Parallel FPGA Implementation of RSA with Residue Number Systems

— Can side-channel threats be avoided? —

EXTENDED VERSION*

Mathieu Ciet**, Michael Neve[1]***, Eric Peeters[1] and Jean-Jacques Quisquater[1]

UCL Crypto Group
Place du Levant, 3
1348 Louvain-La-Neuve, Belgium.
{mneve, peeters, quisquater}@dice.ucl.ac.be
mathieu.ciet@innova-card.com
http://www.dice.ucl.ac.be/crypto

Abstract. In this paper, we present a new parallel architecture to avoid side-channel analyses such as: timing attack, simple/differential power analysis, fault induction attack and simple/differential electromagnetic analysis. We use a Montgomery Multiplication based on Residue Number Systems. Thanks to RNS, we develop a design able to perform an RSA signature in parallel on a set of identical and independent coprocessors. Of independent interest, we propose a new DPA countermeasure in the framework of RNS. It is only (slightly) memory consuming (1.5 KBytes). Finally, we synthesized our new architecture on FPGA and it presents promising performance results. Even if our aim is to sketch a secure architecture, the RSA signature is performed in less than 160 ms, with competitive hardware resources. To our knowledge, this is the first proposal of an architecture counteracting electromagnetic analysis apart from hardware countermeasures reducing electromagnetic radiations.

Keywords: RSA, Residue Numbers Systems, Side-Channels, SPA, DPA, EMA, Counter-measures, FPGA implementations.

1 Introduction

Implementation of public key cryptography requests the manipulation of large numbers, typically 1024 bits for most current applications like RSA [1]. That is the reason why Residue Number Systems (RNS for short) can be very useful. RNS have the main advantage of fast additions, fast

* The original paper has been published in the Proceedings of the 46th IEEE Midwest Symposium on Circuits and Systems, Dec. 03, Egypt. In the present paper, we inserted explanations in order to provide a more detailed version. This work comes as clarification towards the questions and comments we received.

** The original work has been done when the first author was PhD student at the UCL Crypto Group. Since November 2003, Mathieu Ciet is working at Innova Card, Avenue Coriandre, 13 600 La Ciotat, France.

*** Supported by the FRIA Belgium fund.

multiplications, carry-free, high speed arithmetic, some fault detection, possible error correction and foremost *parallel implementations*.

Many studies have been carried out on RNS [5, 6, 30, 31, 33], but, as far as we know, only Kawamura *et al.* propose a full hardware implementation of RSA with RNS in [30]. They describe a new RNS base extension algorithm and implement the whole system in an LSI prototype based on the Cox-Rower architecture [22] that lays in an efficient bases conversion.

Another important objective for crypto-algorithms implementation is to counteract side-channel analysis. Physical and side-channel attacks refer to attacks that exploit the system implementation. Avoiding these attacks requires hardware and software countermeasures. We consider here countermeasures that can be directly added in the design of a processor. Kocher *et al.* introduced the notion of *side-channel analysis* in [23, 24] and showed the importance for an implementation to be resistant against side-channel analysis and leakages from power consumption. Resistance against fault analysis [11, 12] is another issue: sensitive information may leak when the cryptosystem operates under unexpected conditions. More recently, in [17, 29] a new type of analysis has been found, based on electromagnetic radiations of the processor when a crypto-algorithm is processed, (see also [3]), called *ElectroMagnetic Analysis*.

In this paper, we try to tackle the problem at its root in order to design an architecture that can resist some side-channels attacks. A design able to perform an RSA signature in parallel, in a set of identical and independent coprocessors denoted *cells*, is presented here. The concept has originally been proposed in [27].

This paper is organized as follows: Section 2 briefly introduces basics on Residue Number Systems. Then, in Section 3, Montgomery multiplication in RNS is reminded. Section 4 deals with implementation strategies. Generalized Mersenne numbers allow us to reduce memory requirements compared to moduli of form $2^\kappa - 1$, but also efficient modular reduction. Then, details on side-channel analysis countermeasures are explained: a new differential power analysis countermeasure, which is for free in complexity, is presented and a "mobile" architecture to avoid electromagnetic analysis is proposed. In Section 5, the trade-off between pre-computations and computing time is presented. Our new architecture has been simulated on FPGA. Further on, in Section 6, implementations strategies and performance results are presented. Finally, in Section 7, we conclude.

2 Residue Number Systems

Let us introduce the basic terminology [6]:

1. The vector $\{m_1, m_2, \ldots, m_k\}$ forms a set of moduli, called the RNS-base β, where the m_i's are relatively prime.
2. M is the product $\prod_{i=1}^{k} m_i$ and defines the dynamic range of the system.

3. The vector $\{x_1, x_2, \ldots, x_k\}$ is the RNS representation of an integer X, less than M, where $x_i = \langle X \rangle_{m_i} = X \bmod m_i$. Any integer X belonging to $\mathbb{Z}/M\mathbb{Z}$ has a unique representation, in base β.
4. The operations of addition, subtraction and multiplication are defined over $\mathbb{Z}/M\mathbb{Z}$ as:

$$A \pm B = (\langle a_1 \pm b_1 \rangle_{m_1}, \ldots, \langle a_k \pm b_k \rangle_{m_k}) \quad ,$$

$$A \times B = (\langle a_1 \times b_1 \rangle_{m_1}, \ldots, \langle a_k \times b_k \rangle_{m_k}) \quad .$$

