A coprocessor for secure and high speed modular arithmetic

Nicolas Guillermin[1,2]

[1] DGA Information Superiority, Bruz, France
[2] IRMAR, Université Rennes 1, France

Abstract. We present a coprocessor design for fast arithmetic over large numbers of cryptographic sizes. Our design provides a efficient way to prevent side channel analysis as well as fault analysis targeting modular arithmetic with large prime or composite numbers. These two countermeasure are then suitable both for Elliptic Curve Cryptography over prime fields or RSA using CRT or not. To do so, we use the residue number system (RNS) in an efficient manner to protect from leakage and fault, while keeping its ability to fast execute modular arithmetic with large numbers. We illustrate our countermeasure with a fully protected RSA-CRT implementation using our architecture, and show that it is possible to execute a secure 1024 bit RSA-CRT in less than 0.7 ms on a FPGA.

Keywords: countermeasure, fault analysis, side channel analysis, high speed, RNS, FPGA

1 Introduction

The same time cryptographic applications have flooded our all day life, security of cryptographic implementations has become a concern for the research community. First proposed by Kocher et al. [16], passive side channel analysis attend to discover the embedded secrets from activity of the component. This threat involves both symmetric and asymmetric implementations. To protect against side channel exploitation, developers use countermeasures such as desynchronisation [1], masking [7] or multiple rail logic [21].

In parallel fault analysis has become a major threat for cryptographic devices. First introduced in [6], fault analysis showed its efficiency against a huge range of cryptographic services : symmetric [4] and asymmetric [6].

On another topic, the Residue Number System (RNS), introduced in [15], has demonstrated its ability to execute efficient modular arithmetic with large prime and composite numbers, first theoretically [2], and then in practice, on RSA cryptosystem [14] as well as on elliptic curves [20, 13].

Bajard et al. [3] proposed a countermeasure called Leak-Resistant Arithmetic (LRA), lying on the RNS. The main idea of this paper is to randomize the way of representing numbers, i.e the RNS base. This countermeasure has been proven to add few overhead compared to the fixed base. Nevertheless, it has never been to our knowledge any complete hardware implementation of it. The only attempt was done for a full RSA-CRT on a FPGA in [9], we explain in subsection 5.2 why this attempt is not satisfying, neither for side channel analysis, nor for fault analysis.

Our contribution : We present the first full hardware implementation of the Bajard et al. [3] LRA countermeasure. We show that considering some adaptations, it is perfectly suitable with the

Cox Rower architecture first introduced in [14]. Moreover, we introduce a new technique to detect faults during a modular computation, called the RNS Fault Detection (RFD). The RFD detects fault during the computation, by using the redundancy of RNS representation, and by checking it at each step during the whole computation. The two countermeasures can be used together, as well as with other known countermeasures. Eventually we prove the low cost of our countermeasure by exhibiting a full 1024 bit RSA-CRT implementation in around 0.6 ms on a Altera Stratix III FPGA, with less than 15% area overhead and around 6% time overhead for our two countermeasures. Target application of such special purpose designs are all the fields where high speed, low latency and high level resistance against attacks are required (example : IPSEC set-top box, PKCS#11 tokens,...).

Structure of the paper : section 2 reminds all the previous work and introduces prerequisites and notation for the rest of the paper. It deals with mathematical backgrounds of RNS, and presents the Cox Rower architecture, first introduced in [14]. If the reader is familiar with these work, we suggest him to go directly to section 3. Section 3 describes the adaptation of the Bajard [3] LRA countermeasure, and shows its compliance with the Cox Rower architecture. We prove in this part that the LRA countermeasure can be adapted for high speed cryptoprocessor with very few overhead. Section 4 introduces the RFD countermeasure, an improvement of the Cox Rower giving an fault detection ability. Eventually section 5 applies the two countermeasures on a Cox Rower, and shows implementation overhead brought by these two countermeasures in the case of a 1024 bits RSA-CRT implementation. In this section we also give a brief overview of the state-of-art countermeasure to protect RSA-CRT.

2 The Residue Number System

Notations : In all the paper, for $(a, b) \in \mathbb{N}^2$, we denote by $|a|_b$ the result of a modulo b.
We also define the following function :

$$\begin{array}{l|l} \texttt{dec}(X) & \mathbb{R} \longrightarrow [0;1[\\ & X \longrightarrow X - \lfloor X \rfloor \end{array}$$

2.1 RNS

Let $\mathcal{B} = \{m_1, \cdots, m_n\}$ be a set of coprime natural integers, and $M(\mathcal{B}) = \prod_{i=1}^{n} m_i$. The residue number system (RNS) representation $< X >_{\mathcal{B}}$ of $X \in \mathbb{N}$ is the set of positive integers $\{x_1 = |X|_{m_1}, \cdots, x_n = |X|_{m_n}\}$. This representation allows fast arithmetic in $\mathbb{Z}/M(\mathcal{B})\mathbb{Z}$:

$$\begin{array}{l} < X + Y >_{\mathcal{B}} = \{|x_1 + y_1|_{m_1}, \cdots, |x_n + y_n|_{m_n}\} \\ < X - Y >_{\mathcal{B}} = \{|x_1 - y_1|_{m_1}, \cdots, |x_n - y_n|_{m_n}\} \\ < X \times Y >_{\mathcal{B}} = \{|x_1 \times y_1|_{m_1}, \cdots, |x_n \times y_n|_{m_n}\} \\ < X \ / \ Y >_{\mathcal{B}} = \{|x_1 \ / \ y_1|_{m_1}, \cdots, |x_n \ / \ y_n|_{m_n}\} \end{array}$$

The last line is only available for Y coprime with $M(\mathcal{B})$. The integer $|X|_M(\mathcal{B})$ is recovered thanks to the Chinese remainder theorem :

$$X = \left| \sum_{i=1}^{n} |x_i \times \xi_i|_{m_i} \times \frac{M(\mathcal{B})}{m_i} \right|_M (\mathcal{B}) \text{ with } \xi_i = |\frac{M(\mathcal{B})}{m_i}^{-1}|_{m_i}. \quad (1)$$

For the rest of the paper we call $\mathcal{B}$ a RNS base, and $M(\mathcal{B})$ its module.

Algorithm 1 $\texttt{Red}(X, p, \mathcal{B}_1, \mathcal{B}_2)$

Require: B and $\mathcal{B}_2$ RNS bases with $M(\mathcal{B}_1) > \alpha p$ and $M(\mathcal{B}_2) > 2p$
Require: p coprime with $M(\mathcal{B}_1)$ and $M(\mathcal{B}_2)$
Require: $< X >_{\mathcal{B}_1}$ and $< X >_{\mathcal{B}_2}$ with $X < \alpha p^2$
Require: precomputations : $< |-p^{-1}|_{M(\mathcal{B}_1)} >_B$,$< |M(\mathcal{B}_1)^{-1}|_{M(\mathcal{B}_2)} >_{\mathcal{B}_2}$ and $< p >_{\mathcal{B}_2}$
Require: algorithm $\texttt{Switch}(A, \beta, \beta')$
Ensure: $< S >_B$ and $< S >_{\mathcal{B}_2}$ such that $|S|_p = |XM(\mathcal{B}_1)^{-1}|_p$ and $S < 2p$
1: $< Q >_B \leftarrow < X \times |-p^{-1}|_{M(\mathcal{B}_1)} >_{\mathcal{B}_1}$
2: $\quad \texttt{Switch}(Q, \mathcal{B}_1, \mathcal{B}_2)$
3: $< R >_{\mathcal{B}_2} \leftarrow < X + p \times Q' >_{\mathcal{B}_2}$
4: $< S >_{\mathcal{B}_2} \leftarrow < R \times M(\mathcal{B}_1)^{-1} >_{\mathcal{B}_2}$
5: $\quad \texttt{Switch}(S, \mathcal{B}_2, \mathcal{B}_1)$
6: **return** $< S >_{\mathcal{B}_1}$ and $< S >_{\mathcal{B}_2}$

