

Montgomery reduction within the context of residue number system arithmetic

Jean-Claude Bajard, Julien Eynard, Nabil Merkiche

▶ **To cite this version:**

Jean-Claude Bajard, Julien Eynard, Nabil Merkiche. Montgomery reduction within the context of residue number system arithmetic. Journal of Cryptographic Engineering, 2018, 8 (3), pp.189-200. 10.1007/s13389-017-0154-9 . hal-01494547

HAL Id: hal-01494547

https://hal.sorbonne-universite.fr/hal-01494547

Submitted on 23 Mar 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Noname manuscript No.
(will be inserted by the editor)

Montgomery Reduction within the Context of Residue Number System Arithmetic

Jean-Claude Bajard · Julien Eynard · Nabil Merkiche

Keywords Montgomery Reduction · Residue Number System · Chinese Remainder Theorem · RSA · ECC · Lattice-based Cryptography · Hardware Architecture

1 Introduction

Many of the most common public-key cryptosystems (RSA, DSA, Diffie-Hellman, etc) involve intensive computations in finite fields/rings. Within such schemes, modular reduction of large values is a core operation. Its efficiency is a critical issue for the effectiveness of practical implementations. In 1985, Peter L. Montgomery proposed a modular reduction algorithm whose particularity is to not depend on costly divisions [37]. Since then, this approach has been widely adopted for implementing many asymmetric cryptographic primitives.

Regarding the efficiency of asymmetric primitives, Residue Number Systems (RNS) are interesting alternatives to classical radix based representations for performing basic arithmetic operations on large values. However, RNS are of a non-positional nature. Thus, defining an RNS modular reduction is not straightforward. Within such context, Montgomery's reduction has turned out to be well appropriate for such representation. Yet, many issues had to be overcome in order to obtain the state-of-the-art version of RNS modular reduction. RNS version of Montgomery's key idea (replacing costly divisions by simple shifting by powers of the radix) relies on an exact division by the product of elements of an RNS base. Such exact division can be handled through a multiplication by a modular inverse. However, it cannot be performed in the underlying RNS base. A possible solution is to use a second RNS base within which this computation is possible. Hence, a lot of works dedicated to RNS modular reduction have been focused on such kind of structure involving two RNS bases. In particular, the conversion procedures between the two bases have always been a central topic.

A first publication dealing with RNS modular multiplication was proposed by Posch and Posch in 1995 [45], where the use of an auxiliary base was suggested. For their approach, Posch and Posch needed an RNS base extension procedure, which is a costly operation. To efficiently achieve it, they proposed to use the classical constructive proof of Chinese Remainder Theorem, whose principle looks like Lagrange's interpolation. However, this approach introduced another issue. Their base extension required a correction which is hardly compatible with RNS representation. To solve this problem, a floating-point computation was used to make the correction. In 1998 [5], another result enabled to use integer operations only, by using a Mixed Radix representation (relying, this time, on Newton's like interpolation) for the first conversion and an approach due to Shenoy-Kumaresan [47] (which involves an extra modulus) for the second conversion. In 2000, Kawamura et al. [33] refined Posch and Posch's method by using some properties of the RNS base. The floating-point computation was replaced by approximations involving some of the most significant bits of residues. Besides, they proposed a hardware architecture fitting with this

Jean-Claude Bajard
Sorbonne Universités, UPMC, CNRS, LIP6, Paris, France
E-mail: jean-claude.bajard@lip6.fr

Julien Eynard
ECE Dept., University of Waterloo, Ontario, Canada
E-mail: jeynard@uwaterloo.ca

Nabil Merkiche
DGA Maîtrise de l'Information
E-mail: merkiche.nabil@gmail.com

***Acknowledgements**: This work has been supported in part by the French ANR ARRAND 15-CE39-0002-01.*

kind of approach, called Cox-Rower architecture. It is well adapted to hardware implementation and is the most used for practical implementations on embedded device. In 2001 [6], the question of efficiency for the RNS Montgomery reduction was taken a step further by showing how the correction within Lagrange's like approach can be skipped. Even though such strategy is intended to improve overall efficiency of RNS Montgomery reduction/multiplication, it may cause the result requiring further complete reduction. In 2014 [17], an improvement of the Cox-Rower architecture was suggested by introducing a second level of Montgomery reduction within each RNS unit. This enables to increase the set of RNS moduli for which good performance can be reached. As a consequence, cryptosystems involving larger finite fields can be implemented in the same architecture. In 2015 [20], a new approach has been proposed for implementing RNS modular reduction. It does not involve Montgomery's reduction but rather resorts to a hybrid representation, mixing RNS and a classical radix-based positional system, in order to get Mersenne like reductions. Beyond the theoretical novelty, this allows to divide by two the area needed for implementing an RNS Montgomery reduction for specific fields. However, this last approach is less general than the Montgomery one. Indeed, it requires some property to be verified between the RNS base and the modulo.

The paper is organised as follows. Section 2 introduces Residue Number Systems (RNS) and details how the modular Montgomery reduction is adapted to such kind of representation. In Sec. 3, an overview of the most common variants of this reduction, which differ by the way RNS bases are extended, is provided. Section 4 is dedicated to hardware implementations like the classical Cox-Rover architecture. Section 5 depicts how the RNS Montgomery reduction can be used within most of the public key cryptography approaches. Finally, Sec. 6 concludes with an overview of the different kind of protections the RNS Montgomery reduction offer against side-channel analysis and fault attacks.

2 Residue Number Systems and Modular Arithmetic

2.1 Introducing Residue Number Systems

Residue Number Systems are non-positional numeral systems used to represent integers. They were independently introduced in [48] and [27]. Such a system is not based on a radix representation, but it relies on a set of pairwise coprime integers $\mathcal{B} = (m_1, \ldots, m_n)$, called "moduli" (their product is usually denoted by M). The Chinese Remainder Theorem states the existence of a ring isomorphism $\mathbb{Z}_{m_1} \times \ldots \times \mathbb{Z}_{m_n} \simeq \mathbb{Z}_{m_1 \times \ldots \times m_n}$.

Concretely, knowing a remainder $x_M = |x|_M = (x \bmod M)$ is strictly equivalent to knowing the n remainders $x_i = |x|_{m_i} = (x \bmod m_i)$. Furthermore, the isomorphism preserves basic modular arithmetic. Given (x, y) in $[0, M)^2$ and $\star \in \{+, \times\}$,

$$|x \star y|_M \leftrightarrow (|x_1 \star y_1|_{m_1}, \ldots, |x_n \star y_n|_{m_n}) \ .$$

An important advantage is that computations on residues are parallel and independent. Since no carry propagation is required between the residues, additions and multiplications can be performed concurrently. This is also the case of exact division $\frac{x}{y}$ as long as y is coprime to M. Indeed, this operation can be replaced by a multiplication by modular inverses $(|y^{-1}|_{m_i})_{i \in [1,n]}$.

Effect on Complexity of Computations: Given a large interval of integers $[0, p)$ in which some computations have to be made, it is sufficient to choose an RNS base $\mathcal{B}$ with $M \geqslant p$ to be able to represent all the values between 0 and $p-1$. Then, operations over large integers can be replaced by concurrent operations on small residues (an auxiliary RNS Base could be useful depending of the approach). This is advantageous within the context of asymmetric cryptography for optimizing computations in large prime finite fields $\mathbb{F}_p$. By using an RNS base $\mathcal{B}$ with n moduli on r bits ($2^{nr} \gtrsim p$ or $\log_2(p) \lesssim nr$), the cost for multiplying two integers x and y in $[0, p)$ is decreased from $\mathcal{O}(\log_2(p)^2)$ to $\mathcal{O}(nr^2)$ bit operations[1]. Concurrency features of RNS also enable to reduced the time complexity to $\mathcal{O}(r^2)$.

Usually, the complexity of an RNS computation is given in terms of Elementary Modular Multiplications `EMM` (e.g. $(x_i \times y_i \bmod m_i)$ is 1 `EMM`). In practice, it is important to select moduli for which modular reduction is efficient. Among the zoo of such interesting moduli, pseudo Mersenne numbers [16] are a common choice for practical implementations. Such a modulus can be written like $m = 2^r - c$ where $c < 2^{r/2}$.

Reducing an integer x ($x < m^2$) modulo m can be done with roughly $\frac{3}{4}$ `EM` (Elementary Multiplication of two r-bit integers; e.g. $(x_i \times y_i)$ is 1 `EM`). Indeed, by decomposing $x = x_1 2^r + x_0 = (x_1, x_0)$ in base 2^r, one simply computes $x_0 + x_1 \cdot c = (y_1, y_0)$ with $\frac{1}{2}$ `EM` and gets a smaller value, still congruent to x modulo m. Noticing that $y_1 < 2^{r/2}$, a last $\frac{1}{4}$ `EM` allows to get $y_1 \cdot c < 2^r$. The reduction is then achieved with few additions and comparisons to m. Furthermore, the cost of these

[1] Here, we consider a quadratic multiplication. More generally, the complexity goes from $\mathcal{O}(\texttt{Mul}(\log_2(p)))$ to $\mathcal{O}(n \times \texttt{Mul}(r))$, where the best complexities are $\texttt{Mul}(\log_2(p)) \in \mathcal{O}(\log_2(p) \log_2 \log_2(p))$ and $\texttt{Mul}(r) \in \mathcal{O}(r^{1+\epsilon})$, depending of the size of p and r.

elementary reductions can be reduced thanks to lazy reductions (i.e. reducing only when necessary: a sum of k products can be reduced only once instead of k successive times).

Nonetheless, the use of RNS for accelerating finite field computations requires an efficient modular reduction (modulo p) algorithm. For this purpose, Montgomery's approach has turned out to be particularly well appropriate. In further discussions, the residues of an integer x in $\mathcal{B}$ can be denoted by $\boldsymbol{x}_{\mathcal{B}}$. Furthermore, within a clear context, a computation made modulo M means that it is performed in RNS, at the residue level.

