A Full RNS Implementation of RSA

Laurent Imbert, Jean-Claude Bajard

A Full RNS Implementation of RSA

Laurent Imbert **Jean-Claude Bajard**

Research Report – LIRMM No 02068

May 2002

Submitted to

IEEE Transactions on Computers

Special Issue on Cryptographic Hardware and Embedded Systems

Corresponding author:

Laurent Imbert

Laboratoire d'**I**nformatique de **R**obotique et de **M**icroélectronique de **M**ontpellier

161 rue Ada - 34392 Montpellier cedex 5 - France

Tel. (33) 467 418 549 – Fax. (33) 467 418 500

`Laurent.Imbert@lirmm.fr`

A Full RNS Implementation of RSA

Laurent Imbert Jean-Claude Bajard

LIRMM - CNRS

Montpellier – France

Abstract

In this paper we propose an efficient hardware implementation of RSA based on the Residue Number System (RNS) which allows for fast parallel arithmetic. We propose RNS versions of Montgomery multiplication and exponentiation algorithms and illustrate the efficiency of our approach with two implementations of RSA. For the very first time a very attractive conversion-free RSA encryption/decryption scheme is proposed. Compared to previously proposed methods our solution requires less elementary operations and is very promising.

Keywords: RSA, Montgomery multiplication, RNS, base extension.

1 Introduction

During the last decade fast hardware implementations of public-key cryptosystems have been widely studied [2, 3, 12] while confidentiality and security requirements were becoming more and more important. From this time, key-length has kept growing. Nowadays it is assumed that a 1024-bit key-length makes a reasonable choice for RSA [11] and current analysis predict that 2048-bit or 4096-bit key will become the standard in a near future. The ability to perform fast arithmetic on large integers is then still a major issue for the implementation of public key cryptography and digital signature, particularly from an hardware design viewpoint.

Different approaches have been proposed to accelerate the implementation of RSA. For the decipherment 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) [10]. 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 step. More recently, other CRT based solutions have been proposed [9, 4, 8, 1]. They all use a quite similar version of Montgomery multiplication based on the Residue Number System (RNS) [15] which is well adapted to fast parallel arithmetic. The complexity of all those algorithms mainly depends on two RNS base extensions that are required for each modular multiplication.

In this paper we detail the implementation and complexity of an efficient Montgomery multiplication algorithm based on the Residue Number System (RNS) previously proposed by the authors in [1]. This algorithm uses two different techniques for the first and second base extensions and requires less elementary operations than other previous methods. We illustrate the

efficiency of our approach with two full textbook implementations of RSA in RNS. In the first implementation (RSA with conversions) we consider the message we want to encrypt as a number, convert it in RNS, perform the exponentiation and convert the result back to a classical binary notation. The second solution we propose is a lot more attractive since it does not require any conversion to and from the residue number system. We directly consider the message as a value represented in RNS. For this technique to work, both parties must have agreed beforehand on the RNS bases since the message will transit between them in its RNS form.

2 The Residue Number System

In a residue number system (RNS) [14, 15, 5] numbers are represented according to a base $\mathcal{B} = (m_1, m_2, \ldots, m_k)$ of relatively prime moduli which size k is its number of elements. An integer x is represented by the sequence $(x_1, x_2, \ldots, x_k)$ of positive integers, where $x_i = x \mod m_i,\ i = 1...k$. The Chinese Remainder Theorem (CRT) ensures the uniqueness of this representation within the range $0 \leqslant x < M$, where $M = \prod_{i=1}^{k} m_i$. The constructive proof of this theorem can be used to convert x back from its residue representation:

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

where $M_i = \dfrac{M}{m_i}$ and $\left|M_i^{-1}\right|_{m_i}$ is the inverse of M_i modulo m_i. In the following of the paper we shall use $|x|_m$ to denote the value $(x \mod m)$.

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 have

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

On the other hand, one of the disadvantages of this representation is that we can not easily decide whether $(x_1, \ldots, x_k)$ is greater[1] than $(y_1, \ldots, y_k)$. Consequently divisions are very difficult to perform[2] and overflows that can occurred during computations are not easily detected.

From a cryptographic viewpoint, these difficulties are not to be considered as real drawbacks. In public key cryptography most of the algorithms perform the computations in a finite field or ring which eliminates the overflow problem. Moreover, they do not require comparisons and divisions. Modular reduction (the computation of $x \mod m$), multiplication ($xy \mod m$) and exponentiation ($x^y \mod m$) are the most important operations. They can be efficiently computed without division using Montgomery's algorithms [7].

