randomizing the moduli between two sets of RNS bases, and then implies the on-the-fly calculus of the required pre-computed constants. For the modular exponentiation, the Montgomery Ladder algorithm is considered even if other algorithms can be executed by our co-processor. The successive modular multiplications are computed with the RNS Montgomery algorithm [10] that needs two sets of k moduli due to the base extension part. For this crucial operation in the Montgomery multiplication, one considers the fast approximation method [11], which is derived from the Chinese Remainder Theorem. Moreover, hardware countermeasures are adopted with randomization of the RAM addresses during the reading and writing operations.

The rest of the paper is organized as follows. Section 2 give a brief state-of-art about the use of RNS for the integration of public-key algorithms. Section 3 describes the hardware module we have designed and mapped into an FPGA. Section 4 gives experimental results about the robustness of the RNS-RSA implemented on our crypto-module. Finally, a conclusion is drawn in section 5.

2 Preliminaries

2.1 Residue Number System

In the Residue Number System [12], an integer X, is represented according to a base $\mathcal{B} = (b_1, b_2, ..., b_k)$ of relatively prime integers, called moduli. The number X in base $\mathcal{B}$ is thus represented by a k-tuple of positive integers $\langle X \rangle_{\mathcal{B}} = (x_1, x_2, \ldots, x_k)$, where $x_i = X \bmod b_i$, *i.e.* the remainder of the division of X by the modulo b_i, denoted $|X|_{b_i}$ in the sequel. Arithmetic operations ($\pm, \times$) are then performed modulo $B = \prod_{i=1}^{k} b_i$. To recover the original number X (modulo B), given the residues x_i, one may apply the Chinese Remainder Theorem (CRT):

$$|X|_B = \left| \sum_{i=1}^{k} B_i |x_i B_i^{-1}|_{b_i} \right|_B \text{, where } B_i = \frac{B}{b_i}$$

The forward conversion is a key step before starting any computation in RNS. From the radix-2^w representation of $X = \sum_{j=0}^{n-1} X_j 2^{wj}$, the residues x_i are obtained, for all $b_i \in \mathcal{B}$, by:

$$x_i = |X|_{b_i} = \left| \sum_{j=0}^{n-1} \left| X_j |2^{wj}|_{b_i} \right|_{b_i} \right|_{b_i} , \qquad (1)$$

where the constants $|2^{wj}|_{b_i}$ are pre-computed for all i, j to speed up the forward conversion in RNS hardware modules by computing all residues in parallel.

2.2 RNS Montgomery Exponentiation

The core of any RSA implementation is a modular exponentiation of x, namely $x^e \bmod N$ is computed and e is the private exponent. This is the operation to be protected! To deal with timing and SPA attacks, in this work we adopted the Montgomery Ladder exponentiation version in RNS, as given in Algorithm 1. One may observe that the computations are performed over two RNS bases $\mathcal{A}$ and $\mathcal{B}$.

The pre-computed terms for the modular exponentiation are $B \bmod N$ and $B^2 \bmod N$ in bases $\mathcal{A}$ and $\mathcal{B}$. The operation $MM(x, y, N, \mathcal{B}, \mathcal{A})$ returns the RNS Montgomery Multiplication result $xyB^{-1} \bmod N$ in the two RNS bases $\mathcal{A}$ and $\mathcal{B}$. For this crucial operation, the recent improvement proposed in [13] was adopted to accelerate the original method [11] by 18%. This acceleration is provided by rearranging the computations within the so-called base extensions (BE). In [13], two different strategies are proposed for that operation and, in our approach, we adopted the fast approximation method, also called as Posch-Posch method [14]. Given x_i the elements of X in base $\mathcal{B}$, where $x_i = X \bmod b_i$ for $i = 1..k$, the fast approximation method ensures the existence of a certain integer $\lambda < k$, a CRT-correction coefficient, such that:

$$X = \left| \sum_{i=1}^{k} B_i |x_i B_i^{-1}|_{b_i} \right|_B = \sum_{i=1}^{k} B_i |x_i B_i^{-1}|_{b_i} - \lambda.B \qquad (2)$$

and λ can be calculated by:

$$\lambda = \left\lfloor \sum_{i=1}^{k} \frac{B_i |x_i B_i^{-1}|_{b_i}}{B} \right\rfloor = \left\lfloor \sum_{i=1}^{k} \frac{|x_i B_i^{-1}|_{b_i}}{b_i} \right\rfloor = \left\lfloor \frac{1}{2^w} \sum_{i=1}^{k} |x_i B_i^{-1}|_{b_i} \right\rfloor \qquad (3)$$

In Equation 3, $|x_i B_i^{-1}|_{b_i}/b_i$ may be approximated by $|x_i B_i^{-1}|_{b_i}/2^w$ as $b_i = 2^w - c_i$, and $c_i > 0$. The resulting RNS Montgomery algorithm using the fast approximation base extension, with a cost of $2k+7$ single multiplications at each RNS moduli, is shown in Algorithm 2.

The RNS Montgomery algorithm requires a set of precomputed terms in RNS bases $\mathcal{A}$ and $\mathcal{B}$. The term $B_{i,j}NB^{-1}$ refers to the computation $|BNB^{-1}/b_i|_{a_j}$ and $A_{i,j}$ refers to the computation $|A/a_i|_{b_j}$, for $\forall i, j$. The modular exponentiation employing the Algorithm 2 ensures that $X^e \bmod N < 2N$.

```
Algorithm 1: RNS Montgomery Ladder Exponentiation
Data: $x$ in $\mathcal{A} \cup \mathcal{B}$, where $\mathcal{A} = (a_1, a_2, ..., a_k)$, $\mathcal{B} = (b_1, b_2, ..., b_k)$, $A = \prod_{i=1}^{k} a_i$,
      $B = \prod_{i=1}^{k} b_i$, $\gcd(A, B) = 1$, $\gcd(B, N) = 1$ and $e = (e_{n-1} ... e_1 e_0)_2$.
Result: $z = x^e \bmod N$ in $\mathcal{A} \cup \mathcal{B}$
Pre-Computations: $|B \bmod N|_{\mathcal{A}\cup\mathcal{B}}$ and $|B^2 \bmod N|_{\mathcal{A}\cup\mathcal{B}}$
$A_0 = B \bmod N$    (in $\mathcal{A} \cup \mathcal{B}$)
$A_1 = MM(x, B^2 \bmod N, N, \mathcal{B}, \mathcal{A})$    (in $\mathcal{A} \cup \mathcal{B}$)
for $i = n-1$ to $0$ do
    $A_{\overline{e_i}} = MM(A_{\overline{e_i}}, A_{e_i}, N, \mathcal{B}, \mathcal{A})$    (in $\mathcal{A} \cup \mathcal{B}$)
    $A_{e_i} = MM(A_{e_i}, A_{e_i}, N, \mathcal{B}, \mathcal{A})$    (in $\mathcal{A} \cup \mathcal{B}$)
end
$A_0 = MM(A_0, 1, N, \mathcal{B}, \mathcal{A})$    (in $\mathcal{A} \cup \mathcal{B}$)
```

