Hierarchical Approach in RNS Base Extension for Asymmetric Cryptography

Libey Djath and Karim Bigou
Université de Bretagne Occidentale
Lab-STICC, UMR CNRS 6285
F-29200 Brest, France
libeyokonfu.djath@univ-brest.fr,
karim.bigou@univ-brest.fr

Arnaud Tisserand
CNRS
Lab-STICC, UMR 6285
Centre Recherche UBS, rue St Maudé, Lorient, France
arnaud.tisserand@univ-ubs.fr

***Abstract*—Base extension is a critical operation in RNS implementations of asymmetric cryptosystems. In this paper, we propose a new way to perform base extensions using a hierarchical approach for computing the Chinese remainder theorem. For well chosen parameters, it significantly reduces the computational cost and still ensures a high level of internal parallelism. We illustrate the interest of the proposed approach on the cost of typical arithmetic primitives used in asymmetric cryptography. We also demonstrate improvements in FPGA implementations of base extensions on typical elliptic curve cryptography field sizes using high-level synthesis tools.**

***Index Terms*—computer arithmetic; residue number system; modular reduction; hardware implementation.**

I. Introduction

Current asymmetric cryptosystems require an efficient support for *arithmetic over large operands*. For instance, RSA [1] requires modular arithmetic over integers larger than 2000 bits and *elliptic curve cryptography* (ECC) [2], [3] deals with *finite field* elements larger than 200 bits. See [4] for references.

In these cryptographic applications, the *residue number system* (RNS) [5], [6] is increasingly suggested to provide more internal parallelism. RNS uses a set of small *co-prime moduli*, called the *base*, to "split" some arithmetic operations into *independent* and much *smaller* ones over the *residues*. This independence leads to faster operations without carry propagation between the moduli [7]. RNS can also help to improve the security against some physical attacks [8].

In RNS, addition, subtraction and multiplication of large integers are *parallel operations*. The *Chinese remainder theorem* (CRT) is often used for converting the residues into the standard representation, see for instance [7, Chap. 3].

But operations such as comparison, division and modular reduction are costly operations since RNS is a *non-positional representation*. To avoid the conversion to the standard representation for these operations, the use of *base extensions* (BE) is often proposed [9].

To reduce the cost of RNS implementations, two main directions have been explored: reducing the number of BEs at application level (*e.g.*, specific formulas for point addition/doubling in ECC [10]), or reducing the cost of a BE (for instance [11]). This work deals with the second direction and proposes a *new BE algorithm*. It uses a *hierarchical decomposition* with *partial applications of small CRTs*. Our algorithm, called *hierarchical BE* (HBE), allows to reduce the cost of modular arithmetic in RNS.

Definitions and notations are presented in Section II. Section III briefly recalls related elements from the state of the art. Our HBE algorithm is detailed in Section IV. The interest of this algorithm is analyzed in Section V for a few asymmetric cryptography applications. Section VI presents our FPGA implementations results for ECC over $\mathbb{F}_P$ using high-level synthesis (HLS). Finally, Section VII concludes the paper.

II. Definitions and Notations

We use a 2-dimension notation for RNS bases and elements instead of the usual 1-dimension one.

- $|X|_m = X \bmod m$
- $\mathcal{A}$ is an RNS base of $n = r \times c$ moduli of w bits:

$$\mathcal{A} = \begin{pmatrix} a_{1,1} & \cdots & a_{1,c} \\ \vdots & \cdots & \vdots \\ a_{r,1} & \cdots & a_{r,c} \end{pmatrix}$$

- $A_i = \prod_{j=1}^{c} a_{i,j}$ is the product of the i-th row moduli
- $A = \prod_{i=1}^{r} A_i$ is the product of all moduli of $\mathcal{A}$
- $\overline{A_i} = A/A_i$ is the product of all rows except the i-th one
- $\overline{a_{i,j}} = A_i/a_{i,j}$ is the product of all moduli in row i except the j-th one
- The integer X is represented in RNS base $\mathcal{A}$ by:

$$X_A = \begin{pmatrix} x_{a_{1,1}} & \cdots & x_{a_{1,c}} \\ \vdots & \cdots & \vdots \\ x_{a_{r,1}} & \cdots & x_{a_{r,c}} \end{pmatrix} \quad \text{with} \quad x_{a_{i,j}} = |X|_{a_{i,j}}$$

- $X_{A_i} = |X|_{A_i}$ and it can be computed from $(x_{a_{i,1}}, \cdots, x_{a_{i,c}})$ using the CRT
- $T_{a_{i,j}} = \left| \left(\frac{A}{a_{i,j}} \right)^{-1} \right|_{a_{i,j}}$
- Elementary binary operations with (w, w') bits operands are: addition, subtraction, multiplication, *modular multiplication* (MM) and *modular reduction* (MR)
- Their costs are denoted:
  - $\texttt{CADD}(w, w')$ for a $(w \pm w')$-bit addition/subtraction
  - $\texttt{CMUL}(w, w')$ for a $(w \times w')$-bit multiplication

- `CMR`(w', w) for a $(w' \bmod w)$-bit MR
- `CMM`(w, w) for a $(w \times w \bmod w)$-bit MM (in one w-bit RNS channel)

RNS base $\mathcal{B}$ and the related values B_i, B, $\overline{B_i}$, $\overline{b_{i,j}}$, X_B, X_{B_i} and $T_{b_{i,j}}$ are similarly defined.

III. State of the Art

A. Residue Number System

In RNS [5], [6], the integer X is represented by its *residues*, denoted $x_{a_{i,j}}$, modulo a set of *coprime moduli*, denoted $a_{i,j}$, for all (i, j) in the RNS *base* $\mathcal{A}$. To convert X from a *standard positional representation* to RNS, one computes $|X|_{a_{i,j}}$, possibly *independently*, for each moduli in $\mathcal{A}$. The reverse conversion is performed using the *CRT formula*:

$$X = \left| \sum_{i=1}^{r} \sum_{j=1}^{c} |x_{a_{i,j}} \times T_{a_{i,j}}|_{a_{i,j}} \times \frac{A}{a_{i,j}} \right|_A \quad (1)$$

With X and Y represented in RNS base $\mathcal{A}$, their addition, subtraction and multiplication, $X \diamond Y$ where $\diamond \in \{\pm, \times\}$, is performed *independently* on each residue by $|x_{i,j} \diamond y_{i,j}|_{a_{i,j}}$ for all (i, j) in $\mathcal{A}$. Computations related to one modulo are performed in a *channel* (*i.e.* w-bit datapath). Clearly, RNS offers a high level of parallelism for $\pm$ and $\times$ operations. But for other operations, especially comparison, division and MR, the situation is more complex. Their cost in RNS is more important than for a positional representation. Multiplication modulo a large integer M is crucial for asymmetric cryptography ($M = P$ is prime for ECC over $\mathbb{F}_P$ and $M = PQ$ the product of 2 primes for RSA). In RNS, the cost of this operation is mainly the cost of 2 successive base extensions (see Section V).

