A Full RNS Implementation of RSA

Laurent Imbert, *Member*, *IEEE*, and
Jean-Claude Bajard, *Member*, *IEEE*

Abstract—We present the first implementation of RSA in the Residue Number System (RNS) which does not require any conversion, either from radix to RNS beforehand or RNS to radix afterward. Our solution is based on an optimized RNS version of Montgomery multiplication. Thanks to the RNS, the proposed algorithms are highly parallelizable and seem then well suited to hardware implementations. We give the computational procedure both parties must follow in order to recover the correct result at the end of the transaction (encryption or signature).

Index Terms—Cryptography, RSA, Montgomery multiplication, Residue Number Systems.

1 Introduction

During the last decade, fast hardware implementations of public-key cryptosystems have been widely studied [3], [5], [15] while confidentiality and security requirements were becoming more and more important. As a consequence, key-length has kept growing. Nowadays, it is assumed that a 1,024-bit key-length makes a reasonable choice for RSA [14] and current analysis predicts that 2,048-bit or 4,096-bit key will become the standard in the very near future. The ability to perform fast arithmetic on large integers is then still a major issue for the implementation of public key cryptography, particularly from a hardware viewpoint.

Different approaches have been proposed to accelerate the implementation of RSA. For the deciphering, a well-known solution performs the computations over $\mathbb{Z}/p\mathbb{Z}$ and $\mathbb{Z}/q\mathbb{Z}$ independently and reconstructs the final result via the Chinese Remainder Theorem (CRT) [13]. This first application of the CRT to RSA was restricted to this special case (the isomorphism $\mathbb{Z}/n\mathbb{Z} \simeq \mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z}$, with $n = pq$), but it can also be useful in other situations and is not restricted to the decipherment or signature steps, i.e., when the secret quantities p and q are known. More recently, other CRT-based solutions have been proposed [12], [7], [11], [1]. They all use a quite similar version of the Montgomery multiplication based on the Residue Number System (RNS) [18] which is well-adapted to fast parallel arithmetic. The computational complexity of all those algorithms mainly depends on two RNS base extensions that are required for each modular multiplication.

In this paper, we refine and detail the implementation and complexity of an efficient RNS version of the Montgomery multiplication algorithm previously proposed by the authors in [1]. The proposed algorithm uses two different techniques for the first and second base extensions. In terms of elementary operations, its cost is similar to the previous proposed methods. However, in the previous approaches, different conversion techniques, from binary to RNS and RNS to binary, were proposed. Although the same conversion techniques apply for our algorithm, the real novelty of this paper is a full RNS implementation which does not require any conversion. We directly consider the message as a value represented in RNS and we perform all the computations within this system. Unlike other RNS Montgomery-based implementations of RSA, the final correction step of our technique does not require a full precision comparison. We illustrate this fact with a complete textbook implementation of RSA in RNS. Moreover, the proposed solution is also interesting for those who already have an RNS Montgomery multiplication procedure or architecture since it can be easily adapted to any RNS-based multiplication algorithm.

• *The authors are with the Laboratoire d'Informatique, Robotique et Microélectronique de Montpellier, LIRMM, 161 rue Ada, 34392 Montpellier cedex 5, France.*
E-mail: {laurent.Imbert}@lirmm.fr.

Manuscript received 15 Apr. 2003; accepted 5 Sept. 2003.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0011-0403.

2 The Residue Number System

Residue Number Systems (RNS) have been widely studied and used in many applications, from digital signal processing to multiple precision arithmetic [17], [18], [8]. In the RNS, an integer x is represented according to a base $\mathcal{B} = (m_1, m_2, \ldots, m_k)$ of k relatively prime moduli (k is the size of the base), by the sequence $(x_1, x_2, \ldots, x_k)$ of positive integers, where $x_i = x \bmod m_i$ for $i = 1 \ldots k$. The Chinese Remainder Theorem (CRT) ensures the uniqueness of this representation within the range $0 \leq x < M$, where $M = \prod_{i=1}^{k} m_i$. A constructive proof of this theorem gives an algorithm to convert x from its residue representation to the classical radix representation:

$$x = \sum_{i=1}^{k} x_i M_i \left| M_i^{-1} \right|_{m_i} \bmod M, \tag{1}$$

where $M_i = M/m_i$ and $\left|M_i^{-1}\right|_{m_i}$ is the inverse of M_i modulo m_i. In the following portions of the paper, we shall use $|x|_m$ to denote the remainder of x in the division by m, i.e., the value $(x \bmod m) < m$.

The advantages of RNS is that addition, subtraction, and multiplication are very simple and can be implemented in constant time on a parallel architecture. If x and y are given in their RNS form $(x_1, \ldots, x_k)$ and $(y_1, \ldots, y_k)$, one has

$$x \pm y = \Big(|x_1 \pm y_1|_{m_1}, \ \ldots \ , |x_k \pm y_k|_{m_k} \Big),$$
$$x \times y = \Big(|x_1 \times y_1|_{m_1}, \ \ldots \ , |x_k \times y_k|_{m_k} \Big).$$

An important remark is that the final result of a computational task must belong to the interval $[0, M)$ to admit a valid RNS representation in the base $\mathcal{B}$. However, all the intermediate computations can always be performed within the same system, even if the dynamic range provided by the RNS base is not large enough.

On the other hand, one of the disadvantages of this representation is that we cannot easily decide whether $(x_1, \ldots, x_k)$ is greater or less[1] than $(y_1, \ldots, y_k)$ and overflows that can occur during computations are not easily detected.

In cryptographic applications, modular reduction (the computation of $x \bmod m$), multiplication ($xy \bmod m$) and exponentiation ($x^y \bmod m$) are the most important operations. They can be efficiently computed without division using Montgomery's algorithms [10].

3 Modular Exponentiation

Let us briefly recall the principles of Montgomery's techniques. Given two integers R, N such that $\gcd(R, N) = 1$, and $0 \leq x < RN$, the Montgomery reduction technique evaluates $xR^{-1} \bmod N$ by computing the value $q < R$ such that $x + qN$ is a multiple of R. Hence, the quotient $y = (x + qN)/R$ is exact and easily performed. It satisfies $y < 2N$ and $y \equiv xR^{-1} (\bmod N)$. In the same way, Montgomery modular multiplication algorithm computes $xyR^{-1} \bmod N$. For practical implementations, the Montgomery constant R is chosen as a power of 2 to reduce the division by R to

1. According to the CRT testing, the equality of two RNS numbers is trivial.

0018-9340/04/$20.00 © 2004 IEEE Published by the IEEE Computer Society

Algorithm 1 : MM(a, b, N), *RNS Montgomery Multiplication*

Input : Two RNS bases $\mathcal{B} = (m_1, \ldots, m_k)$, and $\mathcal{B}' = (m_{k+1}, \ldots, m_{2k})$, such that $M = \prod_{i=1}^{k} m_i$, $M' = \prod_{i=1}^{k} m_{k+i}$ and $\gcd(M, M') = 1$; a redundant modulus m_r, $\gcd(m_r, m_i) = 1$, $\forall i = 1...2k$; a positive integer N represented in RNS in both bases and for m_r such that $0 < (k+2)^2 N < M$, M' and $\gcd(N, M) = 1$; (Note that M can be greater or less than M'.) two positive integers a, b represented in RNS in both bases and for m_r, with $ab < MN$.

Output : A positive integer $\hat{r}$ represented in RNS in both bases and m_r, such that $\hat{r} \equiv abM^{-1} \pmod{N}$, and $\hat{r} < (k+2)N$.

1: $t = ab$ in $\mathcal{B} \cup \mathcal{B}' \cup \{m_r\}$

2: $q = t(-n^{-1})$ in $\mathcal{B}$

3: $[q \text{ in } \mathcal{B}] \longrightarrow [\hat{q} \text{ in } \mathcal{B}' \cup \{m_r\}]$ *First base extension*

4: $\hat{r} = (t + \hat{q}N)M^{-1}$ in $\mathcal{B}' \cup \{m_r\}$

5: $[\hat{r} \text{ in } \mathcal{B}] \longleftarrow [\hat{r} \text{ in } \mathcal{B}']$ *Second base extension*

Fig. 1. Algorithm 1.

simple shifts. A more detailed discussion on Montgomery reduction and multiplication algorithms can be found in [9], [4].

In the next sections, we present an RNS version of Montgomery multiplication and the conditions for its use within a modular exponentiation algorithm based on the classical technique which combines Montgomery reduction and a binary or k-ary method. If we want to evaluate $x^a \bmod N$, the input x is first transformed into $x' = xR \bmod N$. x' is often called the Montgomery representation of x, or the N-residue of x according to R. It is easy to see that this can be done using a Montgomery multiplication with x and $(R^2 \bmod N)$ as inputs. This representation has the advantage of being stable over Montgomery multiplication:

$$x' \times y' \bmod N = xyR \bmod N.$$

At the end of the exponentiation, the value $z' = x^a R \bmod N$ is converted back into $z = x^a \bmod N$ using a last call to Montgomery multiplication with z' and 1 as inputs. The efficiency of the exponentiation clearly relies on the ability to perform the Montgomery modular multiplication.

4 RNS Montgomery Multiplication

In the RNS version of the Montgomery multiplication algorithm proposed here, the value

$$M = \prod_{i=1}^{k} m_i, \tag{2}$$

is chosen as the Montgomery constant. Hence, the RNS Montgomery multiplication of a and b yields

$$r = abM^{-1} \bmod N, \tag{3}$$

where r, a, b, and N are represented in RNS according to a predefined base $\mathcal{B}$. As in the classical Montgomery algorithm, we look for an integer q such that $(ab + qN)$ is a multiple of M. Hence, the resulting division $(ab + qN)/M$ is exact and is easily performed in RNS by multiplying $(ab + qN)$ by M^{-1}. Unfortunately, the inverse of M does not exist modulo M. Then, the multiplication by M^{-1} cannot be performed in the base $\mathcal{B}$. We define an extended base $\mathcal{B}'$ of k extra relatively prime moduli and perform the multiplication by M^{-1} within this new base $\mathcal{B}'$. For simplicity, we will consider in the rest of the paper that both $\mathcal{B}$ and $\mathcal{B}'$ are of size k. Let us define $\mathcal{B} = (m_1, \ldots, m_k)$ and $\mathcal{B}' = (m_{k+1}, \ldots m_{2k})$, with $M' = \prod_{i=1}^{k} m_{k+i}$, $\gcd(M, M') = 1$.

Now, in order to determine q, we use the fact that $(ab + qN)$ must be a multiple of M. Clearly, its representation in the base $\mathcal{B}$ is merely composed of 0. The RNS representation of q (in the base $\mathcal{B}$) is then given by the solutions of the equations

$$(a_i b_i + q_i n_i) \equiv 0 \pmod{m_i}, \ \forall i = 1 \ldots k, \tag{4}$$

which gives

$$q_i = \left| a_i b_i \right| - n_i^{-1} \Big|_{m_i} \Big|_{m_i}, \ \forall i = 1 \ldots k. \tag{5}$$

As a result, we have computed a value $q < M$ such that $q = -abN^{-1} \bmod M$.

As pointed out previously, we compute $(ab + qN)$ in the extra base $\mathcal{B}'$. Before we can evaluate $(ab + qN)$, we have to know the product ab in base B' and extend q—which has just been computed in base $\mathcal{B}$ using (5)—in base $\mathcal{B}'$. We shall discuss this first base extension in detail in Section 4.1. We then compute $r = (ab + qN)M^{-1}$ in base $\mathcal{B}'$ and extend the result back to the base $\mathcal{B}$ for future use (the next call to Montgomery multiplication). The second base extension is discussed in Section 4.2. Algorithm 1 (shown in Fig. 1) summarizes the computations of our RNS Montgomery multiplication. It computes the Montgomery product $abM^{-1} \bmod N$, where a, b, and N are represented in RNS in both bases $\mathcal{B}$ and $\mathcal{B}'$.

Steps 1, 2, and 4 of Algorithm 1 consist of full RNS operations and can be performed in parallel. As a consequence, the complexity of the algorithm clearly relies on the two base extensions of lines 3 and 5. This algorithm is very similar to those of Posch and Posch [12] and Kawamura et al. [7] which also require two base extensions. However, in their approaches, the same technique is applied for both the first and second base extensions, whereas our solution uses different algorithms. In the next two sections, we give the details of each base extension and we show that our choice requires approximately the same number of elementary operations as those previously proposed methods.

4.1 First Base Extension

In Step 3 of Algorithm 1, we convert q obtained in its RNS form $(q_1, \ldots, q_k)$ in base $\mathcal{B}$ to its RNS representation in base $\mathcal{B}'$ and for an extra-modulus m_r (we explain the reason for this redundant

Algorithm 2 : First (approximated) RNS base extension
Input : $(q_1, \ldots, q_k)$, the RNS representation of q in the base $\mathcal{B}$,
Output : $(\hat{q}_{k+1}, \ldots, \hat{q}_{2k})$ and $\hat{q}_r$, the RNS representation of $\hat{q}$ in $\mathcal{B}' \cup \{m_r\}$.

1: $\sigma_i = q_i \left|M_i^{-1}\right|_{m_i} \bmod m_i$, [in // for $i = 1...k$]
2: $t_0 = 0$
3: **for** $i = 1...k$ **do**
4: $t_i = (t_{i-1} + \sigma_i \left|M_i\right|_{m_j}) \bmod m_j$, [in // for $j = k+1...2k$ and $j = r$]
5: $\hat{q}_j = t_k$ [in // for $j = k+1...2k, r$]

Fig. 2. Algorithm 2.

modulus in the next section). We first compute in parallel, for all $i = 1 \ldots k$, the values

$$\sigma_i = q_i \left|M_i^{-1}\right|_{m_i} \bmod m_i, \tag{6}$$

where the $\left|M_i^{-1}\right|_{m_i}$ are precomputed constants. From (1), we have

$$q = \sum_{i=1}^{k} \sigma_i M_i - \alpha M, \tag{7}$$

where $\alpha < k$.

In the context of Algorithm 1, it is important to note that we do not need to compute the exact value of q for each modulus of $\mathcal{B}'$. We only extend

$$\hat{q} = q + \alpha M$$

by computing, in parallel, for $j = k+1 \ldots 2k$ and for $j = r$, the residues

$$\hat{q}_j = \left| \sum_{i=1}^{k} \sigma_i \left|M_i\right|_{m_j} \right|_{m_j}, \tag{8}$$

where the values $\left|M_i\right|_{m_j}$ are also precomputed constants. In Step 4, we then compute

$$\hat{r} = (ab + \hat{q}N)M^{-1} = (ab + qN)M^{-1} + \alpha N, \tag{9}$$

which yields

$$\hat{r} \equiv abM^{-1} \pmod{N}. \tag{10}$$

The computed value $\hat{r}$ is less than M' and then has a valid RNS representation in base $\mathcal{B}'$. Actually, the conditions $\alpha < k$, $q < M$, and $ab < MN$ give $\hat{q} < (k+1)M$ and, thus, $\hat{r} < (k+2)N < M'$. However, in order to use Algorithm 1 within an exponentiation algorithm (see Section 3), we must be able to reuse the output $\hat{r} < (k+2)N$ as inputs in $\mathrm{MM}(\hat{r}, \hat{r}, N)$. The condition $ab < MN$ of our algorithm implies $(k+2)^2 N^2 < MN$, which rewrites:

$$(k+2)^2 N < M. \tag{11}$$

If N is a 1,024-bit number and if we use 32-bit moduli, we need base $\mathcal{B}$ to be of size $k \geq 33$. In fact, condition (11) is verified as soon as $k \geq 34$.

As we shall see later, the second base extension requires the knowledge of $\hat{q}$ for an additional modulus. This is done by extending $\hat{q}$ using (8) for a redundant modulus m_r. The computational steps are given in Algorithm 2, shown in Fig. 2.

Compared to previous methods, our approach offers the advantage that it does not require the exact computation α in (7). This is a major difference from [12] and [7], where a dedicated hardware, called the cox unit in [7], is used to evaluate a real approximation of α.

As in [7], we evaluate the cost of our algorithms in terms of elementary operations which, in this case, is a modular multiplication of size of the operands; for instance, 32-bit numbers. The first base extension then requires $k^2 + 2k$ modular multiplications. An interesting implementation option is to choose a power of 2 for the redundant modulus m_r, which reduces the cost of the first base extension to $k^2 + k$.

4.2 Second Base Extension

For the second base extension, we use a different algorithm due to Shenoy and Kumaresan [16], described in Algorithm 3 shown in Fig. 3. As previously, we first evaluate in parallel, for all $j = k+1 \ldots 2k$, the values

$$\xi_j = \hat{r}_j \left|M_j'^{-1}\right|_{m_j} \bmod m_j. \tag{12}$$

Again from (1), we have

$$\hat{r} = \sum_{j=k+1}^{2k} \xi_j M_j' - \beta M', \tag{13}$$

where $\beta < k$. Once β is known, we can extend $\hat{r}$ back in base $\mathcal{B}$ by evaluating, in parallel, for all $i = 1 \ldots k$,

$$\left|\hat{r}\right|_{m_i} = \left| \sum_{j=k+1}^{2k} \xi_j \left|M_j'\right|_{m_i} - \left|\beta M'\right|_{m_i} \right|_{m_i}. \tag{14}$$

In order to compute β, we have to know the value of $\hat{r}$ for an additional modulus. This is done by evaluating $\hat{r}$ (Step 4 of Algorithm 1) for the redundant modulus m_r for which we have computed $\hat{q}_r$ in the first base extension. From (13), we have

$$\begin{aligned} \beta M' &= \sum_{j=k+1}^{2k} \xi_j M_j' - \hat{r}, \\ \left|\beta M'\right|_{m_r} &= \left| \sum_{j=k+1}^{2k} \xi_j \left|M_j'\right|_{m_r} - \left|\hat{r}\right|_{m_r} \right|_{m_r} \\ \left|\beta\right|_{m_r} &= \left| \left|M'^{-1}\right|_{m_r} \left(\sum_{j=k+1}^{2k} \xi_j \left|M_j'\right|_{m_r} - \left|\hat{r}\right|_{m_r} \right) \right|_{m_r}. \end{aligned}$$