Algorithm 2: RNS Montgomery Multiplication with Fast Approx. BE [13]

```
Data: $x, y$ in $\mathcal{A} \cup \mathcal{B}$, where $\mathcal{A} = (a_1, a_2, ..., a_k)$, $\mathcal{B} = (b_1, b_2, ..., b_k)$, $A = \prod_{i=1}^{k} a_i$,
      $B = \prod_{i=1}^{k} b_i$, $\gcd(B, A) = 1$, $1 \leq x, y < N$, $B > 4N$ and $A > 2N$
Result: $w = xyB^{-1} \bmod N$ (in $\mathcal{A} \cup \mathcal{B}$)
Pre-Computations in $\mathcal{A}$: $B^{-1}$, $B_{i,j}N.B^{-1}$ for $i, j = 1..k$, $-B.N.B^{-1}$, $A_j^{-1}$ for $j = 1..k$
Pre-Computations in $\mathcal{B}$: $-N^{-1}B_i^{-1}$ for $i = 1..k$, $A_{i,j}$ for $i, j = 1..k$, $-A$
$s = |x.y|_{\mathcal{B} \cup \mathcal{A}}$
———————————— Base extension 1 ——————————————
$q_{b_i} = |s_i(-N^{-1}B_i^{-1})|_{b_i}$ for i=1..k
$f = \left\lfloor \left(\sum_{i=1}^{k} q_{b_i}\right) / 2^w \right\rfloor$
$w_{a_i} = |s_i B^{-1} + \sum_{j=1}^{k} q_{b_j}(B_{i,j}NB^{-1}) - f.B.N.B^{-1}|_{a_i}$ for i=1..k
———————————— Base extension 2 ——————————————
$q_i = |w_{a_i}(A_i^{-1})|_{a_i}$ for i=1..k
$f = \left\lfloor \left(2^{w-1} + \sum_{i=1}^{k} q_{b_i}\right) / 2^w \right\rfloor$
$w_{b_i} = |\sum_{j=1}^{k} q_j A_{i,j} - f.A|_{b_i}$ for i=1..k
```

2.3 RNS Bases Randomization - The LRA Countermeasure

DPA attacks explore the relation between the power consumption and the internal variables to recover the bits of the private key. The leak resistant arithmetic (LRA) countermeasure [9] provides a way for completely masking the internal computations and then protect against differential or correlation power (or EM) analysis at arithmetic level.

Before each modular exponentiation, the two set of bases $\mathcal{A}$ and $\mathcal{B}$ (each of size k) are randomly selected among a set of $2k$ integers. In this way, an integer w (an intermediate result in the modular exponentiation represented in the Montgomery domain) has $C_k^{2k} \approx 2^{2k}/\sqrt{\pi k}$ different RNS representations in bases $\mathcal{A}$ or $\mathcal{B}$ and it offers a high-level of randomization. These randomly selected RNS bases are then used during the entire computation. The authors of [9] also suggested to reinforce the robustness by selecting new bases during the exponentiation, possibly before each MM. However, this second approach may become much slower; it implies two additional MM each time new RNS bases are chosen, or even four extra MM if the Montgomery Ladder is used for the exponentiation. Here, the bases randomization are performed once before each exponentiation, using Montgomery Powering Ladder as depicted in Algorithm 3. In the application of LRA countermeasure, the on-the-fly computation of Montgomery constants $B \bmod N$ and $B^2 \bmod N$ is solved by using the pre-computed term $AB \bmod N$ (in $\mathcal{A} \cup \mathcal{B}$) in the two first Montgomery multiplications. Note the order of $\mathcal{A}$ and $\mathcal{B}$ in these two first calls of MM in Algorithm 3.

The RNS Montgomery Multiplication needs pre-computed constants related to the random choice of RNS bases $\mathcal{A}$ and $\mathcal{B}$. These pre-computed constants must be obtained on-the-fly before each modular exponentiation. The LRA pre-computations necessary for the Montgomery multiplication are:

a) $|B_i^{-1}|_{b_i} = \left|\prod_{j=1}^{k} b_j^{-1}\right|_{b_i} = \left||...||b_0^{-1}.b_1^{-1}|_{b_i}.b_2^{-1}|_{b_i}...b_k^{-1}|_{b_i}\right|_{b_i}$

Algorithm 3: RNS Montgomery Powering Ladder with LRA [9]

```
Data: $x$ in $\mathcal{A} \cup \mathcal{B}$, where $\mathcal{A} = (a_1, a_2, ..., a_k)$, $\mathcal{B} = (b_1, b_2, ..., b_k)$, $A = \prod_{i=1}^{k} a_i$,
      $B = \prod_{i=1}^{k} b_i$, $\gcd(A, B) = 1$, $\gcd(B, N) = 1$ and $e = (e_{n-1}...e_1e_0)_2$.
Result: $z = x^e \bmod N$ in $\mathcal{A} \cup \mathcal{B}$
Pre-Computations: $|AB \bmod N|_{\mathcal{A}\cup\mathcal{B}}$
$A_0 = MM(1, AB \bmod N, N, \mathcal{A}, \mathcal{B})$    (in $\mathcal{A} \cup \mathcal{B}$)
$A_1 = MM(x, AB \bmod N, N, \mathcal{A}, \mathcal{B})$    (in $\mathcal{A} \cup \mathcal{B}$)
for $i = n - 1$ to $0$ do
    $A_{\overline{e_i}} = MM(A_{\overline{e_i}}, A_{e_i}, N, \mathcal{B}, \mathcal{A})$    (in $\mathcal{A} \cup \mathcal{B}$)
    $A_{e_i} = MM(A_{e_i}, A_{e_i}, N, \mathcal{B}, \mathcal{A})$    (in $\mathcal{A} \cup \mathcal{B}$)
end
$A_0 = MM(A_0, 1, N, \mathcal{B}, \mathcal{A})$    (in $\mathcal{A} \cup \mathcal{B}$)
```

