Efficient RNS Bases for Cryptography

Jean-Claude Bajard, Nicolas Méloni, Thomas Plantard

Efficient RNS bases for Cryptography

Jean-Claude Bajard, Nicolas Meloni and Thomas Plantard
LIRMM UMR 5506, University of Montpellier 2, France,
{bajard,meloni,plantard}@lirmm.fr

***Abstract*— Residue Number Systems (RNS) are useful for distributing large dynamic range computations over small modular rings, which allows the speed up of computations. This feature is well known, and already used in both DSP and cryptography. In this paper we deal with implementation for huge numbers like those used for ciphering as with RSA or ECC on prime finite fields. Modular multiplication is the main operation of these protocols. We find very interesting modular multiplication algorithms in RNS where the conversion from an RNS basis to another represents the main part of the complexity. Hence, we propose in this paper an analysis of the criteria for selecting some bases giving efficient conversions. We conclude by giving methods for constructing an efficient basis in function of the size of different parameters like the basic operators, the key of the cryptosystem, etc. Residue Number Systems (RNS) are useful for distributing large dynamic range computations over small modular rings, which allows the speed up of computations. This feature is well known, and already used in both DSP and cryptography. In this paper we deal with implementation for huge numbers like those used for ciphering as with RSA or ECC on prime finite fields. Modular multiplication is the main operation of these protocols. We find very interesting modular multiplication algorithms in RNS where the conversion from an RNS basis to another represents the main part of the complexity. Hence, we propose in this paper an analysis of the criteria for selecting some bases giving efficient conversions. We conclude by giving methods for constructing an efficient basis in function of the size of different parameters like the basic operators, the key of the cryptosystem, etc. -**

***Index Terms*— RNS, MRS, modular reduction, division by a constant, bases conversion.RNS, MRS, modular reduction, division by a constant, bases conversion.-**

I. Introduction

The Residue Number Systems (RNS) are based on the very old Chinese Remainder Theorem (CRT) [Knu81] [Gar59]. This theorem can be written as following:

Theorem 1 (Chinese Remainder Theorem): We consider a set of coprime numbers $(m_1, m_2, \ldots, m_n)$. We note $M = \prod_{i=1}^{n} m_i$, If we consider $(x_1, x_2, \ldots, x_n)$ of integer such that $x_i < m_i$. Then there exits an unique X which verifies:

$$\begin{array}{l} 0 \leq X < M \text{ and} \\ x_i = X \bmod m_i = |X|_{m_i} \text{ for } 1 \leq i \leq n. \end{array} \quad (1)$$

The set $(m_1, m_2, \ldots, m_n)$ of coprimes is generally called RNS basis.

An obvious remark is that the main interest of the Residue Number Systems is to distribute integer operations on the residues values. Thus an operation with large integers is made on the residues which are small numbers and where computations can be executed independently for each modulo allowing a complete parallelization of the calculus.

One domain where those systems are helpful, is cryptography [PP95] [BDK98]. The numbers used are huge, 160 bits for Elliptic Curves Cryptography (ECC) and 1024 bits for RSA. These systems offer a good alternative to implemente parallel computing.

Addition and multiplication are easily parallelizable with RNS. But some operations, like division [Gam89] or modular multiplication[PP95], [BDK01], [BI04], need conversions from RNS to an other RNS basis. Those operations have been studied in [BP04] where some criteria for RNS basis was given. We propose to take into account in this approach, the basic modulo algorithms used on a RNS operator on one modulo.

The organization of the paper is the following. We first present conversion algorithms from RNS to RNS, analyzing them through the point of view of the RNS bases. Then, we give algorithms for addition and multiplication implemented in the basic modulo operators. We show that the condition for efficient basic operators are compliant to the criteria useful for the bases conversion in RNS.

II. Conversion from RNS to RNS

There are two main methods for conversion: one directly based on the CRT, and one which uses an auxiliary representation named Mixed Radix Representation (MRS).

We note $\mathcal{B} = (m_1, m_2, \ldots, m_n)$ and $\widetilde{\mathcal{B}} = (\widetilde{m_1}, \widetilde{m_2}, \ldots, \widetilde{m_n})$ two RNS bases.