   These equations illustrate the parallel carry-free nature of the RNS.
5. The reconstruction of X from its residues $\{x_1, x_2, \ldots, x_k\}$ is based on the Chinese Remainder Theorem:

$$X = \left\langle \sum_{i=0}^{k} \langle \gamma_i x_i \rangle_{m_i} M_i \right\rangle_M \quad ,$$

   where

$$M = \prod_{i=1}^{k} m_i; \quad M_i = \frac{M}{m_i}; \quad \gamma_i = \langle M_i^{-1} \rangle_{m_i} \quad .$$

6. The vector $\{x'_1, \ldots, x'_k\}, 0 \le x'_i < m_i$ is the Mixed Radix System (MRS) representation of an integer X smaller than M, such that:

$$X = x'_1 + x'_2 m_1 + x'_3 m_1 m_2 + \ldots + x'_k \prod_{i=1}^{k-1} m_i \quad .$$

7. Comparison and division are very difficult operations to perform on the RNS representation [19, 20, 39]. That is the reason why Montgomery multiplication is well suited to RNS.

3 Montgomery Multiplication in RNS

In 1985, Montgomery introduced a method [26], widely used nowadays, for modular multiplication that requires no division. Let R be an integer such that $\gcd(R, N) = 1$ and $N' = -N^{-1} \mod R$ then the following equation holds:

$$\frac{P + (PN' \mod R)N}{R} \equiv PR^{-1} \mod N \quad . \tag{1}$$

This method was adapted to RNS by Posch and Posch [31]. Referring to Eq. (1), they propose to pick $\widetilde{M}$ (product of elements of a base $\widetilde{\beta}$) as R. Thereby, all operations of multiplication and addition can be performed in parallel. However, division requires a base extension. Two RNS bases β and $\widetilde{\beta}$ are chosen large enough to represent all intermediate results.

If $4N < M, \widetilde{M}$ then output length matches input length of the next modular multiplication of an exponentiation.

In Algorithm 1, we briefly recall how the Montgomery multiplication algorithm in RNS proceeds [22].

$R = MM(A, B, N, M, \widetilde{M})$

Input: $\langle A \rangle_{\beta \cup \widetilde{\beta}}, \langle B \rangle_{\beta \cup \widetilde{\beta}}$ (where $A, B < 2N$)

Output: $\langle R \rangle_{\beta \cup \widetilde{\beta}}$ (where $R \equiv AB\widetilde{M}^{-1} \mod N, R < 2N$)

Base β Operation	Base $\widetilde{\beta}$ Operation
$\langle s \rangle_\beta \leftarrow \langle A \rangle_\beta . \langle B \rangle_\beta$	$\langle s \rangle_{\widetilde{\beta}} \leftarrow \langle A \rangle_{\widetilde{\beta}} . \langle B \rangle_{\widetilde{\beta}}$
–	$\langle q \rangle_{\widetilde{\beta}} \leftarrow \langle s \rangle_{\widetilde{\beta}} . \langle -N^{-1} \rangle_{\widetilde{\beta}}$
$\langle q \rangle_{\beta \cup \widetilde{\beta}} \Longleftarrow \langle q \rangle_{\widetilde{\beta}}$	
$\langle r' \rangle_\beta \leftarrow \langle q \rangle_\beta . \langle N \rangle_\beta$	–
$\langle r'' \rangle_\beta \leftarrow \langle s \rangle_\beta + \langle r' \rangle_\beta$	–
$\langle R \rangle_\beta \leftarrow \langle r'' \rangle_\beta . \langle \widetilde{M}^{-1} \rangle_\beta$	–
$\langle R \rangle_\beta \Longrightarrow \langle R \rangle_{\beta \cup \widetilde{\beta}}$	

Algorithm 1. Montgomery Multiplication algorithm in RNS

Many different methods (more or less efficient depending on the number of elements in the base) were proposed to perform the base extension step. The most common are: conversion using MRS [6], conversion using an extra modulus [36], conversion allowing an offset [5], approximate base conversion [31] and error-free approximate base conversion [22].

4 Hardware Implementations Strategies

In this section, we present the various strategies used to implement RSA on FPGA using RNS. In the rest of this paper we implicitly suppose that the Chinese Remainder Theorem (CRT for short) is used to compute an exponentiation [28].

4.1 Mersenne numbers

When $L = 2^\kappa - 1$ is prime, L is usually said to be a *Mersenne number* (or a *Mersenne prime*). Throughout the paper, for the sake of simplicity, we extend this definition to any number of the form $2^\kappa - 1$, be it prime or not. Its particular form allows modular reduction $A \mod L$ with at most two κ-bit additions, for any $A < L^2$. This shortcut provides a major speed-up in algorithms based on modular arithmetic.

RNS base elements uniquely composed of Mersenne numbers are in a large range since they must be relatively prime. This leads to inefficient hardware implementations since every operation should be thought for the longest element. For example, considering two suiting bases for 1024-bit representations: $\beta = \{2^{\kappa_1} - 1, \ldots, 2^{\kappa_{12}} - 1\}$ with $\kappa_i = 37, 47, 59, 67, 73, 83, 97, 103, 109, 119, 123, 125$ and similarly for $\widetilde{\beta}$ ($\widetilde{\kappa}_i$ from 43 to 127). To be timing analysis resistant, the implementation must process all data as 126-bit element.

Pairwise elements $(2^{\kappa_1} \pm 1)$ as suggested in [37] could lead to better balanced bases. However, it does not completely fix this issue and more seriously adds higher design complexity.