b) $|A_i^{-1}|_{a_i} = \left|\prod_{j=1}^{k} a_j^{-1}\right|_{a_i} = \left||...||a_0^{-1}.a_1^{-1}|_{a_i}.a_2^{-1}|_{a_i}...a_k^{-1}|_{a_i}\right|_{a_i}$

c) $|B^{-1}|_{a_i} = \left|\prod_{j=1}^{k} b_j^{-1}\right|_{a_i} = \left||...||b_0^{-1}.b_1^{-1}|_{a_i}.b_2^{-1}|_{a_i}...|b_k^{-1}|_{a_i}\right|_{a_i}$

d) $|A^{-1}|_{b_i} = \left|\prod_{j=1}^{k} a_j^{-1}\right|_{b_i} = \left||...||a_0^{-1}.a_1^{-1}|_{b_i}.a_2^{-1}|_{b_i}...|a_k^{-1}|_{b_i}\right|_{b_i}$

e) $|B|_{a_i} = \left|\prod_{j=1}^{k} b_j\right|_{a_i} = \left||...||b_0.b_1|_{a_i}.b_2|_{a_i}...b_k|_{a_i}\right|_{a_i}$

f) $|A|_{b_i} = \left|\prod_{j=1}^{k} a_j\right|_{b_i} = \left||...||a_0.a_1|_{b_i}.a_2|_{b_i}...a_k|_{b_i}\right|_{b_i}$

And then, we obtain:

1. $|-N^{-1}B_i^{-1}|_{b_i} = |-N^{-1}|.|B_i^{-1}|_{b_i}$ and $|-N^{-1}A_i^{-1}|_{a_i} = |-N^{-1}|.|A_i^{-1}|_{a_i}$, for $i = 1..k$
2. $|B_i|_{a_j} = |B|_{a_j}.|b_i^{-1}|_{a_j}$ and $|A_i|_{b_j} = |A|_{b_j}.|a_i^{-1}|_{b_j}$, for $j = 1..k$
3. $|B_iNB^{-1}|_{a_j} = |B_i|_{a_j}.|N|_{a_j}.|B^{-1}|_{a_j}$ and $|A_iNA^{-1}|_{b_j} = |A_i|_{b_j}.|N|_{b_j}.|A^{-1}|_{b_j}$, for $j = 1..k$
4. $|-B|_{a_i} = |B|_{a_i}.|-1|_{a_i}$ and $|-A|_{b_i} = |A|_{b_i}.|-1|_{b_i}$, for $i = 1..k$
5. $|-BNB^{-1}|_{a_i} = |-B|_{a_i}.|N|_{a_i}.|B^{-1}|_{a_i}$ and $|-ANA^{-1}|_{b_i} = |-A|_{b_i}.|N|_{b_i}.|A^{-1}|_{b_i}$, for $i = 1..k$

Then, all constants $|b_i^{-1}|_{b_j}$, $|a_i^{-1}|_{a_j}$, $|b_i^{-1}|_{a_j}$, $|a_i^{-1}|_{b_j}$ for all i, j, $|-N^{-1}|_{\mathcal{A}\cup\mathcal{B}}$, $|N|_{\mathcal{A}\cup\mathcal{B}}$, $|-1|_{\mathcal{A}\cup\mathcal{B}}$ and the RNS base sets $\mathcal{A}$ and $\mathcal{B}$ should be pre-computed.

After the modular exponentiation, the result must be converted back to radix. For the LRA countermeasure, the reverse conversion using CRT-based method needs the on-the-fly computations of the values B_i and B in radix-2^w form and it represents a high level of complexity. In this case, it is adopted the Mixed-Radix System (MRS) [12] for the RNS to radix conversion. The mixed-radix system is a weighted representation of a RNS number. This method is computed in two steps: first, the MRS representation of x_i (RNS representation of X in $\mathcal{B}$) is obtained using the optimized Garner's Algorithm [17], and all the pre-computed values, the inverses $|b_i^{-1}|_{b_j}$, are obtained independently of RNS bases randomizations; second, the MRS result is converted to radix by applying the Horner's scheme, as also presented in [17]. The reverse conversion implies carry-based arithmetic. However, the time spent for these operations is negligible compared to the modular exponentiation.

3 Proposed and Developed Hardware

The proposed hardware computes the forward conversion (radix to RNS), the LRA pre-computations, the modular exponentiation and the reverse conversion (radix to RNS), using the same set of independent data-paths called RNS Units depicted in Fig. 1. The implementation follows a similar schematic than that proposed in [11] and improved in [16], called cox-rower architecture.

As described in section 2, the required LRA pre-computations, which computes the pre-computed constants for the Montgomery multiplication in the RNS bases $\mathcal{A}$ and $\mathcal{B}$, needs a set of pre-computed values. To store them, the RNS Units contain dual-port RAM memories. Then, each RNS Unit contains all pre-computed elements of all moduli of $\mathcal{A}$ and $\mathcal{B}$. It causes an overhead in terms of memory, however speeds-up the on-the-fly pre-computations.