A. With the Chinese Remainder Theorem

In the proof of the Chinese Remainder Theorem we show, that the system given by (1) has a solution X such

that:

$$X = \left| \sum_{i=1}^{n} \left| x_1 \left| M_i \right|_{m_i}^{-1} \right|_{m_i} M_i \right|_M \quad (2)$$

Where, $M_i = \dfrac{M}{m_i}$ for $1 \leq i \leq n$ and $|M_i|_{m_i}^{-1}$ represents the inverse of M_i modulo m_i .

The conversion from $\mathcal{B}$ to $\widetilde{\mathcal{B}}$, is obtained from the equation (2) which becomes for $j \in \{1, ..., n\}$:

$$X = \left| \sum_{i=1}^{n} \left| x_1 \left| M_i \right|_{m_i}^{-1} \right|_{m_i} |M_i|_{\widetilde{m_j}} \right|_{\widetilde{m_j}} - \left| \alpha \left| M \right|_{\widetilde{m_j}} \right|_{\widetilde{m_j}} \quad (3)$$

where α is such that: $X + \alpha M = \sum_{i=1}^{n} \left| x_1 \left| M_i \right|_{m_i}^{-1} \right|_{m_i} M_i$

The main drawbacks of this approach are due to two elements: first the difficulty to evaluate α [SK89] [HP94], [PP95], then the random aspect of the constant factors $|M_i|_{m_i}^{-1}$ and $|M_i|_{\widetilde{m_j}}$ in (3).

B. Using a mixed radix system

Like for RNS, we define as a MRS basis a set of integers $(m_1, m_2, \ldots, m_n)$ and an integer X, inferior to M, is represented by $(x'_1, ..., x'_n)$, with $x'_i < m_i$ for $i \in \{1, ..., n\}$, such that:

$$X = x'_1 + x'_2 m_1 + x'_3 m_1 m_2 + ... + x'_n m_1 ... m_{n-1} \quad (4)$$

A naive approach [ST67] consists in applying to the RNS representation of X , the extraction of a residue and the division by the radix (which is different at each step). We note $m_{i,j}^{-1}$ the inverse of m_i modulo m_j such that: $m_i . m_{i,j}^{-1} \equiv 1 \pmod{m_j}$. Thus the conversion can be described by the following equations:

$$\begin{cases} x'_1 = x_1 \bmod m_1 \\ x'_2 = (x_2 - x'_1) m_{1,2}^{-1} \bmod m_2 \\ x'_3 = ((x_3 - x'_1) m_{1,3}^{-1} - x'_2) m_{2,3}^{-1} \bmod m_3 \\ \vdots \\ x'_n = (.(x_n - x'_1) m_{1,n}^{-1} \cdots - x'_{n-1}) m_{n-1,n}^{-1} \bmod m_n \end{cases} \quad (5)$$

There is another approach [Knu81] where we factorize all the inverses in (5). But in this case we loose all the possibilities to parallelize.

When the MRS representation is known we apply the expression (4) to each modulo:

$$\begin{aligned} \widetilde{x_j} &= X \bmod \widetilde{m_j} \\ \widetilde{x_j} &= |x'_1 + m_1 (x'_2 + m_2 (x'_3 + ... + m_{n-1} x'_n) ...)| \widetilde{m_j} \end{aligned} \quad (6)$$

We will see that, with the conversion using MRS, it is possible to use some properties of the elements of the two bases to reduce the cost of this operation.

III. Algorithms for basic modulo operators

All the RNS complexities are given in number of basic modulo operations on small moduli m_i (or $\widetilde{m_j}$). It is clear that costs of RNS algorithms depends also of the efficiency of such operators. We present in this section the addition and the multiplication modulo m_i (or $\widetilde{m_j}$). We will see in the next section that another operator; a multiplier by an inverse, could be useful for the conversions.

In this section we consider that the moduli of the RNS bases used are represented with the same number k of digits. We note, for $i = 1...n$, $m_i = 2^k - c_i$ (resp. $\widetilde{m_i} = 2^k - \widetilde{c_i}$). For the moment c_i is just a number lower than 2^k, we will show that for certain values the operators could be very efficient.

A. The addition

We consider two integers a and b such that $0 \leq a, b < m_i$ and their sum s that we want to reduce modulo m_i.

We have the following identities:

$$a + b = s = s_1 2^k + s_0 = s_1 m_i + c_i s_1 + s_0$$

So considering the reduction modulo m_i we obtain $a + b \equiv c_i s_1 + s_0 \pmod{m_i}$. Furthermore, as $0 \leq a, b < m_i$, we know that $a + b < 2m_i - 1$, in other words $a + b < 2^k + (2^k - 2c_i - 1)$. We can obviously deduce that $s_1 \leq 1$. It means that only one reduction by m_i could be needed. We note that we have two possible results $a + b$ if this sum is lower than m_i, or else $a + b + c - 2^k$. We remark that this second possibility implies that $a + b + c$ is greater than 2^k. Thus we only have to test the overflow bit to know which is the right solution.

The addition modulo m_i is given by the following algorithm

Algorithm 1: modadd(a, b, m_i)

Data: $0 \leq a, b < m_i$
Result: $s = a + b \bmod p$
$d_0 \leftarrow a + b$;
$d_1 \leftarrow d_0 + c_i$;
if $d_1 \geq 2^k$ **then**
  $s \leftarrow d_1 \mod 2^k$;
else
  $s \leftarrow d_0$;
end

To resume algorithm 1, the moduli addition is equivalent to two additions, the comparison $d_1 > 2^k$ corresponds to the overflow which is used for selecting the result.

B. The multiplication

As for the addition, the operands are a and b such that $0 \leq a, b < m_i$. We note $p = a \times b$ the product of those two values, and we will reduce p modulo m_i. For this, we decompose p such that: $p = p_1 2^k + p_0 = p_1 m_i + c_i p_1 + p_0$. Thus, $a \times b \equiv c_i p_1 + p_0 \pmod{m_i}$. We note $p' = c_i p_1 + p_0$

The condition $0 \leq a, b < m_i$, gives that $p < m_1^2 - 1 < 2^{2k}$ and $p_1 < 2^k$. Now, if we suppose that $c_i < 2^t$, then we obtain with this first reduction $p' = c_i p_1 + p_0 < 2^k(2^t - 1) + 2^k = 2^{k+t}$.

A second reduction is needed. We decompose p' such that $p' = p'_1 2^k + p'_0$. Like previously, we obtain $a \times b \equiv c_i p'_1 + p'_0 \pmod{m_i}$. We note $p'' = c_i p'_1 + p'_0$ and we remark immediately that , as $p' < 2^{k+t}$, $p'_1 < 2^t$. Now if we consider that $2t < k - 1$ then we assume that $p'' < 2^k + 2^{2t} < 2m_i$. Thus we just need to reduce p'' like in the addition algorithm to find the final result.

For $1 < t < (k-1)/2$ we have the following algorithm for the moduli multiplication operator.

Algorithm 2: modprod(a, b, m_i)

Data: $0 \leq a, b < m_i$
with $m_i = 2^k - c_i$ and $c_i < 2^t$ and
$1 < t < (k-1)/2$
Result: $r = a \times b \bmod p$
$p \leftarrow a \times b$;
$p_1 \leftarrow p \div 2^k$; $p_0 \leftarrow p \bmod 2^k$;
$p' \leftarrow c_i p_1 + p_0$;
$p'_1 \leftarrow p' \div 2^k$; $p'_0 \leftarrow p' \bmod 2^k$;
$p'' \leftarrow c_i p'_1 + p'_0$;
$\rho \leftarrow p'' + c_i$;
if $\rho \geq 2^k$ **then**
  $r \leftarrow \rho \mod 2^k$;
else
  $r \leftarrow p''$;
end

This algorithm gives the modular product with one multiplication of two k-bits integers, one multiplication of a k-bits number by a t-bits constant, one multiplication of a t-bits number by a t-bits constant, and three additions of k-bits numbers. The operations $\div 2^k$ and $\bmod 2^k$ are pure register handling.

If, we consider that a multiplier of a k-bits number by a t-bits constant is equivalent to the half of a multiplier of two k-bits integers, and a multiplier of a t-bits number by a t-bits constant is equivalent to the quarter of a multiplier of two k-bits integers, (we note that here we did not take into account the constant operands which could have some properties useful for improving the complexity) then, the multiplication modulo m_i (or $\widetilde{m_j}$) is done with a complexity equivalent to one plus three quarters of a k-bits multiplier. If we note $M(k)$ the complexity of a k-bits multiplier, the obtained complexity for the modulo operator is $\frac{7}{4}M(k)$.

The conclusion of this section is that with $m_i = 2^k - c_i$ such that $c_i < 2^t$ and $1 < t < (k-1)/2$, we assume that addition and multiplication modulo m_i are efficient.

IV. A new useful algorithm for the basic modular operators

Most of the studies on RNS propose to use a basis like $\{2^k - 1, 2^k, 2^k + 1\}$. This kind of basis is very useful in digital signal processing where the values are bounded and not too huge. The algorithms proposed in the literature [Pie94], [CN99] use generally the CRT approach. But it is not clear that it is the best choice. Conversion using Mixed Radix System (MRS) are often a good approach even for a three elements basis $\{2^k - 1, 2^k, 2^k + 1\}$. If we observe the equation (5) we can consider $m_1 = 2^k + 1$, $m_2 = 2^k - 1$, and $m_3 = 2^k$, then , since $m_1 \bmod m_2 = 2$, $m_1 \bmod m_3 = 1$ and $m_2 \bmod m_3 = -1$, we deduce that $m_{1,2}^{-1} = 2^{n-1}$, $m_{1,3}^{-1} = 1$ and $m_{2,3}^{-1} = -1$. Thus the conversion from binary to MRS can be reduced to one shift and four additions, then conversion to binary needs at most two shifts and four additions.

Now if we consider huge numbers like those used in cryptography, then we show in this section, that the use of MSR can be efficient for some RNS bases. Like in the previous section we consider that for $i = 1...n$, $m_i = 2^k - c_i$ (resp. $\widetilde{m_i} = 2^k - \widetilde{c_i}$) with $0 \leq c_i < 2^t$ (resp. $0 \leq \widetilde{c_i} < 2^t$) and $1 < t < (k-1)/2$. The conversion via MRS use the equations (5) and (6).We find in them two operations: modular multiplication by the modular inverse of a small number for (5) and modular multiplication by a small number for (6).

A. Modular multiplication by a modular inverse of a small number

In fact, in the equations (5), we have to multiply by $|m_j|_{m_i}^{-1}$ for $1 \leq j < i \leq n$. Considering that, $m_i = 2^k - c_i$ we have $|m_j|_{m_i}^{-1} = |c_i - c_j|_{m_i}^{-1}$. We know that $-2^t < c_i - c_j < 2^t$. Thus, as $1 < t < (k-1)/2$, we can consider $c_i - c_j$ like a small value represented on t bits. The sign is not a real problem but to simplify the study we assume that the elements of the basis are decreasing ordered, $m_j > m_i$ for $1 \leq j < i \leq n$.

To resume, the main operations in (5) are of the form $y = x * |d|_m^{-1} \bmod m$ where x and m are two k-bits integers and d a small value (ie t bits number).

P. Montgomery [Mon85] has proposed a reduction where an inverse factor appears in the result. We propose to use it to compute $y = x * |d|_m^{-1} \bmod m$. Thus, we will construct a multiple of the value d by adding to x a multiple of m, this multiple of d is equivalent to x modulo m. Then we divide it by d, which is an exact operation, and we obtain y a reduced value lower than m, which is equivalent to $x * |d|_m^{-1}$ modulo m.

In other words, we must construct μ such that: $\mu < d$ and $x + \mu m$ is a multiple of d, then, as the division of $x + \mu m$ by d is exact, we obtain the expected value y.

This evaluation is done with the algorithm 3 where scdiv(x, d) returns (r_x, q_x) which are the remainder and the quotient of the euclidian division by d (such that: $x = r_x + q_x \times d$ with $0 \leq r_x < d$). The procedure scdiv(x, d) is described in the algorithm **??**.

Algorithm 3: : moddiv(x,d,m)

Data: two given integers d, m with $0 < d < m$ and $gcd(m, d) = 1$
and an integer x, $0 \leq x < m$
Precomputed: r_m, q_m *and* I_m *such that:*
$m = r_m + q_m d$ *with* $r_m < d$ *and*
$I_m = (-m)^{-1} \bmod d$
Result: an integer y with $y = x \times |d|_m^{-1} \bmod m$

1 $(r_x, q_x) \leftarrow$ scdiv(x, d);
2 $(\mu, .) \leftarrow$ bardiv$(r_x \times I_m, d)$;
3 $\nu \leftarrow (r_x + \mu \times r_m)$;
4 $(0, \rho) \leftarrow$ bardiv(ν, d) ;
5 $y \leftarrow \rho + q_x + \mu \times q_m$;

Now, we analyze the complexity of this algorithm step by step, we note $2^{\delta-1} \leq d < 2^\delta \leq 2^t$:

1) r_x and q_x are the remainder and the quotient of the euclidian division of a k-bits integer x by a small given integer d (δ-bits number) . For this operation we call a specific procedure scdiv(x, d) which will be described below.
2) r_x and I_m are smaller than d, so μ is obtained with a classical Barret modular reduction [Bar86] on a 2δ-bits number, applied to the product $r_x \times I_m$. This supposes also that we will store $\left\lfloor \frac{2^{2\delta}}{d} \right\rfloor$ for each possible value of d. μ is the remainder obtained with bardiv$(r_x \times I_m, d)$
3) ν is evaluated with one product of a δ-bits by a $(k - \delta)$-bits numbers and one addition of a k-bits number with a δ-bits one.
4) ρ is obtained with bardiv which evaluates the quotient of the euclidian division of ν a 2δ-bits integer by a given δ-bits integer d.
5) y is the result of one product of a δ-bits by a $(k - \delta)$-bits numbers, one addition of two δ-bits integers and one addition.

We can easily verify that:

$$\begin{aligned} y &= \tfrac{r_x + \mu r_m}{d} + q_x + \mu q_m \\ &= \tfrac{x + \mu \times m}{d} \end{aligned}$$

and

$$y = \frac{x + \mu \times m}{d} < m \text{ since } x < m \text{ and } \mu < d$$

Hence, as the division is exact, $x + \mu \times m$ is a multiple of d, we get $y = x \times |d|_m^{-1} \bmod m$.

We note that, for this algorithm 3, some values are pre-computed, and need to be stored. For each value of d, we will have to store r_m, q_m and I_m, and this for each element m_i of the basis. Thus taking into account that $d < 2^t$ and the equation (5), the size of the tables needed is equal to $\frac{n^2-n}{2} \times (k + 2t)$ bits[1].

1) the procedure scdiv of division by a small constant d: The procedure scdiv evaluates the remainder r_x and the quotient q_x of the euclidian division of an integer x by a small given integer d.

As d is known, we are in a context close to the one of Barrett's algorithm [Bar86]. We propose to adapt this algorithm to our purpose.

The main idea of the Barrett algorithm is based on finding the quotient q of the division by d for $x < 2^{2\delta}$ using the following equation:

$$q = \left\lfloor \frac{\left(\left\lfloor \frac{2^{2\delta}}{d} \right\rfloor \left\lfloor \frac{x}{2^{\delta-1}} \right\rfloor \right)}{2^{\delta+1}} \right\rfloor + \epsilon \text{ , with, } \epsilon = 0, 1, 2$$

$\left\lfloor \frac{2^{2\delta}}{d} \right\rfloor$ is a pre-computed value, thus we just have one mutiplication of two $(\delta + 1)$-bits numbers, other operations are truncations, finding ϵ can need one or two substraction.

The algorithm $bardiv(x, d)$ returns (q, r) such that $q = \left\lfloor \frac{\left(\left\lfloor \frac{2^{2\delta}}{d} \right\rfloor \left\lfloor \frac{x}{2^{\delta-1}} \right\rfloor \right)}{2^{\delta+1}} \right\rfloor$ and $r = x - dq$, for $x < 2^{\delta+l}$ and $2^{\delta-1} \leq d < 2^\delta$.

