# Leak Resistant Arithmetic

Jean-Claude Bajard[1], Laurent Imbert[2], Pierre-Yvan Liardet[3], and Yannick Teglia[3]

[1] LIRMM, CNRS, Université Montpellier II
161 rue Ada, 34392 Montpellier cedex 5, France
bajard@lirmm.fr

[2] LIRMM, CNRS, France, and
ATIPS*, CISaC**, University of Calgary
2500 University drive N.W, Calgary, T2N 1C2, Canada
Laurent.Imbert@lirmm.fr,

[3] STMicroelectronics, Smartcard ICs
Z.I. Rousset, 13106 Rousset Cedex, France
{Pierre-Yvan.Liardet,Yannick.Teglia}@st.com

**Abstract.** In this paper we show how the usage of Residue Number Systems (RNS) can easily be turned into a natural defense against many side-channel attacks (SCA). We introduce a Leak Resistant Arithmetic (LRA), and present its capacities to defeat timing, power (SPA, DPA) and electromagnetic (EMA) attacks.

**Keywords:** Side channel attacks, residue number systems, RNS Montgomery multiplication.

## 1 Introduction

Side-channel attacks rely on the interactions between the component and the real world. Those attacks formerly appeared in the network security world and eventually came within the smartcard and embedded system world to become the most pertinent kind of attacks on secure tokens. Some attacks monitor the computation through its time execution or its power consumption in order to discover secrets, as shown by P. Kocher in [19, 18]. Some others try to modify the component's behavior or data, through fault injection as pointed out first by D. Boneh, R. A. DeMillo, and R. J. Lipton in [6] in the case of public-key protocols, and extended to secret-key algorithms by E. Biham and A. Shamir in [5]. From noise adding to whole algorithm randomization, different ways have been considered to secure the implementations against side channels [19], and especially against power analysis [18, 12]. One difficult task when preventing from SCA is to protect from differential power analysis (DPA) introduced by

* Advanced Technology Information Processing Systems
** Center for Information Security and Cryptography

P. Kocher in [18] and its equivalent for electromagnetic attacks (EMA) [9, 1], and recent multi-channel attacks [2]. These specific attacks take advantage of correlations between the internal computation and the side channel information.

The purpose of Leak Resistant Arithmetic (LRA) is to provide a protection at the arithmetic level, i.e. in the way we represent the numbers for internal computations. We show how the usage of Residue Number Systems (RNS), through a careful choice of the modular multiplication algorithm, can easily be turned into a natural defense against SCA. In the same time, our solution provides fast parallel implementation and enough scalability to support key-size growth induced by the progress of classical cryptanalysis and computational power. In addition, another advantage of LRA is that classical countermeasures still apply at the upper level. We illustrate this fact in Sect. 3.3 through an adaptation to LRA of the Montgomery ladder [23], which has been analyzed in the context of side-channels [15], and which features (C safe-error and M safe-error protected) make it a first-class substitute to the square-and-multiply algorithm.

This paper puts together previous works from J.-C. Bajard and L. Imbert on RNS Montgomery multiplication and RSA implementation [3, 4], and P.-Y. Liardet original idea of addressing SCA using RNS, proposed in September 2002 in [20]. The same idea has been independently investigated by M. Ciet, M. Neeve, E. Peeters, and J.-J. Quisquater, and recently published in [8]. In Sect. 3.1, we address the problem of the Montgomery factor when RNS bases are randomly chosen, and we propose solutions which make it possible to randomly select new RNS bases during the exponentiation in Sect. 3.3.

## 2 The Residue Number Systems

In RNS, an integer $X$ is represented according to a base $\mathcal{B} = (m_1, m_2, \ldots, m_k)$ of relatively prime numbers, called moduli, by the sequence $(x_1, x_2, \ldots, x_k)$, where $x_i = X \bmod m_i$ for $i = 1...k$. The conversion from radix to RNS is then easily performed. 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 provides an algorithm to convert $X$ from its residue representation to the classical radix representation:

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

where $T_i = M/m_i$ and $\left|T_i^{-1}\right|_{m_i}$ is the inverse of $T_i$ modulo $m_i$. In the rest of the paper, $|X|_m$ denotes the remainder of $X$ in the division by $m$, i.e. the value $(X \bmod m) < m$.

One of the well known advantages of RNS is that additions, subtractions and multiplications are very simple and can be efficiently implemented on a parallel architecture [17]. Furthermore, only the dynamic range of the final result has to be taken into account since all the intermediate values can be greater than $M$.

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[4] than $(y_1, \ldots, y_k)$.

For cryptographic applications, modular reduction ($X \bmod N$), multiplication ($XY \bmod N$) and exponentiation ($X^E \bmod N$) are the most important operations. Many solutions have been proposed for those operations. For example, it is well known that they can be efficiently computed without trial division using Montgomery algorithms [22].