2.2 RNS Montgomery Modular Reduction

Contrary to Barrett's algorithm [18], the principle of Montgomery's modular reduction goes well with RNS integer arithmetic. The only divisions occurring within the process are exact integer divisions, which are easily doable in RNS.

In original Montgomery's reduction, the fundamental tool is the use of an integer b for which modular reduction and division by b are easy. For instance, when dealing with binary representation, b is typically a power of 2. Thus, to compute $x \bmod p$ with $x < p^2$ (e.g. when reducing a product of elements in $\mathbb{F}_p$), the idea is to add an integer a to x such that (i) $x+a$ is a multiple of b, (ii) $x + a \equiv x \bmod p$, and (iii) $\frac{x+a}{b}$ is smaller than $2p$. Taking $a = p \times q$ with $q = ((x \times |-p^{-1}|_b) \bmod b)$ satisfies the requirements (i) and (ii). When $|-p^{-1}|_b$ can be precomputed, q is easily computable. If b is chosen large enough so that $x < b \times p$, then (iii) is satisfied too ($\frac{x+a}{b} < 2p$).

This approach has been adapted to RNS in [5, 45]. Let $\mathcal{B}$ be the underlying RNS base used to represent $\mathbb{F}_p$. The previous key ideas are transposable in the following way. First, by replacing b by M, computing modulo M is simply achieved by classical RNS computations in $\mathcal{B}$. Thus, $q = (x \times |-p^{-1}|_M) \bmod M$ is obtained efficiently by computing at the residue level. Second, the exact division by M cannot be performed in the base $\mathcal{B}$. So, an auxiliary base $\mathcal{B}'$, coprime to $\mathcal{B}$, is introduced. For the computation to be ended in $\mathcal{B}'$, the RNS representation of q in $\mathcal{B}$ has to be extended towards $\mathcal{B}'$. For that purpose, a procedure called base extension, and denoted `Bex`, is used. Sometimes, such extension can only provide the residues in $\mathcal{B}'$ of $q+kM$ for some positive small integer k such that $q+kM \leqslant (1+\lambda)M$. This can cause the final result to contain some extra multiple of p. Basically, one could say the smaller the λ (and so the k), the costlier the extension procedure. But there are many refinements that will be reviewed throughout the paper. Once the result is obtained in the auxiliary base, it is sent back to the main base $\mathcal{B}$. The size of M is usually chosen such that $x < Mp$. Finally, the result is $(x+p(q+kM))/M$ and it is expected to be smaller than $(2+\lambda)p$. Regarding M', it has to be large enough to fully contain the final result. Thus, taking $M' > (2+\lambda)p$ is sufficient in this case. Finally, Alg. 1 summarizes the principle of an RNS Montgomery modular reduction.

Remark 1 As in the original reduction algorithm, the RNS Montgomery reduction of x modulo p provides a result which is congruent to $xM^{-1} \bmod p$. Thus, the use of a Montgomery representation is still topical.

Algorithm 1 $\texttt{RnsMR}(\boldsymbol{x}_{(\mathcal{B},\mathcal{B}')}, p, \mathcal{B}, \mathcal{B}')$

Require: coprime RNS bases $\mathcal{B}$, $\mathcal{B}'$, $x < Mp$, $(2+\lambda)p < M'$ with $\lambda \geqslant 0$ related to `Bex1`.
Ensure: $\boldsymbol{s}_{(\mathcal{B},\mathcal{B}')}$, $s \equiv xM^{-1} \bmod p$.

1: $\boldsymbol{q}_{\mathcal{B}} \leftarrow \left|-xp^{-1}\right|_M$ ▷ $\|$ in $\mathcal{B}$
2: $\widehat{\boldsymbol{q}}_{\mathcal{B}'} \leftarrow \texttt{Bex1}(\boldsymbol{q}_{\mathcal{B}}, \mathcal{B}, \mathcal{B}')$ ▷ $\widehat{q} < (1+\lambda)M$
3: $\boldsymbol{t}_{\mathcal{B}'} \leftarrow |x + \widehat{q}p|_{M'}$ ▷ $\|$ in $\mathcal{B}'$
4: $\boldsymbol{s}_{\mathcal{B}'} \leftarrow \left|tM^{-1}\right|_{M'}$ ▷ $\|$ in $\mathcal{B}'$; $s < (2+\lambda)p$
5: $\boldsymbol{s}_{\mathcal{B}} \leftarrow \texttt{Bex2}(\boldsymbol{s}_{\mathcal{B}'}, \mathcal{B}', \mathcal{B})$ ▷ exact extension
6: **return** $\boldsymbol{s}_{(\mathcal{B},\mathcal{B}')}$

3 Main Variants of RNS Montgomery Reduction

The general structure of RNS Montgomery reduction is given in Alg. 1. In practice, the two base extension procedures are the costliest parts of the computation. Different types of extensions can be used in practice, each one owning pros and cons. The choice of extension methods within RNS Montgomery reduction really depends on the context. In this section, some of the main practical variants are reviewed.

3.1 Extensions Based on a Mixed Radix System (MRS)

Any (ordered) RNS base $\mathcal{B} = (m_1, \ldots, m_n)$ is associated to mixed radix base composed of n elements $(1, m_1, m_1m_2, \ldots, m_1 \ldots m_{n-1})$. The principle of computation of MRS coefficients from RNS residues [49] is very similar to Newton's interpolation, where some kind of (modular) divided differences are performed. These coefficients, for an integer $x \in [0, M)$, are denoted by $\breve{\boldsymbol{x}}_{\mathcal{B}} = (\breve{x}_1, \ldots, \breve{x}_n)$. They are obtained from residues $\boldsymbol{x}_{\mathcal{B}}$ by inverting the following formula for each $m_i \in \mathcal{B}$:

$$x = \breve{x}_1 + \breve{x}_2 m_1 + \ldots + \breve{x}_n m_1 m_2 \ldots m_{n-1} \ . \quad (1)$$

Consequently, $\breve{x}_1 = x_1$, $\breve{x}_2 = |(x_2 - \breve{x}_1)m_1^{-1}|_{m_2}$, and so on and so forth. Therefore, it is a sequential process which is contrary to the parallel nature of RNS arithmetic. Thus, it is a burdensome operation.

MRS is a positional numeral system. Hence, once the MRS coefficients of two numbers are known, comparing them is straightforward. $\breve{x}_1$ is the least significant coefficient, while $\breve{x}_n$ is the most significant one.

This conversion between RNS and MRS enables a base extension $\texttt{Bex}_{\texttt{mrs}}$ towards $\mathcal{B}'$. It simply consists in computing (1) in $\mathcal{B}'$. Despite the bad performance of such extension, it presents the advantage of providing an exact extended value, i.e. $\texttt{Bex}_{\texttt{mrs}}(\boldsymbol{x}_{\mathcal{B}}, \mathcal{B}, \mathcal{B}')$ always gives the residues of $|x|_M$ in $\mathcal{B}'$.

Montgomery Reduction with MRS Extensions: As previously noticed, $\texttt{Bex}_{\texttt{mrs}}$ enables to get an exact extension. It means that λ in Alg. 1 can be set to 0. In other words, the integer represented by $\boldsymbol{s}_{\mathcal{B}'}$ at line 4 satisfies $s < 2p$. Furthermore, using $\texttt{Bex}_{\texttt{mrs}}$ for the second extension enables a comparison to p in order to get a full reduction. This comparison is simply performed with the precomputed MRS coefficients of p. The subsequent variant of RNS reduction really sticks to the original Montgomery reduction with respect to Montgomery factor M. However, using two of such extension is far from being efficient.

Next, another kind of extension and its use within RNS reduction are introduced. Despite a better efficiency, these extensions introduce possible "overflows" which have to be cautiously handled.

3.2 Extensions Based on the CRT

The classical constructive proof of Chinese Remainder Theorem is based on the following formula, which gives a way to compute $x \in [0, M)$ from the residues $\boldsymbol{x}_{\mathcal{B}}$:

$$x = \sum_{i=1}^{n} |x_i M_i^{-1}|_{m_i} M_i \bmod M = \sum_{i=1}^{n} \tilde{x}_i M_i - \kappa M \ . \tag{2}$$

In (2), M_i stands for $\frac{M}{m_i}$, and $\tilde{x}_i$ for $|x_i M_i^{-1}|_{m_i}$. The integer κ lies in $[0, n-1]$. It is related to the reduction of the sum modulo M. Formally, it is given by the following formula:

$$\kappa = \left\lfloor \tfrac{1}{M} \sum_{i=1}^{n} \tilde{x}_i M_i \right\rfloor = \left\lfloor \sum_{i=1}^{n} \tfrac{\tilde{x}_i}{m_i} \right\rfloor \ . \tag{3}$$

Computing κ (i.e. reducing modulo M in (2)) is an issue for getting an exact base extension procedure from (2). It is challenging to compute it without leaving RNS representation. Several base extension procedures have been proposed so far. Their interest depends on the context.

3.2.1 Fast CRT-Based Extension

Within an appropriate context, computing κ may not be necessary at all. In this case, the extension denoted by $\texttt{Bex}_{\texttt{crt}}$ is a simple matrix-vector product.

$$\begin{aligned} \texttt{Bex}_{\texttt{crt}}(\boldsymbol{x}_{\mathcal{B}}, \mathcal{B}, \mathcal{B}') &= \left(\textstyle\sum_{i=1}^{n} \tilde{x}_i |M_i|_{m'} \bmod m'\right)_{m' \in \mathcal{B}'} \\ &= \left(|x + \kappa M|_{m'}\right)_{m' \in \mathcal{B}'}, \kappa \in [0, n-1] \ . \end{aligned} \tag{4}$$

This extension can be used for implementing Bex1. It provides $\widehat{q} = q + \kappa M < nM$ (i.e. $\lambda = n-1$), because, by definition of q, one has $q < M$. So, $\boldsymbol{s}_{\mathcal{B}'}$ at line 4 satisfies $s < (n+1)p$. Comparatively to the previous approach with MRS-based extension where one is able to get $s < 2p$, the result might need to be further reduced modulo p. However, the efficiency is improved a lot. Indeed, the extension remains completely and easily parallelizable.

