Modular Multiplication and Base Extensions in Residue Number Systems

Jean-Claude Bajard
LIRMM
Montpellier, France
bajard@lirmm.fr

Laurent-Stephane Didier
Université de Bretagne Occidentale
Brest, France
laurent-stephane.didier@univ-brest.fr

Peter Kornerup
SDU/Odense University
Odense, Denmark
kornerup@imada.sdu.dk

Abstract

We present a new RNS modular multiplication for very large operands. The algorithm is based on Montgomery's method adapted to residue arithmetic. By choosing the moduli of the RNS system reasonably large, an effect corresponding to a redundant high-radix implementation is achieved, due to the carry-free nature of residue arithmetic. The actual computation in the multiplication takes place in constant time, where the unit of time is a few simple residue operations. However, it is necessary twice to convert values from one residue system into another, operations which take $\mathcal{O}(n)$ time on $\mathcal{O}(n)$ processors, where n is the number of moduli in the RNS systems. Thus these conversions are the bottlenecks of the method, and any future improvements in RNS base conversions, or the use of particular residue systems, can immediately be applied.

1. Introduction

Many cryptosystems [16, 5, 11] employ modular multiplications and exponentiation on very large numbers (possibly one or two thousand bits), and various algorithms have been proposed [3, 9, 23, 21, 20, 12]. Most of them use redundant (possibly high-radix) standard number systems and Montgomery's modular multiplication [10]. On the other hand the Residue Number System (RNS) is also of particular interest, because of the parallel and carry free nature of its arithmetic [19, 22].

Note that the Montgomery modular multiplication takes place in a modified residue system, where operands and results contain an extra factor M, for some suitably chosen value of M. Mapping in and out of this residue system is simple, and its cost may be amortized over many multiplications, when these are used for modular exponentiation. However, if applied to RSA encryption [16] as well as decryption (i.e., both ends using the same RNS system), we may just as well assume that the message itself is considered the RNS representation of a number, thus mapping in and out of the RNS system is not necessary. This is particularly interesting for the Fiat-Shamir authentification protocol [5, 11], where only modular multiplications are used (no exponentiation). We shall thus not further discuss the implications of using this modified residue system.

We have previously [1, 2] proposed two RNS versions of the Montgomery algorithm for modular multiplication. To compute $A * B \bmod N$, an intermediate value Q is to be determined such that $A * B + Q * N$ is a multiple of M, the product of the moduli of the RNS base. The quotient Q was computed digit-wise in a Mixed Radix System (MRS). The result of one pass of the algorithm was then obtained in an auxiliary RNS base, using $\mathcal{O}(n)$ (the size of the RNS base) RNS computations. The first version was a direct translation of the classical Montgomery algorithm for weighted representations to RNS. We just used MRS as a weighted system associated with the RNS. The second version then was an improvement of the first: we showed that a MRS representation of A was not necessary and that we could precompute some values to reduce the complexity of the algorithm.

Here we propose to compute Q with a single parallel RNS calculation in one RNS base, but to be able to divide out the factor M it is neces-

Proceedings of the 15th IEEE Symposium on Computer Arithmetic (ARITH'01)
1063-6889/01 $10.00 © 2001 IEEE

sary to convert Q into an auxiliary RNS base, such that we can evaluate the result of the algorithm, $R = (A*B + Q*N)*M^{-1}$, in the auxiliary RNS base. The major costs now lie in conversions from one base into another. Two such $\mathcal{O}(n)$-time parallel conversion algorithms are described, where the first and classical one based on [17] unfortunately cannot be employed for the conversion of Q. But it can be used to convert the result R back into the original base, allowing it to be used as an operand for another multiplication. For the first conversion it turns out to be sufficient to allow an offset to be present in the residue, i.e., it need not be properly modulo reduced, it just has to belong to the correct residue class. Then using the above mentioned method for converting back to the original system removes the un-wanted offset. Any other base extension algorithm (without extra modulus) may be appropriate. Note that for regularity purposes, we want base conversions which can be executed on simple cells (the n residue "channels"), which excludes the use of $\mathcal{O}(\log(n))$-time algorithms where multi-operand addition in a cell is performed in a tree structure.

Section 2 introduces the notation used in the residue and the mixed radix systems employed. In Section 3 the Montgomery algorithm is briefly introduced and its adaption to the RNS system is discussed, together with a brief proof of correctness. Section 4 then introduces the two conversion algorithms and their use for our new modular multiplication algorithm. Section 5 combines the basic RNS multiplication with the conversions, and finally Section 6 contains some conclusions.

2. The Residue Number Systems

We begin with a short summary of the RNS system, and introduce our terminology:

- 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 mutually prime.
- M is the value of the product $\prod_{i=1}^{n} m_i$.
- The vector $\{x_1, \cdots, x_n\}$ is the RNS representation of X, a positive integer less than M, where
$$x_i = |X|_{m_i} = X \bmod m_i$$

Due to the Chinese Remainder Theorem, any X less than M has one and only one RNS-representation. Addition and multiplication modulo M can be implemented in parallel in linear space ($\mathcal{O}(n)$ channels), and performed in one single step without any carry propagation, by defining $+_{RNS}$ and $\times_{RNS}$ as component-wise operations [8, 19, 22]:

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

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

We also define "exact division", $A \div_{RNS} B$, assuming that B divides A, $\gcd(B, M) = 1$:

$$R = A \div_{RNS} B \quad \sim \quad \hat{r}_j \text{ for } j \in \{1, \cdots, n\}$$

where $\hat{r}_j$ is computed as:

$$\hat{r}_j = \left| a_j \times (B)^{-1}_{m_j} \right|_{m_j}, \tag{1}$$

where $(X)^{-1}_{m_j}$ denotes the inverse of X modulo m_j for X and m_j relatively prime.

We shall also introduce an auxiliary base $\widetilde{\mathcal{B}}_{\tilde{n}} = \{\widetilde{m}_1, \widetilde{m}_2, \cdots, \widetilde{m}_{\tilde{n}}\}$ with $\widetilde{M} = \prod_{i=1}^{\tilde{n}} \widetilde{m}_i$, where $\widetilde{M}$ is coprime to M. In this system the RNS representations of an integer X is:

$$X_{RNS} = \{\widetilde{x}_1, \widetilde{x}_2, \cdots, \widetilde{x}_{\tilde{n}}\}$$

and we shall assume that $\widetilde{M} > M$.

3. The RNS algorithm

Based on the original M-reduce algorithm by Montgomery [10] we want to compute the modular product, $ABM^{-1} \bmod N$, for given A, B, N and M, where $N < M$ and M is chosen such that reductions modulo M are "easy", which will be the case when M is the product of the moduli of the RNS system used. As we shall see below, to be able to perform the computations in RNS arithmetic, we will have to use two RNS systems, as also used in [1, 2, 15, 7]. Hence the operands A and B must be available in both systems. And when for exponentiations the result of a multiplication may be used again as input for other multiplications, the result should also be delivered in both systems.

In the M-reduce algorithm we compute an intermediate value Q, $Q < M$, such that: $A*B + Q*N$ is a multiple of M. Then in RNS the representation of $A*B + Q*N$ in $\mathcal{B}_n$ is composed only of zeros. As we have $Q < M$, Q can be easily obtained in the RNS base $\mathcal{B}_n$. For $i = 1..n$, we have $(a_i * b_i + q_i * n_i) \bmod m_i = 0$, and thus deduce that for $i = 1..n$ we have:

$$q_i = (-a_i * b_i) * (n_i)^{-1}_{m_i} \bmod m_i. \tag{2}$$

But note that since $A*B + Q*N$ is a multiple of M, it cannot be represented in the system with base $\mathcal{B}_n$. Hence to compute the final result $R = (A*B + Q*N)*M^{-1}$ it is necessary to compute its value in an RNS system using another base $\widetilde{\mathcal{B}}_{\tilde{n}}$, and thus not only to have A, B and N available in $\widetilde{\mathcal{B}}_{\tilde{n}}$, but also to convert Q into that system. Hence we obtain the following algorithm:

Algorithm 1 *RNS Modular Multiplication*

Stimulus: *A residue base* $\mathcal{B}_n$, $\{m_1, m_2, \cdots, m_n\}$, *where* $M = \prod_{i=1}^{n} m_i$
A residue base $\widetilde{\mathcal{B}}_{\tilde{n}}$, $\{\widetilde{m}_1, \widetilde{m}_2, \cdots, \widetilde{m}_{\tilde{n}}\}$, *where* $\widetilde{M} = \prod_{i=1}^{\tilde{n}} \widetilde{m}_i$
where $\gcd(M, \widetilde{M}) = 1$ *and* $M < \widetilde{M}$

A modulus N *expressed in RNS in the two bases, with* $\gcd(N, M) = 1$, *and* $0 < 2N < M$
Integer A *given in RNS in the two RNS bases*
Integer B *given in RNS in the two RNS bases with* $A * B < M * N$

Response: *An integer* $R < 2N$ *expressed in the two RNS bases such that* $R \equiv ABM^{-1} (\bmod N)$

Method: $Q \leftarrow (-A \times_{RNS} B) \times_{RNS} N^{-1}$ *in* $\mathcal{B}_n$
Conversion of the representation of Q *from* $\mathcal{B}_n$ *to* $\widetilde{\mathcal{B}}_{\tilde{n}}$
$R \leftarrow (A \times_{RNS} B \ +_{RNS} Q \times_{RNS} N) \times_{RNS} M^{-1}$ *in* $\widetilde{\mathcal{B}}_{\tilde{n}}$
Conversion of the representation of R *from* $\widetilde{\mathcal{B}}_{\tilde{n}}$ *to* $\mathcal{B}_n$

Since $Q < M$ and $AB < MN$ it follows that $R < 2N$, and it is easy to see that the result R satisfies $R \equiv ABM^{-1} (\bmod N)$. With this version of Montgomery's algorithm, base conversions are the major operations of the algorithm, as the two RNS computations can be performed in parallel on all the individual residues.

Remarks:

The direct construction of the result $AB \bmod N$ (say for the Fiat-Shamir Algorithm) needs a second pass of the algorithm with R and $(M^2 \bmod N)$ (a precomputed value) as inputs. With $A, B < N$, as $R < 2N < M$ and $(M^2 \bmod N) < N$, all the conditions of the algorithm are satisfied. But if we want to use this algorithm for exponentials, we must note that it is necessary to require $4N < M$, since the repeated squarings requires results of the algorithm to be used as operands, and thus A and B will only be bounded by $2N$.

4. Base conversion

All conversions of RNS representations from one base $\mathcal{B}_n$ into another $\widetilde{\mathcal{B}}_{\tilde{n}}$, satisfying $\gcd(M, \widetilde{M}) = 1$, where $M, \widetilde{M}$ are the products of the moduli of the systems, must in some way or other implicitly calculate the value of the numbers represented. For our purpose here we want conversion algorithms, which can be executed on a set of simple processors available for the RNS computations (the "channels").

4.1. Using an extra modulus

We consider $X_{RNS} = \{x_1, x_2, \ldots, x_n\}$ represented in the system $\mathcal{B}_n$ with $X \in [0, M[$ and construct X using the Chinese Remainder Theorem (CRT) [8] by the following expression:

$$X = \left(\sum_{i=1}^{n} x_i \, |M_i|_{m_i}^{-1} \, M_i \right) \bmod M \qquad (3)$$

where $M_i = \frac{M}{m_i}$, and $|M_i|_{m_i}^{-1}$ is the inverse of M_i modulo m_i. Thus we have:

$$(x_i \, |M_i|_{m_i}^{-1} \, M_i) \bmod m_j = \begin{cases} x_i & \text{if } j = i \\ 0 & \text{else} \end{cases}$$

The normal use of this method is to reconstruct the integer value of X in a classical number system. Now if we only want to obtain the residue of X modulo $\widetilde{m}_i$, we could use the expression (3). But the modulo-M reduction gives some problems evaluating the residues modulo $\widetilde{m}_i$. However, an alternative form of the CRT allow us to write

$$\sum_{i=1}^{n} \left| x_i \, |M_i|_{m_i}^{-1} \right|_{m_i} M_i = X + \alpha M \qquad (4)$$

for some value of α where $0 \leq \alpha < n$.

In 1989 Shenoy et Kumaresan [17], proposed to use an extra modulus m_x to evaluate α:

$$\alpha = \left| |M|_{m_x}^{-1} \left(\sum_{i=1}^{n} \left| \left| x_i \, |M_i|_{m_i}^{-1} \right|_{m_i} M_i \right|_{m_x} - |X|_{m_x} \right) \right|_{m_x} \qquad (5)$$

Thus it is now possible to compute $\tilde{x}_j = |X|_{\widetilde{m}_j}$ by

$$\tilde{x}_j = \left| \sum_{i=1}^{n} \left| \left| x_i \, |M_i|_{m_i}^{-1} \right|_{m_i} M_i \right|_{\widetilde{m}_j} - |\alpha M|_{\widetilde{m}_j} \right|_{\widetilde{m}_j} \qquad (6)$$

for $j = 1..\tilde{n}$. Observing that the constants $|M_i|_{m_i}^{-1}$, $|M_i|_{m_x}$, $|M|_{m_x}^{-1}$, $|M|_{\widetilde{m}_j}$ and $|M_i|_{\widetilde{m}_j}$ can be precomputed, then α and $\tilde{x}_j$ are evaluated with $n+1$ multiplications and n additions. The only dependence on α is in the last multiplication and addition to compute $\tilde{x}_j$. Thus, in parallel using $\max(n, \tilde{n}) + 1$ channels, α and all the $\tilde{x}_j$ can be evaluated in $n+2$ multiplications and $n+1$ addition steps.

Note that the value of α is bounded by n, which in practice is much smaller than the other moduli

$m_i, i = 1..n$. Hence the extra channel computing α can operate with a modulus $m_x \geq n$ smaller than the rest, possibly even a power of 2.

As $\alpha < n$, the term $|\alpha M|_{\widetilde{m}_j}$ in (6) and the product by $|M|_{m_x}^{-1}$ in (5) could be read from tables, thus only n multiplications will be needed.

However, a major drawback of this method is that, to compute α, one must know one extra residue, $|X|_{m_x}$, which cannot be computed by (2) for Q in our algorithm, since $(|A|_{m_x}|B|_{m_x} + |Q|_{m_x}|N|_{m_x}) \bmod m_x$ is unknown. But if an extra residue for R can be computed by some means then this algorithm can be used to convert the representation of R.

4.2. Allowing an offset in the residue

Considering again equation (4), it expresses which residue class modulo M that X belongs to, and when applied to Q we may not need to know the value of α to proceed. Thus by the CRT

$$\widehat{Q} = \sum_{i=1}^{n} \left| q_i \, |M_i|_{m_i}^{-1} \right|_{m_i} M_i = Q + \alpha M \qquad (7)$$

for some value of α where $0 \leq \alpha < n$.

When $\widehat{Q}$ has been computed it is possible to compute $\widehat{R}$ as

$$\begin{aligned}\widehat{R} = (AB + \widehat{Q}N)M^{-1} &= (AB + QN + \alpha MN)M^{-1} \\ &= (AB + QN)M^{-1} + \alpha N\end{aligned}$$

so that $\widehat{R} \equiv R \equiv ABM^{-1} \pmod N$, which is sufficient for our purpose. Also, assuming that $AB < NM$ we find that $\widehat{R} < (n+2)N$ since $\alpha < n$.

Given the residue representations of A, B and N in the system $\mathcal{B}$, it is thus possible to compute the residue representation of Q in $\mathcal{B}$ by (2), and by (7) to convert it into the representation of $\widehat{Q}$ in the other residue system $\widetilde{\mathcal{B}}$, including possibly an extra residue. In this system $\widehat{R}$ can now be computed, and finally converted back to the system $\mathcal{B}$ using the method of Shenoy and Kumaresan. For applications like in RSA where many modular multiplications are needed, it is not necessary to have the intermediate results perfectly reduced, it is sufficient at the very end of the computation to find the value of α. But the value of N has to be bounded $(n+2)^2 N < M$, since this together with $AB < NM$ assures $\widehat{R} < (n+2)N$. For single multiplications the first Montgomery pass can be performed using (4) obtaining $\widehat{R} < (n+2)N$ with $\widehat{R} = ABM^{-1} \bmod N + \beta N$. For the second pass the inputs are $\widehat{R}$ and $M^2 \bmod N$, and if an algorithm with exact conversion is used then an R is found saitisfying $R < 2N$.

5. The RNS multiplication algorithm

Figure 1 describes the execution of our algorithm, illustrating the time and area complexities. Although it is possible to use maximal parallelism in the form of $n + \tilde{n} + 1$ channels, $\max(n, \tilde{n}) + 1$ will be sufficient most of the time. The only place where more processors could be employed is to perform the computation of the product AB in the system $\widetilde{\mathcal{B}}_{\tilde{n}}$, so that it is available when $\widehat{Q}$ has been converted. The time complexity is approximately $2(n + \tilde{n})T$, where T is the time for one table look-up plus one modular multiply-add operation. For applications in cryptology, say with $N \sim 2^{1024}$, it is possible to choose $n = \tilde{n} = 33$, where each channel is realized by a very simple 32-bit processor, i.e., a total of 34 processors. Each processor has to be able to compute additions and multiplications, modulo some specific 32-bit primes, and to store some 32-entry look-up tables of 32-bit constants.

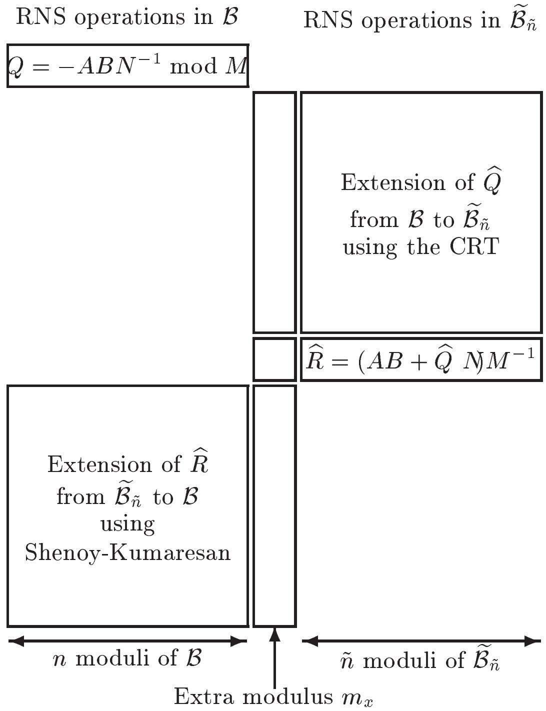

Figure 1. Evaluation of $A \times B \times M^{-1}$ in RNS

Example We consider the systems $\mathcal{B}_5 = \{3, 7, 13, 19, 29\}$, $\widetilde{\mathcal{B}_5} = \{5, 11, 17, 23, 31\}$, the extra modulus $m_e = 8$ and operands A, B and N. Thus, we have $M = 150423$ and $\widetilde{M} = 666655$.

	In $\mathcal{B}_n$					m_x	In $\widetilde{\mathcal{B}}_n$					Base 10
	3	7	13	19	29	8	5	11	17	23	31	
A	1	3	9	14	25	2	1	8	2	5	5	26386
B	1	5	1	9	25	3	1	1	1	21	19	72931
N	1	2	6	11	27	7	2	7	9	14	19	14527

The computation of $A \times B \times M^{-1} \bmod N$ is detailed as shown in the following table.

In $\mathcal{B}_n$					m_x	In $\widetilde{\mathcal{B}}_n$					Computation
2	3	5	11	8							$Q \leftarrow (-A \times B) \times N^{-1} = 143993$
					5	1	3	8	1	14	Extension of Q from $\mathcal{B}_n$ to $\widetilde{\mathcal{B}}_n$
					1	2	5	12	6	7	
					0	2	0	16	16	16	
					6	0	2	14	17	29	
					7	4	10	0	19	20	$\widehat{Q} = 444839 = 143993 + 2 * 150423$
					1	3	5	10	1	15	$\widehat{R} \leftarrow (A \times B \ + \ \widehat{Q} \times N) \times M^{-1}$
0	6	9	5	25	1						Conversion of R through
2	3	12	12	4	3						Shenoy and Kumaresan algorithm
2	2	2	6	17	5						
0	2	10	13	10	3						
1	0	2	13	24	6						
					3						$(= \alpha)$
1	5	9	7	15							$\widehat{R} = 55753 = (12172 + 3 * N) \bmod M$
											$= (A \times B \times M^{-1} \bmod N + 3 * N) \bmod M$
1	1	9	2	23	4	0	7	0	22	25	$M^2 \bmod N = 12580$ with $\widehat{R}$ as input
											Montgomery with exact extension
2	6	9	11	19							$AB \bmod N = 9257$

5.1. Using specific sets of moduli

The use of specific moduli can improve conversion methods using either the Chinese Remainder Theorem or a Mixed Radix System. Many such systems have been previously published using small sets of three or more moduli like $\{2^k-1, 2^k, 2^k+1\}$ for themost popular set. The benefit of such systems is twofold. First, since operations modulo 2^k-1, 2^k or 2^k+1 are reduced to simple logic operations, RNS operations are easier in such systems. Second, the values involved in CRT or MRS conversion algorithms are simple numbers which greatly simplifies the conversion computations.

A recent result published in [6] shows that a conversion from RNS to binary using the set $\{2^k - 1, 2^k, 2^{k-1}-1\}$ has a $\mathcal{O}(k)$ space complexity, and a delay equal to one $2k-1$-bit adder and two $k-1$ adders without any lookup table.

However, as discussed above for applications in crypto-algorithms, much larger systems are needed. The only two systems we have found with a parameterizable number of moduli of the form $2^k \pm 1$, are the systems $\{2^m - 1, 2^{2^0 m} + 1, 2^{2^1 m} + 1, \cdots, 2^{2^k m} + 1\}$ proposed in [14], and systems $\{2^{n_1} \pm 1, 2^{n_2} \pm 1, \cdots, 2^{n_L} \pm 1\}$ in [18] where the exponents $n_1, n_2, \cdots, n_L$ are mutually prime, not divisible by 3.

The first kind of systems are extremely un-balanced. Actually, the largest modulus approximately equals the product of all the remaining moduli, i.e., it is of about the same size as $\sqrt{M}$, where M is the system modulus. The second kind of systems are also un-balanced when a dynamic range of say 1024 bits is needed. It is even simply impossible with 32-bit processors to find sufficiently many such mutually prime moduli pairs of the form $2^k \pm 1$. And with 64-bit processors at least 20 pairs of moduli of the form $2^k \pm 1$ will be needed, in which case the system can hardly be considered balanced.

5.2. Using 32-bit prime moduli in the channels

As the operations inside the channels are completely independent, we can use any representation for the residues there, that we may find convenient. Employing say 32-bit processors, addition modulo a 32-bit prime is not that difficult, the problem is modular multiplication. Just as it is possible to use Montgomery modular multiplication for the very large operands employed in cryptographic applications, it is also possible to use this type of modular multiplication for the operations inside the individual channels of the proposed system. But when information is transfered in and out of the channels (e.g., for broadcasts), it is then necessary to convert into and from the representation used in the particular channel.

The trick in Montgomery modular multiplication is to substitute difficult modular reductions (say by a 32-bit prime) by simpler reductions, here by 2^{32}. This is done by mapping operands into another residue system, so with m prime let

$$\text{for} \quad 0 \le a < m \quad \text{let} \quad [a]_m = a\, 2^{32} \bmod m$$

then it is easy to see that

$$[a+b]_m = |[a]_m + [b]_m|_m$$

where the outer reduction is an ordinary reduction modulo m.

Multiplication in this system can be performed by Montgomery's M-reduce algorithm:

Algorithm 2 *M-reduce(t)*

Stimulus: *An integer t such that* $0 \le t < rm$. *Integer constants* m, r, m', r^{-1} *such that* $\gcd(m,r) = 1, r > m > 2$ *and* $rr^{-1} - mm' = 1$.

Response: *An integer u,* $u = (tr^{-1}) \bmod m$.

