# A general approach for improving RNS Montgomery exponentiation using pre-processing

Filippo Gandino, Fabrizio Lamberti, Paolo Montuschi, Jean-Claude Bajard

HAL Id: hal-01098806

https://hal.sorbonne-universite.fr/hal-01098806v1

Submitted on 29 Jan 2015

# A general approach for improving RNS Montgomery exponentiation using pre-processing

**Abstract**

The hardware implementation of modular exponentiation for very large integers is a well-known topic in digital arithmetic. An effective approach for obtaining parallel and carry-free implementations consists in using the Montgomery exponentiation algorithm and executing the necessary operations in RNS. Two efficient methods for performing the RNS Montgomery exponentiation have been proposed by Kawamura et al. and by Bajard and Imbert. The above approaches mainly differ in the algorithm used for implementing the base extension.

This paper presents a modified RNS Montgomery exponentiation algorithm, where several multiplications are moved outside the main execution loop and replaced by an effective pre-processing stage producing a significant saving on the overall delay. Since the proposed modification can be applied to both the above algorithms, two versions are specifically discussed.

In order to evaluate the effect of the designed modification, the implementations by Kawamura et al. and by Bajard and Imbert. are selected as a reference, and a sharp comparison between the common part of these algorithms and the proposed approach is carried out. This analysis focuses on the arithmetic-algorithmic level, and it is aimed at quantifying the specific weight of each multiplication operation, by taking into account the impact of the number of parallel multipliers in each RNS cell and the level of pipelining. The analysis shows that the application of the proposed approach could produce a consistent reduction in the number of execution steps, thanks to the removal of the most computationally expensive multiplications. The improvement is particularly relevant for fast implementations, as the reduction factor increases with the number of pipeline stages and parallel multipliers per arithmetic cell.

Since the base extension methods used in the two reference algorithms require different arithmetic cells, an analysis at the architecture level is also carried out, with the aim of further characterizing the improvement provided by the proposed approach. Thus, a cell architecture based on work by Kawamura et al. is presented and analyzed at first. Then, a novel architecture adapted to the algorithm by Bajard and Imbert. is designed. The overall comparison shows that with the proposed approach, a $18.5\%$ speedup can be achieved for an implementation over 1024 bits, without any significant area overhead.

**Index Terms**

RNS, Montgomery Multiplication.

# I. INTRODUCTION

The computation of the Montgomery exponentiation (ME) in the Residue Number System (RNS) [1] allows limiting the delay due to carry propagation and reaching a high degree of parallelism [2]. This approach mainly requires the execution of a set of Montgomery multiplications (MMs) [3]. However, in RNS, some operations (e.g. division, comparison, modulo) are natively difficult to execute. Hence, several approaches have been proposed in order to fully exploit the potential of RNS for modular exponentiation, by minimizing the impact of related drawbacks.

A fast RNS ME algorithm has been presented in [4]. This approach requires two Base Extensions (BEs) for each RNS MM. The algorithm employed for performing BE, that has been presented in [5], is particularly expensive from the computational point of view. It executes BE through iterated approximations. The approximation is correct in the majority of the cases, and when there is an offset, it is corrected.

In [6], Kawamura et al. proposed a RNS ME technique applied to RSA, with an improved BE algorithm. The new BE is characterized by a summation that provides a result modulo a small multiple of the base, which is corrected after the sum of each element. The proposed approach has been detailed in [7], where an architecture has also been presented.

In [8], Bajard et al. proposed an implementation of the MM based on both the RNS and the Mixed Radix Number System (MRS), where the MRS corresponds to a weighted system associated with the RNS. Then, in [9], the same authors proposed a Montgomery multiplication method fully implemented in RNS. This approach employs an approximated BE and the algorithm proposed in [10], where the result is approximated and corrected by using an extra modulo. Finally in [11], Bajard and Imbert detailed the application of the previous ME approach in the context of RSA.

The RNS MM has also been studied in different contexts, out of the exponentiation. In [12], an implementation for elliptic curve cryptography on FPGA is presented. This implementation employs the algorithm proposed in [6] with some pre-computations suitable to the particular architecture presented in the paper. In [2], an implementation of RNS MM on GPU is presented. Experimental results achieved in the above context show that the algorithm described in [9] provides the best performance.

This paper proposes a modified ME algorithm for RNS. The novelties of the algorithm are in the pre-processing stage, which is used to reduce the number of multiplications performed in the loop of the exponentiation algorithm. It is worth observing that, even though in the specific case the ME is considered, the basic approach used here is more general and can be easily applied in other contexts. In particular, the

analysis of the designed pre-processing method for generic modular multiplication is presented in [13].

The differences with respect to previous approaches consist in the values that are pre-computed, and in the new ME and MM algorithms. The modifications discussed in the current work are applicable to both the state-of-the-art MM algorithms [6], [9], which mainly differ for the BE correction methods. Hence, in the following, the proposed modifications will be presented in two versions specifically tailored to the characteristics of these methods.

The authors of the above works evaluate the time performance of their RNS MM algorithms according to the number of modular multiplications. By calling $k$ the number of elements for RNS base, the number of modular multiplications is reduced from $2k^2 + 9k$ and $2k^2 + 8k$, to $2k^2 + 6k$ and $2k^2 + 5k$ by Kawamura and Bajard BE, respectively.