The core of each RNS Unit is the arithmetic logic unit (ALU), which computes the modular addition/subtraction, modular products and carry-based arithmetic operations in the reverse conversion (CRT or MRS). To accelerate the modular reductions, we adopted the method proposed in [15]. This solution uses pseudo-Mersenne numbers of the form $b_i = 2^w - c_i$, where $c_i < 2^{w/2}$, for the chosen set of RNS moduli. Then, to compute $x \bmod b_i$ one first performs the following step twice:

$$x \leftarrow (x \bmod 2^w) + c_i \cdot (x/2^w) \qquad (4)$$

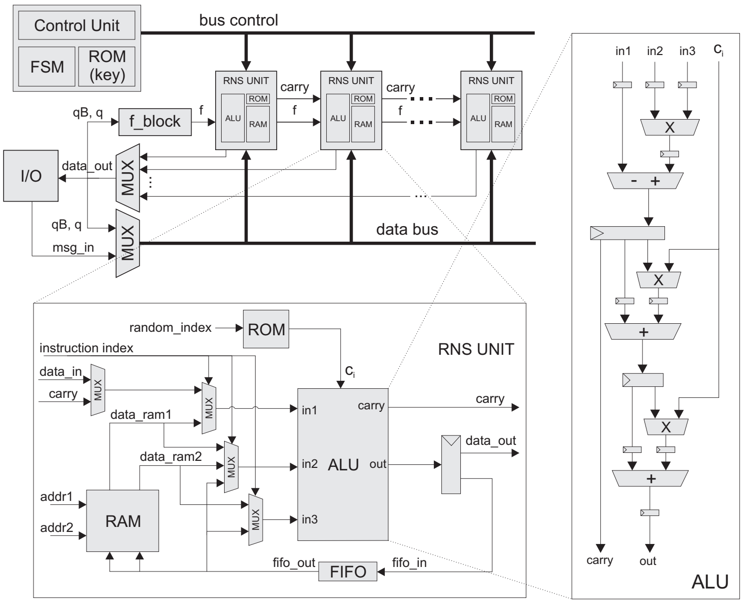

Fig. 1. RNS architecture block diagram.

Then x will be in the range of $[0, 2^{w+1}]$ and a final conditional subtraction by b_i returns the residual value. The coefficient c_i is also an input of the ALU block. As RNS bases randomizations (LRA) makes the RNS Units operate in different moduli, all c_i (for $i = 1..2k$) are stored in a ROM memory. Each RNS Unit performs operations for one RNS channel of $\mathcal{A}$ and one of $\mathcal{B}$; the selection of these channels, and the respective coefficient c_i, is defined by the *random index* input from the control unit.

The architecture also contains an adder block, called *f_block*, for computing the f values in the two base extensions. This block basically sums up all input values (q_B in the first base extension and q in the second base extension) and returns the k most significant bits of this sum, named f.

The hardware countermeasure also relies on the RAM access protection. According to the Algorithm 3 there are four registers ($A0$ in $\mathcal{A}$, $A0$ in $\mathcal{B}$, $A1$ in $\mathcal{A}$ and $A1$ in $\mathcal{B}$) for storing the intermediate values, resulting of modular multiplication or squaring executions in the binary loop of the Montgomery Ladder. So, for example, if a modular multiplication $A_0 = MM(A_0, A_1, N, \mathcal{B}, \mathcal{A})$ is executed when the exponent bit is 1, the reading and writing operations will be:

1. *read*($|A0|_{\mathcal{A}}$,$|A0|_{\mathcal{B}}$,$|A1|_{\mathcal{A}}$,$|A1|_{\mathcal{B}}$)
2. *write*($|A0|_{\mathcal{A}}$,$|A0|_{\mathcal{B}}$)

On the other hand, if a modular multiplication $A_1 = MM(A_0, A_1, N, \mathcal{B}, \mathcal{A})$ is executed when the exponent bit is 0, the reading and writing operations will be:

1. *read*($|A0|_{\mathcal{A}}$,$|A0|_{\mathcal{B}}$,$|A1|_{\mathcal{A}}$,$|A1|_{\mathcal{B}}$)
2. *write*($|A1|_{\mathcal{A}}$,$|A1|_{\mathcal{B}}$)

Note that same registers are read and different registers are written. EM analysis based on localized EM radiations [18] or on the control and RAM leakages [20] show that if the RAM accesses are unprotected, the private key bits can be recovered using sophisticated SEMA or location-based EM attacks. In order to randomize the register's position, and consequently the addresses, where the intermediate results $A0$ and $A1$ (in $\mathcal{A}$ and $\mathcal{B}$) are stored, we propose the scheme depicted in Fig. 2 in all RNS Units.

Considering the first modular multiplication $A_0 = MM(A_0, A_1, N, \mathcal{B}, \mathcal{A})$. The control reads the registers $A0$ and $A1$ (in $\mathcal{A}$ and $\mathcal{B}$) from the RAM address *0h+j*, *1h+j*, *2h+j* and *3h+j* (indicated by 'r') and instead of storing the modular multiplication result $A0$ (in $\mathcal{A}$ and $\mathcal{B}$) in the same positions (*0h+j* and *1h+j*), $A0$ is stored in random positions *5h+j*, *6h+j*, indicated by 'w'. Since the exponent bit $e_i = 1$, the next operation is a modular squaring $A_1 = MM(A_1, A_1, N, \mathcal{B}, \mathcal{A})$. The control reads the registers A_1 (in $\mathcal{A}$ and $\mathcal{B}$) from addresses *2h+j* and *3h+j* and instead of storing the result in the same position, it is placed at random address spaces *4h+j*, *7h+j*. In the next modular multiplication, the registers $A0$ and $A1$ will be read from the previous random positions. With this hardware countermeasure, the storing position of intermediate values changes during the modular exponentiation, blurring the EM emanations. Then, the side-channel leakage due RAM memory addressing is suppressed, because the results are

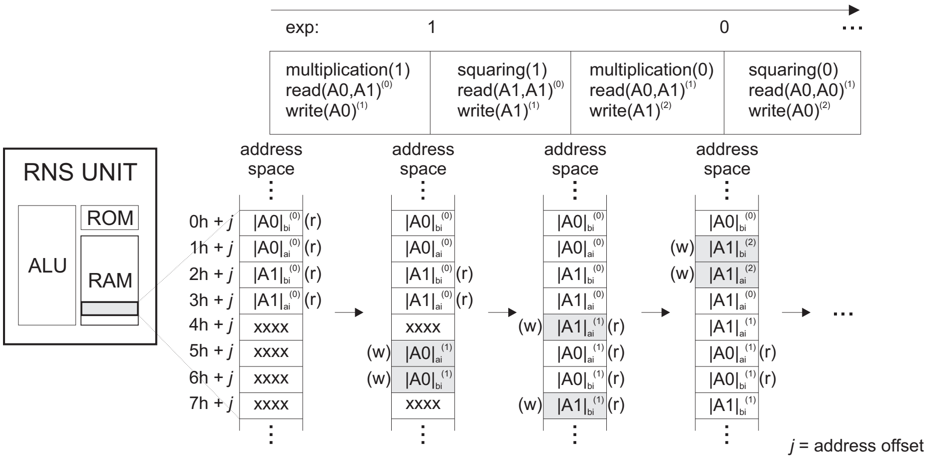

Fig. 2. RAM memory addressing randomization.