Let us briefly recall the principles of Montgomery multiplication algorithm. Given two integers $\beta^k, N$ such that $\gcd(\beta^k, N) = 1$, and $0 \leqslant XY < \beta^k N$, Montgomery multiplication evaluates $XY(\beta^k)^{-1} \bmod N$ by computing the value $Q < \beta^k$ such that $XY+QN$ is a multiple of $\beta^k$. Hence, the quotient $(XY+QN)/\beta^k$ is exact and easily performed. The result is less than $2N$. More detailed discussions on Montgomery reduction and multiplication algorithms can be found in [21, 7].

### 2.1 RNS Montgomery Multiplication

In this section we recall a recent RNS version of the Montgomery multiplication algorithm, previously proposed in [3, 4]. In the RNS version of the Montgomery multiplication, the value

$$M_1 = \prod_{i=1}^{k} m_i, \tag{2}$$

is chosen as the Montgomery constant (instead of $\beta^k$ in the classical representation). Hence, the RNS Montgomery multiplication of $A$ and $B$ yields

$$R = ABM_1^{-1} \bmod N, \tag{3}$$

where $R, A, B$ and $N$ are represented in RNS according to a predefined base $\mathcal{B}_1$. As in the classical Montgomery algorithm we look for an integer $Q$ such that $(AB+QN)$ is a multiple of $M_1$. However, the multiplication by $M_1^{-1}$ cannot be performed in the base $\mathcal{B}_1$. We define an extended base $\mathcal{B}_2$ of $k$ extra relatively prime moduli and perform the multiplication by $M_1^{-1}$ within this new base $\mathcal{B}_2$. For simplicity we shall consider that both $\mathcal{B}_1$ and $\mathcal{B}_2$ are of size $k$. Let us define $\mathcal{B}_1 = (m_1, \ldots, m_k)$ and $\mathcal{B}_2 = (m_{k+1}, \ldots m_{2k})$, with $M_2 = \prod_{i=1}^{k} m_{k+i}$, and $\gcd(M_1, M_2) = 1$.

Now, in order to compute $Q$, we use the fact that $(AB + QN)$ must be a multiple of $M_1$. Clearly $Q = -ABN^{-1} \bmod M_1$, and thus

$$q_i = -a_i b_i n_i^{-1} \bmod m_i, \ \forall i = 1 \ldots k. \tag{4}$$

As a result, we have computed a value $Q < M_1$ such that $Q = -ABN^{-1} \bmod M_1$. As pointed out previously we compute $(AB + QN)$ in the extra base $\mathcal{B}_2$. Before we can evaluate $(AB + QN)$ we have to know the product $AB$ in base $B_2$ and extend $Q$, which has just been computed in base $\mathcal{B}_1$ using (4), in base $\mathcal{B}_2$. We then compute $R = (AB + QN)M_1^{-1}$ in base $\mathcal{B}_2$, and extend the result

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

back to the base $\mathcal{B}_1$ for future use (the next call to Montgomery multiplication). Algorithm 1 describes the computations of our RNS Montgomery multiplication. It computes the Montgomery product $ABM_1^{-1} \bmod N$, where $A, B$, and $N$ are represented in RNS in both bases $\mathcal{B}_1$ and $\mathcal{B}_2$.

---

**Algorithm 1** : MM$(A, B, N, \mathcal{B}_1, \mathcal{B}_2)$, *RNS Montgomery Multiplication*

---

**Input :** Two RNS bases $\mathcal{B}_1 = (m_1, \ldots, m_k)$, and $\mathcal{B}_2 = (m_{k+1}, \ldots, m_{2k})$, such that $M_1 = \prod_{i=1}^{k} m_i, M_2 = \prod_{i=1}^{k} m_{k+i}$ and $\gcd(M_1, M_2) = 1$; a positive integer $N$ represented in RNS in both bases such that $0 < 4N < M_1$, $M_2$ and $\gcd(N, M_1) = 1$; (Note that $M_1$ can be greater or less than $M_2$.) two positive integers $A, B$ represented in RNS in both bases, with $AB < M_1 N$.

**Output :** A positive integer $R$ represented in RNS in both bases, such that $R \equiv ABM_1^{-1} \pmod N$, and $R < 2N$.

1: $T \leftarrow A \otimes_{RNS} B$ in $\mathcal{B}_1 \cup \mathcal{B}_2$
2: $Q \leftarrow T \otimes_{RNS} (-N^{-1})$ in $\mathcal{B}_1$
3: Extend $Q$ from $\mathcal{B}_1$ to $\mathcal{B}_2$
4: $R \leftarrow (T \oplus_{RNS} Q \otimes_{RNS} N) \otimes_{RNS} M_1^{-1}$ in $\mathcal{B}_2$
5: Extend $R$ back from $\mathcal{B}_2$ to $\mathcal{B}_1$

---

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.