Next, the problem is that such base extension cannot be used for Bex2 which has to be exact. It would not make sense to obtain the residues of an integer like $s + \kappa M'$ in base $\mathcal{B}$. Instead of using $\texttt{Bex}_{\texttt{mrs}}$ in this context, Shenoy and Kumaresan have proposed an efficient way to compute the integer κ related to $\boldsymbol{s}_{\mathcal{B}'}$ [47]. The idea is to add an extra redundant modulus besides $\mathcal{B}'$, denoted by m_{sk}, and to invert (2) modulo m_{sk}. This requires to know $x \bmod m_{sk}$. Then, by having $m_{sk} \geqslant n > \kappa$, it follows that

$$|(\textstyle\sum_{i=1}^{n} |x_i M_i^{-1}|_{m_i} M_i - x) M^{-1}|_{m_{sk}} = |\kappa|_{m_{sk}} = \kappa \ . \tag{5}$$

This approach cannot be used for extending q. Indeed, q is computed modulo M. So, there is no known efficient way to compute $q \bmod m_{sk}$. However, $\texttt{Bex}_{\texttt{sk}}$ (which is basically $\texttt{Bex}_{\texttt{crt}}$ coupled with computation of (5)) can be used to extend s towards $\mathcal{B}'$. This extension is exact and remains more efficient than an MRS-based extension. The reduction procedure is depicted by Alg. 2.

Algorithm 2 $\texttt{FastRnsMR}(\boldsymbol{x}_{(\mathcal{B},\mathcal{B}',m_{sk})}, p, \mathcal{B}, \mathcal{B}', m_{sk})$

Require: coprime RNS bases $\mathcal{B}$, $\mathcal{B}'$, m_{sk}, with $x < Mp$, $(n+1)p < M'$, $m_{sk} \geqslant n'$.
Ensure: $\boldsymbol{s}_{(\mathcal{B},\mathcal{B}',m_{sk})}$, $s \equiv xM^{-1} \bmod p$, $s < (n+1)p$.

1: $\boldsymbol{q}_{\mathcal{B}} \leftarrow \left|-xp^{-1}\right|_M$ ▷ $\|$ in $\mathcal{B}$
2: $\widehat{\boldsymbol{q}}_{(\mathcal{B}',m_{sk})} \leftarrow \texttt{Bex}_{\texttt{crt}}(\boldsymbol{q}_{\mathcal{B}}, \mathcal{B}, (\mathcal{B}'; m_{sk}))$ ▷ $\widehat{q} < nM$
3: $\boldsymbol{t}_{(\mathcal{B}',m_{sk})} \leftarrow |x + \widehat{q}p|_{m_{sk}M'}$ ▷ $\|$ in $(\mathcal{B}', m_{sk})$
4: $\boldsymbol{s}_{(\mathcal{B}',m_{sk})} \leftarrow \left|tM^{-1}\right|_{m_{sk}M'}$ ▷ $\|$ in $(\mathcal{B}', m_{sk})$
5: $\widehat{\boldsymbol{s}}_{(\mathcal{B},m_{sk})} \leftarrow \texttt{Bex}_{\texttt{crt}}(\boldsymbol{s}_{\mathcal{B}'}, \mathcal{B}', (\mathcal{B}, m_{sk}))$
6: $\kappa \leftarrow |(\widehat{s}_{m_{sk}} - s_{m_{sk}}) \times (M')^{-1}|_{m_{sk}}$
7: $\boldsymbol{s}_{\mathcal{B}} \leftarrow \widehat{\boldsymbol{s}}_{\mathcal{B}} - \kappa |M'|_M$ ▷ $\|$ in $\mathcal{B}$
8: **return** $\boldsymbol{s}_{(\mathcal{B},\mathcal{B}',m_{sk})}$

3.2.2 Towards an Exact Reduction Modulo p

Algorithm 3 $\texttt{ExactRnsMR}(\boldsymbol{x}_{(\mathcal{B},\mathcal{B}',\tilde{m})}, p, \mathcal{B}, \mathcal{B}')$

Require: coprime RNS bases $\mathcal{B}$, $\mathcal{B}'$, $\tilde{m}$, with $x < Mp$, $2p < M'$, $\tilde{m} \geqslant n+1$.
Ensure: $\boldsymbol{s}_{(\mathcal{B},\mathcal{B}',\tilde{m})}$, $s = x(\tilde{m}M)^{-1} \bmod p$ $(s \in [0,p))$.
1: $\boldsymbol{q}_{\mathcal{B}} \leftarrow \left|-xp^{-1}\right|_M$ ▷ $\|$ in $\mathcal{B}$
2: $\widehat{\boldsymbol{q}}_{(\mathcal{B}',\tilde{m})} \leftarrow \texttt{Bex}_{\texttt{crt}}(\boldsymbol{q}_{\mathcal{B}}, \mathcal{B}, (\mathcal{B}', \tilde{m}))$ ▷ $\widehat{q} < nM$
3: $\boldsymbol{t}_{(\mathcal{B}',\tilde{m})} \leftarrow |x + \widehat{q}p|_{\tilde{m}M'}$ ▷ $\|$ in $(\mathcal{B}', \tilde{m})$
4: $\boldsymbol{s}_{(\mathcal{B}',\tilde{m})} \leftarrow \left|tM^{-1}\right|_{\tilde{m}M'}$ ▷ $\|$ in $(\mathcal{B}', \tilde{m})$
5: $\tilde{q} \leftarrow |-sp^{-1}|_{\tilde{m}}$
6: $\boldsymbol{s}_{\mathcal{B}'} \leftarrow |(s+\tilde{q}p)\tilde{m}^{-1}|_{M'}$ ▷ $\|$ in $\mathcal{B}'$
7: $(\boldsymbol{s}_{\mathcal{B}}, \check{\boldsymbol{s}}_{\mathcal{B}'}) \leftarrow \texttt{Bex}_{\texttt{mrs}}(\boldsymbol{s}_{\mathcal{B}'}, \mathcal{B}', (\mathcal{B}, \tilde{m}))$
8: **if** $\check{\boldsymbol{s}}_{\mathcal{B}'} \geqslant \check{\boldsymbol{p}}_{\mathcal{B}'}$ **then**
9: $\quad \boldsymbol{s}_{(\mathcal{B},\mathcal{B}',\tilde{m})} \leftarrow \boldsymbol{s}_{(\mathcal{B},\mathcal{B}',\tilde{m})} - \boldsymbol{p}_{(\mathcal{B},\mathcal{B}',\tilde{m})}$
10: **end if**
11: **return** $\boldsymbol{s}_{(\mathcal{B},\mathcal{B}',\tilde{m})}$

Within a context where it is required to get a complete reduction modulo p (for instance at the end of a modular exponentiation), there are some final comparisons with p to perform. In such a case, an MRS-based extension can be used. In order to have only one comparison to perform despite the use of $\texttt{Bex}_{\texttt{crt}}$ as first extension, an efficient solution for reducing s from $[0,(n+1)p)$ to $[0,2p)$ has been proposed in [11].

The principle is to use a second small Montgomery reduction on s within base $\mathcal{B}'$. To achieve it, an extra modulus $\tilde{m}$ is introduced, besides $\mathcal{B}'$. If $\tilde{m} \geqslant n+1$, then the reduction allows to obtain a result in $[0,2p)$ as expected. Moreover, when $\tilde{m}$ is smaller than any modulus of $\mathcal{B}'$, a base extension from $\tilde{m}$ to $\mathcal{B}'$ is a simple duplication of the residue. Finally, an MRS-based extension enables to obtain an exact reduction modulo p through only one comparison. Alg. 3 implements this technique. It can be noticed that the Montgomery representation is now associated to the factor $\tilde{m}M$.

3.2.3 Computational Efficiency of RNS vs. Multi-precision Modular Arithmetic

For implementing arithmetic in $\mathbb{F}_p$, it can be noticed that the three previously introduced versions of RNS Montgomery reduction require two bases $\mathcal{B}$ and $\mathcal{B}'$ with basically $M \sim M' \sim p$. Thus, both bases are usually considered to own n moduli each.

The advantage of RNS (comparatively to classical positional representation) is that the multiplication is linear in n. More precisely, computing $x \times y$ in $(\mathcal{B},\mathcal{B}')$ is achieved with $2n$ $\texttt{EMM}$, whereas computing $x \times y$ with $x, y < p$ basically requires n^2 $\texttt{EM}$.[2]

Asymptotically, the cost of any kind of CRT-based extension is $(n^2+\mathcal{O}(n))$ $\texttt{EMM}$. Thus, an (efficient) RNS Montgomery reduction represents $(2n^2+\mathcal{O}(n))$ $\texttt{EMM}$. On the other side, such reduction, when performed in multi-precision, requires $(n^2+\mathcal{O}(n))$ $\texttt{EM}$, and it is therefore a bit more efficient. Consequently, the RNS can noticeably enhance the efficiency of finite field computations by using lazy reduction patterns, as often as possible, in order to take advantage of its efficient multiplication by minimizing the number of its costly reduction. Concretely, using RNS and a lazy reduction to compute $\sum_{i=1}^{k} a_i b_i \bmod p$ would enable a cost reduction from $((k+1)n^2+\mathcal{O}(\log(k)n))$ $\texttt{EM}$ to $(2n^2+\mathcal{O}(kn))$ $\texttt{EMM}$. With a classical multi-precision computation, the cost of multiplications is more important than the one of reduction (for example with pseudo-Mersenne modulo). Thus, the effect of a lazy computation is less significant than for an RNS approach.

4 Towards Efficient Embedded Hardware Implementations

4.1 Approximated Base Extensions

In prior sections, it has been pointed out that the base extensions represent the core computations of a modular reduction. Contrary to MRS-based extensions, the CRT-based methods have a better compatibility with RNS properties. They allow to keep a good parallelism all along the computations. This is at least true for the sum in (2). Actually, all the difficulty is concentrated in the computation of the integer κ which, in some variants of RNS modular reduction, is totally forgotten within the first extension. However, as the second extension has to be exact, this problem has to be overcome. In Alg. 2 for instance, it is necessary to introduce an extra redundant modulus to solve this issue.

Posch and Posch in 1993 [44] (in the context of a base extension), and Kawamura et al. in 2000 [33] (in the context of RNS Montgomery reduction) adopted a different strategy, by suggesting to compute an approximation of κ. Such approximation is based on (3). The hardest thing to manage is the rational numbers $\frac{1}{m_i}$. Posch and Posch introduced a floating-point approach to deal with them. Then, Kawamura et al. used an approximation which allows to manage only small integer arithmetic. This is the approach described hereafter.

If r is the bit-size of the moduli, i.e. $2^{r-1} < m_i < 2^r$, then $\frac{1}{m_i}$ is approximated by $\frac{1}{2^r}$. That way, computing

[2] by considering a quadratic multiplication.

(3) can be achieved by summing the carries beyond 2^r of the sum $\sum_{i=1}^{n} \tilde{x}_i$. Kawamura et al. even go further by keeping only the h most significant bits of each coefficient of $\tilde{x}_i$. Hence, the final approximation is given by (6). This provides an lower bound. It is possible to correct it (in part) by adding a correcting coefficient α.

$$\begin{aligned}\tilde{\kappa}_\alpha &\leftarrow \lfloor \alpha + \sum_{i=1}^{n} \frac{\tilde{x}_i \gg (r-h)}{2^h} \rfloor, \alpha \in 2^{-h}\{0,\ldots,2^h-1\} \\ &= \left(\alpha 2^h + \sum_{i=1}^{n} (\tilde{x}_i \gg (r-h))\right) \gg h \ .\end{aligned} \tag{6}$$

The parameter α enables a correction of the approximation in certain cases. More precisely, it is possible to adjust the parameter h such that the error term $\Delta = \sum_{i=1}^{n} (\frac{\tilde{x}_i}{m_i} - \frac{\tilde{x}_i \gg (r-h)}{2^h})$ lies in $[0,1)$. This implies that $\tilde{\kappa}_0$ is in $\{\kappa-1, \kappa\}$. Hence, when a correction is required, one chooses $\alpha_{\mathtt{kw}} \in [\Delta, 1)$. However, such correction is not effective for every RNS number. For instance, if the value represented by $\boldsymbol{s}_{\mathcal{B}'}$ at line 4 of Alg. 1 satisfies $s < (1-\alpha_{\mathtt{kw}})M'$, then in this case the correction will be successful. Such correction is then not possible for q in $\mathcal{B}$, since it cannot be restricted to an interval like $[0,(1-\alpha)M)$ for a certain $\alpha \in [\Delta, 1)$. Thus, Kawamura et al.'s idea is to compute $\tilde{\kappa}_0$ for the first extension (i.e. without correction). Thus, the extension denoted by $\mathtt{Bex}_{\mathtt{kw}}(\boldsymbol{q}_{\mathcal{B}}, \mathcal{B}, \mathcal{B}', \alpha = 0)$ will produce $\widehat{q} \in \{q, q+M\}$ which furthermore satisfies $\widehat{q} < (1+\Delta)M$. However, this last overflow can be counterbalanced by having $x < (1-\Delta)Mp$.

Finally, a corrected and an uncorrected variant of $\mathtt{Bex}_{\mathtt{kw}}$ (i.e. when $\alpha = \alpha_{\mathtt{kw}}$ or $\alpha = 0$) can be used within RNS modular reduction so that the result is guaranteed to be in $[0, 2p)$, and that the second base extension is exact under certain conditions on the moduli of the RNS bases (see 4.3). The full procedure is given in Alg. 4.

Algorithm 4 $\mathtt{KWRnsMR}(\boldsymbol{x}_{(\mathcal{B},\mathcal{B}')}, p, \mathcal{B}, \mathcal{B}')$

Require: coprime RNS bases $\mathcal{B}$, $\mathcal{B}'$, $x < Mp(1-\Delta)$, $2p < (1-\alpha_{\mathtt{kw}})M'$ with $0 \leqslant \Delta \leqslant \alpha_{\mathtt{kw}} < 1$.
Ensure: $\boldsymbol{s}_{(\mathcal{B},\mathcal{B}')}$, $s < 2p$, $s \equiv x\mathcal{M}^{-1} \bmod p$.
1: $\boldsymbol{q}_{\mathcal{B}} \leftarrow \left|-xp^{-1}\right|_M$ ▷ $\|$ in $\mathcal{B}$
2: $\widehat{\boldsymbol{q}}_{\mathcal{B}'} \leftarrow \mathtt{Bex}_{\mathtt{kw}}(\boldsymbol{q}_{\mathcal{B}}, \mathcal{B}, \mathcal{B}', \alpha = 0)$ ▷ $\widehat{q} < (1+\Delta)M$
3: $\boldsymbol{t}_{\mathcal{B}'} \leftarrow |x + \widehat{q}p|_{M'}$ ▷ $\|$ in $\mathcal{B}'$
4: $\boldsymbol{s}_{\mathcal{B}'} \leftarrow \left|tM^{-1}\right|_{M'}$ ▷ $\|$ in $\mathcal{B}'$; $s < 2p < (1-\alpha_{\mathtt{kw}})M'$
5: $\boldsymbol{s}_{\mathcal{B}} \leftarrow \mathtt{Bex}_{\mathtt{kw}}(\boldsymbol{s}_{\mathcal{B}'}, \mathcal{B}', \mathcal{B}, \alpha = \alpha_{\mathtt{kw}})$ ▷ exact extension
6: **return** $\boldsymbol{s}_{(\mathcal{B},\mathcal{B}')}$

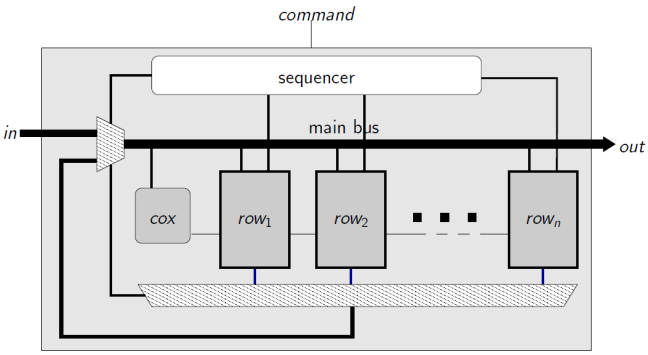


Fig. 1: General design of a Cox-Rower architecture.

4.2 Cox-Rower Architecture

In [33], an architecture dedicated to the implementation of Alg. 4 is proposed. It is depicted in Fig. 1. n arithmetic cells, called Rowers, run in parallel. They are specifically designed to computation patterns like $\sum_{i=1}^{n} a_i b_i \bmod m$ for a modulus m over r bits. Thus, any computation in $\mathcal{B}$ or $\mathcal{B}'$ can be run in parallel. Each Rower implements one channel $\mathbb{Z}_{m_i}$ of $\mathcal{B}$ and another one, $\mathbb{Z}_{m'_i}$, belonging to $\mathcal{B}'$. The total number of Rowers can be smaller than n, each arithmetic cell being in charge of more RNS channels, but at the cost of an increase of computation time. The architecture is then very flexible and easily scalable.

A specific unit called Cox computes (6) during each base extension. The second line of (6) shows that this computation can be executed by a simple h-bit adder. Each carry over 2^h is streamed throughout the Rowers to trigger a subtraction by M or M', according to the direction of current extension (i.e. $\mathcal{B} \to \mathcal{B}'$ or $\mathcal{B}' \to \mathcal{B}$).

4.3 Double Level of Montgomery Reduction

In the initial proposal of [33], the base extension using (6) implies choosing moduli close to 2^r in order to have a good approximation of coefficient κ. However, this consideration can be refined in order to provide a larger set of usable moduli.

On one hand, efficient reduction by moduli is usually insured by selecting pseudo Mersenne numbers ($m = 2^r - c$, $c < 2^{r/2}$). This kind of consideration leads to a set of moduli for which elementary RNS operations are efficient. On the other hand, in [17] the authors establish a tight bound over c allowing the use of Kawa-

mura's like extensions. This bound looks like $c \leqslant 2^r \frac{\alpha}{n}$ (where α is the correction term in (6)).

Consequently, if $2^r \frac{\alpha}{n} > 2^{r/2}$, the set of r-bit pseudo Mersenne moduli may be not sufficient to cover the need in r-bit moduli which are suitable for Kawamura's extension. Thus, a limitation can appear. For example, 2 RNS bases $\mathcal{B}$ and $\mathcal{B}'$ with 17-bit moduli can be found to implement Alg. 4 only for $\log_2(p)$ up to 506 [28]. It means that the use of a Cox-Rower architecture with ALU's designed for 17-bit pseudo Mersenne moduli cannot be used to implement computations in finite fields with characteristic composed of more than 506 bits.

Algorithm 5 Inner Montgomery modular reduction

Require: $m = 2^r - c$, $a \leqslant 2^r$, $b \leqslant m$.
Ensure: $s = (ab2^{-r}) \bmod m$.
1: $c \leftarrow ab = c_1 2^r + c_0$ $\quhd c < m2^r$
2: $q \leftarrow (c_0 \times |-m^{-1}|_{2^r}) \bmod 2^r$
3: $t \leftarrow c + q \times m = s_1 2^r$ $\quhd t < m2^{r+1}$
4: $s \leftarrow (t \gg r) = s_1$ $\quhd s \leqslant 2m, s \equiv ab2^{-r} \bmod m$
5: **if** $s \geqslant m$ **then**
6: $\quad s \leftarrow s - m$
7: **end if**
8: **return** s

A new strategy is proposed in [17]. The goal is to design new ALU's for implementing RNS Montgomery reduction Alg. 4, and which admits more general moduli than pseudo Mersenne, while guaranteeing an efficient inner reduction yet. Thus, the idea is to insert a second level of Montgomery reduction. The elementary reductions are performed by using Alg. 5. For instance, this new approach allows to extend the operating range of a 17-bit Cox-Rower architecture to $\log_2(p)$ up to 1118 bits [28]. In particular, RSA-1024 algorithm can be implemented in this architecture.

5 Applications of RNS to Asymmetric Cryptography

5.1 RSA

Using RNS for Modular Exponentiation: Nozaki et al. [40] used the Montgomery reduction described by Alg. 4 as the fundamental brick of an efficient RNS modular exponentiation. Then, they proposed to use it for implementing the RSA scheme by using a Cox-Rower architecture.

In such kind of approach, RNS representation is only involved to get an access to the modular exponentiation based on RNS Montgomery modular multiplier. Thus, some costly conversions between binary positional system and RNS have to be performed at the beginning and the end of a full procedure encryption+decryption or signature+verification.

RNS is not only useful for accelerating computations. Guillermin [30] integrated some leak resistant arithmetic in such RNS implementation of RSA.

Using RNS as the Main Representation: In [13], the authors proposed a clever way to get a full RNS implementation of RSA (Alg. 6 and 7). The modular multiplication is based on Alg. 2.

Two RNS bases $\mathcal{B}$ and $(\mathcal{B}', m_{sk})$ are required. In order to reach a high throughput, a method is proposed to avoid any conversion from binary integers to RNS. If the moduli are over r bits, any string of $n \times (r-1)$ bits is seen as a set of n residues in $\mathcal{B}$. The integers represented by these bit strings are smaller than the moduli of $\mathcal{B}$ so that any reduction modulo a modulus m_i is not necessary. Thus, these residues represent a certain number x in $[0, M)$. The corresponding residues in $(\mathcal{B}', m_{sk})$ are obtained through a MRS-based extension.

Except this initial MRS-based extension, the whole scheme (encryption+decryption) is completely implemented in RNS representation. To achieve that, the problem of having an incomplete reduction modulo N ($= pq$, the RSA modulus) by using Montgomery reduction, and in particular Alg. 2, had to be solved.

With public key $(a, N = p \times q)$, an encryption of x provides $y = x^a M \bmod N + \alpha N$ with α an integer in $[0, n]$ (the factor M is due to the Montgomery representation). In this case, the incomplete reduction is not a problem. Next, by applying the same modular exponentiation algorithm with secret key b, the decryption would provide $y^b \bmod N + \alpha' N = xM \bmod N + \alpha' N$. Making a last Montgomery modular multiplication by 1 enables to get rid of Montgomery representation, then providing $x \bmod N + \gamma N = x + \gamma N$, still with γ an integer in $[0, n]$. Therefore, the challenge was to correct γ without leaving RNS representation.

The solution requires to reduce a bit the size of input string, but it allows to avoid any comparison (and then MRS-base extensions) at the end of the decryption. The idea is to get only $n-1$ packets of $r-1$ bits, denoted by $(x_1, \ldots, x_{n-1})$, and to set the last residue to 0, i.e. $x_n = 0$. That way, x is now an integer in $[0, M)$ and it is a multiple of m_n. This information allows us to efficiently recover γ. Given adequate conditions on M relatively to N, the nullity of $x_n (= x \bmod m_n)$ indicates that we recover the original message. Otherwise, $\gamma \neq 0$ and it can be recovered by using the residue x_n.

The function $\texttt{RnsModExp}(\boldsymbol{x}_{(\mathcal{B},\mathcal{B}'_{sk})}, e)$ involved in Alg. 6 and 7 is a modular exponentiation with exponent e.

Algorithm 6 $\texttt{Enc}_{\texttt{RSA_RNS}}(x_1, x_2, \ldots, x_{n-1})$

Require: $(N = p \times q, a)$ RSA pub. key; $\mathcal{B}, \mathcal{B}'_{sk} = (\mathcal{B}', m_{sk})$, m_{sk} coprime with the m_i's, $(n+2)^2 N < M \leqslant (m_n - (n+2))N$.

1: $\boldsymbol{x}_{(\mathcal{B},\mathcal{B}'_{sk})} \leftarrow \texttt{Bex}_{\texttt{mrs}}((x_1, \ldots, x_{n-1}, 0)_{\mathcal{B}}, \mathcal{B}, \mathcal{B}'_{sk})$
2: $\boldsymbol{c}_{(\mathcal{B},\mathcal{B}'_{sk})} \leftarrow \texttt{RnsModExp}(\boldsymbol{x}_{(\mathcal{B},\mathcal{B}'_{sk})}, a) = |x^a M|_N + \alpha N$
3: **return** $\boldsymbol{c}_{(\mathcal{B},\mathcal{B}'_{sk})}$

Algorithm 7 $\texttt{Dec}_{\texttt{RSA_RNS}}(\boldsymbol{c}_{(\mathcal{B},\mathcal{B}'_{sk})})$

Require: b RSA secret key.

1: $\boldsymbol{x'}_{(\mathcal{B},\mathcal{B}'_{sk})} \leftarrow \texttt{RnsModExp}(\boldsymbol{c}_{(\mathcal{B},\mathcal{B}'_{sk})}, b) = |xM|_N + \alpha' N$
2: $\boldsymbol{x}_{\mathcal{B}} \leftarrow \texttt{FastRnsMR}(\boldsymbol{x'}_{(\mathcal{B},\mathcal{B}'_{sk})}) = x + \gamma N$
3: $t \leftarrow |-x_n N^{-1}|_{m_n}$
4: **if** $t \geqslant m_n - (n+2)$ **then**
5: $\quad t \leftarrow t - m_n$
6: **end if**
7: $\boldsymbol{x}_{\mathcal{B}} \leftarrow \boldsymbol{x}_{\mathcal{B}} + tN$
8: **return** $(x_1, \ldots x_{n-1})$

It uses the RNS Montgomery modular multiplication as a basic step. Then, as for a classical positional approach the complexity of the exponentiation is related to the exponentiation algorithm used (e.g. Montgomery ladder, etc.) and to the way the exponent e is encoded (binary, NAF (non adjacent form), etc.).

5.2 Elliptic Curve Cryptography and Pairings

5.2.1 Fast Implementations of ECC

One of the first efficient ECC implementations was done by N. Guillermin [29]. He optimized an hardware architecture to compute scalar multiplications of points of elliptic curves defined over a prime field $\mathbb{F}_p$. The RNS representation is used to speed up the computation thanks to a wide parallelization. In his approach, the conversion algorithms are optimized as much as possible by smartly precomputing all the values involved in constant known parameters. An implementation on Altera FPGA for some elliptic curves defined over different finite fields for classical cryptographic security level was proposed. This implementation uses resistant algorithms against SPA (simple power analysis) attacks [38] together with a leak resistant arithmetic (cf. Sec. 6.1) which protects against DPA (differential power analysis) attacks [14]. The inherent parallelism of elliptic curve operations allows to easily fill a large pipelines (6 stages). Furthermore, this architecture supports a high clock frequency for the different curve sizes.

The same strategy can be used for GPU or multi-core CPU implementations. In [1], the authors proposed an implementation on an Nvidia 285 GTX GPU. It deals with 224-bit underlying finite field $\mathbb{F}_p$. The experimental results showed a maximum throughput of 9827 EC point multiplications per second and minimum latency of 29.2ms, which was in 2012 one of the best approaches in terms of latency and throughput. They also analysed the implementation on multi-core CPU (with 4 cores) by programming the algorithms with OpenCL. Four cores were not enough to provide a sufficient parallelism for a RNS approach but due to the burdensome memory transfers between the CPU and the GPU, it can be more interesting to execute the whole decryption procedure on CPU. Thus, with the increasing number of cores in current hardware platforms, the approach could be more efficient on CPU.

Another implementation was done on FPGA in [26]. It improves Guillermin's approach. The main idea is to use three or four moduli of the form $2^k - 2^{t_i} - 1$, based on the study on RNS bases given in [15], and six- and four-stage of pipeline.

5.2.2 How to Accelerate ECC and Pairing

The main idea of this part was first given for ECC in [8]. It comes from the remark that, in RNS, additions and multiplications are very cheap in time and area. More precisely, this can be done, in time, at the cost of a machine word operation and, in area, with a linear function of the size of the prime p characterizing the finite field. The drawback of RNS arithmetic comes from the modular reduction which requires two base extensions. So, interesting formulas for RNS arithmetic are modular sum of products $\sum_{i=0}^{t-1} A_i \times B_i \bmod p$, where the t multiplications-additions can be done before a single final modular reduction. To apply this remark in the context of ECC, some transformations must be done in the addition and doubling point expressions.

In [7], authors gave a survey of a method dealing with new formulas which are well adapted to the use of the RNS arithmetic on elliptic curves. They optimized formulas for basic operations arising in leak resistant arithmetic on elliptic curves (unified addition, Montgomery ladder) in order to minimize the number of modular reductions. These two points were developed in this paper where, by reformulating addition formulae on elliptic curves, some solutions which are up to 30% better than the classical approaches, were proposed. They dealt in particular with different addition formulas for elliptic curves: Hessian form [32], Jacobi quartic [25, 35], short Weierstrass form [22, 23], Montgomery form [31, 38].

Roughly speaking, the cost of a multiplication in a classical representation can be compared to the one of a modular reduction in RNS. Reciprocally, for some given p, modular reduction is easy in a classical representation and can be estimated similar to a multiplication in RNS. When looking at the Hessian form, Jacobi quartic or short Weierstrass form, the original point addition formulas are well adapted to RNS.

Curves	Mult	Mod Red
Hessian form	12	9
Jacobi quartic	12	10
Weierstrass form	18	14

For the Montgomery form, it is less clear, because a modular multiplication is needed at each step of the Montgomery ladder algorithm. But in [7] the authors showed that it is possible to use a Montgomery ladder strategy on a short Weierstrass form by using some results coming from the theory of Kummer varieties. Thus, the formulas of addition and doubling have been transformed. They obtained, for each step of the Montgomery ladder algorithm for curves with small coefficients, 12 modular reductions compared to 17 multiplications in classical representation.

The algorithms used for operating pairings are particularly welcome for this kind of "lazy reduction" approach. In [24], two FPGA-based high speed pairing designs using RNS and lazy reduction were presented. Lazy reduction in pairing computation was introduced for classical arithmetic by Scott [46] and then generalized by Aranha et al. in [2]. The speed of pairing computation in hardware is largely increased for hardware implementations of optimal ate pairing at 126-bit security level in 0.573 ms, which is 2 times faster than all previous hardware implementations at the same security level. Moreover, the first hardware pairing implementation at 192-bit security level was also reported. In [51], the authors combined lazy reduction, Karatsuba like formulas and optimal pipeline scheduling.

5.3 Lattices and Rounding-off

Besides cryptosystems based on finite field computation, the RNS Montgomery multiplier has been used in the context of lattices, where many computations involve operations on matrices and vectors. Thus, RNS is a natural candidate for accelerating computations.

In [11, 12], solutions have been proposed for adapting Babai's rounding-off [3] algorithm in RNS. Given a (full-rank integer) lattice $\mathcal{L} = \boldsymbol{r}_1\mathbb{Z} \oplus \dots \boldsymbol{r}_\ell\mathbb{Z}$ in $\mathbb{R}^\ell$ described by the (integer) matrix basis $\boldsymbol{R}$ (whose rows are the $\boldsymbol{r}_i$'s), and given a target vector $\boldsymbol{c}$, this operation allows to compute a vector of the lattice $\mathcal{L}$ close to $\boldsymbol{c}$. The principle is to express $\boldsymbol{c}$ in the basis $\{\boldsymbol{r}_1, \dots, \boldsymbol{r}_\ell\}$, to round-off the coordinates to the nearest integers, and to come back in canonical basis of $\mathbb{R}^\ell$. To do it, one computes $\boldsymbol{R}\lfloor \boldsymbol{R}^{-1}\boldsymbol{c}\rceil$.

The problem of doing this operation into RNS is then related to the computation of a rounding-off in RNS. In others words, given two integers a, b $(b > 0)$, the problem is to efficiently and exactly compute $\lfloor \frac{a}{b} \rceil$, where b is known in advance. The initial idea in [11] is to rewrite the rounding-off as a formula involving only integers and exact division:

$$\left\lfloor \frac{a}{b} \right\rceil = \left\lfloor \frac{a}{b} + \frac{1}{2} \right\rfloor = \frac{2a + b - |2a + b|_{2b}}{2b} \ .$$

The exact division is doable in RNS. So, the bottleneck is the computation of the exact modular reduction $|2a + b|_{2b}$. This can be performed by using the RNS Montgomery modular reduction operator.

Since the reduction must be complete, a first solution in [11] consists in using Alg. 3. But the use of a RNS to MRS conversion negatively impacts the performance. In [12], another strategy is suggested. It allows to compute the reduction by using the fastest modular reduction algorithm (Alg. 2). The possible error due to an incomplete reduction is corrected by introducing an integer γ. The key idea is based on the equality

$$\lfloor \gamma \tfrac{a}{b} \rceil = \gamma \lfloor \tfrac{a}{b} \rceil + \lfloor \gamma \tfrac{[a]_b}{b} \rceil$$

where $[a]_b$ is the remainder of a modulo b in $[-b/2, b/2)$. And the quantity $\lfloor \gamma \frac{a}{b} \rceil$ is approximated by a flooring:

$$\lfloor \gamma \tfrac{a}{b} \rceil = \lfloor \gamma \tfrac{a}{b} \rfloor - e, e \in \{0, 1\} \ .$$

The flooring is performed in RNS through the formula $\frac{\gamma a - |\gamma a|_b}{b}$. Then, $|\gamma a|_b$ is given by Alg. 2, plus a possible error $e' < n$. Finally, one can easily compute, in RNS, $\gamma \lfloor \frac{a}{b} \rceil + E$, with $E = \lfloor \gamma \frac{[a]_b}{b} \rceil - e - e'$.

At this point, correcting E is easily done by using the residue modulo γ if γ is large enough so that $\gamma/2 > E$. Under this condition, one can access $[E]_\gamma = E$, and correct the rounding. A last exact division by γ allows to recover $\lfloor \frac{a}{b} \rceil$. The procedure is given by Alg. 8.

For the matter of efficiency, γ should not be larger than other moduli, so that the computations for the purpose of correction, which are made modulo γ, are performed with a cheap cost. Formally, the size of γ depends on n (i.e. the cardinality of main base RNS $\mathcal{B}$) and on the gap $\varepsilon = \frac{1}{2} - |\frac{[a]_b}{b}|$. These values should verify $\gamma\varepsilon \geqslant n + \frac{1}{2}$. Consequently, such correction technique is optimal when $\frac{1}{2} - |\frac{[a]_b}{b}| \sim n2^{-r}$.

Algorithm 8 `ExactRNSRoundOff`(a, b)

Require: residues $\boldsymbol{a}_{(\mathcal{B},\mathcal{B}'_{sk})}$ of a in $(\mathcal{B}, \mathcal{B}'_{sk})$ (b fixed)
Ensure: residues $\tilde{\boldsymbol{s}}_{(\mathcal{B},\mathcal{B}'_{sk})}$ of $\lfloor \frac{a}{b} \rceil$ in $(\mathcal{B}, \mathcal{B}'_{sk})$
1: $\boldsymbol{r}_{(\mathcal{B},\mathcal{B}'_{sk},\gamma)} \leftarrow \texttt{FastRnsMR}(\gamma|M|_b a, b, \mathcal{B}, (\mathcal{B}'_{sk}, \gamma))$
2: $\boldsymbol{s}_{(\mathcal{B},\mathcal{B}'_{sk},\gamma)} \leftarrow (\gamma \boldsymbol{a}_{(\mathcal{B},\mathcal{B}'_{sk},\gamma)} - \boldsymbol{r}_{(\mathcal{B},\mathcal{B}'_{sk},\gamma)}) \cdot b^{-1}$ ▷ $\|$ in $(\mathcal{B}, \mathcal{B}'_{sk}, \gamma)$
3: $\tilde{s}_\gamma \leftarrow [s_\gamma]_\gamma$ ▷ "centered" residue in $[-\gamma/2, \gamma/2)$
4: $\tilde{\boldsymbol{s}}_{(\mathcal{B},\mathcal{B}'_{sk})} \leftarrow (\boldsymbol{s}_{(\mathcal{B},\mathcal{B}'_{sk})} - \tilde{s}_\gamma) \cdot \gamma^{-1}$ ▷ $\|$ in $(\mathcal{B}, \mathcal{B}'_{sk})$
5: **return** $\tilde{\boldsymbol{s}}_{(\mathcal{B},\mathcal{B}'_{sk})}$

6 RNS Montgomery Reduction Against Side-channel Analysis and Fault Attacks

Beyond computational efficiency, the alliance RNS and Montgomery reduction has been fruitful for the emergence of practical solutions for defeating side-channel analysis and fault attacks. In particular, RNS Montgomery reduction is a core operation for creating a leak resistant arithmetic, and an arithmetic robust to fault attacks too.

6.1 Leak Resistant Arithmetic

RNS Montgomery multiplier owns an interesting property for the purpose of protection against side-channel attacks, namely the Montgomery representation. Given the main base $\mathcal{B}$ involved in RNS reduction, any data to be handled in the context of computations in $\mathbb{F}_p$ for instance has to be pre-multiplied by $|M|_p$.

In order to counteract side-channel attacks such as analysis of electromagnetic leakage, in [14] the authors suggested to randomize the choice of $\mathcal{B}_\alpha$. Thus, the factor $|M|_p$ would act as a random mask. In practice, the base $\mathcal{B}_\alpha$ is chosen by picking up n moduli in a pool of $2n$ pairwise coprime integers $\{m_1, \ldots, m_{2n}\}$. The remaining moduli are used to create $\mathcal{B}'_\alpha$. By proceeding like that, $\binom{2n}{n} \sim \frac{2^{2n}}{\sqrt{\pi n}}$ masks are available. To make this principle more interesting, the authors had to find an efficient way to re-randomize $\mathcal{B}_\alpha$ on-the-fly during a modular exponentiation.

Let $\mathcal{M}$ be the product of all moduli $\prod_{i=1}^{2n} m_i$. Let M_α denote a product choice of n random moduli for $\mathcal{B}_\alpha$, $M'_\alpha = \mathcal{M}/M_\alpha$ be the product of the remaining n moduli of $\mathcal{B}'_\alpha$, and ditto with M_β, M'_β corresponding to another random choice. The challenge was to switch, efficiently, between the RNS representation of $|xM_\alpha|_p$ in $(\mathcal{B}_\alpha, \mathcal{B}'_\alpha)$ and the one of $|xM_\beta|_p$ in $(\mathcal{B}_\beta, \mathcal{B}'_\beta)$. The technique relies on the fact that $\mathcal{M} = M_\alpha M'_\alpha = M_\beta M'_\beta$.

Given $|xM_\alpha|_p$ in $(\mathcal{B}_\alpha, \mathcal{B}'_\alpha)$, the first step is a Montgomery multiplication by $\mathcal{M}$ with $\mathcal{B}_\alpha$ as the main base, which provides a value congruent to $|xM_\alpha\mathcal{M}M_\alpha^{-1}|_p = |x\mathcal{M}|_p$. The second step consists in applying a Montgomery reduction with $\mathcal{B}'_\beta$ as the main base. Thus, this enables to obtain $|x\mathcal{M}(M'_\beta)^{-1}|_p = |xM_\beta|_p$.

This leak resistant arithmetic has been optimized, implemented and tested in practice in several works [30, 39, 42, 43]. It contributes to make RNS, together with Montgomery reduction, a candidate for secure and efficient finite field arithmetic.

Algorithm 9 On-the-fly RNS Montgomery representation switching

Require: a random couple of n-moduli bases $(\mathcal{B}_\alpha, \mathcal{B}'_\alpha)$, $(\mathcal{B}_\beta, \mathcal{B}'_\beta)$ among $\{m_1, \ldots, m_{2n}\}$, $\mathcal{M} = \prod_{i=1}^{2n} m_i$; residues of $|xM_\alpha|_p$ in $(\mathcal{B}_\alpha, \mathcal{B}'_\alpha)$.
Ensure: residues of $|xM_\beta|_p$ in $(\mathcal{B}_\beta, \mathcal{B}'_\beta)$.
1: $\boldsymbol{y}_{(\mathcal{B}_\alpha,\mathcal{B}'_\alpha)} \leftarrow \texttt{RnsMR}(|xM|_p \times \mathcal{M}, \mathcal{B}_\alpha, \mathcal{B}'_\alpha)$ ▷ $y \equiv x\mathcal{M} \bmod p$
2: $\boldsymbol{z}_{(\mathcal{B}'_\beta,\mathcal{B}_\beta)} \leftarrow \texttt{RnsMR}(y, \mathcal{B}'_\beta, \mathcal{B}_\beta)$ ▷ $z \equiv xM_\beta \bmod p$
3: **return** $\boldsymbol{z}_{(\mathcal{B}_\beta,\mathcal{B}'_\beta)}$

6.2 RNS Montgomery Reduction for Defeating Fault Attacks

Besides side-channel analysis, fault attacks are another serious threat to any device embedding a cryptographic primitive. One of the most famous attack, the Bellcore attack, was performed on CRT-RSA signature scheme [21], allowing to recover the RSA factorisation thanks to hardware faults. Since then, many fault attacks have been discovered against various schemes (ECC, pairings, etc [19, 41, 50]), and protecting embedded cryptosystems against faults is of the highest importance.

RNS is a good fit for providing a fault resistant arithmetic. This is because the information is divided up through the residues, which are processed in distinct hardware units (cf. Cox-Rower architecture). Furthermore, redundancy can be easily added by introducing extra moduli. For instance, adding k redundant moduli r_i to a base $\mathcal{B}$ enables to detect up to k faults as long as the redundant moduli are greater to the m_i's. The principle is very similar to error correcting code. Here, a codeword is a set of residues $(\boldsymbol{x}_\mathcal{B}, \boldsymbol{x}_\mathcal{R})$ ($\mathcal{R}$ is the redundant base), such that the integer x it represents lies in the "legitimate" range $[0, M)$. It means that there is a "consistency" between $\boldsymbol{x}_\mathcal{B}$ and $\boldsymbol{x}_\mathcal{R}$, or again that $\boldsymbol{x}_\mathcal{R}$ is a truly redundant information of $\boldsymbol{x}_\mathcal{B}$. The detection process consists in an exact base extension from $\mathcal{B}$ to $\mathcal{R}$ in order to check this consistency.

Such redundant RNS works as long as the consistency check is performed after "standard" RNS operations, i.e. parallel additions, subtractions, multiplications. Indeed, the independence of residues is required in order to avoid any propagation of a fault, and to preserve consistency for codewords. By consequence, the principle of redundant RNS Montgomery reduction was quite contradictory at a first sight.

In 2013 [9] and 2016 [10], it has been shown that RNS Montgomery reduction can be efficiently adapted to redundant RNS. For implementing such a redundant RNS Montgomery-based modular arithmetic, any kind of algorithm among Alg. 1, Alg. 2 or Alg. 4 can be used, making the approach flexible. In [10], Alg. 4 was preferred because an adequate architecture, based on Cox-Rower, was proposed.

In any kind of approach, it can be shown that the structure of the RNS Montgomery reduction algorithm allows to keep the consistency very straightforwardly by adding a redundant base $\mathcal{R}$ besides $\mathcal{B}$ and $\mathcal{B}'$. Hence, the second (exact) base extension can be used to perform the consistency check. During the whole modular reduction, the computations in $\mathcal{R}$ run completely in parallel (in particular, during the base extensions, $\mathcal{R}$ is not a part of the input base, but only of the output). Thus, as soon as enough area is available for $\mathcal{R}$ on the device, the computational time is not impacted by the fault detection features. The principle of an RNS Montgomery reduction is summarized in Alg. 10.

The fault detection process described here has also the advantage that it is compliant with a leak resistant arithmetic. Indeed, the on-the-fly switching between Montgomery representations (cf. Alg. 9) can be performed by using Alg. 10. The redundant base $\mathcal{R}$ stays the same all along the process.

Algorithm 10 $\texttt{RedundantRnsMR}(\boldsymbol{x}_{(\mathcal{B},\mathcal{B}',\mathcal{R})}, p, \mathcal{B}, \mathcal{B}', \mathcal{R})$

Require: coprime RNS bases $\mathcal{B}$, $\mathcal{B}'$, $\mathcal{R}$, with $r_j > m_i, m'_i$, $x < Mp$, $(2+\lambda)p < M'$ with $\lambda \geqslant 0$ related to `Bex1` (cf. Alg. 1).

Ensure: $\boldsymbol{s}_{(\mathcal{B},\mathcal{B}',\mathcal{R})}$, $s \equiv xM^{-1} \bmod p$ if no fault is detected.

1: $\boldsymbol{q}_{\mathcal{B}} \leftarrow \left|-xp^{-1}\right|_M$ ▷ $\|$ in $\mathcal{B}$
2: $\widehat{\boldsymbol{q}}_{\mathcal{B}'} \leftarrow \texttt{Bex1}(\boldsymbol{q}_{\mathcal{B}}, \mathcal{B}, (\mathcal{B}', \mathcal{R}))$
3: $\boldsymbol{t}_{(\mathcal{B}',\mathcal{R})} \leftarrow |x + \widehat{q}p|_{M'R}$ ▷ $\|$ in $(\mathcal{B}', \mathcal{R})$
4: $\boldsymbol{s}_{(\mathcal{B}',\mathcal{R})} \leftarrow \left|tM^{-1}\right|_{M'R}$ ▷ $\|$ in $(\mathcal{B}', \mathcal{R})$
5: $(\boldsymbol{s}_{\mathcal{B}}, \widehat{\boldsymbol{s}}_{\mathcal{R}}) \leftarrow \texttt{Bex2}(\boldsymbol{s}_{\mathcal{B}'}, \mathcal{B}', (\mathcal{B}, \mathcal{R}))$ ▷ exact extension
6: **if** $\boldsymbol{s}_{\mathcal{R}} \neq \widehat{\boldsymbol{s}}_{\mathcal{R}}$ **then**
7: **return** "`Corrupted data; abort`"
8: **else**
9: **return** $\boldsymbol{s}_{(\mathcal{B},\mathcal{B}',\mathcal{R})}$
10: **end if**

7 Conclusion

Peter L. Montgomery's reduction algorithm has offered a beautiful and useful tool for RNS arithmetic. Adapting it to this particular number system has been rich in terms of application. Initially, it was applied to the classical RSA cryptographic approach, but soon it was intensively used for ECC or Pairings.

Nowadays within the post-quantum era, some applications to lattice-based crypto-systems have been proposed. For instance, RNS are used for accelerating computations in algebraic structures which are useful for schemes based on the Ring-Learning With Errors problems [34, 36]. In a recent work, the RNS Montgomery reduction helps for designing a full RNS somewhat homomorphic encryption scheme [4]. To conclude, Montgomery's reduction is a powerful operator which is still at the heart of contemporary research works in the domain of applied cryptography.