Table 1. Cycle Count and Synthesis Results.

	RSA without protection	RSA with LRA	Overhead
RSA-512 Clock Cycles			
LRA latency	0	1060	100%
Radix to RNS	48	48	0%
Mont. Expo.	78210	78210	0%
RNS to Radix	685 (CRT)	840 (MRS)	18%
Total	78943	80158	1%
Synthesis Results (*FPGA Utilization)			
4-Input LUTs	17124 (28%*)	17769 (29%*)	3%
Slices	8717 (27%*)	9510 (30%*)	8%
18×18 Mults	104 (100%*)	104 (100%*)	0%
KB (RAM)	8.5 (5%*)	118 (66%*)	92%

always stored in different addresses. Next section shows practical EM attacks on both unprotected and secured RAM.

Considering k the number of RNS moduli in each of the bases $\mathcal{A}$ and $\mathcal{B}$, the total number of clock cycles for a Montgomery multiplication is $2k + 37$. The LRA countermeasure needs an amount of $64k + 36$ clock cycles for the pre-computations. Table 1 summarizes the number of clock cycles for the 512 bits RSA, that is able to compute the CRT-RSA 1024 bits, and the synthesis results for FPGA implementation (low-cost Spartan 3E family) including the number of kilobytes that represents the pre-computed terms pre-stored before the exponentiation and the memory space needed during the exponentiation. The results are provided for the two RSA-RNS implementations. As indicated, there is a time overhead of only 1% due to the LRA countermeasure. The memory (kilobytes) and the area overheads (LUTs and Slices) due countermeasures are 92% and 3%, respectively.

4 Robustness to EM Analyses

Collision or chosen-messages pair attacks, threat modular exponentiations by exploiting the existence of identical computations. Correlation electromagnetic analysis (CEMA) seeks to recover the secret information by computing the correlation between the EM traces and some guessed intermediate values manipulated or not by the device according to the exponent bits.

To evaluate the relevance of the LRA and hardware countermeasures, we first applied these attacks on an unprotected hardware design, i.e. an RNS-RSA with fixed bases to set a robustness reference level. Then, we re-applied these attacks on our protected implementation in order to quantify the robustness enhancements. To generalize the notation of the acquired EM traces, we define the following:

$$\mathrm{EM}(\mathrm{T}_E,\mathrm{x},\mathrm{e}) = \left\{\mathrm{EM}(\mathrm{T}_M,\mathrm{x},\mathrm{e}_{n-1}), \mathrm{EM}(\mathrm{T}_S,\mathrm{x},\mathrm{e}_{n-1}), ..., \mathrm{EM}(\mathrm{T}_M,\mathrm{x},\mathrm{e}_0), \mathrm{EM}(\mathrm{T}_S,\mathrm{x},\mathrm{e}_0)\right\}$$

where $\mathrm{EM}(\mathrm{T}_E,\mathrm{x},\mathrm{e})$ is the set of all multiplication and squaring intervals during a modular exponentiation with the exponent $\mathrm{e} = \{\mathrm{e}_{n-1}, \mathrm{e}_{n-2}, ..., \mathrm{e}_1, \mathrm{e}_0\}$, input message x and:

1. $\mathrm{EM}(\mathrm{T}_M,\mathrm{x},\mathrm{e}_i)$ = EM trace of a modular multiplication (M) done during the time window T_M with the exponent bit e_i;
2. $\mathrm{EM}(\mathrm{T}_S,\mathrm{x},\mathrm{e}_i)$ = EM trace of a modular squaring (S) performed during the time window T_S with the exponent bit e_i;
3. T_E = time window of a full modular exponentiation.

We also define $V_{em}(t,x)$ as being the variation of the EM field at the time t of a modular exponentiation having x as input message.

The EM traces were collected with a measurement platform composed of: an oscilloscope (bandwidth: 2.5 GHz; sampling rate: 40 GS/s), an amplifier with a bandwidth of 200 MHz, a 200 μm probe, a motorized stage, an FPGA Spartan-3 XC3S1600 board and a PC to control the whole measurement setup.

4.1 EM Collision Attacks

Collision attacks are SPA like attacks based on the choice of pairs of messages. Basically, an adversary has to measure the power consumption or the EM emanations during the processing of these two chosen messages by the cryptosystem. Then, he has to apply a sliding procedure at the two collected traces to detect, by subtraction, the occurrence of an identical computation. Such collisions typically appear during the squaring operations of modular exponentiations. Several collision attacks have been proposed in the literature. The Doubling Attack (DA) [6] and Yen *et al*'s Attack [7] collisions are observed in squaring operations and apply on left-to-right exponentiation algorithms. Homma *et al*'s Attack [8] is a collision that also applies to right-to-left exponentiations contrarily to the DA and the Yen *et al*' attack. As explained in [8], it is based on a different choice of the input messages to provoke collisions in right-to-left and left-to-right exponentiation algorithms.

Because the Montgomery powering ladder algorithm is a left-to-right algorithm, we did consider the Doubling Attack. Following the DA procedure, we truncated, re-aligned and subtracted the EM traces and we confirmed the occurrence of the same intermediate modular squaring results. Figures 3(a) and (b) show how to select and align traces related to the chosen messages in order to have a reference and a target frame.

The first experiment was done on the unprotected RNS-RSA design, when the RNS bases are always fixed. One averaged EM trace (20 trials) has been necessary for each chosen message for identifying the occurrence of collisions using our EM platform. Fig. 3(c) shows the result of a collision analysis on the target RSA-RNS hardware implemented without countermeasures. Note the amplitude of the differential trace is near to zero where redundant computations are performed (depicted as 'region of interest').