Many different methods have been proposed to perform the base extension. Among those based on the CRT, [24] and [16] use a floating-point like dedicated unit, [26] proposes a version with an extra modulo greater than $k$ (this method is not valid for the first extension of Algorithm 1), [25] perform an approximated extension, and [3] allow an offset for the first base extension which is compensated during the second one. Other solutions have been proposed which use the mixed radix system (MRS) [10]. The great advantage of the MRS approach is that the modification of one modulus, only requires the computation of at most $k$ new values.

In [8], Posch and Posch's RNS Montgomery algorithm [25] is used, together with J.-C. Bajard *et al.* base extensions [3], which requires the computation of $T_i$ and $T_i^{-1}$ for each modulus $m_i$, where $T_i = M/m_i$ is about the same size as $M$, i.e. about 512 bits. In the context of random bases, precomputations are inconceivable (their choices of parameters lead to more than $2^{35}$ possible values for $M$). So we suppose that they evaluate these values at each base selection. Note that our algorithm uses the MRS conversion to avoid this problem.

### 2.2 Modular Exponentiation

The RNS Montgomery multiplication easily adapts to modular exponentiation algorithms. Since the exponent is not represented in RNS, we can consider any classic method for modular exponentiation, from the basic binary (square-and-multiply) algorithm to other fast exponentiation methods [11]. As with any

Montgomery based exponentiation algorithm, the first step in the evaluation of $X^E \bmod N$, is to transform the input $X$ into the so-called Montgomery representation: $X' = XM_1 \bmod N$ ($X'$ is sometimes referred to as the $N$-residue of $x$ according to $M_1$). This is done using a Montgomery multiplication with $X$ and $(M_1^2 \bmod N)$ as inputs. This representation has the advantage of being stable over Montgomery multiplication:

$$MM(X', Y', N, \mathcal{B}_1, \mathcal{B}_2) \equiv XYM_1 \bmod N.$$

At the end of the exponentiation, the value $Z' = X^E M_1 \bmod N$ is converted back into the expected result $Z = X^E \bmod N$ using a last call to Montgomery multiplication with $Z'$ and 1 as inputs.

## 3 Leak Resistant Arithmetic

One advantage of the RNS algorithms presented in previous sections is to offer many degrees of freedom for the randomization of processed data. In this section we propose two approaches based on the random selection of the RNS bases, which provides randomization, both at the circuit level (spatial randomization) and the data level (arithmetic masking). They represent a good trade-off between randomization strength and implementation cost. We consider two approaches:

- **Random choice of the initial bases:** Randomization of the input data is provided by randomly choosing the elements of $\mathcal{B}1$ and $\mathcal{B}2$ before each modular exponentiation.
- **Random change of bases before and during the exponentiation:** A generic algorithm is proposed offering many degrees of freedom in the implementation and at the security level.

### 3.1 Solving the Problem of the Montgomery Factor

A random draw of $\mathcal{B}_1$ and $\mathcal{B}_2$ is seen as a permutation $\gamma$, over the predefined set $\mathcal{B}$ of size $2k$. The first $k$ elements give $\mathcal{B}_{1,\gamma} = \left(m_{\gamma(1)}, \ldots, m_{\gamma(k)}\right)$, and the next $k$ ones give $\mathcal{B}_{2,\gamma} = \left(m_{\gamma(k+1)}, \ldots, m_{\gamma(2k)}\right)$. We denote $M_{1,\gamma}$ and $M_{2,\gamma}$ the products of the elements of $\mathcal{B}_{1,\gamma}$ and $\mathcal{B}_{2,\gamma}$ respectively:

$$M_{1,\gamma} = \prod_{i=1}^{k} m_{\gamma(i)}, \quad M_{2,\gamma} = \prod_{i=k+1}^{2k} m_{\gamma(i)}.$$

Before we give more details on SCA aspects, we solve an important problem due to the random choice of $\mathcal{B}_1$ and $\mathcal{B}_2$. As pointed out before, modular exponentiation of any input $X$ usually starts with an initial modular multiplication to get into the Montgomery representation, according to the so-called Montgomery factor. As mentioned before, we would have to perform the Montgomery multiplication of $X$ and $(M_{1,\gamma}^2 \bmod N)$. But since $M_{1,\gamma}$ is the product of $k$ randomly chosen moduli, we do not know ($M_{1,\gamma}^2 \bmod N$) beforehand. The huge number of possibilities for $M_{1,\gamma}$ (further evaluated) makes the precomputation of the products of all the subsets of $k$ elements of $\mathcal{B}$ unconceivable. On-the-fly evaluation of ($M_{1,\gamma}^2 \bmod N$), after the random choice of $\mathcal{B}_{1,\gamma}$, would be very expensive and would require dedicated hardware.