3 Modular exponentiation

Let us briefly recall the principles of Montgomery's techniques. Given $R > N$, $\gcd(R, N) = 1$ and $0 \leqslant x < RN$, Montgomery reduction technique evaluates $xR^{-1} \mod N$ by computing the value $q < R$ such that $x + qN$ is a multiple of R. Hence $y = (x + qN)R^{-1}$ is performed without division and verifies $y < 2N$ and $y \equiv xR^{-1} \pmod N$. The final result $x \mod N$ can be computed using the same algorithm with inputs y and $(R^2 \mod N)$. In the same way Montgomery modular multiplication algorithm computes $xyR^{-1} \mod N$. For practical implementations the Montgomery constant R

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

[2] Exact division is easy since it simply consists of a multiplication by the inverse.

is chosen as a power of 2 to reduce the multiplication by R^{-1} to simple shifts. A more detailed discussion on Montgomery reduction and multiplication algorithms can be found in [6].

In the next sections we present a 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, also known as the square-and-multiply algorithm. Assuming we aim at computing $x^a \mod n$, the input x is first transformed into $x' = xR \mod n$, often called the Montgomery representation. This is done using a first Montgomery multiplication with x and $(R^2 \mod n)$ as inputs. This representation has the advantage of being stable over Montgomery multiplication:

$$x' \times y' \mod N = xR \times yR \times R^{-1} \mod n = xyR \mod n.$$

The output of the exponentiation, $z' = x^a R \mod n$, is converted back into the expected value $z = x^a \mod n$ using a last call to Montgomery multiplication with z' and 1 as inputs. The efficiency of this exponentiation then clearly relies on the ability to perform the Montgomery modular multiplication.

4 RNS Montgomery multiplication

In the RNS version of Montgomery multiplication algorithm we use

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

as the Montgomery constant (R) and we compute

$$r = abM^{-1} \mod N,$$

where r, a, b and N are represented in RNS according to a predefined base $\mathcal{B}$. As in the classical Montgomery algorithm we are looking for a number q such that $(ab + qN)$ is a multiple of M, which will allow us to perform the division $(ab + qN)/M$. Since this division is exact it reduces to the product of $(ab+qN)$ with the inverse of M. Unfortunately the inverse of M does not exist modulo M, which force us to use an extended base $\mathcal{B}' = (m'_1, \ldots, m'_l)$ with $\gcd(m'_i, M) = 1$ for $i = 1 \ldots l$, $M' = \prod_i m'_i$ and $M' \geqslant M$. Another reason for which we need this second base comes form the fact that the dynamic range provided by the base $\mathcal{B}$ is not large enough to represent $(ab+qN) \geqslant M$. For simplicity we will consider in the rest of the paper that both $\mathcal{B}$ and $\mathcal{B}'$ are of the same size k and we shall denote them

$$\mathcal{B} = (m_1, \ldots, m_k) \quad \text{and} \quad \mathcal{B}' = (m_{k+1}, \ldots m_{2k}).$$

Now, in order to determine q, we use the fact that $r = (ab + qN)$ is a multiple of M, which implies that its representation in the base $\mathcal{B}$ is merely composed of 0:

$$r \equiv 0 \pmod{m_i}, \quad \text{for } i = 1 \ldots k.$$

The RNS representation of q is then given by the solutions of the equations

$$(a_i b_i + q_i n_i) \equiv 0 \pmod{m_i} \quad \forall i = 1 \ldots k, \tag{2}$$

which gives

$$q_i = a_i b_i \,|\, -n_i^{-1}|_{m_i} \quad \forall i = 1 \ldots k. \tag{3}$$

As pointed out previously we can not compute r in base $\mathcal{B}$ but only in base $\mathcal{B}'$. Moreover, one can remark that since r is solely composed of 0, multiplication by M^{-1} would have no effect. Before we evaluate $r = (ab + qN)$ we must extend q in base $\mathcal{B}'$. We shall discuss this first base extension in detail in the section 4.1. We then compute $r = (ab + qN)$ in base $\mathcal{B}'$

and extend the result back to the base $\mathcal{B}$ for future use. The second base extension is discussed in section 4.2. Algorithm 1 clarifies the situation. It computes the product $abM^{-1} \mod N$, with a, b, N represented in RNS in both bases $\mathcal{B}$ and $\mathcal{B}'$.