In order to evaluate the weight of each modular multiplication required by the proposed and by the state-of-the-art algorithms, a detailed algorithmic analysis has been carried out. The analysis compares the proposed approach with the common part of the state-of-the-art algorithm [6], [11]. The evaluation is focused on the number of execution stages required by each multiplication, according to the number of pipeline stages and parallel multipliers present in each arithmetic cell. When an architecture with $k$ cells is considered, the number of execution steps required by the RNS MM algorithm used in [6], [11] (not considering the BE correction) is $2\lceil k/M \rceil - 2 + \lfloor 1/(p + M - 1) \rfloor + 8p$, where $p$ is the number of pipeline stages of the arithmetic cells and $M \leq k$ is the number of parallel multipliers per cell. The proposed approach allows achieving a reduction equal to $4p - \lfloor 1/(p + M - 1) \rfloor$ execution steps; hence, the improvement is proportional to the number of pipeline stages and, for this, it is more suitable to high speed architectures.

An analysis at the architectural level is also carried out, in order to fully exploit the effects of the two versions of the proposed approach. An architecture based on the work presented in [7] is designed and analyzed. Then, a new architecture suitable to the algorithm proposed in [11] is proposed, and an overall comparison is presented, showing that with the proposed approach a $18.5\%$ speedup can be achieved for an implementation over 1024 bits, without any significant area overhead.

The remaining of the paper is organized as follows: in Section II, the proposed RNS ME algorithm is presented, while in Section III, it is analyzed and compared to related works. In Section IV, the implementation of the arithmetic cells is analyzed, and the effects of the proposed approach are discussed. Finally in Section V, some conclusions are drawn.

TABLE I

LIST OF SYMBOLS

| Context | Symbol | Meaning | Symbol | Meaning |
|---|---|---|---|---|
| | $\mathcal{A}$ | RNS base | $\mathcal{B}$ | RNS base |
| | $k$ | Number of base elements | $r$ | Number of bits of each base element |
| RNS bases | $a_j$ | $j^{th}$ element of the base $\mathcal{A}; \forall j; 1 \le j \le k$ | $b_i$ | $i^{th}$ element of the base $\mathcal{B}; \forall i; 1 \le i \le k$ |
| | $A$ | $\prod_{j=1}^{k} a_j$ | $B$ | $\prod_{i=1}^{k} b_i$ |
| | $A_j$ | $\frac{A}{a_j}; \forall j; 1 \le j \le k$ | $B_i$ | $\frac{B}{b_i}; \forall i; 1 \le i \le k$ |
| | $A_j^{-1}$ | Multiplicative inverse of $A_j$ on $a_j$ | $B_i^{-1}$ | Multiplicative inverse of $B_i$ on $b_i$ |
| | $B_A^{-1}$ | Multiplicative inverse of $B$ on $A$ | $B_N^{-1}$ | Multiplicative inverse of $B$ on $N$ |
| | $c_i$ | $2^r - a_i$ | $h$ | Maximum number of bits of $c_i$ |
| MM | $R$ | Montgomery number | $N$ | Modulo of the operation |
| BE Values | $\lambda$ | Approximation $\frac{\tilde{x}-x}{B}$ | | |
| Architecture | $p$ | Stages of pipeline | $M$ | Parallel multipliers |
| Kawamura et al. BE [6] | $c$ | Approximated value of $\frac{q}{A}$ | $f$ | Floor of $c; (\lfloor c \rfloor)$ |
| | $\alpha$ | Initial value of $c; \{0; 0.5\}$ | $\varrho$ | Number of accumulated bits of $q_i$ |
| Bajard et al. BE [11] | $a_r$ | Redundant base element | $A_r^{-1}$ | Multiplicative inverse of $A$ on $a_r$ |
| Accents | $\widetilde{tilde}$ | Approximated values | $\widehat{hat}$ | Values multiplied by $A_j^{-1}$ in $\mathcal{A}$ |
| Operations and Relations | $<<$ | Left shift | $\equiv$ | Equivalent |

TABLE II

ABBREVIATIONS

| Abbreviation | Meaning | Abbreviation | Meaning |
|---|---|---|---|
| RNS | Residue Number System | CRT | Chinese Remainder Theorem |
| ME | Montgomery Exponentiation | MM | Montgomery Multiplication |
| BE | Base Extension | KBE1 | 1st BE of the proposed MM based on Kawamura et al. |
| KBE2 | 2nd BE of the proposed MM based on Kawamura et al. | BBE1 | 1st BE of the proposed MM based on Bajard and Imbert |
| BBE2 | 2nd BE of the proposed MM based on Bajard and Imbert | MAU | Multiplication and adder unit |
| F(S)MRU | First (Second) modular reduction unit | CSA | Carry-save adder |
| $\gcd(x,y)$ | Greatest common divisor | $\text{lcm}(x,y)$ | Least common multiple |

## II. PROPOSED ALGORITHM

This paper presents a new RNS ME. This section illustrates the proposed technique, by analyzing the modifications introduced with respect to previous approaches. Tables I and II provide a description of the main symbols and abbreviations used in the discussion.

### A. RNS

In RNS, a number is represented according to a base $\mathcal{A} = (a_1, a_2, ..., a_k)$, which is made up of $k$ relative prime numbers, where $k$ is called the base size. