Some Improvement on RNS Montgomery Modular Multiplication

Jean-Claude Bajard[a], Laurent-Stéphane Didier[b],
Peter Kornerup[c] and Fabien Rico[a]

[a] LIRMM, 161 Rue Ada, 34392 Montpellier Cedex 5

[b] Department of Computer Science, Université de Bretagne Occidentale,
Avenue Le Gorgeu, BP 809, 29285 Brest Cedex, France

[c] Department of Mathematics and Computer Science, Odense University,
Campusvej 55, DK-5230 Odense M, Denmark

ABSTRACT

In Residue Number Systems (RNS), an integer X is represented by its residues $\{x_0,...,x_{n-1}\}$ modulo a base of relatively prime numbers $\{m_0,...,m_{n-1}\}$. Thus a large number can be represented as a set of small integers. Addition and multiplication can be easily parallelized, there is no carry propagation. The time is reduced to the evaluation of these operations with small numbers. This representation is useful in cryptography and digital signal processing.

Furthermore, in these two domains, modular multiplication ($A \times B \bmod N$) is frequently used. So, in 1998, we have presented in IEEE journal of transactions on computers,[20] a new modular multiplication algorithm in RNS. This algorithm is based on the Montgomery algorithm, using the associated Mixed Radix representation, for the weighted digits. It was the first algorithm of this type.

In this paper, we present two remarks. First, if we develop the different expressions due to the algorithm, we obtain some mathematical simplifications that allow us to suppress some Mixed Radix occurrence in the basic iteration simply with a new initialization of our variables. Thus, in this new version, the complexity of each basic iteration, becomes equivalent to two products of small integers instead of three.

The second remark is that, most of the time, modular multiplications are done with the same modulo N. We can precompute some values and reduce the complexity of each basic iteration to one multiplication of two small integers. Thus, the basic iteration is three time faster, and the global computation, due to the initialization, is 8/5 time faster than the original version.

Sometime after the last basic iteration a Mixed Radix conversion can be needed. Classical parallel methods are linear. We propose an logarithmic parallel algorithm for this translation from RNS to Mixed Radix. For this, we use a result that comes from an RNS division algorithm, we published in Journal of VLSI signal processing systems 1998.[3] We obtain in a logarithmic time an approximation of the Mixed radix representation. The correct representation is then established in a logarithmic time too.

Keywords: Modular Multiplication, Montgomery, Residue Number System, Mixed Radix

1. INTRODUCTION

Modular multiplication with very large numbers is used in many cryptosystems.[13,5,10] Different algorithms have been proposed in the literature.[2,8,19,17,16,11] Most of them use redundant radix number systems and Montgomery's modular multiplication.[9] Recently, using the properties of Residue Number Systems, a modular multiplication algorithm based on the Montgomery's algorithm has been proposed.[20]

A limitation of this algorithm is that an operand has to be expressed in Mixed Radix System, while the others are expressed in RNS. We present in this paper improvements of this algorithm which make all the operands to be expressed in RNS. Furthermore, the step amount for computing the modular multiplication is appreciably reduced.

The final step of the algorithm presented in [20] requires a comparison which can be improved using the principle described in [3]. A generalisation of this method is given in this paper.

Section 2 introduce the notations and the concepts used in the paper. The RNS Montgomery's algorithm is shortly introduced in section 3 and its improvements are described in section 4. Finally a new conversion algorithm generalizing the method of [3] is given in section 5.

2. RESIDUE NUMBER SYSTEM AND MIXED RADIX SYSTEM

Residue number system (RNS) is not similar to positional number system where each digit is related to a specified weight.[15,7,18] This property enables that additions and multiplications are very fast in those systems. The RNS systems are defined as follow:

- The vector $\{m_1, m_2, \cdots, m_n\}$ forms a set of moduli, called the RNS-base $\mathcal{B}_n$, where the m_i's are relatively prime.
- M is the value of the product $\prod_{i=1}^{n} m_i$.
- The set $\{x_1, \cdots, x_n\}$ is the RNS representation of X, an integer less than M, where

$$x_i = |X|_{m_i} = X \bmod m_i$$

Any integer X less than M has one and only one RNS-representation thank to the Chinese Remainder Theorem. Addition and multiplication modulo M can be implemented in parallel in linear space $\mathcal{O}(n)$ and performed in one single step, by defining $+_{RNS}$ and $\times_{RNS}$ as component-wise operations:

$$A +_{RNS} B \sim |a_j + b_j|_{m_j}, \text{ for } j \in \{1, \cdots, n\}$$

$$A \times_{RNS} B \sim |a_j \times b_j|_{m_j}, \text{ for } j \in \{1, \cdots, n\}.$$

The Mixed Radix System (MRS) is a weighted positional number representation which can be associated with a RNS sharing the same base of moduli. Assuming that $(x'_1, \cdots, x'_n)$, $0 \leq x'_i < m_i$ is the MRS representation of X $(X < M)$, then

$$X = x'_1 + x'_2\, m_1 + x'_3\, m_2 m_1 + \cdots + x'_n\, m_1 \cdots m_{n-1}. \tag{1}$$

Thus, the weight associated to a position is the product of the previous weight by a new radix.

Conversion from RNS into MRS representation is often used for comparison of RNS numbers, but the MRS system is not well suited for computations.

3. RNS MONTGOMERY MODULAR MULTIPLICATION

In [20] an RNS modular multiplication algorithm is presented. This algorithm is inspired from the original Montgomery modular multiplication[9] where one reduction is performed at each iteration of the multiplication. The advantage of this algorithm is that it maintains during the computation all the numerical values less than $2N$. Furthermore, the only needed operations are additions and multiplications by a digit. It computes $S = AB \times \beta^{-k} \bmod N$ using standard radix β arithmetic. We obtain $AB \bmod N$ using the same algorithm with as inputs, the previous result and β^{2k}.

Noticing that at each iteration of this algorithm, q_i is computed so that $S + a_i \times B + q_i \times N$ is a multiple of β, the division by β corresponds to a shift. The result obtained is the integer value $S = \frac{AB+QN}{\beta^{k+1}}$ where $Q = \sum_{i=0}^{k-1} q_i \beta^i$.

Since the RNS is not a radix representation, the main problem in adapting the algorithm to RNS arithmetic is to compute Q from the least significant digits of the variables. This difficulty is avoided with the use of a mixed radix system associated (with the same set of moduli).

```
ALGORITHM 1 (RNS MODULAR MULTIPLICATION).
Function:  RNS_Modular_Multiplication
Stimulus:  A residue base $\{m_1, m_2, \cdots, m_n\}$, where $M = \prod_{i=1}^{n} m_i$
           A modulus $N$ expressed in RNS with GCD($N, M$)=1, and satisfying
           $0 \leq N < \frac{M}{3\max_{i\in\{1,\cdots,n\}}(m_i)}$
           Integer $A$ given in MRS
               $A = \sum_{i=1}^{n} a'_i \prod_{j=1}^{i-1} m_j$
           Integer $B$ given in RNS
Response:  An integer $R < 2N$ expressed in RNS, such that:
           $R \equiv ABM^{-1} \bmod N$
Method:    $R \leftarrow 0$
           For $i = 1$ to $n$ do
               $q'_i \leftarrow (r_i + a'_i \times b_i) \times (m_i - n_i)_i^{-1} \bmod m_i$
               $R \leftarrow R +_{RNS} a'_i \times_{RNS} B \ +_{RNS} \ q'_i \times_{RNS} N$
               $R \leftarrow R \div_{RNS} m_i$
           End
```

At each step a MRS digit of q'_i of a number Q is computed and a new value of R using q'_i and a'_i is determined in RNS. At each step R is computed so that it is a multiple of m_i. The moduli are relatively prime numbers. Dividing R by m_i is equivalent to multiplying each residue of R by the modular inverse of m_i. But this division cannot be computed for the i^{th} residue.

One solution is, as R is obtained in a reduced base (the standard base with m_i excluded) to reconstruct the value x_i using the Shenoy-Kumaresan method.[14] This linear time method can be pipelined easily on a ring of processors.[20]

Another solution is to extend the modular system with an auxiliary base $\widetilde{\mathcal{B}}_{\widetilde{n}} = \{\widetilde{m}_1, \widetilde{m}_2, \cdots, \widetilde{m}_{\widetilde{n}}\}$. In this system the RNS and MRS representations of an integer X are:

$$X_{RNS} = \{\widetilde{x}_1, \widetilde{x}_2, \cdots, \widetilde{x}_{\widetilde{n}}\} \text{ and } X_{MRS} = \{\widetilde{x}'_1, \widetilde{x}'_2, \cdots, \widetilde{x}'_{\widetilde{n}}\}$$

The intermediate value R is expressed both in $\mathcal{B}_n$ and $\widetilde{\mathcal{B}}_{\widetilde{n}}$. Thus, R is correctly represented in the tail of the main base and in the auxiliary base. After the n steps of the algorithm, the final value of R is only expressed in the extended base $\widetilde{\mathcal{B}}_{\widetilde{n}}$.

```
ALGORITHM 2.
  Function:  RNS_Modular_Multiplication_Listed
  Stimulus:  A residue base $\{m_1, m_2, \cdots, m_n\}$, where $M = \prod_{i=1}^{n} m_i$
             A residue base $\{\tilde{m}_1, \tilde{m}_2, \cdots, \tilde{m}_{\tilde{n}}\}$, where $\tilde{M} = \prod_{i=1}^{\tilde{n}} \tilde{m}_i$
             A modulus $N$ expressed in RNS in the two bases, with GCD($N, M$)=1,
             and satisfying $0 \leq N < 2M, 2\tilde{M}$
             Integer $A$ given in MRS in $\mathcal{B}_n$
             Integer $B$ given in RNS in the two RNS bases
  Response:  An integer $R < 2N$ expressed in RNS, such that:
             $R \equiv ABM^{-1} \bmod N$
  Method:    $R \leftarrow 0$
             For $i = 1$ to $n$ do
                $q'_i \leftarrow (r_i + a'_i \times b_i) \times (m_i - n_i)_i^{-1} \bmod m_i$
                In parallel for each $j > i$
                   $r_j \longleftarrow (r_j + a'_i * b_j + q'_i * n_j) \times (m_i)_{m_j}^{-1} \bmod m_j$
                In parallel for each $j$
                   $\tilde{r}_j \longleftarrow (\tilde{r}_j + a'_i * \tilde{b}_j + q'_i * \tilde{n}_j) \times (m_i)_{\tilde{m}_j}^{-1} \bmod \tilde{m}_j$
             End
```