Among *Generalized Mersenne numbers* [38, 42], we chose to work with the following special form: $2^{\kappa_1} - 2^{\kappa_2} - 1$. This solution offers to choose bases in smaller range and keeps the efficiency of the modulo reduction. For $L = 2^{\kappa_1} - 2^{\kappa_2} - 1$ and $A < L^2$, the modular reduction $A \mod L$ takes at most 6 additions of κ_1-bits, if $0 < \kappa_2 < \frac{\kappa_1+1}{2}$. Moreover, if κ_2 is chosen bigger than 1, simple combinations of the terms can reduce the number of additions to 4. The careful reader sees that the remainder of the reduction might require a final subtraction of L to be completely reduced. This issue happens to be required only during the base conversions where the moduli are mixed.

4.2 Side-channels countermeasures

The most generally known side-channel analysis is *Timing Attack* presented by Kocher [23], see also [32] in the case of use of CRT. The countermeasures consist of a modification of the well-known Montgomery multiplication [16, 18, 40, 41], *i.e.* avoiding the final substraction such as obtaining timing independent processes.

Another side-channel analysis uses power consumption, suggested by Kocher *et al.* in [24]. Two families have to be considered. The first one uses a single trace of a power consumption and is called *Simple Power Analysis* (SPA for short). The second one is more sophisticated and needs statistical treatment of several power traces, see [2, 8, 13]. This is called *Differential Power Analysis* (DPA for short). To avoid SPA, a variant of the 'square-and-multiply always' algorithm has been used [14]. Randomization of the message and of the exponent are classically applied to defeat DPA. However, thanks to RNS, *independent* randomization for *each* base can be performed. Another efficient DPA countermeasure can be used (that can be combined with the most classical ones): instead of fixing two bases, we propose to use *random* bases for computations. This generic countermeasure can be efficiently applied because special numbers for RNS have been chosen. Indeed, since relatively primes of form $2^{\kappa_1} - 2^{\kappa_2} - 1$ are used, we are able to store several of these bases with (very) small memory requirements and randomly select the bases for each process. Indeed, the use of generalized Mersenne numbers (prime to each others), as already seen, has many nice properties. If these bases are such that $58 \leq \kappa_1 \leq 64$ and $0 < \kappa_2 < \frac{\kappa_1+1}{2}$, at least 69 generalized Mersenne numbers primes to each others can be generated with very small memory requirements: κ_1 is represented as its distance to 64, and κ_2 to 0. As explained below (Section 5), we limit the number of moduli to 63. Only 18 $(= 9{\cdot}2)$ moduli are needed since CRT is used. We wanted that it is very unlikely that two traces have some correlation. In this way, we avoided all permutations inside a base[1] because in the beginning of Algorithm 1 all data are processed in parallel. Then the number of ways that we can choose 2 sets of 9 moduli among 63 is: $\binom{63}{9} \cdot \binom{54}{9} \approx 2^{66}$ possible combinations. Then, each base is randomly chosen for each RNS exponentiation, in combination with the more classical randomization methods of message and exponent *for each base*. Just remark that our countermeasure is only (small) memory consuming[2]. It is also worth noticing that each cell's process can be very simply desynchronized by adding random delays.

Another threat must be taken into account. It is called *Fault Induction Attacks* (also sometimes 'Differential Fault Analysis', DFA for short) [11, 12]. We decided to combine two countermeasures. The first one was first proposed by Yen *et al.* in [43], see also [9, 44]. To our knowledge, this is the best way to prevent fault attack against DFA, since no "if" test is needed contrary to some other efficient methods [4, 34, 35]. The second one is directly related to the use of RNS. Single-error detection can be done using redundant modulus m_r such that: $\forall i \in \{1 \cdots k\}, m_i < m_r$. The error is detected checking if the converted value is outside the range $[0, M-1]$, see [39] for further details.

Finally, recently a new type of attack has been proposed, based on electromagnetic radiations of the chip on which the crypto-algorithm is processed [17, 29]. This attack is called *ElectroMagnetic Analysis* (EMA for short), with its counterpart using a statistical treatment *Differential ElectroMagnetic Analysis* (DEMA for short). It seems difficult to provide software countermeasures to counteract this type of attack. It is not really clear either how to proceed in hardware apart from the reduction of electromagnetic emanations. One of our aims in designing this new architecture is to make an EMA/DEMA harder to mount. The basic idea is to design an architecture with independent cells that randomly proceed. Indeed, one of the advantage of EMA in comparison to power analysis is that better *Signal-to-Noise Ratio* (SNR) is obtained, since the probe can be located above the interesting area with much discarding the emanation produced by the rest of the chip.

Let us now give briefly the skeleton of the global algorithm, with the classical countermeasures included. Let μ be the hashed and padded message to be signed. The secret exponent D (of size d) is given in its CRT form: $D_p = D \bmod p-1$, $D_q = D \bmod q-1$. The principle of our architecture is as follows:

1. Compute $\mu^{(p)} = \mu \bmod p$ and $\mu^{(q)} = \mu \bmod q$,
2. For each $\mu^{(i)}$, randomly choose two bases β: $\{m_1, \cdots, m_k\}$, and $\widetilde{\beta}$
3. Represent each $\mu^{(i)}$ in β as $\{\mu_1^{(i)}, \cdots, \mu_k^{(i)}\}$, where $\mu_j^{(i)} = \mu^{(i)} \bmod m_j$,
4. Add randomizations (to the exponent and to the message) to avoid DPA to each $\mu_j^{(i)}$,

[1] However permutations between the two bases are allowed

[2] This is the first DPA countermeasure for RSA for free in complexity.