The solution we propose is achieved through a call to our RNS Montgomery multiplication where the roles of the bases $\mathcal{B}_{1,\gamma}$ and $\mathcal{B}_{2,\gamma}$ are exchanged. The following proposition holds:

**Proposition 1.** *For every permutation $\gamma$ over $\mathcal{B}$, the Montgomery representation of $X$ according to $\mathcal{B}_{1,\gamma}$, i.e. the value $X\, M_{1,\gamma} \bmod N$, is obtained with (note the order of $\mathcal{B}_{1,\gamma}$ and $\mathcal{B}_{2,\gamma}$ in the call to MM):*

$$MM(X, M \bmod N, N, \mathcal{B}_{2,\gamma}, \mathcal{B}_{1,\gamma}), \tag{5}$$

*where $M = \prod_{i=1}^{2k} m_i$.*

*Proof.* It suffices to remark that $\forall\gamma$, we have $M = M_{1,\gamma} M_{2,\gamma}$. Thus:

$$MM(X, M \bmod N, N, \mathcal{B}_{2,\gamma}, \mathcal{B}_{1,\gamma}) = X M_{1,\gamma} M_{2,\gamma} M_{2,\gamma}^{-1} \bmod N = X M_{1,\gamma} \bmod N.$$

$\square$

It is important to note that $M \bmod N$ does not depend on $\gamma$. This value is precomputed for each $m_i$. We obtain the result in RNS for the two bases, and we continue the exponentiation, with the two bases $\mathcal{B}_{1,\gamma}$ and $\mathcal{B}_{2,\gamma}$ playing their usual role as in $MM(.,.,N, \mathcal{B}_{1,\gamma}, \mathcal{B}_{2,\gamma})$. This solution only requires the precomputation of $2k$ small constants, of the size of the $m_j$s: for $j = 1...2k$, we store the values $|M \bmod N|_{m_j}$, where $M = \prod_{j=1}^{2k} m_j$.

We remark that this problem of the Montgomery factor is not mentioned in [8]. Using their notations, the precomputation of $\widetilde{M}^2 \bmod p$ and $\widetilde{M}^2 \bmod q$ for all the possible RNS bases would require the precomputation of $\binom{69}{9} > 2^{35}$ values of 512 bits each (more than 6.5 TBytes). We must then assume that $\widetilde{M}^2 \bmod p$ and $\widetilde{M}^2 \bmod q$ (or $X\,\widetilde{M} \bmod p$ and $X\,\widetilde{M} \bmod q$) are computed using other techniques, like Barrett or Quisquater, as precisely pointed out in [13]. This would require dedicated hardware (protected against SCA), and thus, drastically increase the size of the circuitry. In this case, the advantages of the RNS solution seems very limited. Using the same parameters, our algorithm only requires 144 Kbytes of memory, and one call to RNS Montgomery (which has to be done anyway).

### 3.2 Initial Random Bases

Taking into account the order of the elements within the bases, a set $\mathcal{B}$ of $2k$ moduli, leads to $2k!$ different bases $\mathcal{B}_{1,\gamma}$ and $\mathcal{B}_{2,\gamma}$ of $k$ moduli each. Since two consecutive exponentiations are performed with two different permutations, $\gamma$ and $\gamma'$, identical input data leak different information through the side-channel.

Actually, after step 2 of algorithm 1, we have computed $Q = (q_{\gamma(1)}, \ldots, q_{\gamma(2)})$ in $\mathcal{B}_{1,\gamma}$, where $q_{\gamma(i)} = q \bmod m_{\gamma(i)}$ for $i = 1...k$. Then, for each $m_{\gamma(j)}$ in $\mathcal{B}_{2,\gamma}$, we evaluate

$$|q|_{m_{\gamma(j)}} = \left|t_1 + m_{\gamma(1)}(t_2 + m_{\gamma(2)}(t_3 + \cdots + m_{\gamma(k-1)}t_k)\cdots)\right|_{m_{\gamma(j)}}, \tag{6}$$

where the $t_i$s are evaluated as follows, with $\mu_{s,t} = m_{\gamma(s)}^{-1} \bmod m_{\gamma(t)}$:

$$t_1 = |q|_{m_{\gamma(1)}} = q_{\gamma(1)} \tag{7}$$

$$t_2 = \left|(q_{\gamma(2)} - t_1)\mu_{1,2}\right|_{m_{\gamma(2)}} \tag{8}$$

$$\vdots$$

$$t_k = \left|(\cdots(q_{\gamma(k)} - t_1)\mu_{1,k} - \cdots - t_{k-1})\mu_{k-1,k}\right|_{m_{\gamma(k)}} \tag{9}$$

From (6) to (9), we remark the influence of $\gamma$ on the computations. It is clear that the values $\mu_{s,t}$ used to evaluate the $t_i$s are different for each new permutation. Moreover, although all of them need to be precomputed, only about half of them (those with $s > t$) are used in (7) to (9). The same remark applies for (6) where all the operands differ from one permutation to another. It is also important to note that equations (6) to (9) require modular arithmetic to be performed modulo different values at each permutation. Initial random bases will thus give very different traces through the side-channel, even with identical input data. This significantly increases the number of experiments the attacker should try in order to retrieve secret information.

Initial random bases also provides data randomization. Selecting two bases of $k$ moduli each within a set of exactly $2k$ moduli, gives $\binom{2k}{k} = \frac{(2k)!}{k!k!}$ pairs $(M_{1,\gamma}, M_{2,\gamma})$, i.e. $\binom{2k}{k}$ different Montgomery representations. Let us explain why this parameter corresponds to the level of randomization of the input data provided by our arithmetic. Randomizations of the message and the exponent are well known techniques to defeat DPA [19]. These randomizations prevent from chosen and known plain-text SCA targeting a secret carried out by the exponent during the exponentiation. In classical arithmetic solutions, such a masking can be obtained by choosing a pair of random values $(r_i, r_f)$, with $r_i \equiv r_f^{-1} \pmod N$, and by multiplying (modulo $N$) the message $X$ by $r_i$ before the exponentiation, such that $Xr_i \bmod N$ has a uniform distribution[5]. Similar techniques are used to randomize the exponent and the modulus. The size of the random factor(s) must be chosen large enough to ensure a good level of randomization.

In our case, $M_{1,\gamma}$ plays the same role as the random factor $r_i$. The first step of the exponentiation which converts $X$ into $XM_{1,\gamma} \bmod N$, thus provides message randomization. Since $M_{1,\gamma}$ is the product of randomly selected moduli, and can take $\binom{2k}{k}$ different values, we can consider that the output $XM_{1,\gamma} \bmod N$ has a uniform distribution (if $k$ is large enough). It is important to note that the

[5] A final multiplication by $r_f^e \bmod N$ is required at the end.

randomization of $X$ is free since the first call to $MM(...)$ must be performed anyway.

Table 1 gives randomization rates of different base sizes, and the corresponding size of the factor $r_i$ in classic arithmetic solutions, computed as $\lfloor \log_2(\binom{2k}{k}) \rfloor$. For example, we remark that $k = 34$ ($|\mathcal{B}| = 68$), provides about the same randomization level as a random factor $r_i$ of 64 bits.

**Table 1.** Randomization level of different base sizes, and their equivalent in classic arithmetic masking solutions.

| size of $\mathcal{B}$ | $\binom{2k}{k}$ | equiv. size of $r_i$ (in bits) |
|---|---|---|
| 36 | 9075135300 | 33 |
| 44 | 2104098963720 | 40 |
| 52 | 495918532948104 | 48 |
| 60 | 118264581564861424 | 56 |
| 68 | 28453041475240576740 | 64 |
| 80 | 107507208733336176461620 | 76 |

In terms of memory requirements, initial random bases require the precomputation of $2k$ moduli $m_i$ of $n$ bits each, $2k-1$ modular inverses $|m_i^{-1}|_{m_j}$ $(i \neq j)$ for each $m_j$, and $2k$ values $|M \bmod N|_{m_i}$; a total of $2k(2k+1)$ $n$-bit integers. Table 2 gives the total memory requirements for different values of $k$ and $n$ and the corresponding RSA equivalent dynamic range (computed as $k(n-1)$, which is the size of the lower bound of $M_{1,\gamma}$ considering $2^{n-1} \leq m_i < 2^n$). For example, a set $\mathcal{B}$ of $2k = 68$ moduli of 32 bits each (which correspond to approximately 1054-bit numbers in classical binary representation) requires about 18 Kbytes of memory.

**Table 2.** Memory requirements for various parameters $k$ and $n$, and the corresponding RSA equiv. size.

| k | $n$ = size of the $m_i$s (in bits) | memory (in KBytes) | dynamic range |
|---|---|---|---|
| 30 | 18 | 8 | $> 510$ |
| 25 | 32 | 10 | $> 775$ |
| 34 | 32 | 18 | $> 1054$ |
| 17 | 64 | 9.5 | $> 1071$ |
| 32 | 64 | 33 | $> 2016$ |

We remark that the space complexity is in $O(k^2 n)$. Thus, it is better to consider smaller bases with larger moduli. Of course, the complexity of the basic cells which perform the arithmetic over each $m_i$ increases at the same time. A tradeoff between the two parameters $k$ and $n$ has to be found, according to the hardware resources.

### 3.3 Random Bases During Exponentiation

In this section, we show how we can randomly change the RNS bases during the exponentiation. As for the initial random bases version presented in the previous section, we must solve another problem, the on-the-fly conversion between two different Montgomery representations.

Let us assume that initial random bases have been selected and that the exponentiation algorithm has computed until, say $Y = X^\alpha M_{1,\gamma} \bmod N$ over the two bases $(\mathcal{B}_{1,\gamma}, \mathcal{B}_{2,\gamma})$. In order to continue with two new random bases $(\mathcal{B}_{1,\gamma'}, \mathcal{B}_{2,\gamma'})$, we have to switch from the old Montgomery representation (according to $M_{1,\gamma}$) to the new one (according to $M_{1,\gamma'}$). In other words, the question is: given $X^\alpha M_{1,\gamma} \bmod N$, how can we compute $X^\alpha M_{1,\gamma'} \bmod N$?

A straightforward solution is to get out of the old Montgomery representation with

$$MM(Y, 1, N, \mathcal{B}_{1,\gamma}, \mathcal{B}_{2,\gamma}) = X^\alpha \bmod N = Z,$$

and to enter into the new Montgomery representation with

$$MM(Z, M \bmod N, N, \mathcal{B}_{2,\gamma'}, \mathcal{B}_{1,\gamma'}) = X^\alpha M_{1,\gamma'} \bmod N$$

using the solution proposed in Sect. 3.2. The exponentiation can then continue according to $M_{1,\gamma'}$ until the next base change. The main drawback of this solution is that we loose the randomization of $X^\alpha \bmod N$ between the two calls to $MM(...)$.

A better solution consists in inverting the order of the two calls to $MM(...)$. Actually, if we first call (note the order of $\mathcal{B}_{1,\gamma}$ and $\mathcal{B}_{2,\gamma}$)

$$MM(X^\alpha M_{1,\gamma} \bmod N, M \bmod N, N, \mathcal{B}_{2,\gamma'}, \mathcal{B}_{1,\gamma'}),$$

we obtain

$$X^\alpha M_{1,\gamma} M_{1,\gamma'} \bmod N.$$

We then call

$$MM(X^\alpha M_{1,\gamma} M_{1,\gamma'} \bmod N, 1, N, \mathcal{B}_{1,\gamma}, \mathcal{B}_{2,\gamma})$$

and get the expected result

$$X^\alpha M_{1,\gamma'} \bmod N.$$

As a result, the value $X^\alpha \bmod N$ is always masked by a random quantity.

In order to illustrate the fact that our arithmetic easily adapts to existing countermeasures at the upper level, Algorithm 2 is a RNS variant of an exponentiation algorithm (adapted from the Montgomery ladder [23]), proposed by M. Joye and S.-M. Yen [15]. The permutations are indiced according to the bits of the exponent. We start with the initial random permutation $\gamma_l$ and we use $\gamma_i$ and $\gamma_{i+1}$ to represent the old and new ones at each iteration (note that $\gamma_{i+1}$ can be equal to $\gamma_i$ if no new permutation is selected.). Note that this generic algorithm offers many implementation options in the frequency of base exchange. Although it is always possible to pay the price for a new permutation of $\mathcal{B}$ at each iteration, this is our feeling that such an ultimate option does not necessarily provides a better security, although this seems difficult to prove at the algorithmic level.

**Algorithm 2** : $RME(X, C, N, \mathcal{B})$, *Randomized Modular Exponentiation*

**Input :** A set $\mathcal{B} = \{m_1, \ldots, m_k, m_{k+1}, \ldots, m_{2k}\}$ of relatively prime integers ; an integer $X$ less than $N$ represented in RNS for all $m_j \in \mathcal{B}$, with $4N < M_{1,\gamma}$, where $M_{1,\gamma} = \prod_{i=1}^{k} m_{\gamma(i)}$ for all permutation $\gamma$ of $\mathcal{B}$ ; a positive exponent $E = \sum_{i=0}^{l-1} e_i 2^i$.

**Output :** A positive integer $Z = X^E \mod N$ represented in RNS over $\mathcal{B}$.

Select randomly $\gamma_l$
$U_0 \leftarrow MM(1, M \bmod N, N, \mathcal{B}_{2,\gamma_l}, \mathcal{B}_{1,\gamma_l})$
$U_1 \leftarrow MM(X, M \bmod N, N, \mathcal{B}_{2,\gamma_l}, \mathcal{B}_{1,\gamma_l})$
**for** $i = l - 1$ **down to** $0$ **do**
  $b \leftarrow \overline{e_i}$
  $U_b \leftarrow MM(U_b, U_{e_i}, N, \mathcal{B}_{1,\gamma_{i+1}}, \mathcal{B}_{2,\gamma_{i+1}})$
  **if** new $\gamma_i$ randomly selected **then**
    $U_0 \leftarrow MM(U_0, M \bmod N, N, \mathcal{B}_{2,\gamma_i}, \mathcal{B}_{1,\gamma_i})$
    $U_0 \leftarrow MM(U_0, 1, N, \mathcal{B}_{1,\gamma_{i+1}}, \mathcal{B}_{2,\gamma_{i+1}})$
    $U_1 \leftarrow MM(U_1, M \bmod N, N, \mathcal{B}_{2,\gamma_i}, \mathcal{B}_{1,\gamma_i})$
    $U_1 \leftarrow MM(U_1, 1, N, \mathcal{B}_{1,\gamma_{i+1}}, \mathcal{B}_{2,\gamma_{i+1}})$
  **else**
    $\gamma_i = \gamma_{i+1}$
  **end if**
  $U_{e_i} \leftarrow MM(U_{e_i}, U_{e_i}, N, \mathcal{B}_{1,\gamma_i}, \mathcal{B}_{2,\gamma_i})$
**end for**
$Z \leftarrow MM(U_0, 1, N, \mathcal{B}_{1,\gamma_0}, \mathcal{B}_{2,\gamma_0})$

## 4 Implementation Aspects

In this section we propose an efficient addressing scheme which shows that our algorithms can be implemented rather efficiently at a reasonable hardware cost.