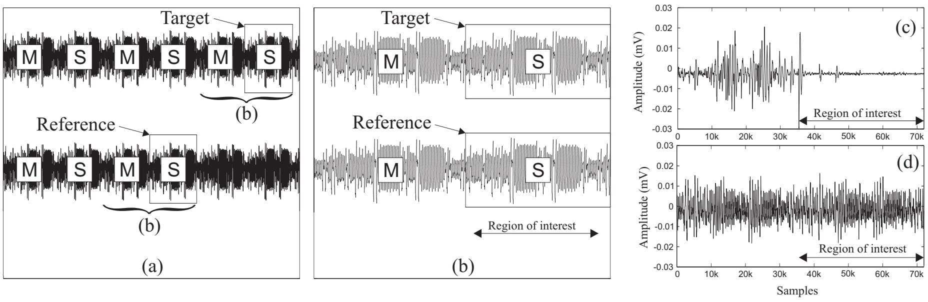

Fig. 3. (a) Electromagnetic traces. (b) Electromagnetic traces alignment for collision detecting. (c) Electromagnetic collision attack on RSA without protection and (d) with RNS bases randomizations (LRA).

To illustrate the effect of our countermeasures, Fig. 3(d) shows the differential traces when DA was applied to the RNS-RSA with randomization of RNS bases. As expected, collisions cannot be detected visually when countermeasures are activated despite the use of average mode of the oscilloscope (20 trials). To demonstrate the efficiency of the DA and quantify the effects of our countermeasures, we define a collision detection criterion by plotting the evolution of the Signal-to-Noise Ratio (SNR) with the number of trials set for the averaging. According to the DA, if the exponent presents consecutive zero bits at e_i and e_{i-1}, the EM traces EM(T_S,x,e_{i-1}) and EM(T_S,x^2,e_i) represent redundant squarings (collision). The SNR was computed according to:

$$SNR = 20.log_{10}\frac{P_{signal}}{P_{noise}} = 20.log_{10}\frac{\sigma^2_{(EM(T_S,x,e_{i-1}))}}{\sigma^2_{(EM(T_S,x,e_{i-1})-EM(T_S,x^2,e_i))}} \quad (5)$$

where $\sigma^2_{(EM(T_S,x,e_{i-1}))}$ is the variance of samples over the time window T_S corresponding to a squaring operation and $\sigma^2_{(EM(T_S,x,e_{i-1})-EM(T_S,x^2,e_i))}$ is the variance of the differential trace samples over the time window T_S. We defined SNR1 when EM(T$_S$,x,e$_{i-1}$) = EM(T$_S$,x^2,e$_i$) (collision) and SNR2 when EM(T$_S$,x,e$_{i-1}$) $\neq$ EM(T$_S$,x^2,e$_i$) (no collision). As shown in Fig. 4a, if a collision occurs, SNR1 is significantly bigger than SNR2 because the denominator of Eq. 5 is almost 0 (suppression of the signal by the collision; only the noise remains) even with no averaging.

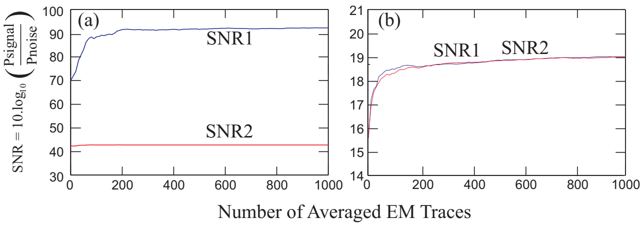

Fig. 4. SNR vs Number of averaged EM traces. (a) RSA without protection. (b) RSA with RNS bases randomizations (LRA).

As shown in Fig. 4(b), collisions cannot be detected when randomization of RNS bases countermeasure is activated, even when averaging over 1000 times the two signals.

4.2 CEMA

Correlation EM Analysis (CEMA) aims at revealing the secret key K manipulated by a circuit by analyzing the correlation between its EM emanations and guesses on the secret key. The most important the correlation is, the most likely the guess is. To apply a CEMA on an RSA the adversary should have the possibility to randomly generate the input data x of the RSA implementation to be attacked or to observe cipher texts. At the same time, he has to measure the variations of the EM field $V_{em}(t,x)$ at time t. This done, he enters in the CEMA procedure that starts by choosing a selection function.

Key Guess and Selection Function: in our case, the adversary, knowing that the considered algorithm is the Montgomery Powering Ladder, may generate 8-bits guesses on the secret key, starting by the MSB. In this way, he has a manageable set of sub-key guesses. These sub-key guesses generated, the adversary computes for each guess k, the corresponding variations of the power consumption at a chosen time of the course of the algorithm, using the Hamming Weight Model $W(x,k)$. This time typically corresponds to the computation of an intermediate value by the algorithm that depends on the sub-key. For any RSA, these intermediate values could be the Montgomery multiplication results. However,

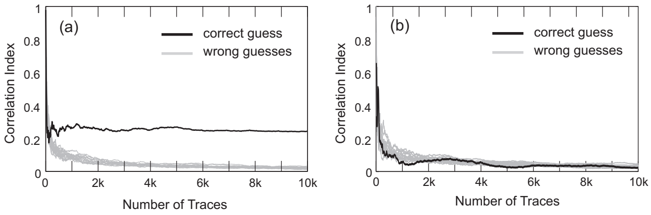

Fig. 5. Correlation Electromagnetic Attack on hardware RSA-RNS without countermeasures and (b) with LRA countermeasure with secured RAM accesses.

for RNS-RSA, the adversary must know the set of bases $\mathcal{A}$ and $\mathcal{B}$. If this is not the case, he has first to perform a long and tedious CEMA on the forward conversion (radix to RNS conversion) to recover them. In this case, the guesses on the selection function are the values of the RNS bases itself, instead of the private key bits as used in the classic CEMA.