Since $\beta < k$, choosing $m_r \geq k$ ensures $\beta < m_r$ and (15) gives the correct result.

$$\beta = \left| \left|M'^{-1}\right|_{m_r} \left(\sum_{j=1}^{k} \xi_j \left|M_j'\right|_{m_r} - \left|\hat{r}\right|_{m_r} \right) \right|_{m_r}. \tag{15}$$

Algorithm 3 : Second RNS base extension
Input : $(\hat{r}_{k+1}, \ldots, \hat{r}_{2k})$ and $\hat{r}_r$, the RNS representation of $\hat{r}$ in $\mathcal{B}' \cup \{m_r\}$
Output : $(\hat{r}_1, \ldots, \hat{r}_k)$, the RNS representation of $\hat{r}$ in $\mathcal{B}$.

1: $\xi_j = \hat{r}_j |M_j'^{-1}|_{m_j} \bmod m_j$, [in // for $j = k+1 \ldots 2k$]
2: $t_0 = 0$
3: **for** $j = 1 \ldots k$ **do**
4: $\quad t_j = (t_{j-1} + \xi_{k+j} |M_j'|_{m_r}) \bmod m_r$
5: $\beta = |M'^{-1}|_{m_r} (t_k - |\hat{r}|_{m_r}) \bmod m_r$
6: $t_0 \leftarrow 0$
7: **for** $j = 1 \ldots k$ **do**
8: $\quad t_j = (t_{j-1} + \xi_{k+j} |M_j'|_{m_i}) \bmod m_i$, [in // for $i = 1 \ldots k$]
9: $\hat{r}_i = (t_k - |\beta M'|_{m_i}) \bmod m_i$ [in // for $i = 1 \ldots k$]

Fig. 3. Algorithm 3.

The sum in (15) requires a total cost of $2k+1$ modular multiplications (Steps 1 to 5 of Algorithm 3, distributed as follows:

- k to compute the values $\xi_j = |\hat{r}_j |M_j'^{-1}|_{m_j}|_{m_j}$,
- k for each $||M_j'|_{m_r} \xi_j|_{m_r}$,
- and 1 for the multiplication by $|M'^{-1}|_{m_r}$.

Since the values ξ_j have already been computed for all j, the number of operations needed to evaluate (14)—which corresponds to Steps 7 to 9 of Algorithm 3—is $k+1$ for each modulus m_i in $\mathcal{B}$, i.e., $k(k+1)$. This results in a total of $k^2 + 3k + 1$ modular multiplications. If m_r is a power of 2, the evaluation of β requires k operations and the total cost of the second base extension reduces to $k + k(k+1) = k^2 + 2k$.

Another option to reduce this cost is to precompute the constants $|M_j' |M_j'^{-1}|_{m_j}|_{m_r}$. Since those values only depends on $\mathcal{B}'$, they can be stored in tables. This reduces the number of modular multiplications to $k(k+2)$, but we must choose $m_r \geq k\tilde{m}$, where $\tilde{m} = \max\{m_j,\ j = k+1 \ldots 2k\}$, to obtain the correct value of β with (15).

In Table 1, we compare our RNS Montgomery multiplication algorithm with the previously proposed method by Kawamura et al. [7] that has itself been demonstrated to be more efficient than another efficient solution proposed in 1995 by Posch and Posch [12]. Without considering the option proposed in the previous paragraph, we obtain similar results.

TABLE 1
Number of Modular Multiplications of
Two RNS Montgomery Multiplication Algorithms

	Our algorithm	Kawamura & al.
Lines 1 and 3 of algo MM	$5k$	$5k$
First base extension	$k^2 + k$	$k^2 + 2k$
Second base extension	$k^2 + 2k$	$k^2 + 2k$
Total	$2k^2 + 8k$	$2k^2 + 9k$

5 RSA Implementation

At the end of the classical Montgomery multiplication, the result is less than $2N$. A subtraction by N is necessary if the result is greater than N. As mentioned previously, in our RNS version, the output $\hat{r}$ of algorithm MM is less than $(k+2)N$. For the same reason, a correction step may be needed. The next two textbook implementations of RSA address this problem. The first version uses conversions to and from the residue number system. We present this version for completeness and since it allows for more freedom in the implementation of the RSA protocol. The second version, without conversion, is a lot more attractive and represents the real novelty of this paper.

5.1 RSA with Conversions

A solution is to perform the first base extension of the very last Montgomery multiplication exactly. This last call to $\mathrm{MM}(x^a M \bmod N, 1, N)$ is the one which suppresses the Montgomery constant M and gives the final result $x^a \bmod n$. This can be done rather efficiently via the Mixed Radix System (MRS) as suggested in 1959 by Garner [6]. For each m_j, we evaluate

$$|q|_{m_j} = \left| t_1 + t_2 m_1 + \cdots + t_k m_1 \ldots m_{k-1} \right|_{m_j}, \tag{16}$$

where

$$\begin{aligned} t_1 &= |q|_{m_1} = q_1 \\ t_2 &= |(q_2 - t_1) c_{12}|_{m_2} \\ &\vdots \\ t_k &= |(\cdots (q_k - t_1) c_{1k} - \cdots - t_{k-1}) c_{(k-1)k}|_{m_k} \end{aligned}$$

and $c_{ij} = |m_i^{-1}|_{m_j}$. This method requires the precomputation of $k(k-1)/2$ constants c_{ij}. As with the original Montgomery multiplication, the result is less than $2N$. A subtraction by N may be required.

5.2 RSA without Conversion

An easy way to consider the binary message $x = \sum_i x_i 2^i$ we want to encrypt as a valid RNS number is to split it into blocks whose sizes depend on the sizes of the moduli of the base $\mathcal{B}$. For example, if $\mathcal{B}$ is composed of 32-bit modulus, splitting x in blocks of at most 31 bits makes it possible to consider each block as a value $x_i < m_i$ and provides what we call a valid RNS number for the base $\mathcal{B}$.

In order to correct the value we obtain at the end of the exponentiations, let us consider one of the moduli of $\mathcal{B}$, say the last one, as a special modulus. This special modulus plays a crucial role in the correction step of our algorithm and, as a consequence, in its validity. Once the message x is expressed in the RNS form $(x_1, \ldots, x_{k-1})$ for a set of $k-1$ moduli, we consider the integer which RNS representation in the base $\mathcal{B} = (m_1, \ldots, m_{k-1}, m_k)$ is:

$$(x_1, \ldots, x_{k-1}, 0). \tag{17}$$

By doing this, we are constructing an integer less than M that we only know in RNS and not in any radix form, which is a multiple of m_k.

To encrypt a message with RSA, we compute $y = x^b \bmod N$. As we have seen previously, our algorithm does not return y exactly, but

$$\hat{y} = x^b \bmod N + \beta N, \quad \text{with} \quad \beta < k+2. \tag{18}$$

At the end of the decryption step, we must obtain a value z congruent to x modulo N and less than N. Our algorithm returns

$$\hat{z} = x^{ba} \bmod N + \gamma N, \quad \text{with} \quad \gamma < k+2. \tag{19}$$

The returned value $\hat{z}$ satisfies $\hat{z} \equiv x (\bmod N)$, but it might be greater than N.

In order to correct this result if necessary, we are going to use the extra information we have thanks to the special modulus m_k. Since we know that the correct result z must be a multiple of m_k, its RNS representation must be $z = (z_1, \ldots, z_{k-1}, 0)$. In other words, we are looking for a value $z < M$ such that $z \equiv x \bmod N$ and $z \equiv 0 \bmod m_k$. From the Chinese remainder theorem, this value z maps back to a unique integer within the interval $[0, m_k N)$. Since $x < M$ by construction, $M \leq m_k N$ is a necessary condition in order for z to have the same RNS representation as x. The algorithm we propose is correct under the slightly more restrictive condition which gives the final conditions for our algorithm:

$$(k+2)^2 N < M \leq (m_k - (k+2))N. \tag{20}$$

Let us consider that the result we have obtained after the second exponentiation has the following RNS representation: $\hat{z} = (\hat{z}_1, \ldots, \hat{z}_{k-1}, \hat{z}_k)$. If $\hat{z}_k = 0$, then (20) ensures that $\hat{z} = x$ and no correction is needed. In the other cases, the solution we propose consists of looking for a value t such that $\tilde{z} = \hat{z} + tN$ is a multiple of m_k less than M and thus satisfies $\hat{z} = x$. The following algorithm computes t in two steps.

1. We compute $t' = \hat{z}_k(-n_k^{-1}) \bmod m_k$, where $n_k = N \bmod m_k$ is given by the RNS representation of N.
2. If $t' < m_k - (k+2)$, then $t = t'$ else $t = t' - m_k$.