[1]It is possible to reduce those tables to $\frac{n^2-n}{2} \times (k + t)$ if we consider that k bits are enough to store r_m and q_m

Algorithm 4: : $bardiv(x, d)$

Data: $x < 2^{2\delta}$ and $2^{\delta-1} \leq d < 2^{\delta}$
Precomputed: $\left\lfloor \frac{2^{2\delta}}{d} \right\rfloor$
Result: (r, q) such that $r = x - dq < 3d$

$q \leftarrow \left(\left\lfloor \frac{2^{2\delta}}{d} \right\rfloor \left\lfloor \frac{x}{2^{\delta-1}} \right\rfloor \right)$;
$q \leftarrow \left\lfloor \frac{q}{2^{\delta+1}} \right\rfloor$;
$r \leftarrow x - qd$ ($r < 3d$);
while $r > d$ **do**
  $r \leftarrow r - d$;
  $q \leftarrow q + 1$;
end

The complexity analysis gives:

1) one product of two $(\delta + 1)$-bits numbers,
2) one shift,
3) one product of $(\delta + 1)$-bits numbers, and one subtraction of $(\delta + 2)$-bits numbers.
4) at most two subtractions of (δ)-bits numbers,

Now in algorithm 3 we need to divide x a k-bits integer by d a δ-bits number. So, bardiv cannot be used directly. We propose the algorithm 5 which consider the values written in radix 2^{δ} and uses bardiv to find the digits of the remainder and the quotient.

Algorithm 5: : Division by a small constant $scdiv(x, d)$

Data: two integers $x < 2^k$ and $2^{\delta-1} \leq d < 2^{\delta}$
Result: two integers R, Q with $R + Qd = x$ and $0 \leq R < d$

$R \leftarrow x$;
$Q \leftarrow 0$;
for $i \leftarrow \lceil \frac{k}{\delta} \rceil - 2$ **to** 0 **do**
  $R' \leftarrow R \div (2^{\delta})^i$; $R \leftarrow R \bmod (2^{\delta})^i$;
  $(R', Q') \leftarrow bardiv(R', d)$;
  $R \leftarrow R'(2^{\delta})^i + R$;
  $Q \leftarrow Q'(2^{\delta})^i + Q$;
end

Analysis of the complexity of algorithm 5:

1) we just consider that R is split into two parts, the 2δ upper bits and the $l\delta$ lower bits.
2) we call *bardiv* to make a reduction of R' (a 2δ-bits number) by d (a δ-bits constant), and to recover Q' which is the quotient of this reduction.
3) we rebuild R with the reduced upper part R'. Since R' is on δ-bits, we just shift R'.
4) we build Q with the quotient of the reduction of R'.

We have to notice that Q' could be on $\delta + 1$-bits. In order to avoid making an addition on each loop, we can consider that $Q = \sum_{i=0}^{(\lceil \frac{k}{\delta} \rceil - 2)} Q'_i (2^{\delta})^i$, where Q'_i is the value of the variable Q' at the i^{th} step of algorithm 5. Then we notice that computing $L = \sum_{i=0}^{(\lceil \frac{k}{\delta} \rceil - 2)} (Q'_i \bmod 2^{\delta})(2^{\delta})^i$ and $U = \sum_{i=0}^{(\lceil \frac{k}{\delta} \rceil - 2)} (Q'_i \div 2^{\delta})(2^{\delta})^i$ can be done without any additions (but only with shifts) because $(Q'_i \bmod 2^{\delta})$ and $(Q'_i \div 2^{\delta})$ are δ-bits integers. As $Q'_i = (Q'_i \bmod 2^{\delta}) + (Q'_i \div 2^{\delta})(2^{\delta})$, we have $Q = L + U(2^{\delta})$. So Q can be computed with only one addition.

Finally the cost of *scdiv* is $\lceil \frac{k}{\delta} \rceil - 1$ call to *bardiv* and one k-bits addition.

Now we can evaluate the complexity of a call to *moddiv*. We begin by a call to *scdiv* with two integers, one of k-bits and one of δ-bits. We also call *bardiv* twice to reduce a 2δ-bits number by a δ-bits number. Moreover, we have to compute two multiplication of δ-bits number and one multiplication of a number of δ-bits by a number of $k - \delta + 1$-bits. At last, we compute one addition of 2δ-bits , one of $\delta + 1$-bits and two of k-bits.

If we set $l = \lceil \frac{k}{\delta} \rceil$, we can evaluate the complexity of *moddiv* to $(l + 1)$ multiplications and $(4l + 1)$ additions of δ-bits number. We also call *bardiv* $(l + 1)$ times, but if we consider that the cost of *bardiv* can be evaluated to two δ-bits multiplications and four δ-bits additions, we have a final complexity of $(3l + 3)$ multiplications and $(8l + 5)$ additions.

So our approach to make a multiplcation by the inverse of a small number is about $(3l + 3)$ δ-bits multiplications. Yet, the classical way to compute a multiplcation by an inverse is to precompute the inverse and to make a modular multiplication. If we use Barrett's algorithm to make the modular multiplication, we obtain a $2l^2 + 4l$ δ-bits multiplications complexity.

B. Analysis of this conversion from RNS to RNS via MRS

To make the whole conversion we have in fact two conversions to do.

First we have a RNS to MRS conversion, made of $\frac{n(n-1)}{2}$ steps. Each step being composed of one call to a k-bits *modadd* and one multiplication of a k-bits numbers by a inverse of a δ-bits number using *moddiv*.

Secondly we have a MRS to RNS conversion, made of $n(n-1)$ steps. Each steps being composed of a modular multiplication of a t-bits number by a k-bits number and a call to a k-bits *modadd*. If we look at the progress of the algorithm of the modular multiplication we have one multiplication of a k-bits number by a t-bits number, one multiplication of a t-bits number by a t-bits number and two k-bits additions.

We can simplify the complexity of the RNS to RNS conversion to $\frac{n(n-1)}{2}(5l + 5)$ t-bits multiplications and

$\frac{n(n-1)}{2}(20l+5)$ t-bits additions. In table I we recapitulate the different compexities of our algorithms (we just give the number of t-bits multiplications). We compare our method to the general case where the m_i's are choosen without any special properties.

Operation	Our method	Classic way
modprod	l^2+l+1	$2l^2+4l$
moddiv	$3l+3$	$2l^2+4l$
RNS-MRS	$\frac{3}{2}n(n-1)(l+1)$	$n(n-1)(l^2+2l)$
MRS-RNS	$n(n-1)(l+1)$	$n(n-1)(2l^2+4l)$
RNS-RNS	$\frac{5}{2}n(n-1)(l+1)$	$3n(n-1)(l^2+2l)$

TABLE I

COMPARAISON OF OUR METHOD AND THE CLASSIC WAY FOR THE COST OF THE DIFFERENT RNS OPERATIONS

V. CONSTRUCTION OF EFFICIENT RNS BASES

Now the problem is to find RNS bases such that those differences are minimal. To simplify this study we can consider that the two bases are ordered such that: $m_1 > m_2 > ... > m_n > \widetilde{m_1} > \widetilde{m_2} > ... > \widetilde{m_n}$. Thus we set δ_{max} the maximum number of bits to represent the max of $m_1 - m_n$ and $\widetilde{m_1} - \widetilde{m_n}$. So, the question is: What is the minimal interval of k-bits integers, which contains $2n$ coprime numbers? Or, how many coprimes can we get in an interval of size δ_{max}?

In the table II, we give the δ_{max} one have to use on 16, 32 or 64-bits systems, in order to obtain RNS basis for classic cryptographic lengths.

		Cryptographic length			
		160	192	320	1024
k	16	6	6	7	10
	32	4	5	6	8
	64	-	3	5	7

TABLE II

DIFFERENT δ_{max} NEEDED TO USE DIFFERENT SYSTEMS AND CRYPTOGRAPHIC LENGTHS

Example 1: If we compute on a 32-bits system and if we want to execute an ECC on 160 bits, we can use the two bases build with $m_i = 2^{32} - c_i$ where c_i is in $[3, 5, 9, 15, 17]$ and $\widetilde{m_i} = 2^{32} - \widetilde{c_i}$ where $\widetilde{c_i}$ is in $[19, 21, 23, 27, 29]$. So we have $\delta_{max} = 4$ and the cost of a multipliciation by an inverse is about 27 4-bits multiplications.