In a highly parallel implementation, the circuit can be built with $2k$ identical basic cells. If $k$ is large it might not be possible to build a circuit having actually $2k$ cells. In this case, it is always possible to implement the algorithms with fewer cells, at the price of less parallelization, by adding control to deal with the available cells. Each elementary cell can perform the basic modular arithmetic operations. It receives three operands $x, y, m$, one control bit and return either the sum or the product[6] (depending on the control bit) of $x$ and $y$ modulo $m$ (see Fig. 1).

The precomputed values are stored in a multiple access memory and are addressed through a permutation table which implements $\gamma$. Each elementary cell has an identification number and performs the operation modulo the value given by $\gamma$ for this number. For example, in Fig. 1, the $j$th cell performs the modular multiplication $x\,\mu_{i,j} \bmod m_{\gamma(j)}$, where $\mu_{i,j}$ (see 3.2 for details) is retrieved from the memory through the permutation table. When the value $\mu_{i,j} = |m_{\gamma(i)}^{-1}|_{m_{\gamma(j)}}$ is required, the indices $i, j$ are passed to $\gamma$ which returns the value stored at the address $(\gamma(i), \gamma(j))$. When $i = j$, the memory blocks can be used to store the $|M \bmod N|_{m_i}$. The advantage of using a permutation table is that the cells

[6] We can also consider a multiply-and-add operation which returns $xy + z \bmod m$.

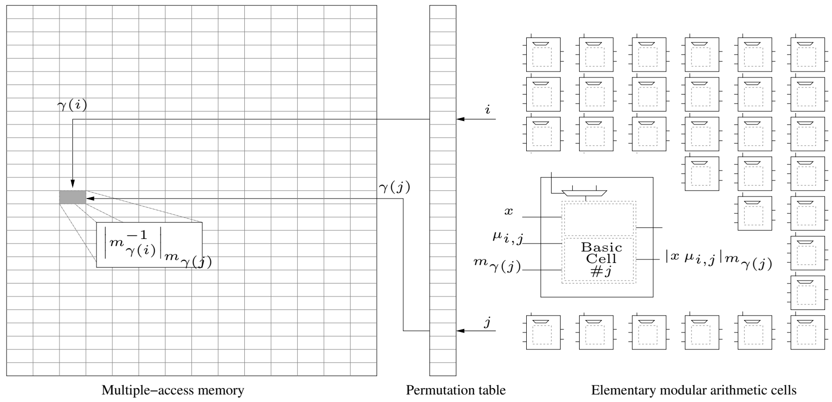


**Fig. 1.** Sketch of a RNS-based cryptoprocessor with a network of elementary cells exchanging data with the memory through a permutation table.

do not have to deal with the permutations themselves. Each time we randomly select a new couple of bases, we just need to reconfigure the permutation table.

## 5 Side-Channel Analysis

In LRA, timing attacks are prevented by masking the input data. Actually, since $M_{1,\gamma}$ comes from a randomly chosen subset of $\mathcal{B}$, the first Montgomery multiplication provides randomization of the message at no extra cost[7]. Note also that timing attacks can still be prevented at the algorithmic level with LRA as presented in Sect. 3.3 with the adaptation of the exponentiation method proposed in [15].

LRA provides natural resistance to SPA, DPA and EMA, by generating a very high level of randomization, both at the data level and the order of the computations.

The first feature brought by the proposed LRA is the randomization of the bases. If we assume that the architecture has exactly $2k$ elementary cells, each cell performs its computations with a randomly drawn modulo. Hence, if the same computation is performed several times, a given cell never computes the same calculation. This leads to protections that act at different levels. First, we have the so-called spatial protection, since a given location, i.e. a cell, behaves differently for the same calculation (same input data); this helps to foil EMA focusing on the activity of an elementary cell. Moreover, the random choice of the bases leads to the randomization of the message. This is a well known technique to defeat DPA as well as SPA.

[7] If further masking is required, a random multiple of $\phi(N)$ can easily be added to the exponent (represented in classical binary representation) before each modular exponentiation.

The second feature, which acts against SPA and DPA, is due to the random bases changes during the exponentiation. Actually, the values from an iteration to another within the exponentiation algorithm are no longer correlated. By the way, it thwarts classical DPA in iterative attacks e.g. on RSA algorithms. Moreover, many implementation options in the frequency of base exchange allow the user to easily increase the level of randomization.