Proof. Since t is computed such that $\hat{z} + tN = x$, the sign of t depends on the order between x and $\hat{z}$. Furthermore, since $t \equiv t' (\bmod m_k)$, we have $-m_k < t < m_k$. If $x > \hat{z}$, then $t > 0$ and then $t = t' > 0$. Otherwise, if $x < \hat{z}$, then $t < 0$ and then $t = t' - m_k < 0$.

Since we only know x and $\hat{z}$ in RNS, it is not possible to decide whether x is less or greater than $\hat{z}$. We must find another way of comparing them. If $x < \hat{z}$, then Algorithm 1 gives $0 < x < (k+2)N$ and $0 < \hat{z} < (k+2)N$. Hence, $|x - \hat{z}| < (k+2)N \Rightarrow |t|N < (k+2)N$. This implies $m_k - t' < k+2 \Rightarrow t' > m_k - (k+2)$. The negation of this implication results in the first condition of our algorithm: If $t' < m_k - (k+2)$, then $x > \hat{z}$ and then $t = t'$. Conversely, if $t' > m_k - (k+2)$, then the only solution is $t = t' - m_k$. □

It is still important to note that the validity of our algorithm is based on an important assumption. Since the message is always considered in RNS and never converted back in binary, even for the transmission, it is clear that both parties must choose a common set of RNS bases, in particular, the same value for m_k. This exchange can be a part of the protocol initialization between the two communicants and is beyond the scope of this paper.

6 Example

We illustrate our algorithm with an implementation of RSA with small values. Let us define the classical RSA parameters.

- $p = 479$, $q = 317$, $n = pq = 151,843$.
- $\phi(n) = 151,048$, $a = 173$, $b = 79,453 = a^{-1} \bmod \phi(n)$.

We also define the RNS bases where the last element of $\mathcal{B}$ is the special modulus $m_k = 73$.

- $\mathcal{B} = (3, 7, 13, 17, 29, 73)$, $M = 9,824,997$,
- $\mathcal{B}' = (5, 11, 19, 23, 31, 37)$, $M' = 27,568,145$.

For the second base extension, we use the redundant modulus $m_r = 8$. Let us first verify the conditions of our algorithm. We have $(k+2)^2 N = 9,717,952 < M, M'$ and

$$M \leq (m_k - (k+2))N = 9,869,795.$$

Let $x = 11010101001101$, the binary representation of the message we want to encrypt. Instead of converting it from its binary representation to its RNS form in base $\mathcal{B}$, we split it into blocks to get a valid RNS number. In this example, the moduli do not all have the same size. We consider an irregular splitting of x. If we express the moduli set in binary we have $\mathcal{B} = (11, 111, 1101, 10001, 11101, 1001001)$, a valid splitting of x is $x = 1\ 10\ 101\ 0100\ 1101$. The size of each block is 1 less than the size of the corresponding modulus. In a real implementation, we can simply consider 32-bit moduli and split x into 31-bit blocks. The RNS message we are going encrypt is $X = (1, 2, 5, 4, 13, 0)_{\mathcal{B}}$. The first operation consists of extending X is the base $\mathcal{B}'$, which must be performed exactly (for example, via the mixed radix representation suggested in the implementation with conversion in Section 5.1). This gives

$$X = (1, 2, 5, 4, 13, 0)_{\mathcal{B}},\ (3, 4, 2, 7, 18, 7)_{\mathcal{B}'},$$

The first step of the exponentiation is the call to $\mathrm{MM}(X, M^2 \bmod N, N)$ which puts X in the Montgomery notation $X' = XM \bmod N$. We get:

$$X' = (1, 2, 12, 16, 0, 10)_{\mathcal{B}},\ (4, 10, 3, 11, 3, 10)_{\mathcal{B}'}.$$

The full exponentiation $Y = X^b M \bmod N$ results in

$$Y = (2, 3, 6, 6, 17, 10)_{\mathcal{B}},\ (2, 4, 0, 11, 3, 27)_{\mathcal{B}'}.$$

At this step, we are still in the Montgomery notation. We send this value to the other party, who decrypts it. Since the transmitted value Y is already in the Montgomery notation, the first call to MM to get into this form can be omitted. We directly perform the exponentiation by computing $Z = Y^a \bmod N$. The last call to $\mathrm{MM}(Z, 1, N)$ removes the Montgomery constant and gives

$$Z = (1, 2, 7, 8, 5, 10)_{\mathcal{B}},\ (0, 3, 12, 1, 6, 1)_{\mathcal{B}'}.$$

Since $Z_k = 10 \neq 0$, we do not have a multiple of $m_k = 73$ and, thus, a final correction step is needed. We compute $t' = Z_k(-N_k^{-1}) \bmod m_k = 21,$. Since $t' = 21 < 73 - (6+2) = 65$, we have $t = t' = 21$. We compute the final result in RNS

$$Z + 21N = (1, 2, 5, 4, 13, 0)_{\mathcal{B}}.$$

If we express it back in binary according to the same splitting and without considering the last residue, we retrieve the original message $x = 11010101001101$.

