Hybrid Position-Residues Number System

Karim Bigou, Arnaud Tisserand

Hybrid Position-Residues Number System

Karim Bigou [3,1] and Arnaud Tisserand [2,1]
[1]IRISA, [2]CNRS, [3]University Rennes 1, INRIA Centre Rennes - Bretagne Atlantique
6 rue Kerampont, CS 80518, 22305 Lannion cedex, FRANCE
{karim.bigou,arnaud.tisserand}@irisa.fr

***Abstract*—We propose an hybrid representation of large integers, or prime field elements, combining both positional and residue number systems (RNS). Our *hybrid position-residues* (HPR) number system mixes a high-radix positional representation and digits represented in RNS. RNS offers an important source of parallelism for addition, subtraction and multiplication operations. But, due to its non-positional property, it makes comparisons and modular reductions more costly than in a positional number system. HPR offers various trade-offs between internal parallelism and the efficiency of operations requiring position information. Our current application domain is asymmetric cryptography where HPR significantly reduces the cost of some modular operations compared to state-of-the-art RNS solutions.**

***Index Terms*—number representation; large integer; finite field; modular arithmetic; residue number system.**

I. Introduction

In computer arithmetic, *representations of numbers* strongly impact algorithms and implementations performances for basic operations. Redundant number systems (see for instance [1]) allow limited carry propagations during additions, and then *fully parallel addition*. Logarithmic number system (LNS, see for instance [2]) swaps the cost of addition/subtraction by the cost of multiplication/division, but it still leads to a large cost difference between these two operations categories. Furthermore LNS is difficult to use in asymmetric cryptography applications with large numbers. *Residue number system* (RNS, see for instance [3], [4]) allows *internal parallelism* for both addition/subtraction and multiplication. Then RNS is commonly used in asymmetric cryptography (see for instance [5], [6], [7], [8], [9], [10]). But RNS is a *non-positional* representation with reduced efficiency for comparisons, modular reductions and modular multiplications.

We propose a new number system, denoted *hybrid position-residues* (HPR) number system for large integers and prime field elements ($\mathbb{F}_p$). HPR "mixes" a high-radix positional representation of numbers and RNS digits. It can be seen as a sort of compromise between a positional representation and RNS. Our hybrid number system offers a good level of internal parallelism for addition, subtraction and multiplication operations (even if it is slightly reduced compared to standard RNS). But it also allows much more efficient comparisons, modular reductions and modular multiplications due to the available position information in the representation.

After the presentation of notations and definitions in Section II, Section III briefly presents the state-of-the-art. Our HPR number system is detailed in Section IV. Applications to modular multiplication and exponentiation for asymmetric cryptography are presented in Sections V and VI respectively. Finally, Section VII concludes the paper.

II. Notations and Definitions

The definitions and notations used in the paper are:

- Capital letters, *e.g.* X, denote large integers or field elements of ℓ bits (in our applications $\ell > 100$)
- $|X|_P$ is $X \bmod P$
- $\mathcal{B} = (m_1, \ldots, m_n)$ is an RNS base composed of n moduli where all m_i are pairwise co-primes of w bits
- $\langle X \rangle$ represents X in the RNS base $\mathcal{B}$ and defined by:
$$\langle X \rangle = (x_1, \ldots, x_n) \text{ where } x_i = |X|_{m_i} \quad (1)$$
- $M = \prod_{i=1}^{n} m_i$ and $M_i = \frac{M}{m_i}$
- EMM a w-bit elementary modular multiplication $|x_i \cdot y_i|_{m_i}$ used as complexity unit for cost analysis as in all works of the literature
- $n = \lceil \ell/w \rceil$, *i.e.* the minimal number of moduli to represent an ℓ-bit value
- the word "*base*" is used in the RNS context
- the word "*radix*" is used in the positional context
- LSD/MSD denote least/most significant digit

We will use two or three small RNS bases denoted $\mathcal{B}_a, \mathcal{B}_b$, etc. In $\mathcal{B}_a$, composed of $n_a < n$ moduli (w-bit integers), X is represented by $\langle X \rangle_a$. Same notations apply for the other bases. The base concatenation is denoted $\mathcal{B}_{a|b}$ and $\langle X \rangle_{a|b}$ is the RNS representation of X using $n_a + n_b$ moduli.

III. State-of-the-Art

A. Residue Number System (RNS)

The RNS representation, proposed in the late 50s in [3], [4], is increasingly used for large modular arithmetic computations and asymmetric cryptography implementations, see for instance [11], [12], [13], [9], [14], [10], [8].

The integers X and Y are represented in the RNS base $\mathcal{B}$ by $\langle X \rangle$ and $\langle Y \rangle$. Multiplications, additions and subtractions are very efficient and natural in RNS. If one wants to perform the $\diamond$ operation in RNS, with $\diamond \in \{+, -, \times\}$, one just computes in parallel over the moduli:

$$\langle X \rangle \diamond \langle Y \rangle = \big(|x_1 \diamond y_1|_{m_1}, \ldots, |x_n \diamond y_n|_{m_n}\big). \quad (2)$$

Computations are performed independently on each modulo m_i without carry propagation. RNS multiplication requires n independent EMMs. Moreover, if Z is an integer coprime with

all m_i, then exact division by Z is computed by multiplying by $\langle Z^{-1} \rangle = \left(|Z^{-1}|_{m_1}, \ldots, |Z^{-1}|_{m_n} \right)$.

We assume $0 \leqslant X < M$ to be able to convert back using the CRT (Chinese remainder theorem) formula:

$$X = |X|_M = \left| \sum_{i=1}^{n} \left| x_i \cdot M_i^{-1} \right|_{m_i} \times M_i \right|_M .$$

An important consequence of the CRT is that each RNS operation performed in base $\mathcal{B}$ is automatically reduced modulo M. As described in Sec. III-C, it is used to compute modular reduction in state-of-the-art.

However, RNS is a *non-positional representation*: comparisons, general divisions and modular reductions are much harder than multiplications in RNS (for instance see [15]).

B. RNS Base Extension

In order to compute more complex operations like RNS modular reduction, the *base extension* (BE) has been proposed in [15]. BE converts $\langle X \rangle_a$ into $\langle X \rangle_b$, avoiding costly conversions to the classical representation. In this paper, we consider only the BE algorithm from [6], which is used in state-of-the-art implementations due to its high level of parallelism. Nonetheless, our propositions are independent of the choice of the BE algorithm, one can choose another algorithm such as [16], [17]. The principle of the BE proposed in [6] is to approximate q in the CRT formula, $X = \sum_{i=1}^{n_a} \left(\left| x_{a,i} \cdot M_{a,i}^{-1} \right|_{m_{a,i}} M_{a,i} \right) - q\, M_a$ where the computations are performed modulo each $m_{b,i}$. The result of this approximated conversion is either $\langle X \rangle_b$ or $\langle X + M_a \rangle_b$ but this is easily managed in state-of-the-art implementations (see details in [6]). This algorithm requires $(n_a\, n_b + n_a)$ EMMs.

C. RNS Modular Reduction and Multiplication

The state-of-the-art RNS modular reduction (RNS-MR) for a generic modulus, proposed in [18] (and optimized in [5], [6], [10]), is based on the Montgomery modular reduction [19] (initially proposed for radix-2).

To perform all required operations in the Montgomery reduction, one needs two bases: $\mathcal{B}_a$ for computing modulo M_a; and $\mathcal{B}_b$ for dividing by M_a (which is not possible in $\mathcal{B}_a$). Algorithm 1 requires two BEs. In most state-of-the-art solutions both bases have n moduli ($n_a = n_b = n$).

Algorithm 1: RNS Montgomery Reduction from [18].

Input: $\langle X \rangle_{a|b}$
Precomp.: $\langle P \rangle_{a|b}$, $\langle -P^{-1} \rangle_a$, $\langle M_a^{-1} \rangle_b$
Output: $\langle S \rangle_{a|b} = \langle \left| X M^{-1} \right|_P \rangle_{a|b} + \delta \langle P \rangle_{a|b}$ in $\mathcal{B}_a$ and $\mathcal{B}_b$ with $\delta \in \{0, 1, 2\}$

1 $\langle Q \rangle_a \leftarrow \langle X \rangle_a \times \langle -P^{-1} \rangle_a$
2 $\langle Q \rangle_b \leftarrow \text{BE}\left(\langle Q \rangle_a, \mathcal{B}_a, \mathcal{B}_b \right)$
3 $\langle R \rangle_b \leftarrow \langle X \rangle_b + \langle Q \rangle_b \times \langle P \rangle_b$
4 $\langle S \rangle_b \leftarrow \langle R \rangle_b \times \langle M_a^{-1} \rangle_b$
5 $\langle S \rangle_a \leftarrow \text{BE}\left(\langle S \rangle_b, \mathcal{B}_b, \mathcal{B}_a \right)$
6 **return** $\langle S \rangle_{a|b}$

Using the optimization from [10], the RNS-MR total cost is $2\, n_a\, n_b + n_a + n_b = 2\, n^2 + 2\, n$ EMMs.

In RNS, the modular multiplication (RNS-MM) is composed of a simple RNS multiplication, on both bases, followed by a RNS-MR and costs $2\, n^2 + 4\, n$ EMMs.

D. Close and Related Representations

The mixed-radix system (MRS [4]) uses, as RNS, a base of moduli to represent integers but is a positional representation with very limited internal parallelism compared to RNS.

The polynomial RNS (PRNS [20]) uses small polynomials for the moduli instead of integers, and as RNS is a non-positional representation.

In the standard radix-2 positional representation, the use of Mersenne and pseudo-Mersenne primes significantly reduces the cost of modular reductions. The representation proposed in [21] generalizes the idea of pseudo-Mersenne primes for any radix but not in the RNS context. Recently, [13] proposes an equivalent to pseudo-Mersenne primes in RNS where the RNS-MR cost is more or less divided by 2.

IV. Presentation of the HPR number system

A. Definition and Properties

In this section, we define our *hybrid position-residues* (HPR) representation and describe its properties. The main idea is to define a representation which makes a link between RNS and the standard positional representation. On the one hand RNS representation is very efficient for some operations, as multiplications, but is not for some others as comparisons. On the other hand, a positional number system allows cheaper comparisons but more costly multiplications than RNS.

Definition 1 (Hybrid Position-Residues Number System (HPR)): Let us assume two coprime RNS bases $\mathcal{B}_a$ and $\mathcal{B}_b$ with $M_a = \prod_{i=0}^{n_a - 1} m_{a,i}$, the *degree* $d \in \mathbb{N}^*$, β_{min} and $\beta_{max} \in \mathbb{R}$ such that $\beta_{max} - \beta_{min} \geqslant 1$ and $\beta_{max} + \beta_{min} \geqslant 0$. The integer X is represented in HPR, with parameters $(\mathcal{B}_a, \mathcal{B}_b, \beta_{min}, \beta_{max})$, in a *high-radix* M_a positional representation by

$$X_{\text{HPR}} = \left(\langle X_{d-1} \rangle_{a|b}, \ldots, \langle X_0 \rangle_{a|b} \right)_{\text{HPR}}$$

where the *digits* $\langle X_i \rangle_{a|b}$, represented in RNS, are such that

$$X = \sum_{i=0}^{d-1} X_i\, M_a^i$$

where $\beta_{min}\, M_a \leqslant X_i \leqslant \beta_{max}\, M_a$.

Figure 1 illustrates the behavior of basic operations in HPR compared to standard positional and RNS representations. The positional representation (top part) has limited internal parallelism due to the carries propagation (arrows), but the implicit weights of the digits provide very accurate information on the magnitude of X (leading to simple comparisons). RNS (bottom part) has significant parallelism due the independent computations on the residues ($\pm, \times$ are fully parallel), but its non-positional characteristic makes comparisons (and related operations) more complex. HPR (middle part) is a trade-off

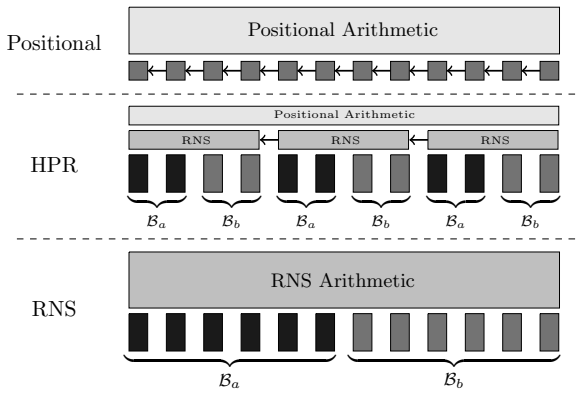

Fig. 1. HPR seen as trade-off between positional and RNS representations.

between these two representations. It still provides internal parallelism at the digit level ($\langle X_i \rangle$), and it provides *coarse* grain information of the integer magnitude.

HPR can be a redundant representation when the digits bounds β_{min} and β_{max} are chosen such that $\beta_{max} - \beta_{min} > 1$. It also allows negative digits when $\beta_{min} < 0$.

HPR is a generalization of both RNS and positional representations (in radix 2^w, $w \in \mathbb{N}$). In the case $d = 1$, $\beta_{min} = 0$ and $\beta_{max} = 1$, HPR is the standard RNS representation $X_{\text{HPR}} = (\langle X_0 \rangle_{a|b})$ with $n_a + n_b$ moduli. In the case $\mathcal{B}_a = (2^w)$ (only one modulo in base $\mathcal{B}_a$) and no second base ($n_b = 0$), HPR is the standard radix-2^w positional representation with d digits (with or without redundancy depending on the values β_{min}, β_{max} like in [1]). In this paper, we only deal with "true" HPR representations where $d > 1$ and $n_b > 1$.

B. Basic Operations

In all HPR operations, there are two levels of computations: the *digit level* using standard positional algorithms and the *residue level* using RNS algorithms (inside the digits).

a) Addition-Subtraction: At residue level, additions and subtractions are performed in parallel over all moduli $\langle X_i \rangle_{a|b} = \langle A_i \rangle_{a|b} + \langle B_i \rangle_{a|b}$ and over all the digits ($0 \leqslant i \leqslant d-1$). In some cases, a digit value X_i can be slightly larger than M_a, then a small carry propagation is required. This carry propagation is detailed in Sec. IV-C. In practice, we use a redundant form of HPR (with appropriate digit bounds β_{min}, β_{max}) to avoid as much as possible such carry propagations. This type of redundancy is widely used in software multi-precision libraries (*e.g.* on a 64-bit machine, a radix "only" equal to 2^{53} is used to allow, most of the time, fully parallel additions).

b) Multiplication: At the digit level, one can use any multiplication algorithm for positional representation (schoolbook, Karatsuba-Ofman [22], etc.). During the multiplication of two digits (at the residue level) $\langle X_i \rangle_{a|b} = \langle A_i \rangle_{a|b} \times \langle B_i \rangle_{a|b}$, one must use a large enough RNS base $\mathcal{B}_b$ (*i.e.* $M_b \geqslant M_a$) to represent the product. The product X_i is very often larger than M_a, the "high" part of X_i must be propagated to X_{i+1}. Extracting the "high" part of X_i is very simple in a standard positional representation (truncation), this is more tricky in RNS. To compute the "high" part, one needs to get the actual value X_i from the residues $\langle X_i \rangle_{a|b} = (x_{i,0}, x_{i,1}, \ldots)$ using the CRT formula (this can be optimized). Then use the euclidean division of X_i by M_a to get the quotient-remainder representation $X_i = Q_i M_a + R_i$. The remainder R_i is kept at this position, and Q_i is added to the next digit X_{i+1}.

The quotient-remainder representation of product X is $(\langle Q \rangle, \langle R \rangle)$ computed using the `Split` function proposed in [23] and presented in Algo. 2. Lines 4–6 are only required when using the approximated BE from [6]. One has $Q < \beta_{max}^2 M_a$ as required for propagating the "high" part of the product but $R < 2M_a$ instead of $R < M_a$ (due to the approximation from [6]) and this is handled using redundancy at digit level (like in additions) thanks to $M_b > \beta_{max}^2 M_a$. Thanks to the analysis presented in [23], the cost of `Split` is $2n_a \times n_b + n_a + n_b$ `EMM`s. In our applications, $n_a = n_b = \frac{n}{d}$ leads to $2\frac{n^2}{d^2} + 2\frac{n}{d}$ `EMM`s. This algorithm is parallel over $\frac{n}{d}$ channels.

Algorithm 2: Decomposition algorithm (`Split`) [23].

Input: $\langle X \rangle_{a|b}$ with $X < (\beta_{max} M_a)^2$ and $M_b > \beta_{max}^2 M_a$
Precomp.: $\langle M_a^{-1} \rangle_b$
Output: $\left(\langle Q \rangle_{a|b}, \langle R \rangle_{a|b}\right)$

1 $\langle R \rangle_a \leftarrow \langle X \rangle_a$ *(virtual operation)*
2 $\langle R \rangle_b \leftarrow \text{BE}\left(\langle R \rangle_a, \mathcal{B}_a, \mathcal{B}_b\right)$
3 $\langle Q \rangle_b \leftarrow \left(\langle X \rangle_b - \langle R \rangle_b\right) \times \langle M_a^{-1} \rangle_b$
4 **if** $\langle Q \rangle_b = \langle -1 \rangle_b$ **then**
5 $\quad \langle Q \rangle_b \leftarrow \langle 0 \rangle_b$
6 $\quad \langle R \rangle_b \leftarrow \langle R \rangle_b - \langle M_a \rangle_b$
7 $\langle Q \rangle_a \leftarrow \text{BE}\left(\langle Q \rangle_b, \mathcal{B}_b, \mathcal{B}_a\right)$
8 **return** $\langle Q \rangle_{a|b}$, $\langle R \rangle_{a|b}$

The `Split` function can be used to propagate the "high" parts of the digits in parallel as described in Algo. 3. The loop in lines 2–3 is as parallel as the standard RNS since we have $d\frac{n}{d} = n$ independent computations (same thing for lines 4–5). The cost of Algo. 3 is the cost of d `Split`s.

Algorithm 3: "High" part propagation in HPR.

Input: $X_{\text{HPR}} = (\langle X_{d-1} \rangle, \ldots, \langle X_0 \rangle)$ with $X_i < (\beta_{max} M_a)^2$
Output: $X_{\text{HPR}} = (\langle X_d \rangle, \ldots, \langle X_0 \rangle)$ with $X_i < (\beta_{max}^2 + 1) M_a$

1 $\langle C_{-1} \rangle \leftarrow \langle 0 \rangle$, $\langle X_{d-1} \rangle \leftarrow \langle 0 \rangle$
2 **for** i **from** 0 **to** $d-1$ **parallel do**
3 $\quad (\langle C_i \rangle, \langle X_i \rangle) \leftarrow \texttt{Split}(\langle X_i \rangle)$
4 **for** i **from** 0 **to** d **parallel do**
5 $\quad \langle X_i \rangle \leftarrow \langle X_i \rangle + \langle C_{i-1} \rangle$
6 **return** $(\langle X_d \rangle, \ldots, \langle X_0 \rangle)$

The output of Algo. 3 is such that $X_i < (\beta_{max}^2 + 2) M_a$ instead of $X_i < \beta_{max} M_a$. In our applications, β_{max} is a small

integer ($\beta_{max} \leqslant 3$), and these extra bits are managed through the redundancy of the digits.

c) Comparison(s): Based on the position of the most-significant (non-zero) digit, a coarse grain comparison is very simple. In RNS such an approximate comparison is very costly (at least n^2 EMMs).

If the two compared operands have the same MSD weight, we can use an iterative process digit by digit. For each position, the comparison requires at least one BE but on $\frac{n}{d}$ moduli instead of n. If the digits at rank i are equal, this process is process repeated on digits at rank $i-1$ (until the LSD). In the worst case, an HPR comparison costs $d(\frac{n}{d})^2 = \frac{n^2}{d}$ EMMs instead of n^2 for standard RNS.

d) Conversions: The conversion from a standard positional representation to HPR is performed by successive divisions by M_a, and then at the digit level the classical RNS conversion is used. The opposite conversion is performed using the CRT for each digit and multiplications by M_a using a Horner scheme. Conversion from RNS to HPR is achieved using iterative divisions by M_a implemented using BEs with a decreasing number of moduli. The opposite conversion uses iterative multiplications by M_a implemented using complete BEs. In our target applications, very few conversions are required, and their costs are negligible compared to the cost of a ECC scalar multiplication or RSA exponentiation.

C. Small Carry Propagation

In the "high" part propagation algorithm 3, the "high" and "low" parts have similar magnitudes (*e.g.*, in the product of 2 digits) and require full BEs from $\mathcal{B}_a$ to $\mathcal{B}_b$ with $n_b \geqslant n_a$. However, after HPR additions, the magnitude of the "high" part is very small. As in [17], and other RNS works, we use BEs with a second base only composed of a single and small modulo m_γ (handled in a small parallel specific unit). Then Algo. 4 costs $d(n_a+n_b)$ EMMs (at lines $\{2,5\}$) and $d(n_a+1)$ small multiplication modulo m_γ (of cost less than 1 EMM).

Algorithm 4: HPR Small Carry Propagation.

Input: X_{HPR} with $X_i < (m_\gamma - 2)M_a \ \forall i \in [0, d-1]$
Precomp.: $\left|M_a^{-1}\right|_{m_\gamma}$
Output: X_{HPR}, $X_i < 2M_a + (m_\gamma - 2) \quad \forall i \in [0, d-1]$

1 **for** i **from** 0 **to** $d-1$ **do**
2 $\quad |R_i|_{m_\gamma} \leftarrow \text{BE}\left(\langle X_i\rangle_a, \mathcal{B}_a, m_\gamma\right)$
3 $\quad |C_i|_{m_\gamma} \leftarrow \left|(X_i - R_i)M_a^{-1}\right|_{m_\gamma}$
4 $\quad$ **if** $|C_i|_{m_\gamma} = m_\gamma - 1$ **then** $|C_i|_{m_\gamma} = 0$
5 $\quad \langle X_i\rangle_b \leftarrow \langle X_i\rangle_b - |C_{i,H}|_{m_\gamma} \times \langle M_a\rangle_b$
6 **for** i **from** 1 **to** $d-1$ **parallel do**
7 $\quad \langle X_i\rangle_{a|b} \leftarrow \langle X_i\rangle_{a|b} + \langle C_{i-1}\rangle_{a|b}$
8 **return** X_{HPR}

In practice, there is a trade-off between the width of m_γ and the application frequency of Algo. 4. Due to the approximation in the BE from [6] at line 4, we choose m_γ such HPR digits verify $X_i < (m_\gamma - 2)M_a$. The additional modulo m_γ limits how frequently Algo. 4 can be used. In case of a large sequence of successive additions, m_γ must be large enough.

In our applications, as modular multiplication in Sec. V-A, m_γ is a small power of two (4–8 bits).

D. A More Complex Operation: $(A \times B) \times (C \times D)$

In the (artificial) operation $R = (A \times B) \times (C \times D)$ over l-bit integers (with $l \leqslant n \times w$), we can show how HPR outperforms RNS in some computations. In a positional representation, a product is obviously twice larger than its operands. RNS is efficient for fixed-size operations (such as finite field ones), but it is not the case when the operands/results sizes vary inside the sequence of operations. In an RNS product, the operands must be represented using the same RNS base than the result, *i.e.*, twice more moduli.

Then in RNS, A, B, C and D are represented by $2n$ moduli of w bits to be able to represent the sub-products AB and CD. Each sub-product costs $2n$ EMMs. To compute R, each sub-product is extended from $2n$ to $4n$ moduli using a BE of cost $(2n)^2 + (2n)$ EMMs. The full product $AB \times CD$ costs $4n$ EMMs. The total cost for computing R in standard RNS is then $2\big((2n) + ((2n)^2 + (2n))\big) + 4n = 8n^2 + 12n$ EMMs.

In HPR, we assume $A_{\text{HPR}} = (\langle A_{d-1}\rangle, \ldots, \langle A_0\rangle)$ and $n_a = n_b = n/d$ (the same for B, C, D). First, we perform AB and CD using the schoolbook multiplication algorithm as the worst case one (faster multiplication algorithms would slightly increase HPR benefit) with d^2 sub-products $\langle A_i\rangle \times \langle B_j\rangle$. Each sub-product is computed in RNS on $n_a + n_b$ moduli and costs $2n/d$ EMMs. AB and CD both costs $d^2 \times \frac{2n}{d} = 2dn$ EMMs. At this point, we have $(2d-1)$ pairs ("high" and "low" parts) where the "high" parts have to be propagated to the next digits using Algo. 3. This requires $(2d-1)$ Splits and costs

$$(2d-1)\left(2\frac{n^2}{d^2} + 2\frac{n}{d}\right) = \left(\frac{4d-2}{d^2}\right)n^2 - \left(\frac{2}{d} + 4\right)n \quad \text{EMMs}.$$

Finally, the computation of $AB \times CD$ costs $(2d)^2 \times \frac{2n}{d} = 8dn$ EMMs. The total cost for computing R in HPR is

$$2\left(\left(\frac{4d-2}{d^2}\right)n^2 - \left(\frac{2}{d} + 4\right)n\right) + 12dn \quad \text{EMMs}.$$

R is fully computed in RNS and HPR but it cannot be used in a new multiplication. But, the redundancy provided by β_{min} and β_{max} parameters allows efficient additions in HPR (same thing applies for RNS). To use R as new multiplication operand, a new large BE is required in RNS (from $4n$ to $8n$ moduli) or a "high" parts propagation is required in HPR (over $4d$ digits). In such a case, HPR will lead to a significantly smaller cost.

Fig. 2 reports the R computation cost for RNS and HPR with $d \in \{2, 3, 4\}$ both for various n. For $d = 2$, there is a cost reduction of 20 % for $n = 6$, 30 % for $n = 10$ and 50 % for $n = 22$ in favor of HPR. The gain is even better for larger d when n increases. Replacing the schoolbook multiplication algorithm by a faster one may slightly increase the benefit for HPR (since d is small in practice).

E. Validation

We used two levels of validation. First, HPR is based on well known and robust properties of RNS and positional arithmetic algorithms. The most "complex" parts in HPR algorithms are the "high" part and small carry propagations for which we have proofs of the main elements. Second, we performed millions of random tests for cryptographic sizes for all our basic and more advanced algorithms and for various sets of parameters in Maple and Sage. We also performed millions of random tests for the applications presented in Sections V and VI.

V. Application 1: ECC Modular Multiplication

This section shows that HPR is interesting for specific finite field applications when the characteristic of the prime field P can be selected as in *elliptic curve cryptography* (ECC, see [24]). In the standard positional representation, the selection of the field characteristic as a (pseudo-)Mersenne prime ($P = 2^\ell - 1$ or $2^\ell - c$ with $c < 2^{\ell/2}$ for some ℓ) allows to significantly speed-up the modular reduction (based on a short sequence of additions in the field instead of multiplications).

There was no equivalent characteristic for RNS up to very recently. In [13], the specific characteristic $P = M_a^2 - 2$ was proposed to reduce the computation cost of modular arithmetic. HPR reduces the cost of modular reductions thanks to available position information, and it offers a significantly larger set of primes for the selection of the field characteristic compared to [13].

A. Algorithm Presentation

We propose to use a prime characteristic of the form $P = Q(M_a)$ where $Q \in \mathbb{Z}[X]$ is irreducible of degree d. In practice, we choose $Q = X^d - Q'$ where Q' is sparse. Our finite field elements are ℓ-bit large then $n_a \times d = n$ (with $n = \lceil \frac{\ell}{w} \rceil$).

Algo. 5 presents our modular multiplication.

In line 1, a degree-d positional product is performed (it is similar to a polynomial one, see for instance [25]).

Line 2 performs a first modular reduction using the identity $M_a^d \equiv Q'(M_a) \bmod P$. The HPR digits X^i with $i \geqslant d$ are multiplied by $Q'(M_a)$ (which is small and sparse), then aligned and added with low-degree digits. Selecting good values of Q is important for efficiency purpose (see Sec. V-D).

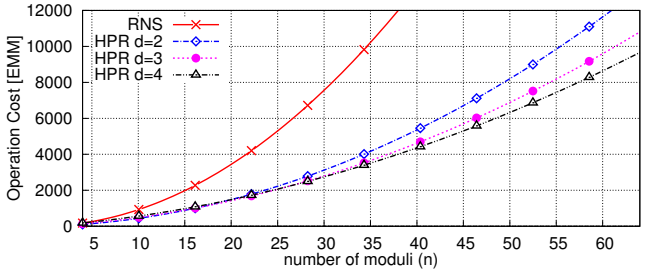

Fig. 2. Cost comparison of HPR and RNS for $(A \times B) \times (C \times D)$.

Algorithm 5: HPR Modular Multiplication.

Parameters: $\mathcal{B}_a$ with $P = Q(M_a)$ and Q of degree d
Input: X_{HPR}, Y_{HPR}
Output: Z_{HPR} with $Z = XY \bmod P$

1 $Z_{\text{HPR}} \leftarrow$ HPR Product($X_{\text{HPR}}, Y_{\text{HPR}}$)
2 $Z_{\text{HPR}} \leftarrow$ Positional Modular Reduction(Z_{HPR}, Q)
3 $Z_{\text{HPR}} \leftarrow$ HPR "High" Part Propagation (Z_{HPR})
4 $Z_{\text{HPR}} \leftarrow$ Positional Modular Reduction(Z_{HPR}, Q)
5 $Z_{\text{HPR}} \leftarrow$ HPR Small Carry Propagation (Z_{HPR})
6 **return** Z_{HPR}

For instance, if $Q' = 2$, the positional reduction consists in $Z_i \leftarrow Z_i + 2Z_{i+d}$ for $i \in [0; d-1]$.

Line 3 propagates the "high" part of the digits using Algo. 3.

Line 4 reduces the highest degree digit Z_d (of weight M_a^d) propagated by line 3. This new reduction is very cheap compared to the one at line 2.

Finally, line 5 propagates the very last small carries.

The cost of Algo. 5, detailed in section V-B, depends on various parameters, the most important ones are: the number of moduli n_a and the form of polynomial Q.

B. Cost Analysis

In ECC applications, the typical field size is $\ell \in [160, \ldots, 600]$ bits. For hardware implementations, the typical width of the moduli is $w \in [16, \ldots, 64]$ bits (DSP blocks in FPGAs commonly have operands from 9 to 25 bits). We assume field elements represented in HPR with d digits and $n_a = n_b = n/d$. We set $\beta_{min} = 0$ and $\beta_{max} = 3$ since the small carry propagation algorithm 4 guarantees output digits less than $3M_a$. As d is small in practice, and to keep simple this case study, we assume that the choice of the positional multiplication algorithm is the schoolbook one.

We explored various forms of the characteristic P. A very interesting one is $P = M_a^d - 2$, *i.e.* $Q = X^d - 2$ and $Q' = 2$. Obviously to find a such P, M_a must be odd as all its modulo, see Sec. V-D which presents the selection of P. Below the reported cost analysis is for this specific form of P.

Let us now analyze the cost of each line of Algo. 5. The positional multiplication at line 1 costs $d^2 \times (2n/d) = 2nd$ EMMs (with schoolbook algorithm). The positional reductions at lines 2 and 4 just compute d additions ($Z_i + 2Z_{i+d}$) and 1 addition ($Z_0 + 2Z_d$) respectively. The "high" part propagation at line 3 (Algo. 3) is more costly:

$$d \times \left(2\frac{n^2}{d^2} + 2\frac{n}{d}\right) = 2n\left(\frac{n}{d} + 1\right) \text{ EMMs}.$$

Using $n_a = n_b = n/d$, the small carry propagation at line 5 costs $2n$ EMMs (we neglect here the n small multiplications modulo m_γ).

Then the complete cost of Algo. 5 is $2n\left(\frac{n}{d} + d + 2\right)$ EMMs. Choosing $d = 2$, we recover the result reported in [13] with $n^2 + 8n$ EMMs using the schoolbook multiplication algorithm (or $n^2 + 7n$ EMMs using Karatsuba-Ofman method [22]).

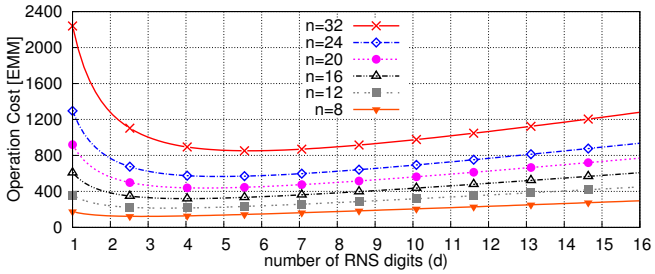

Fig. 3. Cost of HPR modular multiplication for fixed n and various d.

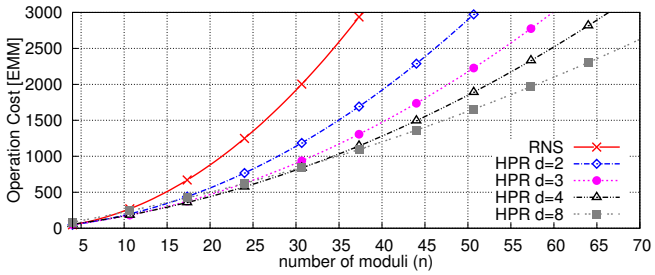

Fig. 4. Cost of modular multiplication in RNS and HPR for various n.

Table I reports the cost of the HPR modular multiplication from Algo. 5 for various d using schoolbook multiplication. For larger d, it can be interesting to use other multiplication algorithms.

Figures 3 and 4 illustrate the behavior of the HPR modular multiplication for various parameters d and n. Fig. 3 shows that for a fixed n, the minimum cost is reached for $d \approx \sqrt{n}$. It comes from the derivative in d of the total cost: $\frac{-2n^2}{d^2} + 2n = 0$ gives $2d^2n = 2n^2$ thus $d = \sqrt{n}$ ($d > 1$). Then in practice, d is small. Fig. 4 shows that HPR is always faster than RNS even if d is not so close from $\sqrt{n}$.

C. Sources of Parallelism in HPR

In RNS, the parallelism naturally comes from the independence of the n residues computations which are often implemented into n parallel units. HPR benefits of this parallelism at the digit-level computations. For a given field size ℓ, the digit-level parallelism reduces when d grows due to fewer moduli in the base.

Actually, using our modular multiplication algorithm 5, the digit level is not the only source of parallelism. First, in the product of two degree-d HPR values, there are d^2 digit by digit sub-products which can be computed in parallel. Each sub-product costs $2\frac{n}{d}$ EMMs performed in parallel (since each digit is an RNS value). The total cost of a degree-d HPR product is $2dn$ parallel EMMs over n parallel units. Second, as seen in Sec. IV-B, the HPR carry and "high" part propagations can be performed over n parallel units. However the number of additions grows with d, and thus introduces data dependencies.

As a conclusion, if d is sufficiently small compared to n, HPR has a similar level of parallelism compared to RNS. For implementations of ECC modular multiplication, a good choice is, at most, $d \approx \sqrt{n}$.

d	2	4	8	16
cost (EMM)	$n^2 + 8n$	$\frac{n^2}{2} + 12n$	$\frac{n^2}{4} + 20n$	$\frac{n^2}{8} + 36n$

TABLE I
COST OF DEGREE-d HPR MODULAR MULTIPLICATION WITH ALGO. 5.

D. Choice of Characteristic P

For hardware implementations one usually uses moduli sizes $w \in [16, \ldots, 64]$ bits and (pseudo-)Mersenne moduli m_i (for fast internal reductions in the computations of the individual residues). Generating characteristics of the form $P = Q(M_a)$ with $Q = X^d - Q'$ and Q' with a very small Hamming weight is not very difficult. In a first time, we use a fast probabilistic primality test (such as the Maple `is_prime` function). Then we select the most convenient candidates with respect to implementation purpose. Finally, one can test only the most relevant final candidate using a very costly deterministic primality test. For the characteristic form $P = M_a^4 - 2$ with 256-bit ECC, using the parameters $n_a = 4$ and $w = 16$, we can generate more than 1000 candidates in less than 7 s using a simple laptop with an Intel I7 processor running at 2.2 GHz. For 512 bits, $n_a = 4$ and $w = 32$, less than 10 s are required to generate more than 1000 candidates.

Using a pseudo-Mersenne prime for the characteristic in the standard binary representation leads to very efficient reductions but a specific irreducible polynomial is required for each field size (due to a very small number of efficient candidates, see for instance [26]). This leads to dedicated (and different) computation units for each field. HPR allows to use the same characteristic form $P = M_a^d - 2$ for different field sizes or for different primes of a given size ℓ. Then the same type of units architecture can be shared for various field sizes or primes characteristics. Only the precomputations are specific to each set of parameters (but the data-path remains the same for a given w parameter).

VI. Application 2: RSA Exponentiation

Below we propose an RSA exponentiation algorithm with a reduced number of operations compared to the best state-of-the-art RNS algorithm [10]. A more efficient RNS exponentiation has been proposed in [23] but it only works for specific modulus and then it cannot be used for RSA where the modulus is the public key. For the same reason, the modular multiplication proposed in Sec. V does not work for RSA.

Our exponentiation algorithm, detailed in Sec. VI-B, is designed for operands and result represented in standard RNS with intermediate computations performed using HPR.

A. Enlarged Conversion from RNS to HPR

Our exponentiation algorithm uses three internal bases $\mathcal{B}_a$, $\mathcal{B}_b$ and $\mathcal{B}_c$ with $n_a = n_b = n/d$ and $n_c = n$. We perform the RNS to HPR conversion using Algo. 6. This conversion is

enlarged because the digits are given in $\mathcal{B}_{a|b}$ but also in $\mathcal{B}_c$, as required by our exponentiation algorithm 7.

Algo. 6 iteratively produces the digits $\langle X_i \rangle$ of the HPR result one by one with successive divisions $X_i = \frac{X_i - X_{i-1}}{M_a}$ using optimized BEs. At each iteration, the division by M_a is not possible in base $\mathcal{B}_a$. Then BE at line 3 converts the current value from base $\mathcal{B}_a$ into the base $\mathcal{B}_{b|c}$. The division can be now computed at line 4. Lines 5 and 6 prevent errors due to BE approximation from [6], as in HPR "high" part propagation and small carry propagation in Sec. IV.

The second BE at line 7 converts back the current value from base $\mathcal{B}_{b|c}$ into base $\mathcal{B}_a$. In this BE, as the current value decreases, less moduli from base $\mathcal{B}_c$ are required. Then $\mathcal{B}_{c(i)}$ denotes the reduced base required at iteration i. At each iteration, $\frac{n}{d}$ moduli are discarded at line 7 in base $\mathcal{B}_c$.

Algorithm 6: RNStoHPR Conversion with 3 Bases.

Parameters: $\mathcal{B}_a, \mathcal{B}_b, \mathcal{B}_c$ with $n_a = n_b = n/d$ and $n_c = n$
Parameters: $\mathcal{B}_{c(i)} = (m_{c,0}, \ldots, m_{c,(d-i)n/d})$
Precomp.: $\langle M_a^{-1} \rangle_{b|c}$
Input: $\langle X \rangle_{a|b|c}$
Output: X_{HPR} with digits in $\mathcal{B}_{a|b|c}$

1 $\langle X_0 \rangle_a \leftarrow \langle X \rangle_a, \quad \langle X_1 \rangle_{b|c} \leftarrow \langle X \rangle_{b|c}$
2 **for** i **from** 1 **to** $d-1$ **do**
3 $\quad \langle X_{i-1} \rangle_{b|c} \leftarrow \text{BE}(\langle X_{i-1} \rangle_a, \mathcal{B}_a, \mathcal{B}_{b|c})$
4 $\quad \langle X_i \rangle_{b|c} \leftarrow \left(\langle X_i \rangle_{b|c} - \langle X_{i-1} \rangle_{b|c}\right) \times \langle M_a^{-1} \rangle_{b|c}$
5 $\quad$ **if** $\langle X_i \rangle_{b|c} = \langle -1 \rangle_{b|c}$ **then**
6 $\quad\quad \langle X_i \rangle_{b|c} \leftarrow \langle 0 \rangle_{b|c}$
7 $\quad \langle X_i \rangle_a \leftarrow \text{BE}(\langle X_i \rangle_b, \mathcal{B}_{c(i)}, \mathcal{B}_a)$
8 **return** $X_{HPR} = \left(\langle X_{d-1} \rangle_{a|b|c}, \ldots, \langle X_0 \rangle_{a|b|c}\right)$

BE at line 3 costs $\frac{n}{d}(\frac{n}{d} + n + 1)$ EMMs and line 4 costs $(n + n/d)$ EMMs. At iteration i, BE at line 7 costs $\frac{in^2}{d^2}$ EMMs. Then the total cost of Algo. 6 is $\left(\frac{-1}{2d} - \frac{1}{d^2} + \frac{3}{2}\right) n^2 + \left(\frac{-2}{d} + 1 + d\right) n$ EMMs. The conversion cost grows with d, for instance when $d = 2$ the cost is $n^2 + 2$ EMMs, and for $d = 3$ it is $\frac{11}{9}n^2 + \frac{10}{3}$ EMMs.

B. Exponentiation Algorithm

Our RSA exponentiation is detailed in Algo. 7 and computes $Z = G^e \bmod N$. We use the RNS Montgomery reduction (RNS−MR) with all values in the RNS Montgomery domain (*i.e.* each is multiplied by $M_{a|b}^{-1}$). The idea is to compute $Z^2 \cdot C$ in HPR where C is fixed and then perform a "partial" modular reduction using the precomputations $|M_a^k C|_N$ for $k \in [0, 2(d-1)]$. This adds $(2d-2) \cdot (nw)$ bits to be stored, which is small compared to the usual precomputations since d is very small in practice. Then one computes the RNS representation of this partially reduced value using

$$|Z^2 G|_N \equiv \sum_{k=0}^{2(d-1)} \sum_{i=0}^{d-1} Z_i Z_{k-i} |M_a^k C|_N \ . \tag{3}$$

Thanks to the precomputation of $|M_a^k C|_N$, the right term of equation 3 is partially reduced modulo N: its size is $\ell + (2\ell/d)$ bits instead of 3ℓ bits for $Z^2 G$. This explains why $n + (2n/d) = n_a + n_b + n_c$ moduli are required to contain this partial reduction.

Algo. 7 is a variant of the *square-and-multiply* algorithm (see [24] for instance). If the bit exponent is 0 then a square is computed, otherwise a square and a multiplication by G are performed. Our algorithm is regular and always computes $|Z^2 C|_N$ with $C = G$ or $C = M_{a|b}^{-1}$ (the Montgomery representation of 1). The "SubProducts" at lines 5 and 8 correspond to the straightforward evaluation of equation 3. Lines 6 and 9 perform one RNS-MR using $\mathcal{B}_{a|b}$ as first base ($2n/d$ moduli) and $\mathcal{B}_c$ (n moduli). The regularity of exponentiation algorithms is important in cryptographic implementations to protect them against some side-channel attacks, as the SPA (simple power analysis) attack.

Algorithm 7: Proposed Regular Modular Exponentiation.

Parameters: $\mathcal{B}_a, \mathcal{B}_b, \mathcal{B}_c$ with $n_a = n_b = n/d$ and $n_c = n$
Input: $\langle G \rangle_{a|b|c}$, e the exponent
Output: $\langle Z \rangle_{a|b|c}$ with $Z = G^e \bmod N$, $Z < 3P$

1 $\langle Z \rangle_{a|b|c} \leftarrow \langle |M_{a|b}^{-1}|_P \rangle_{a|b|c}$
2 **for** i **from** $\ell - 1$ **to** 0 **do**
3 $\quad Z_{\text{HPR}} \leftarrow \text{RNStoHPR}(\langle Z \rangle_{a|b|c}, \mathcal{B}_a, \mathcal{B}_b, \mathcal{B}_c)$
4 $\quad$ **if** $e_i = 0$ **then**
5 $\quad\quad \langle Z \rangle_{a|b|c} \leftarrow \text{SubProducts}(Z_{\text{HPR}}, \langle |M_{a|b}^{-1}|_P \rangle)$
6 $\quad\quad \langle Z \rangle_{a|b|c} \leftarrow \text{RNS-MR}(\langle Z \rangle_{a|b|c}, \mathcal{B}_{a|b}, \mathcal{B}_c)$
7 $\quad$ **else**
8 $\quad\quad \langle Z \rangle_{a|b|c} \leftarrow \text{SubProducts}(Z_{\text{HPR}}, \langle G \rangle)$
9 $\quad\quad \langle Z \rangle_{a|b|c} \leftarrow \text{RNS-MR}(\langle Z \rangle_{a|b|c}, \mathcal{B}_{a|b}, \mathcal{B}_c)$
10 **return** $\langle Z \rangle_{a|b|c}$

C. Cost Analysis

For the evaluation of Algo. 7, we start by evaluating one iteration of the loop. The conversion step has already been evaluated, so we evaluate the cost of the sub-products $Z_i Z_j$. Because a square is performed, $Z_i Z_j = Z_j Z_i$ thus we do not have d^2 products but only $d(d+1)/2$ products to compute on $\mathcal{B}_a$, $\mathcal{B}_b$ and $\mathcal{B}_c$. Then, after the addition of the sub-products of the same weight, one computes the $(2d-1)$ products $|M_a^k C|_N \times (\sum Z_i Z_{k-i})$ and add them. As an example, for $d = 2$ one computes $(Z_1^2) \times |M_a^2 C|_N + (2Z_1 Z_0) \times |M_a C|_N + (Z_0^2) \times |C|_N$. The total cost of the sub-products part is $\left(\frac{d^2}{2} + \frac{7d}{2} - \frac{2}{d} + 4\right) n$ EMMs.

It is possible to reduce the number of products using some factorizations: for instance with $d = 2$ one can compute $Z_1 \times \left(Z_1 |M_a^2 C|_N + (2Z_2) \times |M_a C|_N\right) + (Z_0^2) \times |C|_N$ to save one multiplication on the 3 bases, saving $2n$ EMMs.

The cost of RNS−MR is $2\left(\frac{2n^2}{d}\right) + n$ EMMs, after optimization of the first constant multiplication in RNS−MR which saves $2n/d$ EMMs. Then, one loop iteration costs

$$\left(\frac{7}{2d} - \frac{1}{d^2} + \frac{3}{2}\right) n^2 + \left(\frac{-4}{d} + 6 + \frac{9d}{2} + \frac{d^2}{2}\right) n \text{ EMMs}.$$

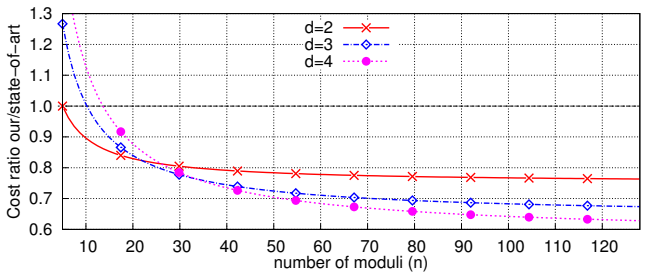

Fig. 5. Comparison of our regular exponentiation algorithm 7 based on HPR and a regular version of the RNS state-of-the-art algorithm from [10].

For instance, it gives $3n^2+15n$ EMMs for $d=2$ (or $3n^2+13n$ with the previous factorizations), and $\frac{37}{16}n^2+31n$ EMMs for $d=4$ against $4n^2+8n$ for state-of-the-art RNS. Fig. 5 presents the gain of this exponentiation with $d=2,3,4$ compared to one RNS regular exponentiation, such as the Montgomery ladder exponentiation from [27]. Values are presented up to 128 moduli, which can be applied for 64-bit implementations of RSA-8192 or 32-bit implementations of RSA-4096. For $n=32$, the gain goes from 20 % to 23 % with $d=2$ to 4. For large n like 128, $d=2,3$ and 4 give a 24 %, 33 % and 38 % cost reduction respectively.

VII. Conclusion

Our *hybrid position-residues* (HPR) number system provides various trade-offs between internal parallelism and the efficiency of operations requiring position information for large integers and prime field elements. In asymmetric cryptography applications, HPR leads to more efficient modular multiplications compared to RNS. For 256–512 bits ECC, HPR offers 40 to 60 % computation cost reduction. For 2048–4096 bits RSA, the computation cost reduction is 20 to 40 %.

We will now work on the selection of the various parameters and complete FPGA implementations for asymmetric cryptography applications.

Acknowledgment

This work has been supported in part by the PAVOIS project (ANR 12 BS02 002 01).

References

[1] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," *IRE Transactions on Electronic Computers*, vol. 10, no. 3, pp. 389–400, Sep. 1961.

[2] E. E. Swartzlander and A. G. Alexopoulos, "The sign/logarithm number system," *IEEE Transactions on Computers*, vol. C-24, no. 12, pp. 1238–1242, Dec. 1975.

[3] A. Svoboda and M. Valach, "Operátorové obvody (operator circuits in czech)," *Stroje na Zpracování Informací (Information Processing Machines)*, vol. 3, pp. 247–296, 1955.

[4] H. L. Garner, "The residue number system," *IRE Transactions on Electronic Computers*, vol. EC-8, no. 2, pp. 140–147, Jun. 1959.

[5] J.-C. Bajard, L.-S. Didier, and P. Kornerup, "An RNS Montgomery modular multiplication algorithm," *IEEE Transactions on Computers*, vol. 47, no. 7, pp. 766–776, Jul. 1998.

[6] S. Kawamura, M. Koike, F. Sano, and A. Shimbo, "Cox-Rower architecture for fast parallel Montgomery multiplication," in *Proc. Internat. Conf. Theory and Application of Cryptographic Techniques (EUROCRYPT)*, ser. LNCS, vol. 1807. Springer, May 2000, pp. 523–538.

[7] H. Nozaki, M. Motoyama, A. Shimbo, and S. Kawamura, "Implementation of RSA algorithm based on RNS Montgomery multiplication," in *Proc. Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, vol. 2162. Springer, May 2001, pp. 364–376.

[8] N. Guillermin, "A high speed coprocessor for elliptic curve scalar multiplications over $\mathbb{F}_p$," in *Proc. Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, vol. 6225. Springer, Aug. 2010, pp. 48–64.

[9] P. M. Matutino, R. Chaves, and L. Sousa, "An efficient scalable RNS architecture for large dynamic ranges," *Journal of Signal Processing Systems*, vol. 77, no. 1, pp. 191–205, Oct. 2014.

[10] F. Gandino, F. Lamberti, G. Paravati, J.-C. Bajard, and P. Montuschi, "An algorithmic and architectural study on Montgomery exponentiation in RNS," *IEEE Transactions on Computers*, vol. 61, no. 8, pp. 1071–1083, Aug. 2012.

[11] B. Gérard, J.-G. Kammerer, and N. Merkiche, "Contributions to the design of residue number system architectures," in *Proc. 22nd Internat. Symp. Computer Arithmetic (ARITH)*. IEEE, Jun. 2015, pp. 105–112.

[12] J.-C. Bajard, J. Eynard, N. Merkiche, and T. Plantard, "RNS arithmetic approach in lattice-based cryptography: Accelerating the rounding-off core procedure," in *Proc. 22nd Internat. Symp. Computer Arithmetic (ARITH)*. IEEE, Jun. 2015, pp. 113–120.

[13] K. Bigou and A. Tisserand, "Single base modular multiplication for efficient hardware RNS implementations of ECC," in *Proc. 17th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, vol. 9293. Springer, Sep. 2015, pp. 123–140.

[14] D. Schinianakis and T. Stouraitis, "Multifunction residue architectures for cryptography," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 4, pp. 1156–1169, Apr. 2014.

[15] N. S. Szabo and R. I. Tanaka, *Residue arithmetic and its applications to computer technology*. McGraw-Hill, 1967.

[16] J.-C. Bajard, L.-S. Didier, and P. Kornerup, "Modular multiplication and base extensions in residue number systems," in *Proc. 15th Symposium on Computer Arithmetic*. IEEE, Apr. 2001, pp. 59–65.

[17] A. P. Shenoy and R. Kumaresan, "Fast base extension using a redundant modulus in RNS," *IEEE Transactions on Computers*, vol. 38, no. 2, pp. 292–297, Feb. 1989.

[18] K. C. Posch and R. Posch, "Modulo reduction in residue number systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 5, pp. 449–454, May 1995.

[19] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, Apr. 1985.

[20] A. Skavantzos and F. Taylor, "On the polynomial residue number system," *IEEE Transactions on Signal Processing*, vol. 39, no. 2, pp. 376–382, Feb. 1991.

[21] J.-C. Bajard, L. Imbert, and T. Plantard, "Modular number systems: Beyond the Mersenne family," in *Proc. 20th International Workshop on Selected Areas in Cryptography (SAC)*, ser. LNCS, vol. 3357. Springer, Aug. 2004, pp. 159–169.

[22] A. Karatsuba and Y. Ofman, "Multiplication of multi-digit numbers on automata," *Doklady Akad. Nauk SSSR*, vol. 145, no. 2, pp. 293–294, 1962, translation in Soviet Physics-Doklady, 44(7), 1963, p. 595-596.

[23] K. Bigou and A. Tisserand, "RNS modular multiplication through reduced base extensions," in *Proc. 25th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, Jun. 2014, pp. 57–62.

[24] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[25] J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*, 2nd ed. Cambridge University Press, 2003.

[26] N. I. of Standards and T. (NIST), "FIPS 186-2, digital signature standard (DSS)," 2000.

[27] M. Joye and S.-M. Yen, "The Montgomery powering ladder," in *Proc. 4th Internat. Workshop Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, vol. 2523. Springer, Aug. 2002, pp. 291–302.