The modular multiplication algorithm computes $ABM^{-1} \bmod N$ in RNS, the result being obtained in the auxiliary base $\widetilde{\mathcal{B}}_{\tilde{n}}$. In order to eliminate the extra factor M^{-1}, it is possible to use the same algorithm from the auxiliary base to the base $\mathcal{B}_n$. The previously computed result $R = ABM^{-1} \bmod N$ takes the place of the operand A and the constant $M\widetilde{M} \bmod N$ takes place of B.

In this algorithm, the conversion of operand A and the extension of operand B into an extended base require n additions and n multiplications steps. At each iteration, the processor i computing the digit q'_i performs two multiplications and one addition while the $2n - i$ others perform three multiplications and two additions (see figure 1, the first version).

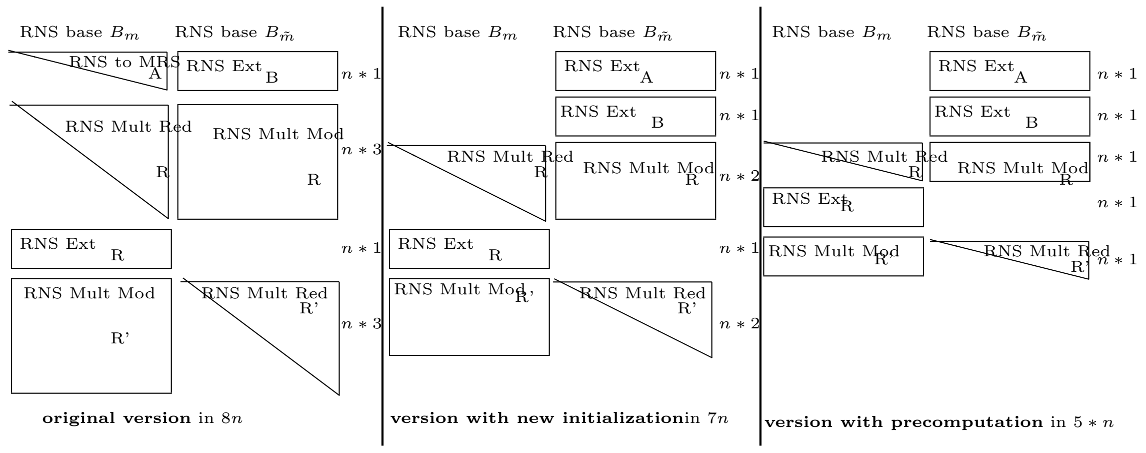

Figure 1. Representation of the computation of the different versions on a bus of processors ($n+\tilde{n}$ processors)

This algorithm can easily be implemented on parallel architectures. In [20], implementations on a bus(with auxiliary base) and on a ring (with Shenoy-Kumaresan) of processors are presented, where space and time complexities are linear. The scheduling of several RNS modular multiplications on such architecture makes possible the computation of modular exponent for large values using relatively small residues. Thus, it is possible to perform computations for cryptographic systems with standard processors more easily.

4. IMPROVEMENTS OF THE ALGORITHM

4.1. First remark

It is possible to initialize the computation of R with the RNS product of A by B. We accumulate in residues r_j and $\tilde{r}_j$, the products $a'_i * b_j \times (m_i)^{-1}_{m_j} \bmod m_j$ and $a'_i * \tilde{b}_j \times (m_i)^{-1}_{\tilde{m}_j} \bmod \tilde{m}_j$. In other words, after the last iteration the value accumulated in the residue r_j is:

$$\begin{aligned} & (\textstyle\sum_{i=1}^{j-1}(a'_i \prod_{k=1}^{i-1} m_k)) * b_j \times (\prod_{i=1}^{j-1} m_i)^{-1}_{m_j} \bmod m_j \\ = \quad & A \bmod m_j \times b_j \times (\textstyle\prod_{i=1}^{j-1} m_i)^{-1}_{m_j} \bmod m_j \\ = \quad & a_j b_j \times (\textstyle\prod_{i=1}^{j-1} m_i)^{-1}_{m_j} \bmod m_j \end{aligned}$$