Example 2: If we compute on a 32-bits system and if we want to execute an RSA on 1024 bits, we can use the two bases build with $m_i = 2^{32} - c_i$ where c_i is in $[3, 5, 17, 23, 27, 29, 39, 47, 57, 65, 71, 75, 77, 83, 93, 99, 105, 107, 113, 117, 129, 135, 143, 149, 153, 159, 167, 168, 173, 185, 189, 195]$ and $\widetilde{m_i} = 2^{32} - \widetilde{c_i}$ where $\widetilde{c_i}$ is in $[285, 297, 299, 303, 309, 315, 323, 327, 329, 339, 353, 359, 363, 365, 369, 383, 387, 395, 413, 419, 429, 437, 453, 465, 467, 479, 483, 485, 489, 497, 507, 509]$. So we have $\delta_{max} = 8$ and the cost of a multiplication by an inverse is about 15 8-bits multiplications.

VI. CONCLUSION

We have seen in this paper that choosing successive coprime numbers can allow to construct efficient RNS implementations. As all the values of an RNS bases are close, we can choose one reference element and define the others by their differences to it, thus we just use small values. With this choice of moduli, the storage is reduced to $nlog_2(t) + k$ bits for the m_i.We also have proposed some specific algorithms, wich take into account our choice of moduli, in order to obtain efficient RNS to RNS base conversions. Moreover, one should notice that it is easy to find such bases for cryptographic applications, as shown in table II. At last, we do not have taken into account the possibility of selecting randomized bases [BILT04] which is interesting in cryptography. It is a new very interesting use of RNS, which needs to be studied in future works.

REFERENCES

[Bar86] P. Barrett. Implementing the rivest, shamir and adleman public key encryption algorithm on a standard digital processor. In A. M. Odlyzko, editor, *Advances in Cryptology, Proceedings of Crypto'86*, pages 311–323, 1986.

[BDK98] J.-C. Bajard, L.-S. Didier, and P. Kornerup. An RNS montgomery modular multiplication algorithm. *IEEE Transactions on Computers*, 47(7):766–776, 1998.

[BDK01] J.-C. Bajard, L.-S. Didier, and P. Kornerup. Modular multiplication and base extension in residue number systems. In L. Ciminiera N. Burgess, editor, *15th IEEE Symposium on Computer Arithmetic*, pages 59–65, Vail Colorado, USA, 2001. IEEE Computer Society Press.

[BI04] J.-C. Bajard and L. Imbert. A full rns implementation of rsa. *IEEE Transactions on Computers*, 53(6):769–774, 2004.

[BILT04] J.-C. Bajard, L. Imbert, P.LY. Liardet, and Y. Teglia. Leak resistant arithmetic. In L. Ciminiera N. Burgess, editor, *CHES 2004*, pages 59–65, Boston MA, USA, 2004. LNCS Kluwer.

[BP04] J.C. Bajard and T. Plantard. Rns bases and conversions. In *SPIE Annual Meeting 2004, Advenced Signal Processing Algorithms, Architectures, and Implementation XIV*, pages 60–69, 2-6 August 2004 Denver, Colorado, USA, 2004.

[CN99] Richard Conway and John Nelson. Fast converter for 3 moduli rns using new property of crt. *IEEE Transactions on Computers*, 48(8):852–860, 1999.

[Gam89] 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.

[Gar59] H. L. Garner. The residue number system. *IRE Transactions on Electronic Computers*, EL-8(6):140–147, June 1959.

[HP94] C. Y. Hung and B. Parhami. An approximate sign detection method for residue numbers and its application to RNS division. *Computers and Mathematics with Applications*, 27(4):23–35, Feb. 1994.

[Knu81] Donald Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, 2 edition, 1981.

[Mon85] Peter Montgomery. Modular multiplication without trial division. *Mathematic of Computation*, 44(170):519–521, April 1985.

[Pie94] Stanislaw J. Piestrak. Design of high-speed residue-to-binary number system converter based on chinese remainder theorem. In *ICCD 1994*, pages 508–511, 1994.

[PP95] K. C. Posch and R. Posch. Modulo reduction in residue number systems. *IEEE Transaction on Parallel and Distributed Systems*, 6(5):449–454, 1995.

[SK89] A. P. Shenoy and R. Kumaresan. Fast base extension using a redundant modulus in RNS. *IEEE Transactions on Computer*, 38(2):292–296, 1989.

[ST67] N. S. Szabo and R. I. Tanaka. *Residue Arithmetic and its Applications to Computer Technology*. McGraw-Hill, 1967.