5. Compute the exponentiation using RNS-Montgomery multiplication for each $\mu_j^{(i)}$ with $D_i^{(j)}$ (by Algorithm1); at each operation randomly choose a cell,
6. Use the CRT to recover $\left(\mu^{(i)}\right)^{D_i} \bmod i$,
7. Use the CRT and output $\mu^D \bmod N$.

More implementation details of our architecture are given in Section 6.

Summarizing our strategies implementations, we can simply say that all side-channel constrains have been taken into account at the beginning of the implementation design in such a way that attacks are defeated/counteracted or at least really more difficult to exploit.

5 Pre and on-the-fly computations of constants

5.1 Choice of a suited conversion algorithm

As quoted above, there are many different base extension algorithms. For the sake of simplicity, we decided to discard solutions using an approximate base conversion. We compare in Table 1 the efficiency of Algorithm 1 with different base conversion methods.

Operations	MRS	Shenoy *et al.*	Bajard *et al.*
Multiplications	$3 \cdot k^2 + 2 \cdot k$	$2 \cdot k^2 + 13 \cdot k + 5$	$2 \cdot k^2 + 10 \cdot k + 4$
Additions	$2 \cdot k^2 - k$	$2 \cdot k^2 + 5 \cdot k + 1$	$2 \cdot k^2 + 3 \cdot k + 1$
Subtractions	$k^2 - 1$	$2 \cdot k + 4$	$k + 1$
Total	$6 \cdot k^2 + k - 1$	$4 \cdot k^2 + 20 \cdot k + 10$	$4 \cdot k^2 + 14 \cdot k + 6$

Table 1. Comparison between the total numbers of operations for Algorithm 1 depending of the base extension method.

As explained in Subsection 6.2, if the basic operations are well organized, the bottleneck of the system is not the operations themselves but the data transmission between the memory and the different basic cells. In Table1, we took into account all basic operations in our comparison. The method of Shenoy *et al.* requires one extra modulo per base in order to evaluate α [36] while Bajard *et al.*'s method needs only one additional modulo in one of the two bases. From these considerations, we can deduce that Bajard *et al.* method is faster than MRS one if $k \geq 7$.

Moreover the main problem with the MRS extension [6] concerns the pre-computation of the constants. As we would have to store all $\langle m_i \rangle_{m_j}$ for all $i, j \in 1, \ldots, 63$ and $i \neq j$, it would lead to significant space of recent smart card memory ($64 * 62 * 63 \simeq 31.2$ kBytes). Hence we focused our attention on the algorithm proposed by Bajard *et al.* The main idea is to let an offset occur during the first base extension and to use the

Shenoy *et al.* conversion for the second one. Then, it adds the condition $(k+2)^2 N < M, \widetilde{M}$.

Basically, the conversion from β to $\widetilde{\beta}$ of Shenoy *et al.* works as follows[3]:

$$\alpha = \left\langle \left\langle M^{-1} \right\rangle_{m_r} \cdot \left\langle \left\langle \sum_{i=1}^{k} \left\langle a_i \cdot \langle M_i^{-1} \rangle_{m_i} \right\rangle_{m_i} \cdot M_i \right\rangle_{m_r} - a_r \right\rangle_{m_r} \right\rangle \quad (2)$$

$$\widetilde{a_j} = \left\langle \left\langle \sum_{i=1}^{k} \left\langle a_i \cdot \langle M_i^{-1} \rangle_{m_i} \right\rangle_{m_i} \cdot M_i \right\rangle_{\widetilde{m_j}} - \left\langle \alpha \cdot M \right\rangle_{\widetilde{m_j}} \right\rangle_{\widetilde{m_j}} \quad (3)$$

The difficulty remains in the computation of $\langle M_i^{-1} \rangle_{m_i}$ for $i = 1, \ldots, 63$. Again, if they are pre-computed and stored, it would fill up a significant part of the memory. A solution is simply to store $K_i = \langle S_i^{-1} \rangle_{m_i}$ where $S = \Pi_{i=1}^{63} m_i$ and $S_i = \frac{S}{m_i}$. Hence, the on-the-fly computation of all $\langle M_i^{-1} \rangle_{m_i}$ is simply obtained by multiplying (in the basic cells) each of the required K_i by all the moduli that were not gathered for the exponentiation. For instance, let us define the sets of the randomly chosen moduli by σ and $\widetilde{\sigma}$. The product of all elements of σ is M. Then $\langle M_i^{-1} \rangle_{m_i}$ is computed in the following way:

Input: K_i
Output: $\langle M_i^{-1} \rangle_{m_i}$

$R = S_i$
for j **from** 1 **to** 63 **do**
  if $m_j \notin \sigma$ **then**
    $R = \langle R \cdot m_j \rangle_{m_i}$
  end
end
return R

Algorithm 2. Algorithm for computing $\langle M_i^{-1} \rangle_{m_i}$.

The memory requirements are $62 \cdot 63 = 496$ Bytes to store all K_i. The computing overhead is negligible compared to the whole algorithm since it requires $18 \cdot 54$ multiplications in parallel.

5.2 Montgomery representation

When starting an exponentiation, the problem is that we should perform the Montgomery product of the initial message μ with $\widetilde{M^2} \bmod N$. But

[3] $\left\langle a_i \cdot \langle M_i^{-1} \rangle_{m_i} \right\rangle_{m_i}$ are computed only once.