B. RNS Base Extension

A BE can be seen as a conversion from one RNS base $\mathcal{A}$ to a second one $\mathcal{B}$. If $\mathcal{B}$ is co-prime with $\mathcal{A}$, then the concatenation of both bases can be seen as an extension of $\mathcal{A}$.

In the literature, two main strategies are used for BEs: using an intermediate representation called *mixed-radix system* (MRS [5]) or computing equation 1 *directly* in base $\mathcal{B}$. MRS requires more operations and introduces strong data dependencies which limit its interest compared to CRT approaches. For CRT based BEs, the main issue is to compute the reduction modulo A from Eq. 1 directly in base $\mathcal{B}$, other operations are just sums and products. Usually, one instead computes the CRT under the form:

$$X = \left(\sum_{i=1}^{r} \sum_{j=1}^{c} |x_{a_{i,j}} \times T_{a_{i,j}}|_{a_{i,j}} \times \frac{A}{a_{i,j}} \right) - hA \quad (2)$$

The issue is to compute h (*i.e.*, how many times A should be subtracted to get the correct MR). Authors of [12] noticed that in some cryptographic applications, reduction modulo A can be skipped after the sum, leading to larger values (less than rcA instead of A). [13] proposed to use an extra modulo to retrieve h. This method cannot be used in some situations but can be combined to [12] to fully implement RSA and ECC in RNS. A third approach proposed in [14], and improved in [9], uses an approximation for h in Eq. 2. In this paper, we focus on [9], which is the most used in state-of-the-art implementations. Our main idea can be adapted to other CRT based BE methods.

Algorithm 1: Base Extension from [9] (KBE).

Input: X_A, $\sigma = 0$ or 0.5
Precomp.: $T_{a_{i,j}}$ $\forall i \in [1, r]$ and $\forall j \in [1, c]$
Output: X_B

1 **for** *i* **from** 1 **to** r **parallel do**
2 **for** *j* **from** 1 **to** c **parallel do**
3 $\widehat{x}_{a_{i,j}} \leftarrow \left| x_{a_{i,j}} \times T_{a_{i,j}} \right|_{a_{i,j}}$
4 **for** *i* **from** 1 **to** r **do**
5 **for** *j* **from** 1 **to** c **do**
6 $\sigma \leftarrow \sigma + \frac{\mathrm{trunc}(\widehat{x}_{a_{i,j}})}{2^w}$
7 $h_{i,j} \leftarrow \lfloor \sigma \rfloor$
8 $\sigma \leftarrow \sigma - h_{i,j}$
9 **for** *k* **from** 1 **to** r **parallel do**
10 **for** *l* **from** 1 **to** c **parallel do**
11 $x_{b_{k,l}} \leftarrow \left| x_{b_{k,l}} + \widehat{x}_{a_{i,j}} \times \left| \frac{A}{a_{i,j}} \right|_{b_{k,l}} + |-h_{i,j}\, A|_{b_{k,l}} \right|_{b_{k,l}}$

Algo. 1 (with our 2D notations). One can see that lines 3 and 11 mainly compute the sum of products from Eq. 2. The lines 6–8 of KBE perform an accumulation of small t-bit values to compute $h_{i,j}$, and finally subtract $h_{i,j}A$ in line 11. The $h_{i,j}$ are 1-bit values and $\sum_{i=1}^{r} \sum_{j=1}^{c} h_{i,j}$ is h or $h - 1$. The function $\mathrm{trunc}(x)$ keeps the t MSBs of $\widehat{x}_{a_{i,j}}$, and sets all the others to 0. For asymmetric cryptography implementations, $t \in [4, 8]$ is very common. KBE can be used into 2 main different modes:

- if input $X < A/2$, choosing $\sigma = 0.5$ leads to $\sum_{i=1}^{r} \sum_{j=1}^{c} h_{i,j} = h$ and the output is *exactly* X in base $\mathcal{B}$;
- if X is close to A with $X < A$, choosing $\sigma = 0$ leads to $\sum_{i=1}^{r} \sum_{j=1}^{c} h_{i,j} = h - 1$ and the output is X or $X + A$ in base $\mathcal{B}$.

Thus, KBE can be used to perform all computations in asymmetric cryptography. KBE is efficiently implemented using the `cox-rower` architecture introduced in [9] and depicted in Fig. 1. A `rower` unit performs all computations in one channel. All `rowers`, one per modulo in the base, operate in *parallel*. The single `cox` unit computes the appropriate reduction factor $h_{i,j}$ and distributes it to the `rowers`. KBE algorithm and `cox-rower` architecture have been used in several hardware implementations of asymmetric cryptosystems using RNS: *e.g.*, [15] for RSA; [16] and [17] for ECC.

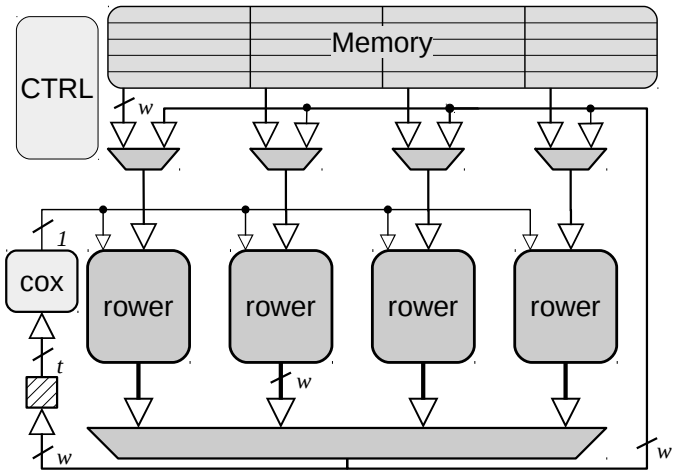

Fig. 1. The `cox-rower` architecture from [16] inspired by the one from [9] (control signals are not represented).

C. Hierarchical Approaches in RNS

[18] and [19] propose to recursively use RNS where computations inside channels are performed in a small RNS base. This does not improve much performances compared to usual RNS, but it provides other properties such as protection against some physical attacks. [19] proposes small RNS channels where all elementary operations are performed using fully precomputed lookup tables (reducing information leakage).

In [20], [21], RNS bases with 3 or 4 moduli are used for signal processing applications (on small values) where some moduli are factorizable into smaller ones. To be efficient, the moduli and their factorization must support fast MR algorithms in the corresponding (sub)-channels.

In this paper, we do not use an RNS hierarchical representation, but we propose a hierarchical approach for the CRT computation to reduce the cost of cryptographic applications. There is no need for factorizable moduli, and all MRs are performed using usual RNS w-bit moduli (see Sec.IV).

To improve the CRT computation in a general context, hierarchical approaches with partial CRT have been proposed to convert back from a set of residues to the corresponding value in $\mathbb{Z}$, see for instance [22], [23]. However, as far as we know, our proposition is the first BE algorithm based on a hierarchical approach for full RNS computations.

IV. Proposed Algorithm

A. New Base Extension Algorithm

Our *Hierarchical Base Extension* (HBE) is detailed in algorithm 2. As in the KBE algorithm, HBE mainly computes the CRT Eq. 2 from the base $\mathcal{A}$ and the result is reduced modulo each element in base $\mathcal{B}$. Lines 1–3 are exactly the same as in KBE. HBE uses intermediate CRT computations (lines 4–7) independently for each row $i \in [1, r]$: $(x_{a_{i,1}}, \ldots, x_{a_{i,c}})$. The result at row i is $\widehat{X}_{A_i} \equiv X_{A_i} \times \left(\overline{A_i}\right)^{-1} \bmod A_i$. No modular reduction is performed at this step, thus $\widehat{X}_{A_i} < cA_i$.

Algorithm 2: Proposed hierarchical base extension HBE.

Input: X_A, $\sigma = 0$ or 0.5
Precomp.: $T_{a_{i,j}}$ $\forall i \in [1, r]$ and $\forall j \in [1, c]$
Output: X_B

1 **for** *i* **from** 1 **to** r **parallel do**
2 **for** *j* **from** 1 **to** c **parallel do**
3 $\widehat{x}_{a_{i,j}} \leftarrow \left|x_{a_{i,j}} \times T_{a_{i,j}}\right|_{a_{i,j}}$
4 **for** *i* **from** 1 **to** r **parallel do**
5 $\widehat{X}_{A_i} \leftarrow 0$
6 **for** *j* **from** *1* **to** c **do**
7 $\widehat{X}_{A_i} \leftarrow \widehat{X}_{A_i} + \widehat{x}_{a_{i,j}} \times \overline{a_{i,j}}$ *(no reduction)*
8 **for** *i* **from** 1 **to** r **do**
9 $\sigma \leftarrow \sigma + \frac{\mathrm{trunc}(\widehat{X}_{A_i})}{2^{w \times c}}$
10 $h_i \leftarrow \lfloor \sigma \rfloor$
11 $\sigma \leftarrow \sigma - h_i$
12 **for** *k* **from** 1 **to** r **parallel do**
13 **for** *l* **from** 1 **to** c **parallel do**
14 $\widehat{x}_{b_{k,l,i}} \leftarrow \left|\widehat{X}_{A_i}\right|_{b_{k,l}}$
15 $x_{b_{k,l}} \leftarrow$ $\left|x_{b_{k,l}} + \widehat{x}_{b_{k,l,i}} \times \overline{A_i} + |-h_i A|_{b_{k,l}}\right|_{b_{k,l}}$

The value $\widehat{X}_{A_i}$ can be seen as a *super-residue*, corresponding to the *super-modulo* A_i.

The nested loops in lines 8–15 replace lines 4–11 of KBE. Lines 9–11 in HBE are an adaptation of the approximation method proposed in [9] to compute h_i from our super-residues $\widehat{X}_{A_i}$ (h_i is $\lceil log_2(c{+}1) \rceil$ bits). Similarly to the definition of $h_{i,j}$ for KBE, we define $\sum_{i=1}^{r} h_i = h$ or $h{-}1$. In practice, it allows the computation of *exact BEs* in the same conditions than KBE. These lines could be removed, leading to a hierarchical version of the *approximated* BE proposed in [12].

Finally comes the most inner loop, at lines 14–15, with cr^2 iterations. Line 14 reduces $\widehat{X}_{A_i} \bmod x_{b_{k,l}}$. Then, line 15 in HBE is strictly equivalent to line 11 in KBE. The factor $\overline{a_{i,j}}$ at line 11 of KBE is already integrated in $\widehat{X}_{A_i}$. Then the multiplication by $\overline{A_i}$ completes the CRT computation in line 15 of HBE. However, line 11 in KBE is executed $c^2r^2 = n^2$ times instead of only cr^2 for HBE.

To summarize, the c residues in a row are pooled to get one *super-residue* (of size $cw + \lceil \log_2(c) \rceil$) using a first CRT computation. Then a CRT computation converts the super-residues $\widehat{X}_{A_i}$ into the second base using an approximation similar to the one from KBE. The number of MM(w, w) (i.e., MMs inside channels) is *divided* by c. But HBE deals with multiplications of $w \times (c-1)w$ bits (line 7) and reductions from $cw + \lceil \log_2(c) \rceil$ to w bits (see details in Section IV-B).

In the papers [11], [16] and [24], the 3 first lines in KBE are hidden inside the computations of the RNS Montgomery reduction to get faster implementations. This could also be applied with our algorithm (exactly the same way).

For our asymmetric cryptography applications, $c \in \{2, 3, 4\}$ seems to be a good choice (see also section IV-B). Below, we

will see that $c = 2$ leads to very interesting simplifications.

B. Validation of HBE Algorithm

We first show that the CRT formula computed in HBE is equivalent to the one computed in KBE. Second, we explain why the same base can be used for both an exact KBE and an exact HBE with $c \in \{2, 3, 4\}$. HBE does not introduce additional constraints. The only change can be introduced by the trunc function. It may require 1 or 2 additional bits to ensure the result after the approximation.

1) Validation of the CRT sum: In KBE and HBE, one can see that the same CRT sum is actually computed:

$$\begin{aligned} S = \sum_{i=1}^{r}\sum_{j=1}^{c} \widehat{x}_{a_{i,j}} \times \frac{A}{a_{i,j}} &= \sum_{i=1}^{r}\sum_{j=1}^{c} \widehat{x}_{a_{i,j}} \times \overline{a_{i,j}} \times \overline{A_i} \\ &= \sum_{i=1}^{r} \left(\sum_{j=1}^{c} \widehat{x}_{a_{i,j}} \times \overline{a_{i,j}} \right) \times \overline{A_i} \\ &= \sum_{i=1}^{r} \widehat{X}_{A_i} \times \overline{A_i} \equiv X \bmod A \end{aligned}$$

To get an exact BE with KBE, h is computed from all $\widehat{x}_{a_{i,j}}$. Two strategies can be used for HBE.