2.2 RNS Montgomery reduction algorithm and base extension

An adaptation of the classical Montgomery reduction algorithm to the context of RNS has been proposed by Bajard et al. in [2]. Algorithm 1 is given for remaindering. Let $p \in \mathbb{N}$. This algorithm needs the definition of two RNS bases $\mathcal{B}_1$ and $\mathcal{B}_2$ with $M(\mathcal{B}_1)$ and $M(\mathcal{B}_2)$ coprime with p and with each other (thus the set of coprimes of $\mathcal{B}_1$ and $\mathcal{B}_2$ defines a RNS base $\mathcal{B}$ with $M(\mathcal{B}) = M(\mathcal{B}_1)M(\mathcal{B}_2)$). For the sake of simplicity, we consider these two bases to be of the same size n, even if this algorithm is completely general. $\mathcal{B}$ is then a RNS base of size $2n$. If $M(\mathcal{B}_1) > \alpha p$ and $M(\mathcal{B}_2) > 2p$, for any $X < \alpha p^2$, algorithm 1 computes the values $\{< S >_{\mathcal{B}_1}, < S >_{\mathcal{B}_2}\} = \texttt{Red}(X, p, \mathcal{B}_1, \mathcal{B}_2)$, with $S = |XM(\mathcal{B}_1)^{-1}|_p + \delta p$ $(\delta = 0 \text{ or } 1)$.

This algorithm needs the input of another function : considering two RNS bases β and β', of size n with $M(\beta)$ and $M(\beta')$ coprime, we define $\texttt{Switch}(X, \beta, \beta')$ by the following :

$$\texttt{Switch}(X, \beta, \beta') \left| \begin{array}{l} [0; M(\beta)[\longrightarrow [0; M(\beta')[\\ < X >_\beta \longrightarrow < |X|_{M(\beta)} >_{\beta'} \end{array} \right.$$

In order to do this transformation, all the former fast implementations using RNS for cryptography[17, 20, 13] used the Kawamura algorithm proposed in [14]. This algorithm uses CRT reconstruction formula (equation 1). The main concern is the computation of the reduction modulo $M(\mathcal{B}_1)$ in equation 1. Kawamura [14] proposes an efficient technique using fixed point division presented on the algorithm 2. In the following we present the work introduced in [14] with the point of view we need for our work, since we will focus on variation of the algorithm 2 and its subroutines.

This algorithm needs a lot of precomputations : if $\beta = \{m_1, \cdots, m_n\}$ and $\beta' = \{m'_1, \cdots, m'_n\}$, one will need of $|(M(\beta)/mi)^{-1}|_{mi}$, $|(M(\beta)/mi)|_{m'_j}$, and $|-M(\beta)|_{m'_j}$ for $(i, j) \in [1, n]^2$. Kawamura's algorithm indeed computes in β' the following value :

$$X = \sum_{i=1}^{n} (\xi_i \times \frac{M(\beta)}{m_i}) - M(\beta) \lfloor \sum_{i=1}^{n} \frac{\xi_i}{m_i} \rfloor \text{ with } \xi_i = |x_i \frac{M(\beta)}{m_i}^{-1}|_{m_i} \tag{2}$$

The evaluation of the quotient $\lfloor \sum_{i=1}^{n} \frac{\xi_i}{m_i} \rfloor$ is done by the line 9. A function $eval(a, b)$ which evaluates the a/b quotient in $\mathbb{R}$ is needed. This function $eval(a, b)$ must give an under-approximation of a/b : $eval(a, b) = a/b - error$. The value $reg\_accumulator$ is initialized at a value $errinit$ (line 4 of algorithm 2), which plays a significant role :

Algorithm 2 $\texttt{Switch}_{\beta,\beta'}(X)$

1: **for** i from 0 to n **do**
2: $\quad \xi_i = |x_i m_i / M(\beta)|_{m_i}$
3: **end for**
4: $reg\_accumulator = errinit$
5: **for** j from 0 to n' **do**
6: $\quad r_j = 0.$
7: **end for**
8: **for** i from 0 to n **do**
9: $\quad reg\_accumulator = reg\_accumulator + eval(\xi_i / m_i)$
10: $\quad$ **for** j from 0 to n' **do**
11: $\quad\quad r_j = r_j + |x_j + \xi_i M(\beta)/m_i|_{\mu_j}$
12: $\quad\quad$ **if** $reg\_accumulator >= 1$ **then**
13: $\quad\quad\quad r_j = |\tilde{r_j} - M|_{\mu_j}$
14: $\quad\quad$ **end if**
15: $\quad\quad reg\_accumulator = \texttt{dec}(reg\_accumulator)$
16: $\quad$ **end for**
17: **end for**

- If $errinit = 0$, then the output of the Kawamura algorithm is $< X >_{\beta'}$ or $< X + \beta >_{\beta'}$ when the branch 13 is executed one time too few. This is the case for the call of $\texttt{Switch}(Q, \mathcal{B}_1, \mathcal{B}_2)$ at line 2 of algorithm 1. We see easily that as $\texttt{Switch}(Q, \mathcal{B}_1, \mathcal{B}_2) =< Q >_{\mathcal{B}_2}$ or $< Q + M(\mathcal{B}_1) >_{\mathcal{B}_2}$ and if $M(\mathcal{B}_2) > 3p$, we have the following : $S = |XM(\mathcal{B}_1)^{-1}|_p + \delta p$ with $\delta \in \{0, 1, 2\}$.
- On the $\texttt{Switch}(S, \mathcal{B}_2, \mathcal{B}_1)$ at line 5, let us introduce the value $errmax$

$$errmax = max(\sum_{i=0}^{n}(\frac{\xi_i}{\tilde{m}_i} - eval(\xi_i, \tilde{m_i})) \tag{3}$$

  If the inequation $errmax \leq errinit < 1 - 3p/M(\mathcal{B}_2)$ holds, then the output of $\texttt{Switch}(S, \mathcal{B}_2, \mathcal{B}_1)$ is $< S >_{\mathcal{B}_1}$, and the result of algorithm 1 is then the $< S >_{\mathcal{B}_1}, < S >_{\mathcal{B}_2}$ we were waiting for $(S < 3p)$.

All previous hardware implementation used the $eval(a, b)$ function of [14] : b was estimated by 2^r, and a was reduced to the 8 most significant bits. This coarse approximation is enough for typical cryptographic sizes.

2.3 Hardware architecture

It has been shown in [17, 13], that this algorithm is easily implementable in high speed hardware implementations for both Elliptic curve (EC) and RSA cryptosystems. The Kawamura's architecture used in [17] for RSA and in [13] for EC is called the Cox Rower. Since we use it in this paper, we present more specifically the architecture. An overview of the architecture is presented on figure 5 in appendix.

The Rowers are synchronous hardware modules parametrized by the word length r and a pseudo-mersenne size r_ε. For any value $(x, y, t) \in [0; 2^r]^3$ any value $m \in [2^r - 2^{r_\varepsilon}; 2^r]$, any $\tau \in \{0, 1\}$, the Rower computes $|x \times y + \tau t|_m$ and adds it or not with the previous result in the accumulator register.

Considering that $r_\varepsilon < r/2$, the reduction phase is a classical reduction by a pseudo mersenne, and algorithm given in [13] works fine.

The Cox module is used to compute the lines 9 and 13 of algorithm 2. Together with the Rowers, it is sufficient to compute the whole algorithm 1, and therefore RSA (CRT or not) [17] or EC [13]. Note that the exponent is treated in the sequencer and does not appear on figure 5. In all previous hardware, the Cox is a simple hardware 8 bit accumulator.