Note that $\sum_{i=1}^{j-1}(a'_i \prod_{k=1}^{i-1} m_k)$ is the operand A expressed in MRS and $(\prod_{i=1}^{j-1} m_i)^{-1}_{m_j} \bmod m_j$ is the extra factor accumulated at each iteration. Similarly the residue $\tilde{r}_j$ in the auxiliary basis is:

$$\begin{aligned} & (\textstyle\sum_{i=1}^{n}(a'_i \prod_{k=1}^{i-1} m_k)) * b_j \times (\prod_{i=1}^{n} m_i)^{-1}_{\tilde{m}_j} \bmod \tilde{m}_j \\ = \quad & A \bmod \tilde{m}_j \times b_j \times (\textstyle\prod_{i=1}^{n} m_i)^{-1}_{\tilde{m}_j} \bmod \tilde{m}_j \\ = \quad & \tilde{a}_j \tilde{b}_j \times (\textstyle\prod_{i=1}^{n} m_i)^{-1}_{\tilde{m}_j} \bmod \tilde{m}_j \end{aligned}$$

Thus, it is possible to initialize the residues of the accumulated value R with $a_i * b_i \bmod m_i$ and $\tilde{a}_i * \tilde{b}_i \bmod \tilde{m}_i$. This leads to the algorithm:

```
ALGORITHM 3 (RNS MODULAR MULTIPLICATION WITH FIRST REMARK).
  Function:  RNS_Modular_Multiplication_Modif1
  Stimulus:  A residue base $\{m_1, m_2, \cdots, m_n\}$, where $M = \prod_{i=1}^{n} m_i$
             A modulus $N$ expressed in RNS with GCD($N, M$)=1, $0 \le N < 2M, 2\tilde{M}$
             Integer $A$ and $B$ given in RNS, $AB < MN, \tilde{M}N$
  Response:  An integer $R < 2N$ expressed in RNS, such that:
             $R \equiv ABM^{-1} \bmod N$
  Method:    In parallel for each $i$
                $r_i \longleftarrow a_i * b_i \bmod m_i$
                $\tilde{r}_i \longleftarrow \tilde{a}_i * \tilde{b}_i \bmod \tilde{m}_i$
             For $i = 1$ to $n$ do
                $q'_i \longleftarrow (r_i)(m_i - n_i)^{-1}_i \bmod m_i$
                In parallel for each $j > i$
                   $r_j \longleftarrow (r_j + q'_i * n_j) \times (m_i)^{-1}_{m_j} \bmod m_j$
                In parallel for each $j$
                   $\tilde{r}_j \longleftarrow (\tilde{r}_j + q'_i * \tilde{n}_j) \times (m_i)^{-1}_{\tilde{m}_j} \bmod \tilde{m}_j$
             End
```

The main gain of this improvement is that we don't use the MRS the presentation of the operand A. The product AB is made in one step during the initialization. Thus we save one multiplication and one addition in the main iteration (see the second version figure 1). Furthermore, the computation of the value q'_i is simplified. But the representation of A must be extended to the auxiliary RNS base.

4.2. Second remark

It possible to improve again the previous algorithm. Indeed, considering the residues r_j and $\tilde{r}_j$, we have after the last step:

$$\begin{aligned} r_j &= a_i * b_i \prod_{k=1}^{j-1} (m_k)_{m_j}^{-1} + q_1 * n_j \prod_{k=1}^{j-1} (m_k)_{m_j}^{-1} + ... \\ &\quad ... + q_t * n_j \prod_{k=t}^{j-1} (m_k)_{m_j}^{-1} + ... + q_{j-1} * n_j * (m_{j-1})_{m_j}^{-1} \\ q'_j &= r_j * (m_j - n_j)_j^{-1} \end{aligned}$$

and

$$\begin{aligned} \tilde{r}_j &= \tilde{a}_i * \tilde{b}_i \prod_{k=1}^{n} (m_k)_{\tilde{m}_j}^{-1} + q_1 * \tilde{n}_j \prod_{k=1}^{n} (m_k)_{\tilde{m}_j}^{-1} + ... \\ &\quad ... + q_t * \tilde{n}_j \prod_{k=t}^{n} (m_k)_{\tilde{m}_j}^{-1} + ... + q_n * \tilde{n}_j * (m_n)_{\tilde{m}_j}^{-1} \end{aligned}$$

We remark that the values $(m_k)_{m_j}^{-1}$ and $(m_n)_{\tilde{m}_j}^{-1}$ are related only to the main and the auxiliary modular basis and are consequently constant values. Considering that the operand N is rather unchanging in the most common application of modular multiplication, we can assume that N is a constant that can be stored in look-up table.

Thus, it is possible to precalculate the following values:

$$\begin{cases} c_{j,0} & = & \prod_{k=1}^{j-1} (m_k)_{m_j}^{-1} * (m_j - n_j)_j^{-1} \bmod m_j \\ . & . & . \\ c_{j,t} & = & n_j \prod_{k=t}^{j-1} (m_k)_{m_j}^{-1} * (m_j - n_j)_j^{-1} \bmod m_j \\ . & . & . \\ c_{j,j-1} & = & n_j * (m_{j-1})_{m_j}^{-1} * (m_j - n_j)_j^{-1} \bmod m_j \end{cases}$$

$$\begin{cases} \tilde{c}_{j,0} & = & \prod_{k=1}^{n} (m_k)_{\tilde{m}_j}^{-1} \bmod \tilde{m}_j \\ . & . & . \\ \tilde{c}_{j,t} & = & \tilde{n}_j \prod_{k=t}^{n} (m_k)_{\tilde{m}_j}^{-1} \bmod \tilde{m}_j \\ . & . & . \\ \tilde{c}_{j,n} & = & \tilde{n}_j (m_n)_{\tilde{m}_j}^{-1} \bmod \tilde{m}_j \end{cases}$$

This leads to the following algorithm.