7 Conclusions

We have presented a new implementation of Montgomery multiplication in RNS and have shown its efficiency toward a new full RNS implementation of RSA. The message is never considered as a binary number, but rather in RNS during the whole process. Thus, no conversion is needed. This approach requires both parties to agree on a set of RNS parameters beforehand. Compared to previously proposed solutions [12], [7], [11], our algorithms require approximately the same number of elementary operations and use only integer arithmetic (no rational approximations of α in (7) are computed). Furthermore, the conditions on our parameters are easier to satisfy than their counterparts in these other methods. The parallel nature of RNS arithmetic can also offer potential advantage in the resistance to side channel attacks. This is currently a work in progress for our team [2].

Acknowledgments

This work was supported by the French Ministry of Education and Research under the ACI 2002, "OpAC, Opérateurs arithmétiques pour la Cryptographie," grant number C03-02. The authors would like to thank the anonymous reviewers for their very careful reading and useful comments.

References

[1] J.-C. Bajard, L.-S. Didier, and P. Kornerup, "Modular Multiplication and Base Extension in Residue Number Systems," *Proc. 15th IEEE Symp. Computer Arithmetic,* N. Burgess, ed., pp. 59-65, June 2001.

[2] J.-C. Bajard, L. Imbert, and P.-Y. Liardet, "Leak Resistant Arithmetic," LIRMM, Research Report 03021, Oct. 2003.

[3] E.F. Brickell, "A Survey of Hardware Implementation of RSA," *Advances in Cryptology, Proc. CRYPTO '89,* pp. 368-370, 1990.

[4] Ç.K. Koç, T. Acar, and B.S. Kaliski Jr., "Analyzing and Comparing Montgomery Multiplication Algorithms," *IEEE Micro,* vol. 16, no. 3, pp. 26-33, June 1996.

[5] S.E. Eldridge and C.D. Walter, "Hardware Implementation of Montgomery's Modular Multiplication Algorithm," *IEEE Trans. Computers,* vol. 42, no. 6, pp. 693-699, June 1993.

[6] H.L. Garner, "The Residue Number System," *IRE Trans. Electronic Computers,* vol. 8, pp. 140-147, June 1959.

[7] S. Kawamura, M. Koike, F. Sano, and A. Shimbo, "Cox-Rower Architecture for Fast Parallel Montgomery Multiplication," *Advances in Cryptology, Proc. EUROCRYPT 2000,* pp. 523-538, May 2000.

[8] D.E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms,* third ed. Addison-Wesley, 1997.

[9] A.J. Menezes, P.C. Van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography.* Boca Raton, Fla.: CRC Press, 1997.

[10] P.L. Montgomery, "Modular Multiplication without Trial Division," *Math. Computation,* vol. 44, no. 170, pp. 519-521, Apr. 1985.

[11] H. Nozaki, M. Motoyama, A. Shimbo, and S. Kawamura, "Implementation of RSA Algorithm Based on RNS Montgomery Multiplication," *Proc. Cryptographic Hardware and Embedded Systems (CHES 2001),* pp. 364-376, Sept. 2001.

[12] K.C. Posch and R. Posch, "Modulo Reduction in Residue Number Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 6, no. 5, pp. 449-454, May 1995.

[13] J.-J. Quisquater and C. Couvreur, "Fast Decipherment Algorithm for RSA Public-Key Cryptosystem," *IEE Electronics Letters,* vol. 18, no. 21, pp. 905-907, Oct. 1982.

[14] R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems," *Comm. ACM,* vol. 21, no. 2, pp. 120-126, Feb. 1978.

[15] M. Shand and J. Vuillemin, "Fast Implementation of RSA Cryptography," *Proc. 11th IEEE Symp. Computer Arithmetic,* E.E. Swartzlander, M.J. Irwin, and G.A. Jullien, eds., pp. 252-259, June 1993.

[16] A.P. Shenoy and R. Kumaresan, "Fast Base Extension Using a Redundant Modulus in RNS," *IEEE Trans. Computers,* vol. 38, no. 2, pp. 292-297, Feb. 1989.

[17] N. Szabo and R.I. Tanaka, *Residue Arithmetic and Its Application to Computer Technology,* 1967.

[18] F.J. Taylor, "Residue Arithmetic: A Tutorial with Examples," *Computer,* vol. 17, no. 5, pp. 50-62, May 1984.