The whole architecture is parametrized by ν the number of Rowers, r and r_ε the word size of the Rower. We only consider RNS bases $\mathcal{B}_1$ and $\mathcal{B}_2$ composed of n coprimes pseudo-Mersenne numbers of size r. To get the best results, we assume that ν divides n, and $\rho = n/\nu$. The modular value p added with the needed α in algorithm 1 must respect $\lceil log_2(\alpha p) \rceil < r \times n$, but is as close as possible to the limit (for the sake of efficiency). Thanks to the Cox Rower architecture, a full length multiplication of two values less than $3p$ is executed in only 2ρ cycles. If we consider the following bases $\mathcal{B}_1$ and $\mathcal{B}_2$ of size n, we can also execute the full algorithm 1 in $\rho(2n+3)$ cycles by mixing the algorithm 2 in the algorithm 1. On figure 1 is the exhaustive list of the precomputations.

Fig. 1. Needed precomputations for algorithm 1 and 2 in Cox Rower

Precomputations		used in
$\lvert(-pM(\mathcal{B}_1)/m_i)^{-1}\rvert_{m_i}$	for $i \in [1,n]$	combined line 1 of `Red` and line 2 of `Switch`$(Q, \mathcal{B}_1, \mathcal{B}_2)$
$\lvert(pm_{n+j}/(m_i M(\mathcal{B}_2))\rvert_{m_{n+j}}$	for $(i,j) \in [1,n]^2$	main loop of `Switch`$(Q, \mathcal{B}_1, \mathcal{B}_2)$
$\lvert -pm_{n+j}/M(\mathcal{B}_2)\rvert_{m_{n+j}}$	for $j \in [1,n]$	used in line 13 of `Switch`$(S, \mathcal{B}_1, \mathcal{B}_2)$
$\lvert(m_{n+j}/(M(\mathcal{B}_1)M(\mathcal{B}_2))\rvert_{m_{n+j}}$	for $j \in [1,n]$	combined line 3 and 4 of `Red` and line 2 of `Switch`$(S, \mathcal{B}_2, \mathcal{B}_1)$
$\lvert M(\mathcal{B}_2)/m_{n+j}\rvert_{m_{n+j}}$	for $j \in [1,n]$	invert line 2 of `Switch`$(S, \mathcal{B}_2, \mathcal{B}_1)$ [3]
$\lvert M(\mathcal{B}_2)/m_{n+j}\rvert_{m_i}$	for $(i,j) \in [1,n]^2$	main loop of `Switch`$(S, \mathcal{B}_2, \mathcal{B}_1)$
$\lvert -M(\mathcal{B}_2)\rvert_{m_i}$	for $i \in [1,n]$	used in line 13 of `Switch`$_{\mathcal{B}_2,\mathcal{B}_1}$

This architecture has proven to reach impressive efficiency while computing RSA and EC. For example in [17], a 1024 bit RSA-CRT is computed in 2.4 ms at 80 MHz using $\rho = 3$ and $r = 32$ on a 130 nm ASIC technology. In [13], Guillermin proposed different variations of n, with $\rho = 1$ for EC. He proposed $r = 36$, well adapted to the high-end FPGA Xilinx or Altera with their 18×18 multipliers. Using $r \simeq 36$ a 160 bit EC is computed in 0.37 ms on an Altera Stratix II, and a 512 in only 2.23 ms.

Nevertheless, this architecture suffers drawback : It is designed to efficiently manipulate only one length of numbers. For example, the 512 bit architecture is efficient only for manipulating 512 bit numbers (and 1024 before reduction). Small numbers can be used, but with much loss of speed compared to their size. This makes them therefore not really suitable for countermeasure like Shamir's [19] and the following [5, 10], which manipulate small numbers at the end of the computation.

In the rest of the paper, we define Cox Rower(n, r, ρ) a Cox Rower design with $\lceil n/\rho \rceil$ rowers of word length r (then able to compute modular arithmetic up to $nr - log_2(\alpha)$ bits). Our purpose is

[3] used to spare pipeline idle states before the main loop of `Switch`$_{\mathcal{B}_2,\mathcal{B}_1}$

to adapt this architecture to provide it with the Bajard countermeasure [3], and a fault detection ability, without losing the advantages of this architecture, neither in speed, nor in size.

3 Leak Resistant Arithmetic using Cox Rower Architecture

In this section we prove that the Leak Resistant Arithmetic (LRA) countermeasure of Bajard [3] is also achievable on a Cox Rower considering some adaptations. Subsection 3.1 reminds the principle of this countermeasure and subsection 3.2 presents the necessary adaptations to compute a LRA variant in a Cox Rower.

3.1 The LRA countermeasure

The countermeasure consists in using a random partition γ of an RNS base $\beta = \{m_1, \cdots m_{2n}\}$ into 2 coprime RNS bases $\mathcal{B}_{1,\gamma} = \{m_{\gamma(1)}, \cdots, m_{\gamma(n)}\}$ and $\mathcal{B}_{2,\gamma} = \{m_{\gamma(n+1)}, \cdots, m_{\gamma(2n)}\}$ respecting the conditions described in subsection 2.2 for any γ. By doing this, we transform any modular value $|X|_p$ of the cryptosystem by its Montgomery representative $|XM(\mathcal{B}_{1,\gamma})|_p$. The number of possible representatives of the same value X is then equal to the number of partitions γ. For a Cox Rower$(15, 36, 1)$ (enough to compute a 1024 bit RSA-CRT), this makes around 2^{37} different masks. This kind of mask needs no expensive inversion step for demasking, because it is automatically done by invert Montgomery transformation (which is a simple call to the `Red` function).

Using the partition γ instead of any completely different RNS base each time allows to reduce to 0 the need of specific precomputations depending on γ. To get this result, Bajard uses the Mixed Radix System (MRS) transformation to implement the $switch(X, \beta, \beta')$. The value X is first transformed in MRS of the base β before being recomputed in β'. The RNS to MRS transformation does only need $\mu_{i,j} = |m_i^{-1}|_{m_j}$. The MRS to RNS transformation uses $m_{i,j} = |m_i|_{m_j}$, and others precomputed values such as $< -p^{-1} >_{M(\mathcal{B}_{1,\gamma})}$ and $< p >_{M(\mathcal{B}_{2,\gamma})}$ are indeed the set of $|-p^{-1}|_{m_{\gamma(i)}}$ and $|p|_{m_{\gamma(n+i)}}$. All these precomputations are independant from γ.

There does not exist speed efficient hardware implementation of this kind of algorithm so far, mostly because there are too many dependencies of temporary results during the MRS transformation. Therefore it is not really suitable for parallel high speed computation.

The only difficulty is the computation of the Montgomery representative $|XM(\mathcal{B}_{1,\gamma})|_p$. We can indeed do it by computing $\texttt{Red}(X \times |M(\mathcal{B})|_p, p, \mathcal{B}_{2,\gamma}, \mathcal{B}_{1,\gamma})$ (note the base swap between $\mathcal{B}_{1,\gamma}$ and $\mathcal{B}_{2,\gamma}$).

3.2 Precomputations

The main difficulty of the Kawamura's algorithm is the number of precomputations that are needed to execute the reduction algorithm. These precomputations are not a problem if the bases $\mathcal{B}_1$ and $\mathcal{B}_2$ are fixed and known.