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 such that $0 < (k+2)^2 N < M$ and $\gcd(N, M) = 1$; two positive integers a, b represented in RNS in both bases, with $ab < MN$.

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

1: $q \leftarrow (a \times b) \times (-N^{-1})$ in $\mathcal{B}$

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

3: $\hat{r} \leftarrow (a \times b + \hat{q} \times N) \times M^{-1}$ in $\mathcal{B}'$

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

Instructions 1 and 3 consist in full RNS operations and can be performed in parallel. As a consequence the complexity of the algorithm clearly relies on the two base extensions on lines 2 and 4. This algorithm is very similar to those of Posch and Posch [9] and Kawamura and al. [4] which also require two base extensions. In their approaches, the same technique is applied for both the first and second base extensions.

We propose a different solution which do not use the same algorithm for the two extensions on lines 2 and 4. We show in section 6 that our choice requires less elementary operations than those previously proposed methods.

4.1 First base extension

The instruction in line 2 consists of converting q obtained in its RNS form $(q_1, \ldots, q_k)$ in the base $\mathcal{B}$ to its RNS representation in base $\mathcal{B}'$. If we evaluate the sum in (1) by first computing the values

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

we have

$$q = \sum_{i=1}^{k} M_i \sigma_i - \alpha M. \tag{4}$$

where α is an integer less than k. But instead of extending the exact value of q in base $\mathcal{B}'$, we only extend

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

by only computing the residues

$$\hat{q}_j = \Big| \sum_{i=1}^{k} \left| M_i \right|_{m_j} \sigma_i \Big|_{m_j}, \quad \forall j = k+1 \ldots 2k. \tag{5}$$

Compared to previous methods the advantage comes from the fact that we do not need to compute the value of α in (4). This is the first difference with [9] and [4] where a rational approximation of α is evaluated.

In instruction 3 we compute in the base $\mathcal{B}'$ the value

$$\hat{r} = (ab + \hat{q}N)M^{-1} = (ab + qN)M^{-1} + \alpha N < M'. \tag{6}$$

After instruction 3 we are then provided with a value $\hat{r}$ such that

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

which is sufficient for our purpose. The conditions $\alpha < k$, $q < M$ and $ab < MN$ gives $\hat{q} < (k+1)M$ and thus $\hat{r} < (k+2)N < M'$.

In order to use algorithm MM within the exponentiation algorithm (see section 3), we must be able to compute $x^2 \mod N$, where x is the output

of algorithm MM verifying $x < (k+2)N$. The condition $ab < MN$ then implies $(k+2)^2N^2 < MN$ which rewrites:

$$(k+2)^2N < M. \tag{7}$$

If N is a 1024-bit number and if we use 32-bit modulus, we need base $\mathcal{B}$ to be of size $k \geqslant 33$. In fact condition (7) is verified as soon as $k \geqslant 34$.

As we shall see further for the second base extension, we need to know the value of $\hat{q}$ for an additional modulus. This is done by extending $\hat{q}$ using (5) for a redundant modulus m_r, which gives $\hat{q}_r = \hat{q} \mod m_r$.

As in [4] we evaluate the cost of our algorithms in terms of elementary operations which, in this case, is a modular multiplication of size the size of the modulus m_i and operands ($q_i, \left|M_i^{-1}\right|_{m_i}$ and $\left|M_i\right|_{m_j}$) ; for instance 32-bit numbers. This easily generalizes to other choices for the bases ; for example smaller bases with larger modulus, say 64-bit numbers. The first base extension then requires $k^2 + 2k$ elementary operations.