The first strategy relies on the fact that the lines 1–3 are the same as in KBE. Additional memories can store $\widehat{x}_{a_{i,j}}$ in addition to $\widehat{X}_{A_i}$. In KBE on the `cox-rower` architecture, $\widehat{x}_{a_{i,j}}$ is broadcasted from one channel to all the other ones through the large multiplexer. It is also sent to the `cox`, which only accumulates its t MSBs (see Fig. 1). Nonetheless, this is not directly possible with HBE where the CRT sum has only r terms instead of rc. Then this strategy seems less promising than the computation proposed in HBE to compute h.

The second strategy uses a `cox` unit directly operating on the super-residues. The values $A_i = \prod_{j=1}^{j=c} a_{i,j}$ can be seen as *super-moduli* and one extracts its MSBs to get h_i. From now, we assume $a_{i,j}$ selected such that they satisfy the constraints from [9] for ensuring an exact BE.

2) Majoration of the approximation error: Below we show that for $c = 2$, a compliant base $\mathcal{A}$ with the KBE constraints from [9] is directly compliant with HBE using a trunc function which keeps at most $t' = t+1$ MSBs. The same property can be shown for $c = 3$ and $c = 4$ with $t' = t + 2$.

Let us assume 2 RNS bases $\mathcal{A}$ and $\mathcal{B}$ with $n = rc$ moduli of w bits where each modulo is a pseudo-Mersenne number selected to satisfy the theorems from [9]. As presented in the proof from [9], with $d_{a_{i,j}} = \frac{\widehat{x}_{a_{i,j}} - \mathrm{trunc}(\widehat{x}_{a_{i,j}})}{a_{i,j}}$, $e_{a_{i,j}} = \frac{2^w - a_{i,j}}{2^w}$, $d_m = \max(d_{a_{i,j}})$ and $e_m = \max(e_{a_{i,j}})$, one has:

$$\sum_{i=1}^{r}\sum_{j=1}^{c} \frac{\widehat{x}_{a_{i,j}}}{a_{i,j}} - r\,c\,(d_m + e_m) < \sum_{i=1}^{r}\sum_{j=1}^{c} \frac{\mathrm{trunc}(\widehat{x}_{a_{i,j}})}{2^w} \quad (3)$$

$$\sum_{i=1}^{r}\sum_{j=1}^{c} \frac{\mathrm{trunc}(\widehat{x}_{a_{i,j}})}{2^w} < \sum_{i=1}^{r}\sum_{j=1}^{c} \frac{\widehat{x}_{a_{i,j}}}{a_{i,j}}. \quad (4)$$

The value $r\,c\,(d_m + e_m)$ is an upper bound of the error committed by using $\mathrm{trunc}(\widehat{x}_{a_{i,j}})$ instead of $\widehat{x}_{a_{i,j}}$ and 2^w instead of $a_{i,j}$. From [9], if $\mathcal{A}$ is chosen such that $r\,c\,(d_m + e_m) < \alpha < 1$ and $X < (1-\alpha)M$, then the BE is exact.

In practice, choosing $r\,c\,(d_m + e_m) < 0.5$ is sufficient to accurately perform an exact BE for an integer $X < A/2$. For $X < A$, KBE computes X or $X + A$ in base $\mathcal{B}$, which is sufficient in various applications such as the second extension in RNS Montgomery reduction (see theorems 1 and 2 in [9]).

Now let us assume RNS bases $\mathcal{A}$ and $\mathcal{B}$ are used in HBE with $c = 2$. We will show that using the same definitions for d_m and e_m, one gets

$$\sum_{i=1}^{r}\sum_{j=1}^{c} \frac{\widehat{x}_{a_{i,j}}}{a_{i,j}} - n(d_m + e_m) < \sum_{i=1}^{r} \frac{\mathrm{trunc}(\widehat{X}_{A_i})}{2^{cw}} \quad (5)$$

and

$$\sum_{i=1}^{r} \frac{\mathrm{trunc}(\widehat{X}_{A_i})}{2^{cw}} < \sum_{i=1}^{r}\sum_{j=1}^{c} \frac{\widehat{x}_{a_{i,j}}}{a_{i,j}}, \quad (6)$$

thus the conclusions from theorems in [9] still hold for HBE.

Let us define $d_{A_i} = \frac{\widehat{X}_{A_i} - \mathrm{trunc}(\widehat{X}_{A_i})}{A_i}$ and $e_{A_i} = \frac{2^{cw} - A_i}{2^{cw}}$, similar to $d_{a_{i,j}}$ and $e_{a_{i,j}}$ for the super-residues.

Let us prove that Eq. 5 holds. If trunc keeps the t MSBs in KBE, then $x - \mathrm{trunc}(x) < 2^{w-t}$ (i.e. at most all the $w-t$ LSBs dropped are 1s). The truncated bits cannot be predicted then $d_m = \max(2^{w-t}/a_{i,j})$ is assumed for the choice of the RNS base in KBE, this assumption is still true for HBE. Assuming $c = 2$ in HBE, our trunc keeps the $t + 1$ MSBs from the $2w + 1$ bits of $\widehat{X}_{A_i}$ and we approximate A_i by 2^{2w}. Then

$$d_{A_i} < \frac{2^{2w+1-(t+1)}}{A_i} \leq \frac{2^w}{a_{i,2}} \times \frac{2^{w-t}}{a_{i,1}} < 2\,d_m$$

because $2^w/a_{i,2} < 2$ (moduli $a_{i,j}$ are w-bit integers) and by definition $2^{w-t}/a_{i,1} < d_m$. Summing all rows leads to

$$\sum_{i=1}^{r} d_{A_i} < n\,d_m. \quad (7)$$

For $c = 2$, $A_i = a_{i,1} \times a_{i,2} = (2^w - u_1)\,(2^w - u_2)$ with u_1 and u_2 small since $a_{i,j}$s are pseudo-Mersenne integers. Thus

$$\begin{aligned} e_{A_i} &= \frac{2^{2w} - a_{i,1}\,a_{i,2}}{2^{2w}} = \frac{2^{2w} - (2^w - u_1)\,(2^w - u_2)}{2^{2w}} \\ &= \frac{(u_1 + u_2)2^w - u_1\,u_2}{2^{2w}} = \frac{(u_1 + u_2)}{2^w} - \frac{u_1\,u_2}{2^{2w}} \\ &= \frac{2^w - a_{i,1}}{2^w} + \frac{2^w - a_{i,2}}{2^w} - \frac{(2^w - a_{i,1})\,(2^w - a_{i,2})}{2^{2w}} \\ &= e_{a_{i,1}} + e_{a_{i,2}} - e_{a_{i,1}} \times e_{a_{i,2}} \\ &< e_{a_{i,1}} + e_{a_{i,2}}. \end{aligned}$$

It follows

$$\sum_{i=1}^{r} e_{A_i} < \sum_{i=1}^{r}\sum_{j=1}^{c} e_{a_{i,j}} < n\,e_m \quad (8)$$

which combined with Eq. 7 gives

$$\sum_{i=1}^{r}(d_{A_i}+e_{A_i}) < n\,(d_m+e_m) \quad . \tag{9}$$

Using the proof in [9], one can easily find:

$$\sum_{i=1}^{r}\sum_{j=1}^{c}\frac{\widehat{x}_{a_{i,j}}}{a_{i,j}} - \sum_{i=1}^{r}(d_{A_i}+e_{A_i}) < \sum_{i=1}^{r}\sum_{j=1}^{c}\frac{\mathrm{trunc}(\widehat{X}_{A_i})}{2^{cw}}$$

which leads to Eq. 5 by applying Eq. 9.

To complete the proof, Eq. 6 directly comes from the definition of the approximations from [9], using 2^w instead of $a_{i,j}$ to maximize the denominators, and trunc minimizes the numerators thus the approximation is always less than the real value.

C. Theoretical Cost Evaluation

To compare with KBE, we evaluate the number of CMMs and CMRs in HBE for a generic c and then we focus on $c=2$.

Lines 1–3 in HBE are the same in KBE, and cost rc $\mathtt{CMM}(w,w)$. Lines 4–7 cost $rc\,\mathtt{CMUL}(w,(c-1)w)$ and $rc\,\mathtt{CADD}(cw,cw)$. This part is hard to evaluate for a generic c without a full implementation because it introduces additions on cw-bit integers (larger than the channels). HBE operations at lines 9–11 are negligible. There are just very small additions on a few bits computed in the cox unit in parallel with the rowers (see Fig. 1). Finally HBE lines 14–15 are performed r^2c times each. HBE line 15 costs the same as KBE line 11, *i. e.,* $1\,\mathtt{CMM}(w,w)+2\,\mathtt{CADD}(w,w)$ per iteration. Line 14 is more difficult to evaluate, because it depends on c (*i.e.,* $\widehat{X}_{A_i}$ has $cw+\lceil log_2(c)\rceil$ bits) and on the form of the moduli (*e.g.,* generic *vs.* sparse numbers).

To sum up, the theoretical cost of HBE Algo. 2 is :

$$r^2c\,(\mathtt{CMM}(w,w)+2\mathtt{CADD}(w,w)+\mathtt{CMR}(cw+\lceil\log_2(c)\rceil,w))+ rc\,(\mathtt{CMM}(w,w)+\mathtt{CMUL}(w,(c-1)w)+\mathtt{CADD}(cw,cw))$$

For $c=2$, then $r=n/2$, using the assumption $\mathtt{CMUL}=\mathtt{CMM}$ (we overestimate CMUL), the cost reduces to:

$$\frac{n^2}{2}\,(\mathtt{CMM}(w,w)+2\mathtt{CADD}(w,w)+\mathtt{CMR}(2\,w+1,w))+ n\,(2\mathtt{CMM}(w,w)+\mathtt{CADD}(2w,2w))$$

Using the simplification $\mathtt{CMUL}=\mathtt{CMM}$ (in reality $\mathtt{CMUL}<\mathtt{CMM}$), the overestimation of the HBE cost is small since it only impacts the linear term in n.

The choice $c=2$ is very interesting since the circuit only deals with values of w or $2w+1$ bits, as in KBE. No new arithmetic operator is required to use HBE with $c=2$.

As in most of the state-of-the-art works, we focus on the number of modular multiplications because additions are usually hidden in the pipeline computing the multiplications, especially in DSP slices of modern FPGAs (they are actually accumulations of products).

This leads to a cost for HBE of

$$\frac{n^2}{2}\mathtt{CMM}(w,w)+\frac{n^2}{2}\mathtt{CMR}(2\,w+1,w)+2\,n\,\mathtt{CMM}(w,w)$$

TABLE I
FPGA IMPLEMENTATION RESULTS FOR $\mathtt{CMR}(2\,w+1,w)$ AND $\mathtt{CMM}(w,w)$ ELEMENTARY OPERATIONS IN A XILINX XC7Z020.

operations	$\mathtt{CMR}(2\,w+1,w)$				$\mathtt{CMM}(w,w)$			
w (bits)	17	20	24	28	17	20	24	28
nb. slices	1	1	24	35	1	24	1	39
nb. DSP	2	2	1	1	3	3	4	5
nb. cycles	1	1	2	2	2	2	2	3
time (ns)	2.4	2.6	9.0	9.6	7.8	10.6	10.6	17.1

against a cost for KBE of

$$n^2\,\mathtt{CMM}(w,w)+n\,\mathtt{CMM}(w,w).$$

If $\mathtt{CMR}(2\,w+1,w)\ll\mathtt{CMM}(w,w)$, significant improvements may be achieved using HBE instead of KBE. To evaluate these costs in real hardware, we implemented on a Xilinx XCV7020 FPGA (using Vivado 2017.4) both operations for numerous randomly chosen pseudo-Mersenne moduli (2^w-u_i) and various w. The results are reported in Tab. I. $\mathtt{CMR}(2\,w+1,w)$ requires about half of the time and half of area than $\mathtt{CMM}(w,w)$. Thus, we assume $\mathtt{CMM}(w,w)/4\leqslant\mathtt{CMR}(2w+1,w)\leqslant\mathtt{CMM}(w,w)/2$ for pseudo-Mersenne moduli.

Finally, Fig. 2 presents the theoretical improvements of HBE using $c=2$ *vs.* KBE for various values of the ratio CMM/CMR and numbers of moduli between 6 and 32 (typical values in our asymmetric cryptography applications). HBE leads to theoretical improvements up to 35% compared to KBE.

D. Parallelism of HBE in a cox-rower Architecture

We want to implement our cryptosystems in RNS using a cox-rower architecture adapted from [9] (see Fig. 1). We also use n rowers (*i.e.*, one physical channel per modulo).

HBE is mainly made of 3 successive loops. The first and the third loops operate on all channels of $\mathcal{A}$ and $\mathcal{B}$ respectively (each of $n=rc$ channels). Thus these loops can be performed on a cox-rower architecture with n parallel rowers as in KBE. However, if $c>2$ the architecture must be modified to reduce values larger than $2\,w$ bits.

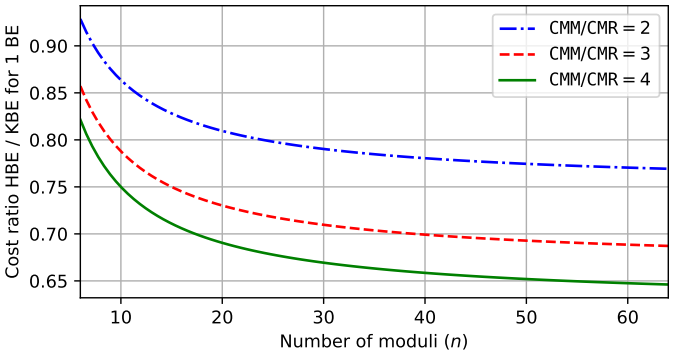

Fig. 2. Theoretical costs of a BE for various base sizes (n), the two compared BE algorithms and $c=2$. Each curve corresponds to cost(HBE)/cost(KBE) for one cost ratio of elementary operations (CMM/CMR).

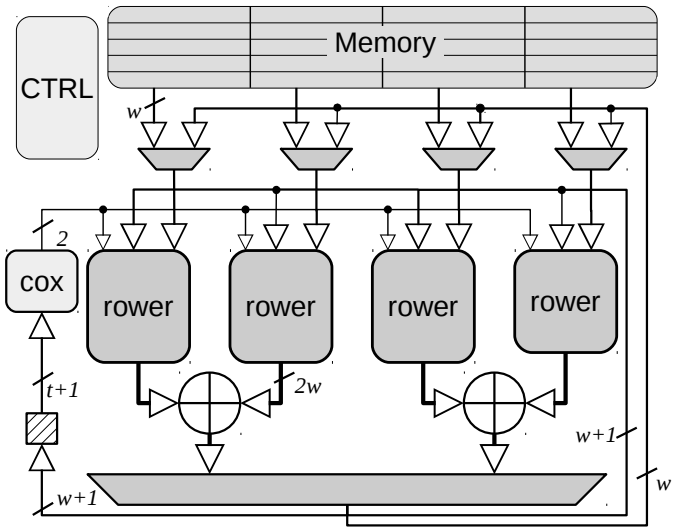

Fig. 3. Our HBE `cox-rower` architecture with $c = 2$ and $n = 4$.

In the second loop, $r = n/c$ independent sums of products are computed: the greater is c, the greater is the reduction of the parallelism. More, $\widehat{X}_{A_i}$ is $cw + \lceil \log_2(c) \rceil$ bits wide and requires larger operators than in the `cox-rower` from [9] architectures for $c > 2$. This could be balanced by the reduction of the cost of the last loop with only n^2/c iterations.

Using $c = 2$, the `cox-rower` can be modified as presented in Fig. 3.

Computing the n multiplications in parallel units and summing their $2w$-bit results allows to compute $\widehat{X}_{A_i}$ with an architecture as parallel as the original `cox-rower` or nearly. The $w + 1$ MSBs are sent to the `cox` (through a truncation) and to all the `rowers`, the w LSBs are also broadcasted to the `rowers`.

V. Cryptographic Applications

RSA exponentiation and ECC scalar multiplication require numerous MRs with a large modulus (200+ bits for ECC and 2000+ bits for RSA). The standard integer MR algorithm used with a generic modulus (*i.e.*, no specific form) is the Montgomery reduction proposed in [25]. It replaces a costly division by 2 multiplications by a constant, one reduction modulo 2^l and one division by 2^l where l is the modulus width in a radix-2 representation.

An RNS version of this algorithm has been proposed in [26], see Algo. 3. Instead of 2^l, it uses reduction and division by A at line 4. Because A cannot be inverted in base $\mathcal{A}$, BEs at lines 2/5 in/from base $\mathcal{B}$ are required to divide by A. The second BE must be *exact*, but the first one can be *approximated* (the result, multiplied by P, does not impact the result modulo P). KBE and HBE algorithms can be used for both BEs.

The cost of Algo. 3 is dominated by the 2 BEs. One RNS reduction modulo P from [9] costs $2n^2+5n$ $\texttt{CMM}(w, w)$ where the 2 BEs cost $2n^2 + 2n$. By reordering internal operations, [11] reduces the cost of one RNS reduction modulo P to $2n^2+2n$ `CMMs` (actually 2 successive BEs).

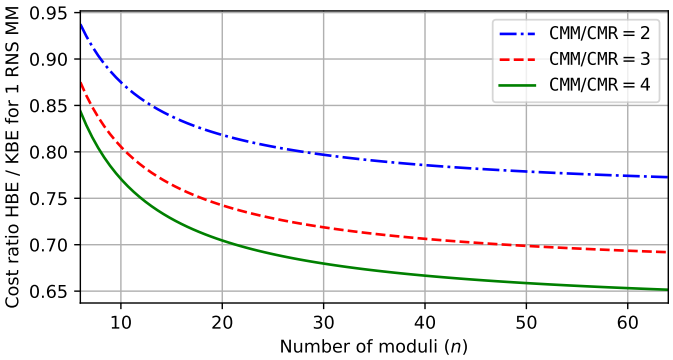

Fig. 4. Theoretical costs of one RNS Montgomery MM from [11] for various base sizes (n) and the two compared BE algorithms for ECC or RSA applications. Each curve corresponds to `cost`(MM w. HBE)/`cost`(MM w. KBE) for one cost ratio of elementary operations (`CMM/CMR`).

One full RNS MM based on KBE costs $2n^2 + 4n$ `CMMs` (one MR and $2n$ `CMMs`, one multiplication per channel per RNS base). The same RNS MM algorithm using HBE instead of KBE only costs $\frac{3n^2}{2} + 6n$ `CMMs` when `CMM/CMR` $= 2$ or $\frac{5n^2}{4} + 6n$ when `CMM/CMR` $= 4$.

Recently, [24] improves RNS MMs for ECC applications. By selecting well-suited bases for prime field characteristic P from standards (*e.g.*, NIST primes), the KBE is reduced from $2n^2 + 4n$ to $2n^2 + 3n$ `CMMs`. This method requires to generate one base per field characteristic. For our applications with multiple fields this method is less interesting than [11].

Figure 4 depicts the theoretical gain when using HBE instead of KBE for one RNS MM using Algo. 3 and optimizations from [11]. The gain is given for various cost ratios `CMR/CMM`. For instance, 256-bit ECC and 1024-bit RSA-CRT both with 17-bit moduli (to fit into one Xilinx DSP multiplier see Sec. VI), respectively requires 16 and 32 moduli for the RNS bases. In these typical cases, HBE leads to 17% up to 32% theoretical gain compared to KBE for MMs.

In RSA, modular exponentiation only performs MMs, then the HBE gain for one RNS MM can be directly transposed to one RNS modular exponentiation. In ECC scalar multiplication, this gain can be slightly reduced because one can sometimes perform 1 MR for 2 multiplications, see [10].

In addition to the comparison with [11], we also evaluate

Algorithm 3: RNS Montgomery reduction modulo P [26].

Input: X_A, X_B
Precomp.: $P_A, P_B, \quad \left(-P^{-1}\right)_A, \quad \left(A^{-1}\right)_B$
Output: S_A and S_B with $S = \left(XA^{-1}\right) \bmod P + \delta P$ and $\delta \in \{0, 1, 2\}$

1 $Q_A \leftarrow X_A \times \left(-P^{-1}\right)_A$
2 $Q_B \leftarrow BE\left(Q_A, \mathcal{A}, \mathcal{B}\right)$
3 $R_B \leftarrow X_B + Q_B \times P_B$
4 $S_B \leftarrow R_B \times \left(A^{-1}\right)_B$
5 $S_A \leftarrow BE\left(S_B, \mathcal{B}, \mathcal{A}\right)$
6 **return** (S_A, S_B)

TABLE II
COSTS FOR THREE RNS MM ALGORITHMS (IN CMMS) WHEN USING KBE AND HBE (WITH TWO RATIOS FOR $r = \texttt{CMM}/\texttt{CMR}$).

BE	MM [11]	HPR $d = 2$ [17]	HPR $d = 4$ [27]
KBE	$2n^2 + 4n$	$n^2 + 8n$	$\frac{n^2}{2} + 12n$
HBE ($r = 2$)	$\frac{3n^2}{2} + 6n$	$\frac{3n^2}{4} + 10n$	$\frac{3n^2}{8} + 14n$
HBE ($r = 4$)	$\frac{5n^2}{4} + 6n$	$\frac{5n^2}{8} + 10n$	$\frac{5n^2}{16} + 14n$

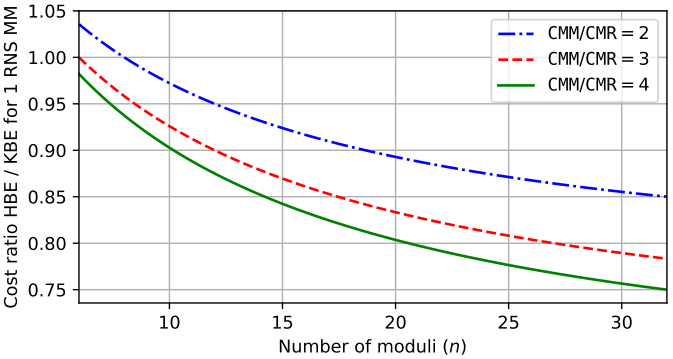

Fig. 5. Theoretical costs of one HPR MM from [17], [27] for $d = 2$, various base sizes (n) and the two compared BE algorithms. Each curve corresponds to `cost`(MM w. HBE)/`cost`(MM w. KBE) for one cost ratio of elementary operations (CMM/CMR).

the interest in using HBE instead of KBE for other RNS MM algorithms: [17] and [27].

A specific RNS MM approach for ECC has been proposed in [17] and generalized in [27]. These methods propose to use specific P values with *pseudo-Mersenne like properties* in RNS. They mainly mix RNS with a polynomial representation of a small degree d. The MM algorithms from [17] and [27] are respectively referred as *HPR with* $d = 2$ and *HPR with* $d = 4$ below.

We compare the cost of all these RNS MM algorithms when using internally KBE and HBE for typical sets of parameters. The results are reported in Tab. II. Fig. 4 illustrates the gain for the RNS MM from [11]. They are illustrated in Fig. 5 for HPR with $d = 2$ from [17]. With this RNS MM algorithm and $n = 16$, HBE leads from 8 to 17% speedup compared to KBE. For $n = 32$, the improvement reaches 15 to 25%. With the HPR width $d = 4$ in algorithm from [27], the gain is illustrated in Fig. 6. The number of moduli is assumed to be at least 16 (below HPR with $d = 2$ is faster). For $n = 32$, HBE is 8 to 15% faster than KBE.

Our evaluations assume a pessimistic ratio $\texttt{CMM}/\texttt{CMR} = 4$. We currently use simple pseudo-Mersenne moduli (see FPGA implementation results in Tab. I). We plan to improve this ratio by using well chosen moduli.

VI. FPGA IMPLEMENTATION RESULTS

We implemented our HBE, Algo. 2, as well as the KBE, Algo. 1, from state of the art [9] using the same environment and effort. We did that to make a fair comparison and because of a lack of experimental results for standalone BE implementations in the literature. We used the Vivado 2017.4

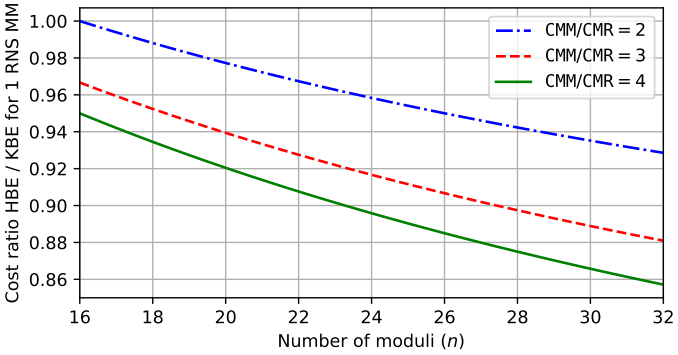

Fig. 6. Similar to Fig. 5 but for $d = 4$.

HLS tools and a XC7Z020 FPGA from Xilinx. HLS allows us to quickly explore various widths for RNS channels and implementation constraints (*e.g.*, loop unrolling, pipelining).

We used RNS bases for ECC over $\mathbb{F}_P$ with 256 and 384 bits elements. We explored several widths for RNS channels: $w \in \{17, 20, 24, 28\}$ bits. DSP slices in Xilinx FPGAs embed hardwired integer multipliers for 18×18 or 18×25 bits operands in 2's complement. For unsigned integers, 17 bits operands should be used to fully exploit DSP slices, see [28], [29]. Currently, we do not support asymmetric widths for the DSP operands (*e.g.*, 17×24). The number n of RNS channels is the smallest multiple of $c = 2$ such that $nw > \log_2(P)$.