Assuming known these RNS bases, the latter may now predict the power consumption variations (and therefore the EM field variations) with the manipulated data x for each key guess k. As the Montgomery multiplication results are obtained in parallel, he has to choose one RNS channel to compute the Hamming weight. Assuming n is the register width, the selection function follows the linear model $d(x,k) = W(x,k) - n/2$. This is done for each guess of the 8-bits sub-key. The CEMA is expected to return an estimate $\widehat{k}$ of the key by identifying the guess leading to the highest correlation value during the course of the algorithm. The correlation is computed between $d(x,k)$ and EM trace $V_{em}(t,x)$ of single measurements as function of time t :

$$c(t,k) = \frac{\sum_i (d(x_i,k) - \overline{d(x_i,k)})(V_{em}(t,x_i) - \overline{V_{em}(x_i,t)})}{\sqrt{\sum_i (d(x_i,k) - \overline{d(x_i,k)})^2}\sqrt{\sum_i (V_{em}(t,x_i) - \overline{V_{em}(t,x_i)})^2}} \tag{6}$$

To illustrate the effect of the RSA countermeasures against CEMA, we evaluated the relation between the number of EM traces and the peak margin observed for the correct guess of the sub-key related to incorrect ones. Figure 5(a) shows the evolution of the peak of the correlation index $c(t,k)$ with the number of EM traces when the architecture performs modular exponentiations with fixed RNS moduli. It is possible to guess the correct hypothesis after the processing of 500 EM traces when RSA presents no countermeasures. With the LRA countermeasure and secured RAM accesses, the correlation curve associated to the secret key has still drowned among the other correlation curves even after the processing of 10k traces.

4.3 RAM Memory Randomization

The LRA countermeasure offers a high level of randomization for the internal variables. Collisions and CEMA attacks are defeated because the Hamming

Weight of an internal variable can not be estimated to find the secret. Considering that an RSA hardware design is usually composed by arithmetic block (ALU), control (CPU), bus and memories (RAM, ROM), one may find some sources of leakages. The control and memories also performs executions depending on the exponent bits, mainly regarding the values of the memory addresses. The RAM leakages, in the case of Montgomery Ladder, will be generated by different addressing values for reading and writing multiplication or squaring results. Then, simple EM analysis, template attacks [22] or attacks based on a single execution (SE) of exponentiations [21][4][19], may explore the leakage caused by RAM addressing in the Montgomery Ladder and others SPA-protected exponentiation algorithms. SE attacks on exponentiation are also a threat against classical algorithmic countermeasures like message or exponent blinding, however they depend on the quality of the measured traces. If the SNR is very reduced, meaning that the trace contains a big amount of noise, the probability of recovering leaking information from a single trace is quite low. The analyses developed here illustrate the design vulnerabilities related to RAM access when the hardware countermeasure by addressing randomization is disregarded.

Initially, an adversary can do as follows: considering the exponentiation is always performed with a fixed exponent. He sends random messages x to the device and collects an averaged EM trace representing the multiplication when the exponent bit is 1 $[EM(T_M, x, 1)]$ and another representing the multiplication when the exponent bit is 0 $[EM(T_M, x, 0)]$. The adversary may then obtains the differential trace $E_{diff} = EM(T_M, x, 0) - EM(T_M, x, 1)$ which may reveals the leakages of control and RAM accesses, as illustrated in Fig. 6(a). The leakage is indicated by higher amplitudes during the RAM reading (r) and writing (w) executions. The procedure adopted by the adversary is:

1. Consider EM(T_E, x, e) the trace samples of a full modular exponentiation;
2. Consider $\{$EM(T_M, x, e_i) $\}$ the set of all trace samples of size T_M corresponding to the multiplications at the exponent bits e_i;
3. Set $EM(T_M, x, e_{n-1})$ as the referential trace, where $e_{n-1} = 1$ and compute the differential traces $E_{diff} = EM(T_M, x, e_{n-1})$ - $EM(T_M, x, e_{n-1-i})$, for $i = 0 : n - 1$.
4. Differential traces E_{diff} with higher amplitudes (higher variance) indicates the subtraction $EM(T_M, x, 1)$-$EM(T_M, x, 0)$.

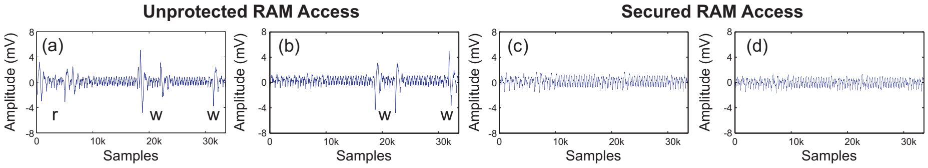

Fig. 6. (a),(c): $EM(T_M, x, 1)$-$EM(T_M, x, 0)$. (b),(d): $EM(T_S, x, 1)$-$EM(T_S, x, 0)$.

The same procedure can be verified in Fig. 6(b) by subtracting the EM traces of modular squarings when the RAM addressing is not randomized. Following the notations of Alg. 2, the amplitudes at the first samples of the differential traces represent the multiplications $s = x.y$ in the two RNS bases $\mathcal{A}$ and $\mathcal{B}$ and RAM memory is accessed in order to read the values x and y. The modular multiplication results w_A and w_B must also be stored in the RAM and this activity is indicated in the differential trace by higher amplitudes representing the RAM writing. Fig. 6(c) and (d) show the differential EM trace obtained after randomizing the RAM addresses. As we can see, these leakages were suppressed.

Now, if the exponent is randomized ($e_r = e + r.\phi(N)$), the attack processes single traces. Template and SE attacks assumes that for each multiplication $EM(T_M, x, 0)$ or $EM(T_M, x, 1)$ there is at least one sampled point in time t_i for which the amplitude of EM emanations follows a normal distribution $N(\mu_{M0}, \sigma_{M0})$ for $EM(T_M, x, 0)$ and $N(\mu_{M1}, \sigma_{M1})$ for $EM(T_M, x, 1)$. In an advantageous scenario, the point t_i may be accurately the amplitude of the EM emanation during the RAM access. To justify this model, we acquired 10000 EM traces from the RSA design mapped on the FPGA, when the private key is known. Fig. 7(a) shows the histogram of the amplitude (in mV) during a fixed point where the architecture performs memory access by writing the multiplication results in the RAM. The sample points t_i during memory accesses follow a normal distribution with different means μ_{M0}, μ_{M1} and standard deviations σ_{M0}, σ_{M1}. Yet, Fig. 7(b) illustrates the histogram during the fixed point t_i where the architecture performs a RAM writing execution after the squarings.