We demonstrate here that they are neither a problem if we have a Cox Rower(n, r, ρ) at our disposal, because all the needed precomputations can be done by the Cox Rower itself, starting from γ independent precomputations (in fact, the same as the one Bajard needs). First we choose for every Rower Row_j, ρ random moduli from $\mathcal{B}_{1,\gamma}$ and ρ from $\mathcal{B}_{2,\gamma}$. Next for each chosen moduli m_x we load the RAM of j^{th} Rower with the following precomputed values : $|p|_{m_x}, |-p^{-1}|_{m_x}, ||M(\mathcal{B})|_p|_{m_x}, m_{i,x}$

and $\mu_{i,x}$ for $i \in [1, 2n]$. The values $m_{i,i}$ and $\mu_{i,i}$ are both set to 1. Next we compute the following values :

$$\begin{array}{rcl}
< A >_{\mathcal{B}_{1,\gamma}} = & < M(\mathcal{B}_{2,\gamma}) >_{\mathcal{B}_{1,\gamma}} & = \{|\prod_{i=n}^{2n} m_{\gamma(i),\gamma(j)}|_{m_{\gamma(j)}}\} \text{ for } j \in [0,n] \\
< B >_{\mathcal{B}_{1,\gamma}} = & < M(\mathcal{B}_{2,\gamma})^{-1} >_{\mathcal{B}_{1,\gamma}} & = \{|\prod_{i=n}^{2n} \mu_{\gamma(i),\gamma(j)}|_{m_{\gamma(j)}}\} \text{ for } j \in [0,n] \\
< C >_{\mathcal{B}_{1,\gamma}} = \{|M(\mathcal{B}_{1,\gamma})/m_{\gamma(i)}\|_{m_{\gamma(i)}}\} & = \{|\prod_{i=0}^{n} m_{\gamma(i),\gamma(j)}|_{m_{\gamma(j)}}\} \text{ for } j \in [0,n] \\
< D >_{\mathcal{B}_{1,\gamma}} = \{|m_{\gamma(i)}/M(\mathcal{B}_{1,\gamma})|_{m_{\gamma(i)}}\} & = \{|\prod_{i=0}^{n} \mu_{\gamma(i),\gamma(j)}|_{m_{\gamma(j)}}\} \text{ for } j \in [0,n]
\end{array}$$

and the same $< A' >_{\mathcal{B}_{2,\gamma}}, < B' >_{\mathcal{B}_{2,\gamma}}, < C' >_{\mathcal{B}_{2,\gamma}}$ and , $< D' >_{\mathcal{B}_{2,\gamma}}$ with $\mathcal{B}_{1,\gamma}$ and $\mathcal{B}_{2,\gamma}$ swapped in the above formulae. The number of cycles needed to compute it in a Cox Rower(n, r, ρ) is $8\rho n$ cycles. From here it is easy to see that the values given in figure 1 can be easily computed through $2\rho n + 6\rho$ cycles :

$$\begin{array}{rl}
\{|(-pM(\mathcal{B}_{1,\gamma})/m_{\gamma(i)})^{-1}|_{m_{\gamma(i)}}\} & = < -p^{-1} \times A >_{\mathcal{B}_{1,\gamma}} \\
\{|pm_{\gamma(n+j)}/(m_i M(\mathcal{B}_2))|_{m_{\gamma(n+j)}}\}_{\mathcal{B}_{2,\gamma}} & = \quad < p\times > \\
|-pm_{n+j}/M(\mathcal{B}_2)|_{m_{\gamma(n+j)}} & \\
\text{finir...} &
\end{array}$$

As we need $\texttt{Red}(X, p, \mathcal{B}_{2,\gamma}, \mathcal{B}_{1,\gamma})$, we add $2\rho n + 6\rho$ cycles more. Eventually if another prime q is needed (like RSA CRT), this adds $2\rho(n + 3)$ for the whole reduction step.

Note that the value of the prime p is automatically masked by a value depending on γ during the base precomputation. Therefore, even if the attacker knows the input X of the `Red` algorithm (which is the case for the Montgomery representation computation), she will not be able to set classical DPA to recover p (useful for RSA cryptosystem).

As a conclusion, the precomputations for the LRA countermeasure can efficiently be executed by a Cox Rower(n, r, ρ). The total cycle cost is $12\rho(n + 1)$ cycles (with an extra cost of $2\rho(n + 3)$ cycles for an additive prime. Therefore the main problem remains to load the complete RAMs with precomputed values independant of γ (we use a simple data transfer to put all the precomputed values corresponding to the moduli of $\mathcal{B}_{1,\gamma}$ and $\mathcal{B}_{2,\gamma}$ in the right Rower, no computation is needed).

4 The RNS Fault Detection countermeasure

In this section we introduce a new countermeasure against faults, which we call the RNS Fault Detection (RFD). In this section we introduce this countermeasure independently from the LRA countermeasure. These two can indeed be used separately. Therefore we start from the notation of section 2.

4.1 Fault model

Let us consider a Cox Rower(n, r, ρ) defined with two fixed bases $\mathcal{B}_1 = \{m_1, \cdots, m_n\}$ and $\mathcal{B}_2 = \{m_{n+1}, \cdots, m_{2n}\}$, not necessarily random, and known by the attacker. All the m_i are pseudo-mersenne numbers, and we define r_ε as in subsection 2.3. Let p be the used modular value.

The fault model we address is a fault on a single Rower: while all the Rowers compute in parallel, only one is targeted by the fault. This fault model is typical of a laser injection. As Rowers is the most important part of the design, it is crucial to protect them, and we propose an efficient way to achieve this goal. A fault on the Cox or on the sequencer of the Cox Rower is not addressed, some ideas are proposed in the subsection 4.6.

The intuition we follow is that if the base is bigger (e.g a Rower is added), the Montgomery reduction algorithm still gives a result between 0 and $3p$, and then we can use the value of the added Rower as a proof that no fault has been realized. We also follow the intuition that two RNS values close to one another (with the same RNS digits on most of the moduli of the base) are not close to one another in classical multiprecision representation (the absolute value of their difference is big). We then just have to exploit this property. To do so, we still use the Kawamura algorithm (algorithm 2) together with the Cox Rower architecture.

4.2 The s limited development Cox

First we need to reconsider the $eval(a, b)$ function needed at line 9 of algorithm 2. Indeed, the proposed $\xi_i/2^r$ of [14] is too approximative as we will see in subsection 4.4. We propose to use a s limited development of ξ_i/m_i :

$$\xi_i/m_i \simeq \xi_i(\sum_{j=0}^{s-1} \frac{(2^r - m_i)^j}{2^{r(j+1)}}) \tag{4}$$

Therefore, with a s limited development design, the approximation $eval(\xi_i, m_i)$ is still an underestimation, the value $errmax$ defined in subsection 2.2 respects the following :

$$errmax \leq \sum_{i=0}^{n} \left(\frac{2^r - m_i}{2^r} \right)^s \leq 2^{-s(r-r_\varepsilon)+\lceil log_2(n) \rceil} \tag{5}$$

The hardware design realizing this operation is given in figure 2. It is constituted of at least two pipeline stages : a multiplication stage and an addition stage. The multiplication stage is constituted of $s-1$ multipliers. One of the operand is the main bus of the Cox Rower, and the second is a precomputed value relying on $\mathcal{B}_1$ or $\mathcal{B}_2$. Considering that r is the maximal size of the multiplier, we easily see that we can have s up to $\lfloor r/r_\varepsilon \rfloor$ without taking any risk on the critical path of the design (the same multipliers are in the Rowers).

The addition part may be more complex to integrate. Indeed the Cox signal must be available two cycles after the computed ξ_i value apparition on the main bus of the Cox Rower, this signal being used on the second pipeline stage of the Rower (see figure 5). This lets us only one cycle to do all the additions of equation 5. Nevertheless, the value of the Cox signal can easily be anticipated if a high value of s is needed.

Note eventually that the original Kawamura's Cox [14, **?**], is actually our proposed Cox reduced to $s = 1$ (with no multiplier).