Table III reports all the corresponding implementation results. "Time" rows refer the BE computation time (*i.e.*, the product of the period by the number of cycles). In most of cases, HBE leads to faster *and* smaller solutions than KBE. HBE always reduces the number of required DSP slices (the largest elements in the FPGA). For instance, for 256-bit $\mathbb{F}_P$ elements and 20-bit channels, HBE is 10% faster and 19% smaller in DSPs. The area reduction can reach 31% for $w = 17$ (the gain in computation time is 2% in that case).

HLS tools allow us to explore various area *vs.* time trade-offs in a much faster way than using VHDL descriptions. For instance, on 384-bit $\mathbb{F}_P$ elements, we are able to speedup the computation by 40% for a 28% increase of the number of DSP slices by increasing the channels width.

VII. CONCLUSION AND FUTURE PROSPECTS

We proposed a new algorithm for RNS base extension called *hierarchical base extension* (HBE). In HBE, the moduli are used in a hierarchical way with partial applications of small intermediate CRTs. The moduli, usually handled as a single vector, are managed as a matrix in HBE. The moduli in each row are used to construct *super-moduli* through small parallel CRTs. At the highest level, all row-wise contributions are used to complete the BE using a last CRT.

HBE reduces the theoretical computation cost up to 35 % compared to the state of the art KBE algorithm from [9]. HBE shares the exact same constraints than KBE for selecting RNS bases. We also proposed a modified `cox-rower` architecture to efficiently implement HBE. For specific shapes of the

TABLE III
HLS IMPLEMENTATION RESULTS ON A XC7Z020 FPGA FOR OUR HBE AND THE KBE (FROM [9]) ALGORITHMS FOR 2 WIDTHS OF PRIME FIELD ELEMENTS AND 4 RNS CHANNELS WIDTHS w.

$\mathbb{F}_P$ width (bits)	BE algo.	KBE	HBE	KBE	HBE	KBE	HBE	KBE	HBE
	w (bits)	17		20		24		28	
256	nb. slices	445	758	1073	784	785	769	753	843
	nb. DSP	51	35	45	39	52	42	76	60
	nb. BRAM	1	1	1	1	1	1	1	1
	period (ns)	9.8	10.3	9.6	8.9	9.6	9.5	9.7	9.6
	nb. cycles	98	91	88	83	89	81	77	71
	time (ns)	960.4	937.3	844.8	738.7	854.4	769.5	746.9	681.6
384	nb. slices	587	644	1215	869	1251	1134	1031	1145
	nb. DSP	81	63	63	54	76	60	104	80
	nb. BRAM	1	1	1	1	1	1	1	1
	period (ns)	7.6	10.1	9.6	9.0	7.6	7.6	9.9	9.4
	nb. cycles	165	143	140	122	163	132	103	93
	time (ns)	1254.0	1444.3	1344.0	1098.0	1238.8	1003.2	1019.7	874.2

matrix of moduli (*i.e.*, two columns), our new architecture preserves the natural RNS parallelism with a slightly deeper pipeline at the `rower` level. FPGA implementation results show significant area reduction in DSP slices as well as a small speedup.

As future work, we intend to study and design full crypto-processors for ECC using HBE. We also plan to study optimizations for other types of decompositions (*e.g.*, $c \in \{3, 4\}$) and well suited moduli forms for HBE.

ACKNOWLEDGMENTS

This work has been supported by a PhD grant from DGA/Pôle de Recherche Cyber.

REFERENCES

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[2] H. Cohen and G. Frey, Eds., *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chapman & Hall/CRC, 2005.

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

[4] ECRYPT, "Algorithms, key size and protocols report," Feb. 2018, h2020-ICT-2014 - Project 645421. [Online]. Available: http://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf

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

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

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

[8] J.-C. Bajard, L. Imbert, P.-Y. Liardet, and Y. Teglia, "Leak resistant arithmetic," in *Proc. Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, vol. 3156. Springer, Aug. 2004, pp. 62–75.

[9] 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.

[10] J.-C. Bajard, S. Duquesne, and M. D. Ercegovac, "Combining leak-resistant arithmetic for elliptic curves defined over Fp and RNS representation," *Publications Mathématiques de Besançon: Algèbre et Théorie des Nombres*, pp. 67–87, 2013.

[11] 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.

[12] J.-C. Bajard and L. Imbert, "A full RNS implementation of RSA," *IEEE Transactions on Computers*, vol. 53, no. 6, pp. 769–774, Jun. 2004.

[13] 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.

[14] K. C. Posch and R. Posch, "Base extension using a convolution sum in residue number systems," *Computing*, vol. 50, no. 2, pp. 93–104, Jun. 1993.

[15] 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.

[16] 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.

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

[18] H. M. Yassine, "Hierarchical residue numbering system suitable for VLSI arithmetic architectures," in *Proc. IEEE International Symposium on Circuits and Systems*, vol. 2, May 1992, pp. 811–814.

[19] H. D. L. Hollmann, R. Rietman, S. de Hoogh, L. Tolhuizen, and P. Gorissen, "A multi-layer recursive residue number system," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, Jun. 2018, pp. 1460–1464.

[20] A. Skavantzos and M. Abdallah, "Implementation issues of the two-level residue number system with pairs of conjugate moduli," *IEEE Transactions on Signal Processing*, vol. 47, no. 3, pp. 826–838, Mar. 1999.

[21] T. Tomczak, "Hierarchical residue number systems with small moduli and simple converters," *International Journal of Applied Mathematics and Computer Science*, vol. 21, no. 1, pp. 173–192, Mar. 2011.

[22] A. Bostan, G. Lecerf, and E. Schost, "Tellegen's principle into practice," in *Proc. International Symposium on Symbolic and Algebraic Computation ISSAC*. ACM, Aug. 2003, pp. 37–44.

[23] J. van der Hoeven, "Fast chinese remaindering in practice," in *Mathematical Aspects of Computer and Information Sciences*. Springer, Nov. 2017, pp. 95–106.

[24] S. Kawamura, Y. Komano, H. Shimizu, and T. Yonemura, "RNS montgomery reduction algorithms using quadratic residuosity," *Journal of Cryptographic Engineering*, Sep 2018.

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

[26] 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.

[27] K. Bigou and A. Tisserand, "Hybrid position-residues number system," in *Proc. 23rd Symposium on Computer Arithmetic (ARITH)*. IEEE, Jul. 2016, pp. 126–133.

[28] Xilinx, "Vivado design suite user guide high-level synthesis (UG902)," Tech. Rep., Apr. 2017.

[29] ——, "7 series DSP48E1 slice user guide (UG479)," Tech. Rep., Mar. 2018.