An interesting implementation option is to choose a power of 2 for the redundant modulus m_r[3]. Since the reduction modulo a power of 2 is a lot easier than for any modulus, we can omit them when counting the elementary operations. This 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 [13]. As previously, we first evaluate

$$\xi_j = \hat{r}_j \left|M_j'^{-1}\right|_{m_j} \mod m_j, \quad \forall j = k+1 \ldots 2k,$$

[3]Of course, since all the moduli have to be relatively prime, we can only choose one even modulus.

and we consider the sum given by the CRT reconstruction algorithm in equation (1)

$$\hat{r} = \sum_{j=1}^{k} M'_j \xi_j - \beta M', \tag{8}$$

where $\beta < k$. Once β is known we can extend $\hat{r}$ back in base $\mathcal{B}$ by evaluating

$$|\hat{r}|_{m_i} = \left| \sum_{j=1}^{k} \left|M'_j\right|_{m_i} \xi_j - \left|\beta M'\right|_{m_i} \right|_{m_i}, \quad \forall i = 1...k. \tag{9}$$

In order to compute β we have to know the value of $\hat{r}$ for an additional modulus. This is done by evaluating $\hat{r}$ (line 3 of algorithm 1) for the redundant modulus m_r for which we have computed $\hat{q}_r$ in the first base extension. From eq. (8) we have

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

Since $\beta < k$, choosing $m_r \geqslant k$ ensures $\beta < m_r$ and equation (10) gives the correct result.

$$\beta = \left| \left|M'^{-1}\right|_{m_r} \left(\sum_{j=1}^{k} \left|M'_j\right|_{m_r} \xi_j - |\hat{r}|_{m_r} \right) \right|_{m_r}. \tag{10}$$

The sum in eq. (10) requires a total cost of $2k + 1$ elementary modular multiplications distributed as follow: k to compute the values $\xi_j = \left|\hat{r}_j \left|M'^{-1}_j\right|_{m_j}\right|_{m_j}$, k for each $\left|\left|M'_j\right|_{m_r} \xi_j\right|_{m_r}$, and one for the multiplication by $\left|M'^{-1}\right|_{m_r}$. Since the values ξ_j have already been computed for all j, the number of operations needed to evaluate eq. (9) is $k + 1$ for each modulus m_i in $\mathcal{B}$, which results in $k^2 + 3k + 1$ elementary modular multiplications. If m_r is a power of 2 the cost of the second base extention reduces to $k + k(k + 1) = k^2 + 2k$.

In the next sections we illustrate our algorithm with two textbook RSA implementations. The first version uses conversions to and from the residue number system. We present this version for completeness and since it allows more freedom in the implementation of the RSA protocol. The second version without conversion is a lot more attractive.

5 RSA Implementations

At the end of the classical Montgomery multiplication the result is less than $2N$. A correction step is necessary if the result is greater than N and in this case a subtraction by N gives the correct value. 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.

In the next paragraphs we present two implementations of RSA which address this problem.

5.1 RSA with conversions

To address the final correction step, a straightforward solution consists of subtracting N from $\hat{r}$ until $\hat{r} < N$. This technique can not be performed efficiently in RNS since it requires a comparison after each subtraction. Furthermore, for practical values of k the overhead is important.

Another solution, however, is to perform, for the very last Montgomery multiplication, (i.e. the last call to MM($x^a M \mod N$,1,N) which suppress the Montgomery constant M and gives the expected result $x^a \mod n$), the first base extension exactly. This can efficiently be done via the Mixed Radix System (MRS) has suggested in 1967 by Szabo and Tanaka [14]. For each

m_j, we evaluate

$$|q|_{m_j} = \Big|t_1 + t_2 m_1 + t_3 m_1 m_2 + \cdots + t_k m_1 ... m_{k-1}\Big|_{m_j}, \tag{11}$$

where

$$\begin{aligned}
t_1 &= q \mod m_1 = q_1 \\
t_2 &= \big(q_2 - t_1\big)c_{12} \mod m_2 \\
&\vdots \\
t_k &= \big(\cdots\big(\big(q_k - t_1\big)c_{1k} - t_2\big)c_{2k} - \cdots - t_{k-1}\big)c_{(k-1)k} \mod m_k
\end{aligned}$$

and $c_{ij} = m_i^{-1} \mod m_j$.

Example

We illustrate our algorithm with an implementation of RSA with small values. We first define the classical RSA parameters. Let us define 2 prime numbers $p = 479$ and $q = 317$. We compute $n = pq = 151843, \phi(n) = (p-1)(q-1) = 151048$; find a value a such that $\gcd(a, \phi(n)) = 1$, and compute $b = a^{-1} \mod n$ using the extended Euclid algorithm: $a = 173, b = 79453$. The couple (b, n) is the public key. $a, p, q, \phi(n)$ are kept secret.