$\widetilde{M}$ is the product of 9 randomly chosen moduli and it seems impossible to store all possible pre-computed value in a memory. Indeed, there are $\binom{63}{9} \approx 2^{34}$ possible sets which leads to a memory space of around 4 TBytes.

We propose here a solution inspired from an idea of Bajard *et al.*[7] As they suggested, we use few calls to our RNS Montgomery multiplication to solve this problem.

Again, we have $S = \Pi_{i=1}^{63} m_i$ and we store $\langle S \bmod N \rangle_{m_i}$ for all i. The two randomly sets of moduli are σ and $\widetilde{\sigma}$. The corresponding products are M and $\widetilde{M}$. Then our idea is to use our RNS Montgomery multiplication to divide $\langle S \bmod N \rangle_{m_i}$ by all not chosen moduli. At each time, we change the second base in order to divide the $\mu \cdot S \pmod N$ by all non selected moduli in $\widetilde{M}$. For that reason, the whole set of moduli must be a multiple of k, then $l = k.r$ where $l, k, r \in \mathbb{N}$ (in our case $r = 7$, $l = 63$ and $k = 9$). The M_i for $i = 3, \ldots, r$ are all formed sets of k moduli that were not chosen.

Input: μ and $S \bmod N$
Output: $\mu_M = \mu \cdot \widetilde{M} \bmod N$

Compute constants: $\langle \widetilde{M}_i^{-1} \rangle_{\widetilde{m}_i}$, $\langle M_i^{-1} \rangle_{m_i}$, $\langle M^{-1} \rangle_{\widetilde{m}_i}$
$\mu_M = MM(\mu, S \bmod N, N, \widetilde{M}, M)$
for j **from** 3 **to** r **do**
  Compute constants: $\langle M_{j,}^{-1} \rangle_{m_i}$, $\langle M_j^{-1} \rangle_{\widetilde{m}_i}$.
  $\mu_M = MM(\mu_M, 1, N, \widetilde{M}, M_j)$
end
return μ_M

Algorithm 3. Algorithm for computing $\mu_M = \mu \cdot \widetilde{M} \pmod N$.

So the following relation holds: $\mu_M = \mu \cdot S \cdot M^{-1} \cdot M_3^{-1} \cdots M_r^{-1} \bmod N = \mu \cdot \widetilde{M} \bmod N$. In terms of memory requirements, we need $\langle S \bmod N \rangle_{m_i}$ for $i = 1, \ldots, 63$, then $64 \cdot 63 = 496$ Bytes.

5.3 RNS SCA Protected Exponentiation Algorithm

We gather all the operations necessary to achieve the SCA protected exponentiation algorithm. Moreover, we give the time and memory cost for the latter.

All pre-computations and computations are processed in parallel. So the number of operation gives a clear idea of the total time for the whole algorithm, since the time depends on the number of operations and not

Input: $m_1, \ldots, m_{63}$ a set of relatively Generalized Mersenne prime, μ the message, N the modulo, E the exponent of size e and $\langle S_i^{-1} \rangle_{m_i}$, $\langle S \bmod N \rangle_{m_i}$, $\langle N \rangle_{m_i}$ for all $i = 1, \ldots, 63$ stored in the memory.

Output: $C = \mu^D \bmod N$

Pick randomly $2k$ moduli and form two bases M and $\widetilde{M}$.

Use Algorithm 3 to obtain μ_M.

Compute constants with Algorithm 2: $\langle M_i^{-1} \rangle_{m_i}$, $\langle \widetilde{M}_i^{-1} \rangle_{\widetilde{m}_i}$, $\langle \widetilde{M}^{-1} \rangle_{m_i}$.

$R_0 := 1$, $R_1 := \mu_M$, $i := 0$, $g := 1$

while $(i \leq d-1)$ **do**

  $g := g \oplus D_i$

  $R_g := MM(R_g, R_1, N, M, \widetilde{M})$

  $i := i + g$

end

$C = MM(R_0, 1, N, M, \widetilde{M})$

return C

Algorithm 4. RNS SCA Protected Exponentiation Algorithm.

on the computing time of each of them. First μ_M is obtained by 6 Montgomery multiplication and some on-the-fly computations of constants, the total of operations is: $9 \cdot 54 \cdot 3 + 6 \cdot 456 + 9 \cdot 54 \cdot 2 \cdot 5 = 9054$. The computation of the constants with Algorithm 2 are evaluated as $9 \cdot 54 \cdot 3 = 1458$ basic operations. The exponentiation algorithm takes on average $1.5 \cdot 512 \cdot 456 = 350208$ basic operations and finally the last conversion from the Montgomery representation needs 456 basic operations. Then the time overhead caused by the pre-computation due to our proposed countermeasure is evaluated as 3% of the global computing time while the memory requirements are evaluated as $3 \cdot 496$ Bytes, *i.e* less than 1.5 KBytes.

6 Implementation Results

6.1 Implementation

The main core of our design is a set of all alike, independent and parallel coprocessors. They can perform basic modular operations using any modulus of the bases. Each cell is connected to one common 16-bit wide communication bus on which they can interact with a defined protocol. They are managed by a *multiplier controller* containing the sequence of operations to perform one 512-bit multiplication of the 'square-and-multiply' algorithm. The combination of the control unit and the set of cells forms a large *multiplication processor*. Following the current key bit, the *exponentiation processor* feeds (A, A) or (A, X) into the multiplication processor. However, the latter processes the given data in the same way, no matter if the operation is a square or a multiply. Fig. 5 illustrates all parts connected together by the *data* bus.