Method: $q := ((t \bmod r)m') \bmod r$;
$u := (t + qm) \operatorname{div} r$;
if $u \ge m$ *then* $u := u - m$;

Thus with $r = 2^{32}$, since M-reduce$([a]_m[b]_m) = ab\, 2^{32} \bmod m = [ab]_m$, the product $[ab]_m$ can be computed with 3 ordinary 32-bit multiplications and 1 or 2 ordinary additions, given suitable constants. Mapping into and out of this residue system is performed by M-reduce, multiplying with constants $|r^2|_m$, respectively 1.

6. Conclusions

With this new approach to modular multiplication in RNS, we observe that base extension is a key operation. All the complexity is due to this transformation. Shenoy and Kumaresan proposed an efficient method based on the CRT, by which a logarithmic time complexity can be obtained with n^2 processors. Utilizing n parallel RNS channels the algorithm can be implemented in $n+1$ steps, using one additional channel for an extra residue.

But since such an extra residue cannot be obtained for the quotient Q, and if the computation is to be performed on the set of parallel RNS processors, some other mechanism must be employed. In [15] and [7] various fixed-point computations were used to obtain approximations to α in (4), and in that way perform the base conversion of both Q and R. We realized that the conversion of Q need not be "exact", an "approximation" $\widehat{Q}$ is sufficient to obtain an $\widehat{R}$ in the same residue class as the result R.

Compared to other number systems, RNS can be considered a real parallel system. Our method is easily implementable with processors connected by a bus, where communication is reduced to at most one broadcast per step.

If modular multiplication is implemented on processors which do not support arithmetic on the full data-width (say 1024 bits), we believe that an RNS implementation is preferable to a high-radix implementation. Given a number of smaller (say 32-bit) processors, parallelism is easier to exploit in RNS with $\mathcal{O}(n)$ processors, than in a redundant ordinary radix system. And it is not necessary to employ the full $n+\tilde{n}+1$ processors for maximal parallelism, or almost as good, $\max(n,\tilde{n})+1$, any number will do, even a single processor.

References

[1] 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.

[2] J.-C. Bajard, L.-S. Didier, P. Kornerup and F. Rico, Some Improvements on RNS Montgomery Modular Multiplication, *SPIE's International Symposium on Optical Science and Technology* 30 July - 4 August 2000, San Diego, California USA.

[3] E.F. Brickell. A Survey of Hardware Implementations of RSA. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89*, LNCS-435, pages 368–370. Springer-Verlag, 1990.

[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] A. A. Hiasat and S. H. Abdel-Aty-Zohdy. Residue-to-binary arithmetic converter for the moduli set $(2^k, 2^k-1, 2^{k-1}-1)$. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 45(2):204-209, 1998.

[7] S. Kawamura, M. Koike, F. Sano and A. Shimbo. Cox-Rower Architecture for Fast Parallel Montgomery Multiplication. *Proc. EUROCRYPT 2000*, LNCS 1807, pages 523–538, Springer Verlag, 2000.

[8] D.E. Knuth. *Seminumerical Algorithms*, Volume 2 of *The Art of Computer Programming*. Addison-Wesley, 2 edition, 1981.

[9] 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.

[10] P. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, April 1985.

[11] 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.

[12] 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.

[13] 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*, LNCS-1560, pages 223-234. Springer Verlag, 1999.

[14] F. Pourbigharaz and H. M. Yassine. A signed-digit architecture for residue to binary transformation. *IEEE Transactions on Computers*, 46(10):1146–50, 1997.

[15] K. Posch and R. Posch. RNS-Modulo Reduction in Residue Number Systems. *IEEE Trans. on Parallel and Distributed Systems*, 6(5):449–454, May 1995.

[16] 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.

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

[18] A. Skavantzos and M. Abdallah. Implementation issues of the two-level residue number system with pairs of conjugate moduli. *IEEE Transactions on Signal Processing*, 47(3):826–38, 1999.

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

[20] 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.

[21] 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.

[22] F.J. Taylor. Residue Arithmetic: A Tutorial with Examples. *COMPUTER*, pages 50–62, May 1984.

[23] C.D. Walter. Systolic Modular Multiplication. *IEEE Transactions on Computers*, C-42(3):376–378, March 1993.