Now, we define the RNS parameters: the bases $\mathcal{B} = (3, 7, 13, 19, 29, 67)$ and $\mathcal{B}' = (5, 11, 17, 23, 31, 37)$, the redundant modulus $m_r = 8$. The dynamic ranges $M = 10078341$ and $M' = 24666235$ provided by the two bases satisfy the conditions $(k+2)^2 n = 64n < M < M'$.

Let $x = 132976$ be the message less than n we want to encrypt. The first step consists of converting the values x and n in their RNS form X and N [4]. We have:

$$X = (1, 4, 12, 14, 11, 48)_{\mathcal{B}}, \quad (1, 8, 2, 13, 17, 35)_{\mathcal{B}'}$$

[4] In the examples, we use uppercases for RNS numbers and lowercases for their corresponding value.

$$N = (1, 6, 3, 14, 28, 21)_{\mathcal{B}}, \quad (3, 10, 16, 20, 5, 32)_{\mathcal{B}'}$$

We compute in RNS the encrypted value $Y = X^b \mod N$:

$$Y = (0, 6, 7, 14, 12, 3)_{\mathcal{B}}, \quad (3, 2, 1, 5, 18, 8)_{\mathcal{B}'},$$

which correspond in decimal to $y = 118593$. According to the RSA algorithm, we decrypt this value by computing $Z = Y^a \mod N$. The result in RNS is

$$Z = (1, 4, 12, 14, 11, 48)_{\mathcal{B}}, \quad (1, 8, 2, 13, 17, 35)_{\mathcal{B}'},$$

and correspond to the original message $z = 132976 = x$.

For this example we have used the same bases for both the encryption and decryption. If this situation occurs in a practical case it is important to note that we do not need to convert the encrypted value Y back in binary before transmission. We can directly send the RNS value Y to the other part.

5.2 RSA without conversion

An easy way to consider the message $x = \sum_i x_i 2^i$ we want to encrypt as a valid RNS number is to split it in blocs, which size depends on the size of the moduli of the base $\mathcal{B}$. For example if $\mathcal{B}$ is composed of 32-bit modulus, splitting x in blocs of at most 31 bits makes it possible to consider each bloc as a value $x_i < m_i$ and provides what we have just called a valid RNS number.

In order to correct the value obtained at the end of the exponentiations we consider the last modulus of $\mathcal{B}$ as a special modulus. This extended 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 the $k-1$ first modulus, we consider the number which

RNS representation in the base $\mathcal{B} = (m_1, \ldots, m_{k-1}, m_k)$ is $(x_1, \ldots, x_{k-1}, 0)$. By doing this we are constructing a number less than M, that we do not know explicitly, which is a multiple of m_k.

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

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

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

$$\hat{z} = x^{ba} \mod N + \gamma N$$

with $\gamma < k + 2$. The returned value $\hat{z}$ verifies $\hat{z} \equiv x \pmod N$ but it may be greater than N.

In order to correct this result if necessary, we use the extra information we have thanks to the last modulus m_k. We know that the result z must be a multiple of m_k. Thus its RNS representation should be

$$z = (z_1, \ldots, z_{k-1}, 0).$$

In practice we will almost always obtain a value

$$\hat{z} = (\hat{z}_1, \ldots, \hat{z}_{k-1}, \hat{z}_k),$$

with $z_k \neq 0$, which is not a multiple of m_k.

The solution we propose consists of looking for a value t such that $\tilde{z} = \hat{z} + tN$ is a multiple of m_k. We compute t by solving the equation $\hat{z}_k + tN \equiv 0 \pmod{m_k}$ which gives

$$t = \hat{z}_k(-n_k^{-1}) \mod m_k,$$

where $n_k = N \mod m_k$ is given by the RNS representation of N. We then compute $tN = (tn_1, \ldots, tn_{k-1}, tn_k)$ and $\tilde{z} = \hat{z} + tN$ in RNS to obtain

$$\tilde{z} = (\tilde{z}_1, \ldots, \tilde{z}_{k-1}, 0).$$

At this point we are provided with a value $\tilde{z}$ such that:

$$\tilde{z} \equiv x \pmod{N}$$

$$\tilde{z} \equiv x \pmod{m_k}$$

Then from the Chinese Remainder Theorem, it maps back to a unique number within the interval $[0, m_k N)$. Since $x = (x_1, \ldots, x_{k-1}, 0) < M$, taking $M \leqslant m_k N$ ensures that the computed RNS number $\tilde{z}$ has actually the same RNS representation than x. We have then obtained the correct result. This gives us the final conditions for our algorithm:

$$(k+2)^2 N < M \leqslant m_k N,$$

which implies $m_k > (k+2)^2$.