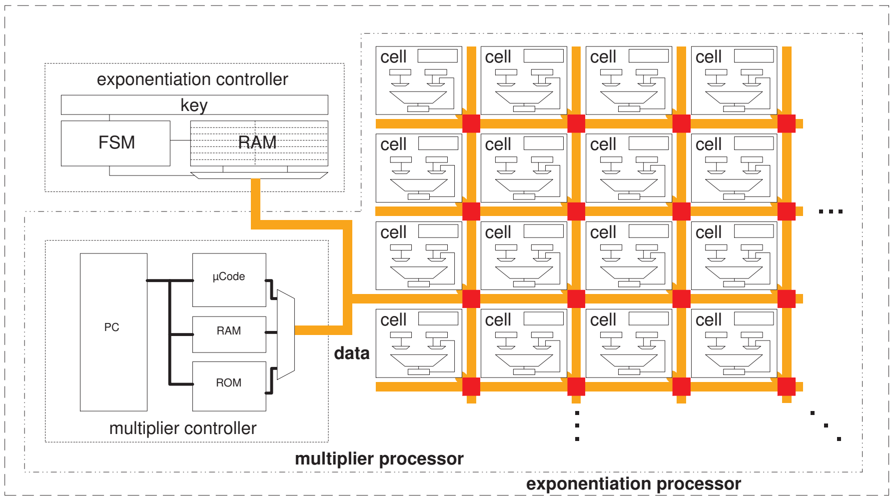

Fig. 5. Architectural structure of the 512-bit exponentiation processor

Cells have been implemented using the less hardware resources possible with minimal loss in frequency. They integrated the three basic operations: the addition and the multiplication are quite similar, while the subtraction requires extra hardware resources (multiplexors). The cell presents a fair tradeoff between complexity and efficiency.

Our first try was based on *Scaling Accumulator* technic. So the principle is to compute a complete multiplication, to store the results in registers, and finally to process the modular reduction by carrying out the different numbers to be added (see [42]). As explained only 4 additions are required. Based on this structure, modular addition and subtraction can also be easily treated.

Another direction is the following. Instead of separating the operation and the reduction, it seems interesting to interleave both. At each step, the result (say S_1) from addition of previous outcome and new partial product is truncated after κ_1 bit (S'_1). The overflowing bits (MSBs of S_1) are then used to produce the remainder of the number (say R_1). Finally S'_1 is added to R_1. If $\kappa_2 \geq 4$ is verified, the remainder generation is simplified. Using some shifters, it has been easily adapted to a pipelined version. Implementation details of this cell are given in Figure 6.

As explained in § 4.1, the remainder might not be fully reduced and it could lead to a problem, especially in bases conversion. An extra final substraction is performed to tackle this issue. If an underflow occurs, the previous value is kept and the other one is discarded. This extra step protects against conversion error in a simple and timing-resistant way.

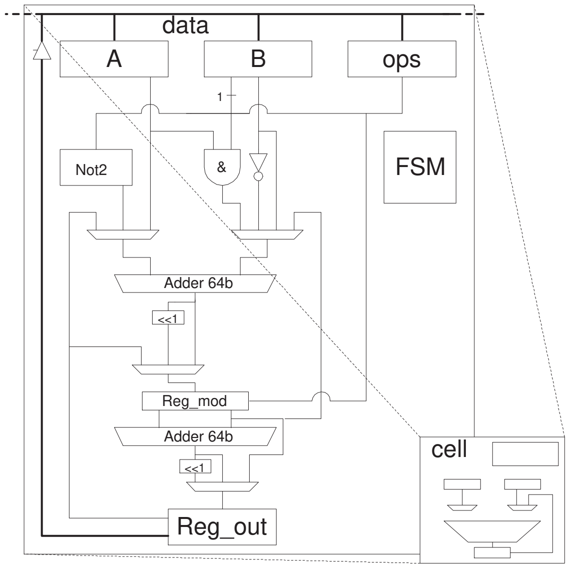

Fig. 6. Detail of a 2-stage pipeline cell.

The purpose of the multiplication processor is to orchestrate the set of cells in order to perform one 512-bit multiplication. The operands expressed in the 9 moduli of both bases are sent through *data* by the exponentiation processor. Once the data are set, the multiplication processor shares out the computations to idle cells. When the result of a cell is available, the corresponding cell uploads its result. And so on till the whole multiplication is completed. The global results in both bases are sent back to the exponentiation processor. There is clearly no difference in the operation sequence whether the general computation is a square or a multiply.

It is basically built as a microcontroller: the 10-bit program counter refers to the current operation and to the address of the operands in the memory; the microcode is fetched and then interpreted by the processor which enables the memory to send or receive data in burst mode.

The 512 bits of the secret key are stored in a register inside the exponentiation processor. The latter is scanned from left to right to perform a variant of the 'square-and-multiply-always' algorithm.

6.2 Performances

The average number of clock cycles per operation (addition, subtraction, multiplication or write back) is about 11. Hence, if the operations are well scheduled in the program, a basic operation in the base β or $\tilde{\beta}$ requires virtually 22 cycles on the data bus (a multiplication requires a bit more than 64 cycles).