Previous attacks on public key protocols using Fault injection [6] works well when the values are stored in the classic positional binary number representation. For example, the attack on non-CRT implementation of RSA makes the assumption that the flip of one bit of a register during the exponentiation changes a value $z$ to $z \pm 2^b$ for an unknown bit $b$. Since RNS is not such a positional number system, this assumption is not valid anymore and known fault attacks do not apply. Moreover, the use of (redundant) residue number and polynomial systems for error checking/correcting has been investigating in the past (see [14], [27]) and would apply perfectly to our system in order to reinforce the resistance against fault-based attacks (in the case of the CRT signature scheme for example). Even though fault injection issues has not been addressed here, it is not unreasonable to think that the LRA could also be used to defeat them. A deeper analysis is required in order to see how this goal can be accurately achieved.

## 6 Conclusions

We presented a new defense against side channel analysis adapted to public key cryptosystems operating over large finite rings or field (RSA, ElGamal, ECC over large prime fields, etc). For that purpose we introduced a Leak Resistant Arithmetic (LRA) based on Residue Number Systems (RNS). We provided concrete algorithms together with example of implementation. Our approach allows the usage of many implementation optimizations at the field operator level without introducing weaknesses. The computation overhead due to our technique is shown to be negligible regarding to the overall computation time. We have shown that the LRA provides self robustness against EMA, DPA and SPA at the field implementation level. Moreover, at an upper level, usual protections against SCA easily adapt.

## References


[1] D. Agrawal, B. Archambeault J. R. Rao, and P. Rohatgi. The EM side-channel(s). In B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2523 of *LNCS*, pages 29–45, Redwood Shores, CA, USA, August 2002. Springer-Verlag.

[2] D. Agrawal, J. R. Rao, and P. Rohatgi. Multi-channel attacks. In C. D. Walter and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *LNCS*, pages 2–16, Köln, Germany, September 2003. Springer-Verlag.

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

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

[5] E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. In *Advances in Cryptology – CRYPTO'97*, number 1294 in LNCS, pages 513–525. Springer-Verlag, 1997.

[6] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults. In W. Fumy, editor, *Advances in Cryptology – Eurocrypt'97: International Conference on the Theory and Application of Cryptographic Techniques*, volume 1233 of *LNCS*, pages 37–51, Konstanz, Germany, May 1997. Springer-Verlag.

[7] Ç. K. Koç, T. Acar, and B. S. Kaliski Jr. Analyzing and comparing montgomery multiplication algorithms. *IEEE Micro*, 16(3):26–33, June 1996.

[8] M. Ciet, M. Neve, E. Peeters, and J.-J. Quisquater. Parallel FPGA implementation of RSA with residue number systems – can side-channel threats be avoided? In *46th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS-2003)*, Cairo, Egypt, December 2003.

[9] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *LNCS*, pages 251–272, Paris, France, May 2001. Springer-Verlag.

[10] H. L. Garner. The residue number system. *IRE Transactions on Electronic Computers*, EC-8:140–147, June 1959.

[11] D. M. Gordon. A survey of fast exponentiation methods. *Journal of Algorithms*, 27(1):129–146, April 1998.

[12] L. Goubin and J. Patarin. DES and differential power analysis – the duplication method. In *Cryptographic Hardware and Embedded Systems – CHES 99*, number 1717 in LNCS, pages 158–172. Springer-Verlag, 1999.

[13] G. Hachez and J.-J. Quisquater. Montgomery multiplication with no final subtraction: Improved results. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2000*, volume 1965 of *LNCS*, pages 293–301. Springer-Verlag, 2000.

[14] W. K. Jenkins. The design of error checkers for self-checking residue number arithmetic. *IEEE Transactions on Computers*, C-32(4):388–396, April 1983.

[15] M. Joye and S.-M. Yen. The montgomery powering ladder. In *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *LNCS*, pages 291–302, 2002.

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

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

[18] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397, Santa-Barbara, CA, USA, August 1999. Springer-Verlag.

[19] P. C. Kocher. Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *Advances in Cryptology - CRYPTO '96*,

volume 1109 of *LNCS*, pages 104–113, Santa-Barbara, CA, USA, August 1996. Springer-Verlag.

[20] P.-Y. Liardet. Masquage de données décomposées dans un système de résidus. Patent, September 2002. Dépôt Français numéro FR0211671, dépôt Européen numéro EP03300126.

[21] 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.

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

[23] P. L. Montgomery. Speeding the pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, January 1987.

[24] 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*, number 2162 in LNCS, pages 364–376, September 2001.

[25] 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.

[26] A. P. Shenoy and R. Kumaresan. Fast base extension using a redundant modulus in RNS. *IEEE Transactions on Computers*, 38(2):292–297, February 1989.

[27] P. R. Turner. Residue polynomial systems. *Theoretical Computer Science*, 279(1-2):29–49, May 2002.