4.3 Fault detection using s limited development Cox

Let us have $\mathcal{B}_2$ largely superior to $3p$, the maximal value of $\{S\}_{\mathcal{B}_2}$ on line 4 of algorithm 1. Let us define σ such that $M(\mathcal{B}_2) > 2^{\sigma+1}3p$. $M(\mathcal{B}_1)$ is still superior to αp. We can prove the following :

Proposition 1. *Considering the* 3 *following conditions :*

- *s the limited development level is such that $s(r - r_\varepsilon) \geq \lceil log_2(n) \rceil + \sigma + 1$,*
- *$errinit = 2^{-\sigma-1}$*
- *X the input of the* `Red` *algorithm is less than αp^2 ,*

the σ most significant bit of the accumulator register of the Cox are equal to 0 *at the end of of the second base change (line 5 of the algorithm 1).*

Proof : As $X \leq \alpha p^2$, we deduce that $S \leq 3p$ at line 4 of algorithm 1. In the $\texttt{Switch}_{\mathcal{B}_2,\mathcal{B}_1}(S)$ (line 5) we can then deduce the following :

$$0 \leq \texttt{dec}\left(\sum_{i=0}^{n} \frac{\xi_i}{m_i}\right) = \frac{S}{M(\mathcal{B}_1)} < 2^{-\sigma-1} \tag{6}$$

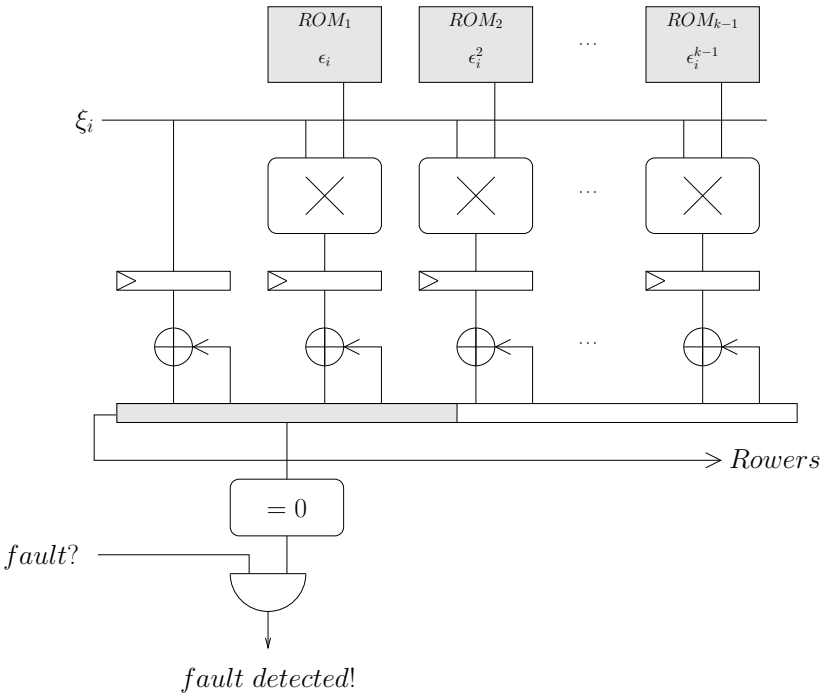
Thanks to the definition of s and $errinit$, we also deduce that at the end of the algorithm 2,

$$\texttt{dec}\left(\sum_{i=0}^{n} \frac{\xi_i}{m_i}\right) \leq reg\_accumulator \leq \texttt{dec}\left(\sum_{i=0}^{n} \frac{\xi_i}{m_i}\right) + errinit \tag{7}$$

We can have then the final inequation : $0 \leq reg\_accumulator < 2^{-\sigma}$ QED.

In figure 2, we present the fault detection capable Cox with a s limited development. Considering that the conditions of proposition 1 are fulfilled, if no fault is introduced in the design, the "$fault\ detected$!" signal remains down during the whole computation. In the following we propose to introduce faults on local variables in the Rowers.

Fig. 2. Fault detection capable Cox



4.4 Fault detection on the base $\mathcal{B}_2$

In this subsection we introduce an error in one Rower, on a value on $\mathcal{B}_2$. It corresponds to a fault either on $\{X\}_{\mathcal{B}_2}$, on the lines 3 and 4 of algorithm 1, on the line 5, 11 and 13 of the first base change, the line 2 of the second base change, or on any precomputed value needed for precomputations in $\mathcal{B}_2$.

We consider s, $M(\mathcal{B}_2)$ and the faultless value X such that the conditions of proposition 1 holds. We can then deduce

Proposition 2. *If $\sigma \geq r+1$, any fault on the base $\mathcal{B}_2$ on a single Rower can be detected by the s limited development Cox, or has no impact on the result.*

Proof : Any unique fault on the upper invoked lines, leads to a value $\xi_i' = \xi_i + \epsilon_i$ after the execution of the line 2 of the second base change. As the registers of the Cox Rower are of size r, we have $-m_{n+i} < -\xi_i \leq \epsilon_i \leq 2^r - \xi_i < 2^r$. At the end of the second base change, the value of $reg\_accumulator'$ is set to $\texttt{dec}(reg\_accumulator + eval(\epsilon_i, m_{n+i}))$. We then have to consider 2 cases :

- $\epsilon_i = m_{n+i}$: We see easily that this fault has no impact on the result : the offset on $< S >_B$ (line 11 of algorithm 2) is $M(\mathcal{B}_2)$ and is automatically corrected by the line 15, which is executed one time too much. The fault detection may work or not, depending on the approximation of $eval(\epsilon_i, m_{n+i})$,
- other ϵ_i : We have the following inequation :

$$\frac{\epsilon_i}{2^r} < eval(\epsilon_i, m_{n+i})) < \epsilon_i \left(\frac{1}{2^r} + \frac{1}{2^{r+1}} \right). \tag{8}$$

Therefore it is easy to see that for any value $\epsilon_i \notin \{0; m_{n+i}\}$, $\texttt{dec}(reg\_accumulator + eval(\epsilon_i, m_{n+i}))$ has at least one 1 in its σ most significant bits, and is automatically detected by the Cox.

4.5 Faults on the base $\mathcal{B}_1$

In this section we introduce fault on the line 1 of algorithm 1, on the line 2 of the first base change, or on the lines 11 and 13 of the second base change (algorithm 2), or on any precomputed value needed for precomputations in $\mathcal{B}_1$. All these faults induce incorrect values on the values ξ_i at line 2 of the first base change. For the fault on the second base change, we suppose that these will be used further in the whole cryptographic algorithm. If not, we can detect them thanks to the RNS to multiprecision transformation algorithm (indeed, the expected value must be less than $3p$. If not, it means that a fault occurred).

Proposition 3. *Considering the 3 following conditions :*

- *s the limited development level is such that $s\rho_\varepsilon \geq \lceil log_2(n) \rceil + \sigma + 1$,*
- *$errinit = 2^{-\sigma-1}$*
- *X the input of the* `Red` *algorithm is less than αp^2 ,*

If $\sigma \geq (r + \lceil log(6n) \rceil + 1) + \phi$, the probability that there exist a potentially undetected fault on a single Rower is

$$Pr(\exists f \ undetected\ ) < 2^{-\phi} \tag{9}$$

Proof : we consider a single fault ϵ_i, with $\xi'_i = \epsilon_i + \xi_i$ at line 2 of the first base change. Let us see how it is propagated during the `Red` algorithm.

- $Q'' = |Q' + \epsilon_i M(\mathcal{B}_1)/m_i|_{M(\mathcal{B}_2)}$ if the line 13 has been executed the same number with or without the fault.
- $Q'' = |Q' + \varepsilon M(\mathcal{B}_1)/m_i - M(\mathcal{B}_1)|_{M(\mathcal{B}_2)}$ if line 13 of algorithm 2 has been executed one time to much,
- $Q'' = |Q' + \varepsilon M(\mathcal{B}_1)/m_i + M(\mathcal{B}_1)|_{M(\mathcal{B}_2)}$ if line 13 of algorithm 2 has been missed once.