It is yet 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.

Example

We use the same parameters than in the previous example:

- $p = 479$, $q = 317$, $n = 151843$
- $\phi(n) = 151048$, $a = 173$, $b = 79453 = a^{-1} \mod \phi(n)$

The RNS bases are also identical ; the last element of $\mathcal{B}$ playing the role of the special modulus $m_k = 67$.

- $\mathcal{B} = (3, 7, 13, 19, 29, 67)$, $M = 10078341$,
- $\mathcal{B}' = (5, 11, 17, 23, 31, 37)$, $M' = 24666235$,

For the second base extension we use the redundant modulus $m_r = 8$.

Let us first verify the conditions of our algorithm:

$$(k+2)^2 N = 9717952 < M = 10078341 \leqslant m_k N = 10173481.$$

Let $x = 11010101001101$ the binary representation of the message we aim at encrypting. Instead of converting it from its binary representation to its RNS form in base $\mathcal{B}$, we split it in blocs such that it corresponds to a valid RNS number. In this example the modulus do not have all the same size which implies us to consider an irregular splitting of x. If we express the moduli set in binary we have $\mathcal{B} = (11, 111, 1101, 10011, 11101, 1000011)$. A valid splitting of x is then $x = 1\ 10\ 101\ 0100\ 1101$. The size of each bloc is 1 minus the size of the corresponding modulus. In a real implementation we can simply consider 32-bit moduli and split x in 31-bit blocs. The value we are going to encrypt is then in RNS

$$X = (1, 2, 5, 4, 13, 0)_{\mathcal{B}}.$$

The first operation consists of extending X is the base $\mathcal{B}'$. This has to be done exactly via the mixed radix representation as explained in section 5.1.

$$X = (1, 2, 5, 4, 13, 0)_{\mathcal{B}}, \quad (1, 10, 11, 18, 8, 22)_{\mathcal{B}'}.$$

The first step in the computation of $Y = X^b M \mod N$ is the call to $\mathrm{MM}(Y, (M^2 \mod N), B)$ which transforms Y into the Montgomery-like notation $YM \mod N + \beta N$

$$YM \mod N + \beta N = (2, 0, 0, 16, 11, 13)_{\mathcal{B}}, \quad (0, 6, 10, 9, 5, 17)_{\mathcal{B}'}.$$

The exponentiation then gives:

$$Y = (1, 4, 7, 2, 7, 21)_{\mathcal{B}}, \quad (1, 7, 11, 11, 13, 33)_{B'}.$$

At this step we are still in the Montgomery notation. We send this value to the other part who decrypt 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 $Z = Y^a \mod N$. The last call to $\mathrm{MM}(Z, 1, B)$ gets out from the Montgomery notation and gives:

$$Z = (2, 6, 2, 3, 8, 26)_{\mathcal{B}}, \quad (2, 8, 13, 16, 22, 28)_{\mathcal{B}'}.$$

Since $Z_k = 26 \neq 0$ we do not have a multiple of $m_k = 67$ and thus a final correction step is needed. We compute

$$t = Z_k(-N_k^{-1}) \mod m_k = 53,$$

and

$$53N = (2, 3, 3, 1, 5, 41)_{\mathcal{B}}.$$

A final addition gives the correct result

$$Z + 53N = (1, 2, 5, 4, 13, 0)_{\mathcal{B}}.$$

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

6 Complexity and comparisons

We compare our approach with a previously proposed method by Kawamura, et al. [4] that has itself been demonstrated more efficient than another efficient solution proposed in 1995 by Posch and Posch [9]. Since the

modular exponentiation algorithms we compare only differ in the way modular multiplication, and more specifically the two embedded base extensions, are performed, we only compare the elementary operations needed for one Montgomery multiplication. Even if the values reported in table 1 do not

	Our algorithm	Kawamura & al.
Lines 1 and 3 of algo MM	$5k$	$5k$
First base extension	$k^2 + k$	$2k^2 + 2k$
Second base extension	$k^2 + 2k$	$2k^2 + 2k$
Total	$2k^2 + 8k$	$4k^2 + 9k$
1024-bit key-length ($k = 34$)	2584	4930

Table 1: Number of elementary modular multiplications needed for two different RNS implementations of Montgomery multiplication. In our algorithm we consider that the redundant modulus m_r is a power of 2

take into account the fact that many of those elementary operation (single-precision modular multiplication) can be performed in parallel, our modular multiplication algorithm is clearly more efficient.

If we compare the whole implementations of RSA our solution without conversion is also more efficient. We can decompose our algorithm and Kawamura et al.'s one in three main parts: the mapping in, the modular exponentiation and the mapping out (see figure 1). As shown in table 1 our modular exponentiation requires less operations. Our mapping in procedure only consists in a base extension and we no not use any operation for the mapping out. Our conversion-free implementation is thus very efficient.

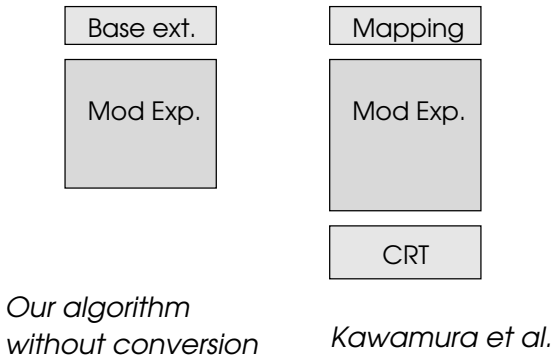

Figure 1: Comparisons of two encryption/decryption implementations of RSA in RNS.

7 Conclusion

We have presented a new implementation of Montgomery multiplication in RNS and have shown its efficiency with two implementations of RSA. The first solution uses an embedded RNS arithmetic which maps the values in RNS before the computations and convert them back in binary at the end. But the real novelty is a full RNS cryptosystem. The message is never considered as a binary number but rather in RNS all along the protocol. Thus no conversion are needed. This approach requires both parties to agree on a set of RNS parameters beforehand. Compared to previously proposed solutions [9, 4, 8] our algorithms requires less elementary operations and use only integer arithmetic (no rational approximation of α in (4) are computed). Furthermore the conditions on our parameters are easier to satisfy than the ones we have in these other methods.

References

[1] J.-C. Bajard, L.-S. Didier, and P. Kornerup. Modular multiplication and base extension in residue number systems. In N. Burgess, editor,

Proceedings of Arith15, the 15th IEEE symposium on Computer Arithmetic, pages 59–65, Vail, Colorado, USA, June 2001.

[2] E. F. Brickell. A survey of hardware implementation of RSA. In G. Brassard, editor, *Advances in Cryptologie - CRYPTO '89*, LNCS-453, pages 368–370. Springer-Verlag, 1990.

[3] S. E. Eldridge and C. D. Walter. Hardware implementation of Montgomery's modular multiplication algorithm. *IEEE Transactions on Computers*, 42(6):693–699, June 1993.

[4] S. Kawamura, M. Koike, F. Sano, and A. Shimbo. Cox-rower architecture for fast parallel montgomery multiplication. In *Advances in Cryptology - EUROCRYPT 2000 (LNCS 1807)*, pages 523–538, May 2000.

[5] D. E. Knuth. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*. Addison-Wesley, Reading, MA, third edition, 1997.

[6] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC Press, 2000 N.W. Corporate Blvd., Boca Raton, FL 33431-9868, USA, 1997.

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

[8] H. Nozaki, M. Motoyama, A. Shimbo, and S. Kawamura. Implementation of RSA algorithm based on RNS montgomery multiplication. In *Cryptographic Hardware and Embedded Systems CHES 2001 (LNCS 2162)*, pages 364–376, September 2001.

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

[10] J.-J. Quisquater and C. Couvreur. Fast decipherment algorithm for RSA public-key cryptosystem. *IEE Electronics Letters*, 18(21):905–907, October 1982.

[11] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[12] M. Shand and J. Vuillemin. Fast implementation of RSA cryptography. In E. E. Swartzlander, M. J. Irwin, and G. A. Jullien, editors, *Proceedings of the 11th IEEE Symposium on Computer Arithmetic*, pages 252–259, Windsor, Canada, June 1993. IEEE Computer Society Press, Los Alamitos, CA.

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

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

[15] F. J. Taylor. Residue arithmetic: A tutorial with examples. *IEEE Computer*, 17(5):50–62, may 1984.