References

1. S. Antao, J.-C. Bajard, and L. Sousa. RNS-Based Elliptic Curve Point Multiplication for Massive Parallel Architectures. *The Computer Journal*, 55(5), 2012.
2. D. Aranha, K. Karabina, P. Longa, C. H. Gebotys, and J. Lopez. Faster explicit formulas for computing pairings over ordinary curves. In *EUROCRYPT Advances in Cryptology*, vol. 6632 of *LNCS*, 2011.
3. L. Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1), 1986.
4. J.-C. Bajard, J. Eynard, A. Hasan, and V. Zucca. A Full RNS Variant of FV like Somewhat Homomorphic Encryption Schemes. In *SAC: The 23rd Conference on Selected Areas in Cryptography*, 2016.
5. J.-C. Bajard, L.-S. Didier, and P. Kornerup. An RNS Montgomery Modular Multiplication Algorithm. *IEEE Transactions on Computers*, 47(7), 1998.
6. J.-C. Bajard, L.-S. Didier, and P. Kornerup. Modular multiplication and base extensions in residue number systems. In *ARITH: 15th IEEE Symposium on Computer Arithmetic*, 2001.
7. J.-C. Bajard, S. Duquesne, and M. Ercegovac. Combining leak-resistant arithmetic for elliptic curves defined over $\mathbb{F}_p$. *Pub. Math. de Besançon. Algèbre et Théorie des Nombres*, 2013.
8. J.-C. Bajard, S. Duquesne, M. Ercegovac, and N. Meloni. Residue systems efficiency for modular products summation: Application to Elliptic Curves Cryptography. In *Proceedings of SPIE Optics & Photonics Symposium*, vol. 6313, 2006.
9. J.-C. Bajard, J. Eynard, and F. Gandino. Fault Detection in RNS Montgomery Modular Multiplication. In *ARITH: 21st IEEE Symposium on Computer Arithmetic*, 2013.
10. J.-C. Bajard, J. Eynard, and N. Merkiche. Multi-fault Attack Detection for RNS Cryptographic Architecture. In *ARITH: 23rd IEEE Symposium on Computer Arithmetic*, 2016.
11. J.-C. Bajard, J. Eynard, N. Merkiche, and T. Plantard. Babai Round-Off CVP method in RNS, Application to Lattice based cryptographic protocols. In *ISIC: IEEE Intern. Symposium on Integrated Circuits*, 2014.

12. J.-C. Bajard, J. Eynard, N. Merkiche, and T. Plantard. RNS Arithmetic Approach in Lattice-Based Cryptography: Accelerating the "Rounding-off" Core Procedure. In *ARITH: IEEE 22nd Symposium on Computer Arithmetic*, 2015.
13. J.-C. Bajard and L. Imbert. A full RNS implementation of RSA. *IEEE Transactions on Computers*, 53(6), 2004.
14. J.-C. Bajard, L. Imbert, P.Y. Liardet, and Y. Teglia. Leak Resistant Arithmetic. In *CHES: Cryptographic Hardware and Embedded Systems*, vol. 3156 of *LNCS*, 2004.
15. J.-C. Bajard, M. E. Kaihara, and T. Plantard. Selected RNS Bases for Modular Multiplication. In *ARITH: 19th IEEE Symposium on Computer Arithmetic*, 2009.
16. J.-C. Bajard, N. Meloni, and T. Plantard. Efficient RNS Bases for Cryptography. In *IMACS: Scientific Computation Applied Mathematics and Simulation*, 2005.
17. J.-C. Bajard and N. Merkiche. Double Level Montgomery Cox-Rower Architecture, New Bounds. In *CARDIS: 13th Smart Card Research and Advanced Application Conference*, vol. 8968 of *LNCS*, 2014.
18. P. Barrett. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In *Advances in Cryptology - CRYPTO*, 1987.
19. I. Biehl, B. Meyer, and V. Müller. Differential Fault Attacks on Elliptic Curve Cryptosystems. In *CRYPTO : Advances in Cryptology*, vol. 1880 of *LNCS*, 2000.
20. K. Bigou and A. Tisserand. Single Base Modular Multiplication for Efficient Hardware RNS Implementations of ECC. In *CHES: 17th International Workshop on Cryptographic Hardware and Embedded Systems*, vol. 9293 of *LNCS*, 2015.
21. D. Boneh, R.A. DeMillo, and R.J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In *EUROCRYPT: Intern. Conf. on the Theory and Application of Cryptographic Techniques*, vol. 1233 of *LNCS*, 1997.
22. E. Brier and M. Joye. Weierstraß Elliptic Curves and Side-Channel Attacks. In *PKC: 5th International Workshop on Practice and Theory in Public Key Cryptosystems*, vol. 2274 of *LNCS*, 2002.
23. E. Brier and M. Joye. Fast point multiplication on elliptic curves through isogenies. In *AAECC : 15th International Symposium, Applied Algebra, Algebraic Algorithms and Error-Correcting*, vol. 2643 of *LNCS*, 2003.
24. R.C.C. Cheung, S. Duquesne, J. Fan, N. Guillermin s, I. Verbauwhede, and G.X. Yao. FPGA Implementation of Pairings Using Residue Number System and Lazy Reduction. In *CHES: 13th International Conference on Cryptographic Hardware and Embedded Systems*, vol. 6917 of *LNCS*, 2011.
25. S. Duquesne. Improving the Arithmetic of Elliptic Curves in the Jacobi Model. *Inf. Process. Lett.*, 104(3), 2007.
26. M. Esmaeildoust, D. Schinianakis, H. Javashi, T. Stouraitis, and K. Navi. Efficient RNS Implementation of Elliptic Curve Point Multiplication Over GF(p). *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(8), 2013.
27. H.L. Garner. The Residue Number System. In *Western Joint Computer Conference*, IRE-AIEE-ACM '59 (Western), 1959.
28. B. Gérard, J. G. Kammerer, and N. Merkiche. Contributions to the Design of Residue Number System Architectures. In *ARITH: IEEE 22nd Symp. on Computer Arithmetic*, 2015.
29. N. Guillermin. A High Speed Coprocessor for Elliptic Curve Scalar Multiplications over $\mathbb{F}_p$. In *CHES: 12th International WorkshopCryptographic Hardware and Embedded Systems*, vol. 6225 of *LNCS*, 2010.
30. N. Guillermin. A Coprocessor for Secure and High Speed Modular Arithmetic. Technical report, Cryptology ePrint Archive, Report 2011/354, 2011.
31. I. Izu and T. Takagi. A Fast Parallel Elliptic Curve Multiplication Resistant against Side Channel Attacks. In *PKC: 5th International Workshop on Practice and Theory in Public Key Cryptosystems*, vol. 2274 of *LNCS*, 2002.
32. M. Joye and J.-J. Quisquater. Hessian Elliptic Curves and Side-Channel Attacks. In *CHES: Third International Workshop on Cryptographic Hardware and Embedded Systems*, vol. 2162 of *LNCS*, 2001.
33. S. Kawamura, M. Koike, F. Sano, and A. Shimbo. Cox-Rower Architecture for Fast Parallel Montgomery Multiplication. In *EUROCRYPT : 19th International Conference on Theory and Application of Cryptographic Techniques*, 2000.
34. T. Lepoint and M. Naehrig. A Comparison of the Homomorphic Encryption Schemes FV and YASHE. In *AFRICACRYPT*, vol. 8469 of *LNCS*, 2014.
35. P.-Y. Liardet and N. Smart. Preventing SPA/DPA in ECC Systems Using the Jacobi Form. In *CHES: Third International Workshop on Cryptographic Hardware and Embedded Systems*, vol. 2162 of *LNCS*, 2001.
36. C. Aguilar Melchor, J. Barrier, S. Guelton, A. Guinet, M.O. Killijian, and T. Lepoint. NFLlib: NTT-Based Fast Lattice Library. In *CT-RSA*, vol. 9610 of *LNCS*, pages 341–356, 2016.
37. P.L. Montgomery. Modular Multiplication without Trial Division. *Math. of Computation*, 44(170), 1985.
38. P.L. Montgomery. Speeding the Pollard and Elliptic Curve Methods of Factorization. *Mathematics of Computation*, 48(177), 1987.
39. C. Negre and G. Perin. Trade-Off Approaches for Leak Resistant Modular Arithmetic in RNS. In *ACISP: Information Security and Privacy: 20th Australasian Conference*, vol. 9144 of *LNCS*, 2015.
40. H. Nozaki, M. Motoyama, A. Shimbo, and S.i Kawamura. Implementation of RSA Algorithm Based on RNS Montgomery Multiplication. In *CHES: Third International Workshop Cryptographic Hardware and Embedded Systems*, vol. 2162 of *LNCS*, 2001.
41. D. Page and F. Vercauteren. A Fault Attack on Pairing-Based Cryptography. *IEEE Transactions on Computers*, 55(9), 2006.
42. G. Perin, L. Imbert, P. Maurine, and L. Torres. Vertical and horizontal correlation attacks on RNS-based exponentiations. *Journal of Cryptographic Engineering*, 5(3), 2015.
43. G. Perin, L. Imbert, L. Torres, and P. Maurine. Practical Analysis of RSA Countermeasures Against Side-Channel Electromagnetic Attacks. In *CARDIS : 12th International Conference Smart Card Research and Advanced Applications*, vol. 8419 of *LNCS*, 2013.
44. K.C. Posch and R. Posch. Base Extension Using a Convolution Sum in Residue Number Systems. *Computing*, 50(2), 1993.
45. K.C. Posch and R. Posch. Modulo Reduction in Residue Number Systems. *IEEE Transactions on Parallel and Distributed Systems*, 6(5), 1995.
46. M. Scott. Implementing Cryptographic Pairings. In *Pairing-Based Cryptography - Pairing 2007*, vol. 4575 of *LNCS*, 2007.
47. P. P. Shenoy and R. Kumaresan. Fast Base Extension Using a Redundant Modulus in RNS. *IEEE Transactions on Computers*, 38(2), 1989.
48. A. Svoboda and M. Valach. Operátorové obvody (Operational circuits). In *Strojena Zpracování Informací (Information Processing Machines)*, vol. 3. Sbornik, 1955.
49. N. Szabó and R. Tanaka. *Residue Arithmetic and its Applications to Computer Technology*. McGraw-Hill, 1967.
50. C. Whelan and M. Scott. The Importance of the Final Exponentiation in Pairings When Considering Fault Attacks. In *Pairing: Pairing-Based Cryptography*, vol. 4575 of *LNCS*, 2007.
51. G. X. Yao, J. Fan, R. C. C. Cheung, and I. Verbauwhede. Faster Pairing Coprocessor Architecture. In *Pairing: 5th International Conference Pairing-Based Cryptography*, vol. 7708 of *LNCS*, 2012.