Algorithm 4 (RNS Modular Multiplication With Second Remark).

```
Function:  RNS_Modular_Multiplication_Modif2
Stimulus:  A residue base $\{m_1, m_2, \cdots, m_n\}$, where $M = \prod_{i=1}^{n} m_i$
           A modulus $N$ expressed in RNS with GCD(N, M)=1, $0 \le N < 2M, 2\tilde{M}$
           Integer $A$ and $B$ given in RNS in $\mathcal{B}_n$ and $\widetilde{\mathcal{B}}_{\tilde{n}}$
Response:  An integer $R < 2N$ expressed in RNS, such that:
           $R \equiv ABM^{-1} \bmod N$
Method:    In parallel for each $i$
               $r_i \longleftarrow a_i * b_i * c_{i,0} \bmod m_i$
               $\tilde{r}_i \longleftarrow \tilde{a}_i * \tilde{b}_i * \tilde{c}_{i,0} \bmod \tilde{m}_i$
           For $i = 1$ to $n$ do
               $q'_i \longleftarrow r_i$
               In parallel for each $j > i$
                   $r_j \longleftarrow (r_j + q'_i * c_{j,i}) \bmod m_j$
               In parallel for each $j$
                   $\tilde{r}_j \longleftarrow (\tilde{r}_j + q'_i * \tilde{c}_{j,i}) \bmod \tilde{m}_j$
           End
```

The constants $c_{j,0}$ and $\tilde{c}_{j,0}$ are included in the initialization of the residues of R. Consequently, no addition and no multiplication are needed for the computation of q'_i. The main iteration is reduced only to one multiplication and one addition.

We can remark that, in this version (figure 1) that the reduction part of the multiplication (RNS Mult Red) can be done always in the same RNS base, the auxiliary base for example. Thus the number of precomputed value can be reduced by half.

4.3. Complements

All those remarks are also available for the implementation proposed in [20] using the reconstruction of Shenoy-Kumaresan. This kind of algorithms can be also useful for a sequential implementation. In [12] the author proposed an implementation to boost a crypto-processor close to the third version proposed here.

5. A NEW CONVERSION ALGORITHM

In [3], we proposed to compute an approximation of an RNS number in order to perform a division. In this section we propose a generalization of this method which enable us to compute an approximation of each MRS digit of a given RNS number. By this way, we can do a base extension in order to recover the lost moduli (after a division by m_i) or for a base extension (from $\mathcal{B}_n$ to $\widetilde{\mathcal{B}}_{\tilde{n}}$) or exact comparisons (for example the comparison between R and N at the end of the RNS modular multiplication algorithm).

5.1. Computing the approximation of one digit

We suppose in this section that $\forall i,\ m_i \ge 4$. We note :

$$M^i = \prod_{k=1}^{i} m_i \qquad M^i_k = \frac{M^i}{m_k} \qquad X^i = X \pmod{M^i}$$

X^i is the number represented in the reduced base $(m_1, ..., m_i)$ by the i first RNS digits of $X = \{x_1, \cdots, x_n\}$. We want to compute $(x'_1, \cdots, x'_n)$ the standard mixed radix representation associated to $\mathcal{B}_n$, such that:

$$X \quad = \quad x'_1 + x'_2 m_1 + \cdots + x'_n m_1 \cdots m_{n-1}.$$

Standard MRS conversion computes the least significant digits first [7]. We want to compute the digits in every order using the Chinese remainder theorem and the values $X^i = \{x_1, \cdots, x_i\}$. By definition (1) of the MRS, we have for each i, $1 \leq i \leq n$:

$$X^i \quad = \quad x'_1 + x'_2 m_1 + \cdots + x'_i m_1 \cdots m_{i-1}.$$

The most significant MRS digit of X^i is also the i^{th} MRS digit of X. Thus, if we are able to find this digit, we can compute every MRS digit of X in the same time.

So, considering the formula of the demonstration of the Chinese Remainder Theorem [7] applied to X_i, we obtain:

$$X^i \quad = \quad \sum_{k=1}^{i} \left| x_k \left| M_k^i \right|_{m_k}^{-1} \right|_{m_k} M_k^i \quad (\text{mod } M^i)$$

Thus,

$$\frac{X^i}{M^i} \quad = \quad frac\left(\sum_{k=1}^{i} \left| x_k \left| M_k^i \right|_{m_k}^{-1} \right|_{m_k} \frac{1}{m_k} \right) \tag{2}$$

$$\text{and} \quad x'_i \quad = \quad \left\lfloor m_i \times \frac{X^i}{M^i} \right\rfloor \tag{3}$$

As the floating point like notation in [3], we don't compute the exact value of $\frac{X^i}{M^i}$ which require a large precision, but an approximation with p digits, p being an integer close to $\log(m_i)$. The value of p will be given in the following.

Our goal is to obtain the value of x'_i. In other words, we want to obtain $m_i \times \frac{X^i}{M^i}$ with an error less than $\frac{1}{4}$. Thus it will be possible to know x'_i up to one. To assume this precision, we need to compute $\frac{X^i}{M^i}$ with an error less than $2^{-\lceil \log m_i \rceil - 2}$.

We note R_i^p the value obtained from the sum $\sum_{k=1}^{i} \left| x_k \left| M_k \right|_{m_k}^{-1} \right|_{m_k} \frac{1}{m_k}$ computed by a p digits adder taking into account the p most significant digits of the fractional part. The values $x_k \times \frac{|M_k|_{m_k}^{-1}}{m_k}$ are similarly obtained on p digits from a multiplication by a p-digits approximated value of $\frac{|M_k|_{m_k}^{-1}}{m_k}$.

Thus for p large enough, we will obtained R_i^p or $1 - R_i^p$ close to $\frac{X^i}{M^i}$. As we use a rounding to zero, we obtain:

$$0 \quad \leq \quad \frac{X^i}{M^i} - (R_i^p \text{ or } 1 - R_i^p) \quad \leq \quad 2^{-p + \lceil \log(i) \rceil} \tag{4}$$

In equation (4), we consider two cases. The specific case $1 - R_i^p$ takes into account the fact that $\frac{X^i}{M^i}$ could be close to zero. As we are rounding to zero, R_i^p could be close to 1 since the integer part is not considered. The regular case R_i^p is straightforward.

Supposing that $p \geq \lceil \log(i) \rceil + \lceil \log(m_i) \rceil + 2$, we obtain two cases:

$$\begin{array}{ccccc} 0 & \leq & m_i \frac{X}{M^i} - m_i R_i^p & \leq & \frac{1}{4} \\ & & \text{or} & & \\ 0 & \leq & m_i \frac{X}{M^i} - (m_i R_i^p + 1 - m_i) & \leq & \frac{1}{4} \end{array} \tag{5}$$

The last inequation corresponds to the specific case of inequation (4). Using this, we obtain an approximation $\widetilde{x'_i}$ (don't confused with auxiliary base) of the number x'_i. Indeed:

$$x'_i \quad = \quad \left\{ \begin{array}{l} \widetilde{x'_i} \\ \text{or} \\ \left| \widetilde{x'_i} + 1 \right|_{m_i} \end{array} \right.$$

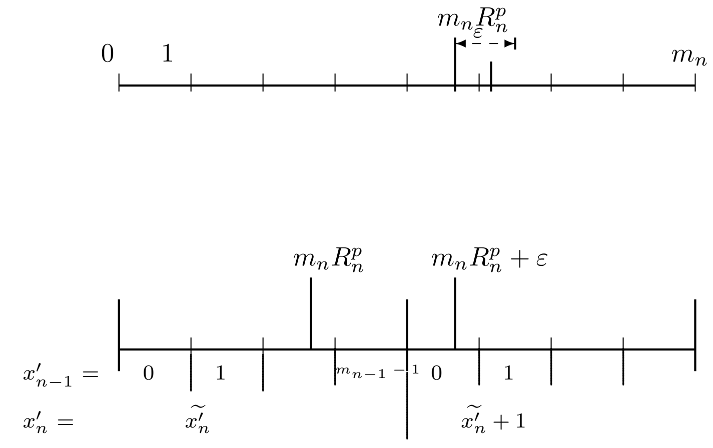

Figure 2. The approximate value of x'_n

Furthermore we know that if $frac\,(m_iR_i^p)$ is in $[0, \frac{3}{4}]$ then $x'_i = \widetilde{x'_i}$. Else the value of x'_i depend on the value of x'_{i-1} (see figure 2).

As $frac\,(R_i^p) \in [\frac{3}{4}, 1]$ involves $m_i\frac{X^i}{M^i} \in [\widetilde{x'_i} + \frac{3}{4}, \widetilde{x'_i} + 1 + \frac{1}{4}]$, we have:

$$\begin{array}{lll|lll}
\text{either} & & & \text{or} & & \\
x'_i = \widetilde{x'_i} & \Longrightarrow m_i\frac{X^i}{M^i} \in [\widetilde{x'_i} + \frac{3}{4}, \widetilde{x'_i} + 1[& & x'_i = \widetilde{x'_i} + 1 & \Longrightarrow m_i\frac{X^i}{M^i} \in [\widetilde{x'_i} + 1, \widetilde{x'_i} + \frac{1}{4}] & \\
 & \Longrightarrow \left\lfloor\frac{3m_{i-1}}{4}\right\rfloor \le x'_{i-1} \le m_{i-1} - 1 & & & \Longrightarrow 0 \le x'_{i-1} \le \left\lfloor\frac{m_{i-1}}{4}\right\rfloor & \\
 & \Longrightarrow \left\lfloor\frac{3m_{i-1}}{4}\right\rfloor - 1 \le \widetilde{x'_{i-1}} \le m_{i-1} - 1 & & & \Longrightarrow 0 \le \widetilde{x'_{i-1}} \le \left\lfloor\frac{m_{i-1}}{4}\right\rfloor \text{ or } \widetilde{x'_{i-1}} = m_{i-1} - 1 &
\end{array}$$

Unfortunately, we may not be able to compute the value of x'_i from $\widetilde{x'_i}$ and $\widetilde{x'_{i-1}}$. This characterization depends of the value x'_{i-1}. Thus, if we know the value of x'_{i-1} or if $\widetilde{x'_{i-1}} \ne m_{i-1} - 1$, then we know the value of x'_i. In the other case, if we know the value of x'_{i-2} or if $\widetilde{x'_{i-2}} \ne m_{i-2} - 1$, then we know the value of x'_i and x'_{i-1}, etc.

This method leads to two conversions from RNS to MRS. The first method computes all the values $(R_i^p)_{1\le i\le n}$ in parallel using $O(n^2)$ floating point adders and multipliers. The result may be obtained trough $O(\log(n))$ steps. The second method needs only $O(n)$ adders and multipliers. It gives the result after $O(n\log(n))$ steps. Those two algorithms are explain in [1].

Since our multiplication algorithm need exact comparisons and base extensions, we will detail this two operations.

5.2. Comparison

Let consider two RNS numbers X and Y :

$$X = \{x_1, x_2, \cdots, x_n\} \qquad Y = \{y_1, y_2, \cdots, y_n\}$$

We compute the approximate MRS digits $\widetilde{x'_n}$, $\widetilde{y'_n}$, $\widetilde{x'_{n-1}}$, $\widetilde{y'_{n-1}}$, $\cdots$, $\widetilde{x'_1}$, $\widetilde{y'_1}$ ordered from most to least significant. The algorithm to compare X^i and Y^i is the following:

- If we know both x'_i and y'_i then
  - if $x'_i = y'_i$ then return $X^{i-1} < Y^{i-1}$

  - else return $x'_i < y'_i$

- elseif we know x'_i and not y'_i then
  - if $\widetilde{y'_i} = m_i - 1$ we must compute the exact MRS digit y'_i in order to continue
  - elseif $x'_i = \widetilde{y'_i}$ or $\widetilde{y'_i} + 1$ we must compute the exact MRS digit y'_i in order to continue
  - else return $x'_i < \widetilde{y'_i}$.
- elseif we know y'_i and not x'_i then
  - if $\widetilde{x'_i} = m_i - 1$ we must compute the exact MRS digit x'_i in order to continue
  - if $\widetilde{x'_i} = y'_i$ or $y'_i - 1$ we must compute the exact MRS digit x'_i in order to continue
  - else return $\widetilde{x'_i} < y'_i$.
- elseif we don't know both x'_i and y'_i
  - if $\widetilde{x'_i} = m_i - 1$ or $\widetilde{y'_i} = m_i - 1$ or $|\widetilde{x'_i} - \widetilde{y'_i}| \leq 1$ then we must compute the exact MRS digits x'_i and y'_i in order to continue
  - else return $\widetilde{x'_i} < \widetilde{y'_i}$.

The worst case of this algorithm is when $X = Y$, all the exact mixed radix digits of X and Y must be generated. In this case the algorithm needs to do $O(n^2)$ operations. But generally, to compare X^i and Y^i, we only need to have the two most significant digits $\widetilde{x'_i}$ and $\widetilde{y'_i}$. The other approximated digits is only needed if we have to compute the exact MRS digits or if we want to do the comparison $X^{i-1} < Y^{i-1}$. This means that at each step we will have to compute the next approximated digits ($\widetilde{x'_{i-1}}$ and $\widetilde{y'_{i-1}}$) only if $\widetilde{x'_i} = m_i - 1$ or $\widetilde{y'_i} = m_i - 1$ or $\widetilde{x'_i} = \widetilde{y'_i} \pm 1$.

5.3. Base extension

Suppose that we have an incomplete representation[6] of X ($0 \leq X \leq \frac{M_n}{2}$) $X = \{x_1, \cdots, x_{n-1}, \bullet\}$. The problem is to recover the last residue.

Let Y be the number represented by $\{x_1, \cdots, x_{n-1}, 0\}$, we have $Y = X + kM_n$ such that $x_n = m_n - kM_n \pmod{m_n}$. And the mixed radix representation of Y is :

$$Y \quad = \quad x'_1 + x'_2 M^1 + \cdots + x'_{n-1} M^{n-2} + kM_n.$$

If we compute

$$R_n^p(Y) = \sum_{k=1}^{n-1} \left| x_k \left| M_k \right|_{m_k}^{-1} \right|_{m_k} M_k \quad \text{and} \quad A_n^p(Y) = m_n R_n^p$$