Our design has been synthesized on a Virtex2 xc2v6000 clocked at 50 Mhz. The synthesis is performed with FPGA Compiler 2 3.7.1 (SYNOPSYS) and the implementation with XILINX ISE-5. Moreover, 9 RAM blocks, 4956 slices of which 2788 registers and 8185 LUTs, are needed for a 512 bits exponentiation. The signature is achieved in 158 ms. Since the goal is to evaluate our methodology on general reconfigurable hardware devices, we did not take the most particular features of Virtex2 (such as internal 18 by 18 bits multipliers). Performing a 1024 bits exponentiation in the same time would only require doubling hardware resources, since CRT is used. The design contains the hardware needed to receive the message μ in RNS and perform the exponentiation. Compare our design with a Cox-Rower architecture seems to be very difficult. Indeed, many choices differ from their implementation, especially all the side-channels considerations. But, we think that implementing their architecture in an FPGA is certainly more hardware consuming than ours regarding the number of adders and multipliers they need for one single cell.

7 Conclusion

A new architecture to avoid side-channels analysis (based on timing, power consumptions and electromagnetic radiations that leak when a crypto-algorithm is processed) but also fault induction resistance has been proposed. Our design is based on Residue Number Systems that permits fast additions and multiplications, carry-free, high speed arithmetic, some faults detection, possible error correction and mostly parallel implementations. Generalized Mersenne numbers provide well balanced bases and offer efficient modular reductions. They are also useful to efficiently implement our new and general DPA countermeasure based on RNS by reducing the memory requirements. Moreover, we proposed different hardware strategies to manage a multiplication within the cells.

Finally, we gave a concrete implementation of our architecture on FPGA. Although our aim is to design a side-channel secure implementation, our FPGA implementation results in promising efficiency: less than 150 ms for a 1024-bit RSA, with competitive hardware resources. Our implementation cannot be compared to those proposed by Nozaki *et al.* as the objectives are really different, different circuit types are used, and less hardware requirements are needed in our case.

References

1. IEEE Std 1363-2000. *IEEE Standard Specifications for Public-Key Cryptography.* IEEE Computer Society, August 29, 2000.

2. Mehdi-Laurent Akkar, Régis Bevan, Paul Dischamp, Didier Moyart. Power analysis, What is now possible.... In T. Okamoto, Ed., *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pp. 489–502. Springer-Verlag, 2000.
3. Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao and Pankaj Rohatgi The EM side-channel(s). In B.S. Kaliski Jr. and Ç.K. Koç, Ed., *Cryptographic Hardware and Embedded Systems (CHES 2002)*, volume 2523 of *Lecture Notes in Computer Science*, pp. 29–45. Springer, 2002.
4. Christian Aumüller, Peter Bier, Wieland Fischer, Peter Hofreiter and Jean-Pierre Seifert. Fault attacks on RSA with CRT: Concrete results and practical countermeasures In B.S. Kaliski Jr. and Ç.K. Koç, Ed., *Cryptographic Hardware and Embedded Systems (CHES 2002)*, volume 2523 of *Lecture Notes in Computer Science*, pp. 260–275. Springer, 2002.
5. Jean-Claude Bajard, Laurent-Stéphane Didier and Peter Kornerup, Modular Multiplication and Base Extensions in Residue Number Systems, *IEEE Symposium on Computer Arithmetic*(2001).
6. Jean-Claude Bajard, Laurent-Stéphane Didier and Peter Kornerup, *An RNS Montgomery modular multiplication algorithm*, IEEE Transactions on Computers, vol. 47, pp. 766–76, July 1998.
7. Jean-Claude Bajard, Laurent Imbert, Pierre-Yvan Liardet and Yannick Teglia. Leak Resistant Arithmetic.
Research report: http://papyrus.lirmm.fr/GEIDEFile.PDF?Archive= 191140-291932&File=RR03021_PDF. 2004.
8. Régis Bevan and Erik Knudsen. Ways to enhance differential power analysis. In P.J. Lee and C.H. Lim, Ed., *Information Security and Cryptology (ICISC 2002)*, volume 2587 of *Lecture Notes in Computer Science*, pp. 327–342. Springer-Verlag, 2002.
9. Johannes Blömer, Martin Otto and Jean-Pierre Seifert. A new CRT-RSA algorithm secure against bellcore attacks. newblock *ACM conference on Computer and communication security (ACM-CCS)*, pp. 311–320. *ACM Press*, 2003.
10. Dan R. Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults. In W. Fumy, Ed., *Advances in Cryptology - EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pp. 37–51. Springer, 1997.
11. Dan R. Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of eliminating errors in cryptographic computations. *Journal of Cryptology*, 14(2):101–119, 2001. An earlier version appears in EUROCRYPT '97 [10].
12. Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In B.S. Kaliski Jr., Ed., *Advances in Cryptology - CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pp. 513–525. Springer, 1997.
13. Eric Brier, Christophe Clavier, and Francis Olivier. Optimal statistical power analysis. Cryptology ePrint Archive, Report 2003/152, 2003. http://eprint.iacr.org/2003/152.
14. Benoît Chevallier-Mames, Mathieu Ciet, and Marc Joye. Low-Cost Solutions for Preventing Simple Side-Channel Analysis: Side-Channel Atomicity. In IEEE Transactions on Computers Vol. 53 No. 6, pp. 760-768, June 2004.
15. Jaewook Chung and Anwar Hasan. More generalized mersenne numbers. *Selected Areas in Cryptography (SAC 2003)*, August 14 & 15, 2003, Carleton University, Ottawa, Ontario, Canada.
16. Jean-François Dhem. *Design of an efficient public-key cryptographic library for RISC-based smart cards.* PhD thesis, Université catholique de Louvain, May 1998.
17. Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In Ç.K. Koç, D. Naccache, and C. Paar, Ed., *Cryptographic Hardware and Embedded Systems (CHES 2001)*, volume 2162 of *Lecture Notes in Computer Science*, pp. 251–261. Springer, 2001.
18. Gael Hachez and Jean-Jacques Quisquater. Montgomery exponentiation with no final subtractions: Improved results. In Ç.K. Koç and C. Paar, Ed., *Cryptographic*