The input of $\texttt{Switch}_{\mathcal{B}_2,\mathcal{B}_1}$ (line 5 of algorithm 1) is therefore :

$$S' = |S + \varepsilon p m_i^{-1}|_{M(\mathcal{B}_2)} \tag{10}$$

$$\text{or } S' = |S + \varepsilon p m_i^{-1} - p|_{M(\mathcal{B}_2)} \tag{11}$$

$$\text{or } S' = |S + \varepsilon p m_i^{-1} + p|_{M(\mathcal{B}_2)} \tag{12}$$

Considering the fault detection as mentionned above. Thanks to the precision of the eval function, it is easy to see that the following proposition is true.

$$\text{No fault detection} \Rightarrow S' < M(\mathcal{B}_2)2^{-\sigma} \text{ or } S' > M(\mathcal{B}_2)(1 - 2^{-\sigma-1}). \tag{13}$$

Eventually as $0 \leq X < 3p < 2^{-\sigma-1}$, equation 13 becomes :

$$\text{No fault detection} \Rightarrow |X' - X|_{M(\mathcal{B}_2)} < M(\mathcal{B}_2)2^{-\sigma} \text{ or } X' > M(\mathcal{B}_2)(1 - 2^{-\sigma}). \tag{14}$$

As X is between 0 and $3p$, we have :

$$\forall i \in [0; n[, \forall \epsilon_i \in]-m_i; 2^r[, \epsilon_i \notin \{0, m_i\}, \forall \zeta \in \{-1, 0, 1\}\ 2^{-\sigma} < \frac{|\varepsilon p m_i^{-1} + \zeta p|_{M(\mathcal{B}_2)}}{M(\mathcal{B}_2)} < 1 - 2^{-\sigma}$$
$$\Rightarrow \text{a fault detection occurs.} \tag{15}$$

We can see in this result that any fault on the base B leads to a deviation depending on the basis B and p. We have $6n2^r$ different possible deviation for a fault on a single Rower. If we consider them equally distributed between 0 and $M(\mathcal{B}_1)$, we can conclude our demonstration. Note that this is a worst case for us : as p is very small compared to $M(\mathcal{B}_2)$, the 3 values $|\epsilon p m_i^{-1} + \zeta p|_{M(\mathcal{B}_2)}/M(\mathcal{B}_2)$ for $\zeta \in \{-1, 0, 1\}$ are close to one another. Note also that if a fixed base is used, the inequation **??** can be exhaustively tested for every i,ϵ_i and ζ.

4.6 Fault on the Cox or on the sequencer

At this point we have shown that if we take in consideration the conditions given in proposition 3 (which is essentially raising the size of $M(\mathcal{B}_2)$ as well as the precision of the Cox) any fault on a single Rower will automatically lead to a fault detection, as long as the sequence of operations is not affected by the fault Unfortunately, it is not possible to prove the same assumption on the Cox or on the sequencer. Nevertheless, considering that the Cox is a small part of the design, and that it is necessary to prevent the sequencer from fault, we can use redundancy, dual rail, or any classical technique to detect execution errors in this subpart of the design.

4.7 Combining both countermeasures

The LRA and the RFD countermeasures can efficiently be used together. Indeed we see that propositions 1 and 2 uniquely rely on n, r and s, and are therefore true for any permutation γ. For the second base change, we have only restricted the probability of the existence of an undetected fault, without proving their non existence. But as seen in equation 10, the fault depends on p but also $M(\mathcal{B}_{1,\gamma})$ and $M(\mathcal{B}_{2,\gamma})$. Randomizing the RNS bases makes the task of the attacker harder.

5 Securing RSA-CRT with LRA and RFD countermeasures

In this section we propose an implementation of the Cox Rower carrying the LRA as well as the RFD countermeasure. We design it to efficiently execute a classical cryptographic operation : the 1024 bit RSA-CRT cipher primitive.

Let $N = pq$ a 1024 bit RSA key (p and q are 512 bit long), d the private exponent and $e = |d^{-1}|_{\phi(N)}$ the public one. Let M be the input of the RSA primitive. The RSA-CRT computes $|M^d|_N$, but using the RNS base $\{p, q\}$ and its fast arithmetic over $\mathbb{Z}/N\mathbb{Z}$. It replaces the expensive 1024 bit exponentiation with two 512 bits exponentiation. The complexity reduction is around 4 (in RNS as well as in the classical multiprecision context). Algorithm 3 describes the whole algorithm to compute RSA-CRT using the Cox Rower. We remark that the reduction step is cheaper for RNS the way it is proposed than the classical Garner's recombination : $C = c_q + q(|q^{-1}|_p(c_p - c_q))$. It only costs 2 reductions which are mixed with the Montgomery inverse transformation instead of 3.

Algorithm 3 $RSA-CRT$ for RNS (p, q, d_p, d_q, X, m)

Require: p, q, d a RSA key, m a message, $\mathcal{B}$ a RNS base,
Ensure: $c = |m^d|_{pq}$
1: `Radix2RNS`$(m, M(\mathcal{B}))$
2: computes γ a partition of $\mathcal{B}$ in $\mathcal{B}_{1,\gamma}$ and $\mathcal{B}_{2,\gamma}$,
3: $M_p =$ `TrMgt`$(m, p, \mathcal{B}_{1,\gamma}, \mathcal{B}_{2,\gamma})$
4: $M_q =$ `TrMgt`$(m, q, \mathcal{B}_{1,\gamma}, \mathcal{B}_{2,\gamma})$
5: $C_p =$ `expo`$(M_p, d_p, p, , \mathcal{B}_{1,\gamma}, \mathcal{B}_{2,\gamma})$
6: $C_q =$ `expo`$(M_q, d_q, q, \mathcal{B}_{1,\gamma}, \mathcal{B}_{2,\gamma})$
7: $c_p =$ `Red`$(C_p * |q^{-1}|_p, p, \mathcal{B}_{2,\gamma}, Bg)$
8: $c_q =$ `Red`$(C_q * |p^{-1}|_q, q, \mathcal{B}_{1,\gamma}, \mathcal{B}_{2,\gamma})$
9: $C = c_p * q + c_q * p$
10: **return** `RNS2Radix`(C, M)

5.1 RSA-CRT physical vulnerability

It is a well known fact that classical RSA-CRT implementation without countermeasures is very sensitive to differential fault analysis. Indeed, if an attacker gets a message and a corresponding faulty cipher text (m, c'), after having injected a fault anywhere between line 3 and 9 she can get the private key be simply executing the following computation :

$$p \text{ or } q = gcd(N, c'^e - m) \tag{16}$$

The simplest way to prevent such an attack is to verify if $c = m^e$. Often, this countermeasure does not cost a lot, since e is small. But in many case, e is not small, and verifying costs a lot (the same as a classical 1024 bit exponentiation). Other times, e is simply not available (for example the javacard RSA API does not provide it).

Design of countermeasure for RSA-CRT has known a lot of attention in the public community. Among them, we can cite the following [5, 8, 12, 18, 22, 10]. All these countermeasure are design for smart card devices, and evaluation of the performance uses a complexity approach. Parallelism is rarely considered, and that makes these countermeasure not completely suitable for high speed designs.