with $p \geq \lceil \log(i) \rceil + \lceil \log(m_i) \rceil + 2$, we obtain $\widetilde{y'_n}$ such that $\widetilde{y'_n} = k$ or $k-1$. In this case we can know directly the exact value of k. Indeed, if $frac(A_n^p) \in [0, \frac{3}{4}[$ then $m_i \frac{Y}{M} \in [\widetilde{y'_n}, \widetilde{y'_n} + 1[$ and $k = \widetilde{y'_n}$. If $frac(A_n^p) \in [\frac{3}{4}, 1]$ then $m_i \frac{Y}{M} \in [\widetilde{y'_n} - \frac{1}{4}, \widetilde{y'_n} + 1 + \frac{1}{4}[$. But, $m_i \frac{Y}{M} \in [\widetilde{y'_n} - \frac{1}{4}, \widetilde{y'_n} + 1[$ is not possible because this mean $\frac{3}{4} \leq m_i \frac{Y}{M} - k = \frac{X}{M_n} \leq 1$ and we have supposed that $0 \leq X \leq \frac{M_n}{2}$. With $n-1$ multiplications and $m-2$ additions, we can compute k and so $x_n = m_n - kM \pmod{m_n}$.

This algorithm is very close to the Shenoy and Kumaresan one [14]. Their algorithm needs a redundant modulus (which means than $0 \leq X \leq \frac{M_n}{2}$) and they use the Chinese Remainder Theorem to extend the base. Our algorithm uses larger operators (on p digits such that $p \geq \lceil \log(n) \rceil + \lceil \max_i(\log(m_i)) \rceil + 2$) but need only n multiplications and n additions to found $x \pmod{m_n}$. The Shenoy and Kumaresan algorithm needs $2n$ multiplications and $2n$ additions to compute the same value with smaller operator (only $\lceil \max_i(\log(m_i)) \rceil$ digits).

6. CONCLUSION

This improvement makes the algorithm suited for Digital Signal Processors which are designed to perform multiplication-accumulations very fast.

REFERENCES

1. J-C. Bajard and F. Rico. How to improve division in residue number systems. In *IMACS*, 2000.
2. E.F. Brickell. A Survey of Hardware Implementations of RSA. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89*, pages 368–370. Springer-Verlag, 1990.
3. J.-C. Bajard, L.-S. Didier, and J.-M. Muller. A new euclidian division algorithm for residue number systems. *Journal of VLSI Signal Processing*, 19(2):167-178, July 1998.
4. S.E. Eldridge and C. D. Walter. Hardware implementation of Montgomery's modular multiplication algorithm. *IEEE Transaction on Computers*, 42(6):693–699, June 1993.
5. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology - Proceedings of Crypto'86*, pages 186–194, 1986.
6. D. Gamberger. Incompletely specified numbers in the residue number system - definition and applications. In M. D. Ercegovac and E. Swartzlander, editors, *9th IEEE Symposium on Computer Arithmetic*, pages 210–215, Santa Monica, U.S.A, 1989. IEEE Computer Society Press.
7. D.E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming.* Addison-Wesley, 2 edition, 1981.
8. P. Kornerup. High-radix modular multiplication for cryptosystems. In G. Jullien M.J Irwin, E. Swartzlander, editors, *11th IEEE Symposium on Computer Arithmetic*, pages 277–283, Windsor, Canada, 1993. IEEE Computer Society Press.
9. P. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, April 1985.
10. S. Micali and A. Shamir. An improvement of the Fiat-Shamir identification and signature scheme. In *Advances in Cryptology - Proceedings of Crypto'88*, pages 244–247, 1988.
11. H. Orup. Simplifying Quotient Determination in High-Radix Modular Multiplication. In S. Knowles and W. H. McAllister, editors, *Proc. 12th IEEE Symposium on Computer Arithmetic.* IEEE Computer Society, 1995.
12. P. Paillier. Low-cost double-size modular exponentiation or how to stretch your cryptoprocessor. In H. Imai and Y. Zheng, editors, *Second International Workshop on Practice and Theory in Public Key Cryptography, PKC'99*, pages 223-234. Springer Verlag, 1999.
13. R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
14. A. P. Shenoy and R. Kumaresan. Fast base extension using a redundant modulus in RNS. *IEEE Transactions on Computer*, 38(2):292–296, 1989.
15. N. Szabo and R. I. Tanaka. *Residue Arithmetic and its application to Computer Technology.* McGraw-Hill, 1967.
16. M. Shand and J. Vuillemin. Fast Implementations of RSA Cryptography. In M.J. Irwin E. Swartzlander and G. Jullien, editors, *Proc. 11th IEEE Symposium on Computer Arithmetic*, pages 252–259. IEEE Computer Society, 1993.
17. N. Takagi. Modular Multiplication Algorithm with Triangle Addition. In M.J. Irwin, E. Swartzlander and G. Jullien, editors, *Proc. 11th IEEE Symposium on Computer Arithmetic*, pages 272–276. IEEE Computer Society, 1993.
18. F.J. Taylor. Residue Arithmetic: A Tutorial with Examples. *COMPUTER*, pages 50–62, May 1984.
19. C.D. Walter. Systolic Modular Multiplication. *IEEE Transactions on Computers*, C-42(3):376–378, March 1993.
20. J.-C. Bajard, L.-S. Didier, and P. Kornerup, "An RNS montgomery modular multiplication algorithm," *IEEE Transaction on Computers* **47**(7), pp. 766–776, 1998.