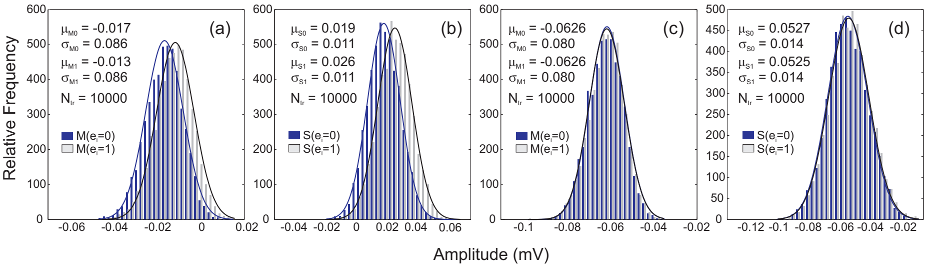

Fig. 7. Histogram and normal distribution of current measurements for (a)(b) non-randomized RAM and (c)(d) randomized RAM addressing.

With RAM addressing randomization, the same points t_i for $EM(T_M, x, 0)$ and $EM(T_M, x, 1)$ present similar distributions, meaning the SNR is reduced and SE attacks are more difficult now. Fig. 7(c) and (d) show the normal distribution for multiplication and squaring, respectively. Note the average and standard deviation are very close even for different exponent bits.

5 Conclusion

In this paper, a performance and robustness evaluation of an RSA cryptocore implemented with RNS was proposed. We evaluated countermeasures at algorithmic, arithmetic and hardware levels in order to provide protection against side-channel analysis. The Montgomery Powering Ladder exponentiation is adopted in order to protect against simple side-channel analysis. We show that collision-based attacks remain efficient against an RSA-RNS. To defeat sophisticated SPA and collision attacks, we implemented countermeasures at arithmetic and hardware levels, by randomizing the RNS bases and the RAM memory addresses, respectively. The time overhead due to countermeasures is about 1%.

References

1. R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and PKC", *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.
2. J.-S. Coron, "Resistance against differential power analysis for elliptic curve cryptography", in *CHES'99*, ser. LNCS, vol. 1717. Springer, 1999, pp. 292–302.
3. M. Joye and S.-M. Yen, "The Montgomery powering ladder", in *CHES'02*, ser. LNCS, vol. 2523. Springer, 2002, pp. 291–302.
4. A. Bauer,E. Jaulmes, E. Prouff and J. Wild "Horizontal and Vertical Side-Channel Attacks against Secure RSA Implementations", Proc. CT-RSA, pp. 1-17, 2013.
5. E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model", in *CHES'04*, ser. LNCS, vol. 3156. Springer, 2004, pp. 16–29.
6. P.-A. Fouque and F. Valette, "The doubling attack - *why upwards is better than downwards*", in *CHES'04*, ser. LNCS, vol. 2523. Springer, 2003, pp. 269–280.
7. S.M. Yen, W.C. Lien, S.J. Moon, and J.C. Ha, "Power Analysis by Exploiting Chosen Message and Internal Collisions?Vulnerability of Checking Mechanism for RSA-Decryption", Proc. Mycrypt ?05, pp. 183-195, Sept. 2005.
8. N. Homma, A. Miyamoto, T. Aoki, A. Satoh, and A. Shamir, "Comparative power analysis of modular exponentiation algorithms", *IEEE Transactions on Computers*, vol. 59, no. 6, pp. 795–807, 2010.
9. J.-C. Bajard, L. Imbert, P.-Y. Liardet, and Y. Teglia, "Leak resistant arithmetic", in *CHES'04*, ser. LNCS, vol. 3156. Springer, 2004, pp. 62–75.
10. J.-C. Bajard, L-Stéphane Didier and P. Kornerup "An RNS Montgomery Modular Multiplication Algorithm", in *IEEE Trans. Computers*, vol. 47, n.7, p. 766-776, 1998, pp. 62–75.
11. S. Kawamura, M. Koike, F. Sano, and A. Shimbo, "Cox-rower architecture for fast parallel Montgomery multiplication", in *Advances in Cryptology, EUROCRYPT'00*, ser. Lecture Notes in Computer Science, vol. 1807. Springer, 2000, pp. 523–538.
12. A. Omondi and B. Prekumar, *Reside Number Systems: Theory and Implementation.* London: Imperial College Press, 2007.
13. F. Gandino, F. Lamberti, P. Montuschi, and J.-C. Bajard, "A general approach for improving RNS montgomery exponentiation using pre-processing", in *ARITH20.* IEEE Computer Society, 2011, pp. 195–204.
14. K. Posch and R. Posch, "Modulo Reduction in Residue Number Systems", *IEEE Trans. Parallel Distrib. Syst.*, vol 6, number 5, pages 449-454, 1995.

15. J.-C. Bajard, N. Meloni, and T. Plantard, “Efficient RNS bases for cryptography”, in *Proceedings 17th IMACS World Congress, Scientific Computation, Applied Mathematics and Simulation*, 2005, pp. 113–119.
16. N. Guillermin, “A coprocessor for secure and high speed modular arithmetic”, Cryptology ePrint Archive, Report 2011/354, 2011.
17. K. Koc, “A fast algorithm for mixed-radix conversion in residue arithmetic”, *IEEE* International Conference on Computer Design: VLSI in Computers and Processors, pages 18-21, October 2-4, 1989.
18. J. Heyszl, S. Mangard, B. Heinz, F. Stumpf and Georg Sigl, “Localized EM Analysis of Cryptographic Implementations”, Proc. CT-RSA, pages 231-244, 2012.
19. J. Heyszl, A. Ibing, S. Mangard, F. Santis and G. Sigl “Clustering Algorithms for Non-Profiled Single-Execution Attacks on Exponentiations”, *IACR Cryptology ePrint Archive*, vol. 2013, pages 438, 2013.
20. G. Perin, L. Torres, P. Benoit and P. Maurine “Amplitude demodulation-based EM analysis of different RSA implementations”, *DATE*, pages 1167-1172, 2012.
21. C. Clavier, B. Feix, G. Gagnerot, M. Roussellet and V. Verneuil “Horizontal Correlation Analysis on Exponentiation”, Proc. ICICS, pp. 46-61, 2010.
22. S. Chari, J.R. Rao and P. Rohatgi, “Template Attacks”, *Cryptographic Hardware and Embedded Systems, CHES’02*, ser. LNCS, vol. 2523. Springer, 2002, pp. 13–28.