5.2 The Ciet *et al.* [9] countermeasure

In [9], the authors proposed a FPGA implementation of a 1024 bit RSA-CRT using $r = 64$ bits. Like this work, they used the Residue Number System together with additional moduli in their RNS base. They also claimed to be SCA-FA resistant, but their approach is not satisfactory for two reasons :

- to avoid side channel, they proposed to use base randomization, but only choosing two bases in a set precomputed ones. This is clearly less powerful than the Bajard *et al.* countermeasure [3] proposed in this work.
- They proposed to use a redundant modulus in the RNS base, but instead of detecting the fault at each reduction, they wait for the RNS to classical multiprecision representation transformation to verify if the value is in the expected range (less than $3p$ in our case). Used alone, this countermeasure is clearly weak. If a fault occurs at the beginning of the exponentiation step, even though the faulty value will be large compared to $3p$, as this value will be multiplied and reduced many time before being turned into classical multiprecision representation, and as $M(\mathcal{B}_1)$ is necessarily more than $3p$, the manipulated value will progressively decrease, and may be eventually in the expected range at the end of the computation. Our approach does not suffer this drawback, since all the temporary variables are tested each time they are reduced.

5.3 Our SCA-FA resistant implementation of a 1024 bits RSA-CRT

In RSA-CRT, no reduction by N is needed except in the reconstruction step (as x'_p and x'_q can be up to $3p$ we have $C < 6p$). This reduction step cannot be done by the Cox Rower, but remains cheap enough to be executed by a small general purpose embedded CPU. Therefore we propose a Cox Rower having the following parameters :

n	r	r_ε	ρ	s	σ	ϕ
16	36	8	1	2	51	8

Note that with these parameters the value of σ is limited by s and not by the size of $M(\mathcal{B}_2)$. But these parameters are clearly enough to ensure that no fault on a single Rower can be achieved without being detected. We implement the architecture of the figure 5 on the Stratix III EP3SL50F484C2, the smallest of the Stratix III series (available in the web edition of Altera Quartus II). We used redundancy to protect both the Cox and the sequencer, as they are not protected by the RFD

countermeasure. The final design is cadenced by a single clock which can run up to 148 MHz at $85°C$, according to the Altera Quartus II software.

Fig. 3. Area consumption of the Cox Rower

Module	number	logic	18×18 DSP	Memory
Rower	16	467 ALM	9	2 M9k
Cox	2	220 ALM	4	0
sequencer	2	416 ALM	0	2 M9k
Cox Rower	1	8385 ALM	148	34 M9k

The total cost of the LRA and RFD countermeasures is 1323 ALM out of 8385 (15%), 13 18×18 DSP blocks out of 148 (9%), and eventually 6 M9k memory block out of 34 (18%).

To protect the exponentiation step against side channel analysis, we used the Montgomery ladder together with the exponent randomization (see algorithm 5 in appendix). It is clearly less efficient than the Rivain countermeasure [18], but does not need on the fly double chain computation (the double chain computation needs to be on the fly because of the exponent randomization). We chose a 36 bit random value. Our Leak Resistant Arithmetic countermeasure provides the operand randomization needed to thwart Fouque attacks [11]. Thanks to the exponent randomization, we are also immune against safe errors.

5.4 Montgomery representation computation

Let now X be an input of the RSA-CRT algorithm. One need first to compute the Montgomery representation $M(X) = |XM(\mathcal{B}_{1,\gamma})|_p$. In [3], the computation of $\texttt{Red}(X \times |M(\mathcal{B})|_p, p, \mathcal{B}_{2,\gamma}, \mathcal{B}_{1,\gamma})$ gives the result, and the only precomputations which are needed do not depend on γ. In section 3 we have shown that our Cox Rower is able to do the same. But for RSA-CRT we need a small adaptation of the Bajard algorithm : as X is 1024 bits long, $X|M|_p$ may be a 1536 bits number, and we cannot use such a number in our Cox Rower. Algorithm 4 shows this adaptation.

Algorithm 4 $\texttt{TrMgt}(X, p, \mathcal{B}_{1,\gamma}, \mathcal{B}_{2,\gamma})$

Require: p a prime, X a message, $\mathcal{B}_{1,\gamma}$ and $\mathcal{B}_{2,\gamma}$ the two RNS bases,
Ensure: $|X^d|_{pq}$
1: $A = \texttt{Red}(X, p, \mathcal{B}_{2,\gamma}, \mathcal{B}_{1,\gamma})$
2: $B = A \times |M(\mathcal{B})|_p$
3: $C = \texttt{Red}(B, p, \mathcal{B}_{2,\gamma}, \mathcal{B}_{1,\gamma})$
4: $D = C \times |M(\mathcal{B})|_p$
5: $E = \texttt{Red}(D, p, \mathcal{B}_{1,\gamma}, \mathcal{B}_{2,\gamma})$
6: **return** E

5.5 CRT reconstruction and RNS to Radix computation

It is easy to see that with our SCA-FA countermeasures, algorithm 3 is protected until line 8. Afterwords, the variables are not masked by the LRA countermeasure. Fortunately, there are no side channel attack to our knowledge able to take benefit of it (except very powerful attacks like templates). But the operation of line 9 is vulnerable to fault attacks, and no reduction is used after that. Attacking after line 9 is useless, since the manipulated values are public.

To protect the line 9, we propose to use the `RNS2Radix` reconstruction proposed by Guillermin [13] together with our RFD countermeasure. Indeed, the main idea of this reconstruction is to consider $\mathcal{B}_1 = \{2^r\}$ and $\mathcal{B}_2 = \mathcal{B} - \{2^r\}$. Indeed the value modulo 2^r are the LSB of the result. This value is subtracted on each module of $\mathcal{B}_2$, and the obtained value is multiplied by $|2^{-}r|_{M(\mathcal{B}_2)}$, and eventually a base switch can be computed between $\mathcal{B}_1$ and $\mathcal{B}_2$. Hence, this base switch presents the same properties as the second base switch of the algorithm 1. In our proposed case, conditions of proposition 2 are fulfilled, therefore any fault on $\mathcal{B}_2$ will be detected. Using the formula of proposition 1 and 3, we find in this case that $\sigma = 50$ probability that there exists a fault on $\mathcal{B}_1$ which may be undetected is majored by 2^{-5}. Moreover, as the base are fixed, it is easy to prove with an exhaustive search that inequation 15 always holds.

5.6 cycle count

In figure 4 we compare two implementations of the RSA-CRT : the first one is a version using a simple Montgomery ladder using 36 bit exponent randomization in a Cox Rower $(15, 36, 1)$ implementing neither the LRA countermeasure nor the RFD. We use it as a benchmark to evaluate the marginal cost of our two countermeasures. The array shows that it costs less than 6% of the cycle count.

The full 1024 bit RSA-CRT computation requires around 0.6 ms.

Fig. 4. cycle count for a RSA-CRT without and with SCA-FA protections

.

subroutine	RSA-CRT without protection	RSA-CRT with protection	overhead
Base precomputation	0	242	242
`Radix2RNS`	254	254	0
`TrMgt`	80	296	216
`expo`	82771	87155	4384
CRT reconstruction	93	97	4
`RNS2Radix`	1073	1209	136
total	84271	89253	4972