Hardware and Embedded Systems (CHES 2000), volume 1965 of *Lecture Notes in Computer Science*, pp. 293–301, 2000.
19. Markus A. Hitz and Erich Kaltofen. Integer division in residue number systems. *IEEE Transaction on Computers* 44(8), pp. 983-989,(1995).
20. Ching Yu Hung and Behrooz Parhami. Fast RNS division algorithms for fixed divisors with application to RSA encryption *Information Processing Letters*, Vol.51, pp. 163-169, 1994.
21. Laurent Imbert, Jean-Claude Bajard. A full RNS implementation of RSA Research Report 02068, LIRMM, available at:
22. Shinichi Kawamura, Masanobu Koike, Fumihiko Sano, and Atsushi Shimbo, Cox-Rower Architecture for Fast Parallel Montgomery Multiplication, In B. Preneel Ed. *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, Springer-Verlag, 2000, pp. 523-538.
23. Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, Ed., *Advances in Cryptology - CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pp. 104–113. Springer, 1996.
24. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In M. Wiener, Ed., *Advances in Cryptology - CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pp. 388–397. Springer, 1999.
25. Olivier Kömmerling and Markus G. Kuhn, Design principles for tamper-resistant smartcard processors, *USENIX Workshop on Smartcard Technology (Smarcard'99)*, pp. 9–20. USENIX Association, 1999
26. Peter L. Montgomery. Modular multiplication without trial division. *Math. Comp.*, 44(170):519–521, April 1985.
27. Michael Neve and Eric Peeters. Challenging a New Attack-Resistant Distributed Architecture for Asymmetric Cryptographic Algorithms on Smart Cards. Final year dissertation, Microelectronics Laboratory DICE, Université catholique de Louvain, June 2002.
28. Jean-Jacques Quisquater and Chantal Couvreur. Fast Decipherment Algorithm for RSA Public-Key Cryptosystem. *Electronics Letters*, 18(21):905–907, 1982.
29. Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In I. Attali and T.P. Jensen, Ed., *Smart Card Programming and Security (E-smart 2001)*, volume 2140 of *Lecture Notes in Computer Science*, pp. 200–210. Springer, 2001.
30. Hanae Nozaki, Masahiko Motoyama, Atsushi Shimbo, and Shinichi Kawamura, Implementation of RSA algorithm based on RNS Montgomery multiplication, In C. Paar ed. *Cryptographic Hardware and Embedded Systems - CHES 2001*, pp. 364376, Springer-Verlag, Berlin, Germany.
31. Karl C. Posh and Reinhard Posh. Modulo reduction in Residue Number Systems. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6, No. 5, May 1995, pp. 449-454.
32. Werner Schindler. A timing attack against RSA with the chinese remainder theorem. In Ç.K. Koç and C. Paar, Ed., *Cryptographic Hardware and Embedded Systems (CHES 2000)*, volume 1965 of *Lecture Notes in Computer Science*, pages 109–124. Springer, 2000.
33. J. Schwemmlein, Karl C. Posch, Reinhard Posch. RNS-modulo reduction upon a restricted base value set and its applicability to RSA cryptography. *Computer & Security*, Vol.17, No.7, pp. 637-650, 1998
34. A. Shamir. How to check modular exponentiation. Presented at Eurocrypt'97 rump session, 1997.
35. A. Shamir. Method and apparatus for protecting public key schemes from timing and fault attacks. United States Patent 5991415, 1999.
36. A.P. Shenoy and R. Kumaresan. Fast base extension using a redundant modulus in RNS. *IEEE Transactions on Computers*, 38 (1989), pp. 292-297.
37. Alex Skavantzos and Mohammad Abdallah. Implementation issues of the two-level residue number system with pairs of conjugate moduli IEEE Transactions on Signal Processing, vol. 47, no. 3, pp. 826–838, 1999.

38. Jérôme A. Solinas Generalized Mersenne numbers. Technical report, The centre for applied cryptographic research, University of Waterloo, 1999. CORR 99-39.
39. Nicholas S. Szabó an Richard I. Tanaka. Residue arithemtic and its application to computer technology. McGraw-Hill,1967
40. Colin D. Walter. An overview of Montgomery multiplication technique: How to make it smaller and faster. In Ç.K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES '99)*, volume 1717 of *Lecture Note in Computer Science*, pp. 80–93 1999.
41. Colin D. Walter. Montgomery exponentiation needs no final subtractions. *Electronics Letters*, 35(21):1831–1832, October 1999.
42. Huapeng Wu. On modular reduction Technical report, CACR, University of Waterloo, 2000. CORR 2000-36.
43. Sung-Ming Yen, Seungjoo Kim, Seongan Lim, and Seongan Moon. RSA Speedup with Residue Number System Immune against Hardware Fault Cryptanalysis. In K. Kim, Ed., *Information Security and Cryptology - ICISC 2001*, volume 2288 of *Lecture Notes in Computer Science*, pp. 397–413. Springer-Verlag, 2001.
44. Sung-Ming Yen, Seungjoo Kim, Seongan Lim and Sang-Jae Moon. RSA Speedup with Chinese Remainder Theorem Immune against Hardware Fault Cryptanalys *IEEE Transactions on Computers* 52(4): pp. 461-472, 2003.