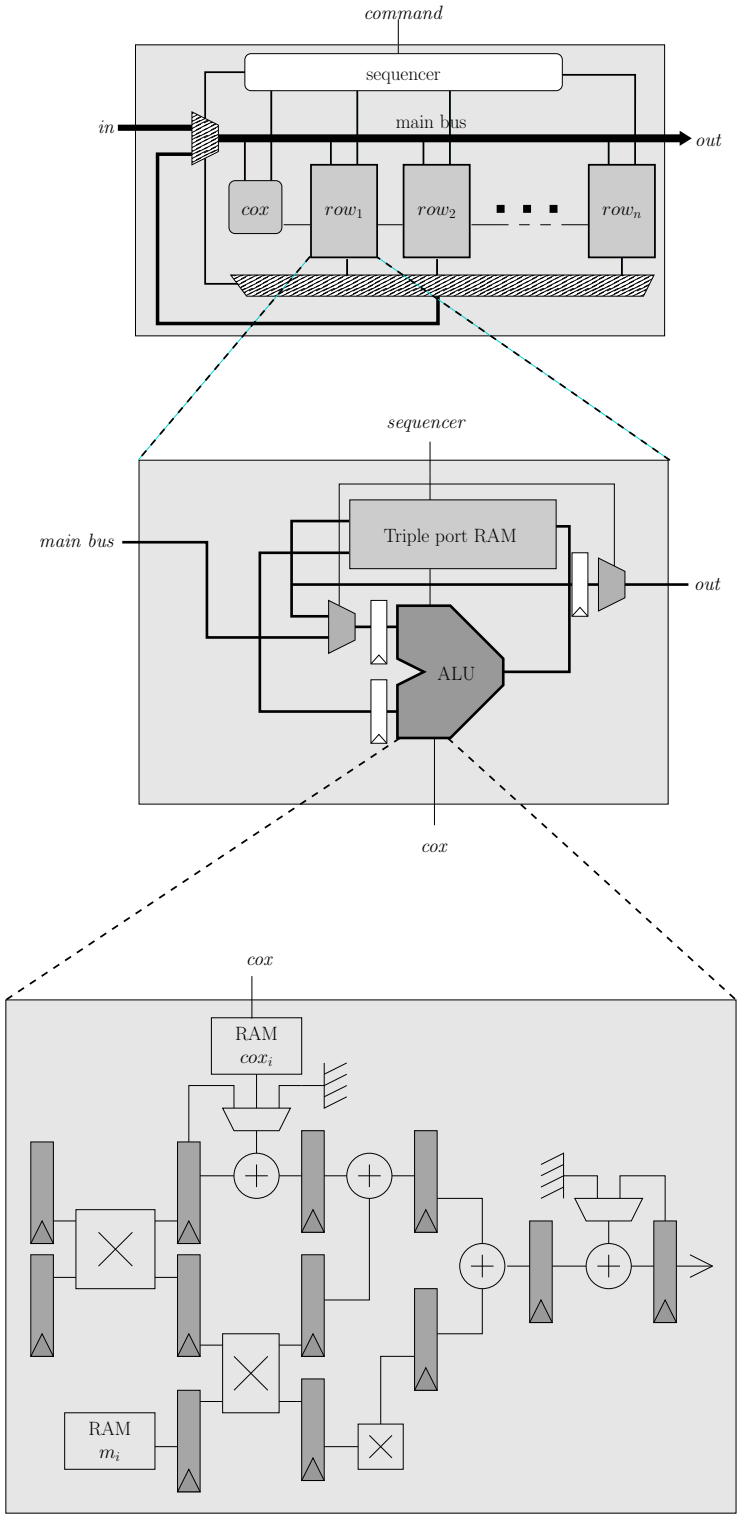
6 Conclusion

In this paper, we proposed an adaptation of the Leak-Resistant Arithmetic (LRA) countermeasure of [3] and proposed a new countermeasure against fault called the RNS Fault Detection (RFD). These two countermeasure are adapted to the Cox Rower architecture and provide a good protection

against side channel and fault analysis for any cryptographic function using modular arithmetic. We implemented our countermeasure in a FPGA in the case of RSA-CRT 1024 bit, showing that our countermeasure induced a small area overhead (15%) as well as a small latency overhead (6%). This makes our countermeasure very affordable for any hardware device requiring both speed and efficient protection against security threats.

Appendix

Fig. 5. General architecture of a Cox Rower



Algorithm 5 The Giraud countermeasure [12]

Require: X, d
Ensure: X^d
1: $R = 1; S = X$
2: **for** i from 1 to $log_2(d)$ **do**
3: **if** $d_i = 0$ **then**
4: $S = S \times R; R = R^2$
5: **else**
6: $R = S \times R; S = S^2$
7: **end if**
8: **end for**
9: **if** $XR \neq S$ **then**
10: throw fault detection
11: **end if**
12: **return** R

References

1. Analysis and improvement of the random delay countermeasure of ches 2009. In *CHES*, volume 6225, page 95, 2010.
2. Jean-Claude Bajard, Laurent-Stphane Didier, and Peter Kornerup. An rns montgomery modular multiplication algorithm. *IEEE Transactions on Computers*, 47(7):766–776, 1998.
3. Jean-Claude Bajard, Laurent Imbert, Pierre-Yvan Liardet, and Yannick Teglia. Leak resistant arithmetic. In *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 116–145. Springer Berlin / Heidelberg, 2004.
4. Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, pages 513–525, London, UK, 1997. Springer-Verlag.
5. Johannes Blomer, Martin Otto, and Jean pierre Seifert. A new crt-rsa algorithm secure against bellcore attacks. In *CCS 2003, ACM SIGSAC, ACM Press*, pages 311–320. ACM Press, 2003.
6. Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In *EUROCRYPT*, pages 37–51, 1997.
7. D. Canright and Lejla Batina. A very compact "perfectly masked" s-box for aes. In *Proceedings of the 6th international conference on Applied cryptography and network security*, ACNS'08, pages 446–459, Berlin, Heidelberg, 2008. Springer-Verlag.
8. Mathieu Ciet and Marc Joye. Practical fault countermeasures for chinese remaindering based rsa (extended abstract). In *IN PROC. FDTC05*, pages 124–131.
9. Mathieu Ciet, Michael Neve, Eric Peeters, and Jean jacques Quisquater. Parallel fpga implementation of rsa with residue number systems can side-channel threats be avoided. In *46 th . International Midwest Symposium on Circuits and Systems: MWSCAS 03*, 2003.
10. Jean-Sébastien Coron, Christophe Giraud, Nicolas Morin, Gilles Piret, and David Vigilant. Fault attacks and countermeasures on vigilant's rsa-crt algorithm. In *FDTC*, pages 89–96, 2010.
11. Pierre-Alain Fouque and Frédéric Valette. The doubling attack - *hy upwards is better than downwards*. In *CHES*, pages 269–280, 2003.
12. Christophe Giraud. An rsa implementation resistant to fault attacks and to simple power analysis. *IEEE Trans. Computers*, 55(9):1116–1120, 2006.
13. N. Guillermin. A high speed coprocessor for elliptic curve scalar multiplications over $\mathbb{F}_p$. In *CHES '10: Proceedings of the Twelvth International Workshop on Cryptographic Hardware and Embedded Systems*, 2010.

14. Shinichi Kawamura, Masanobu Koike, Fumihiko Sano, and Atsushi Shimbo. Cox-rower architecture for fast parallel montgomery multiplication. In *Advances in Cryptology EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 523–538. Springer Berlin / Heidelberg, 2000.
15. Donald Knuth. The art of computer programming.
16. Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO*, pages 104–113, 1996.
17. Hanae Nozaki, Masahiko Motoyama, Atsushi Shimbo, and Shin-ichi Kawamura. Implementation of rsa algorithm based on rns montgomery multiplication. In *CHES '01: Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems*, pages 364–376, London, UK, 2001. Springer-Verlag.
18. Matthieu Rivain. Securing rsa against fault analysis by double addition chain exponentiation. In *CT-RSA*, pages 459–480, 2009.
19. Adi Shamir. Improved method and apparatus for protecting public key schemes from timing and fault attacks, 1998.
20. Robert Szerwinski and Tim Gayneysu. Exploiting the power of GPUs for asymmetric cryptography. In *Cryptographic Hardware and Embedded Systems CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 79–99. Springer Berlin / Heidelberg, 2008.
21. Kris Tiri and Ingrid Verbauwhede. A logic level design methodology for a secure dpa resistant asic or fpga implementation. pages 246–251, 2004.
22. David Vigilant. Rsa with crt: A new cost-effective solution to thwart fault attacks. In *CHES*, pages 130–145, 2008.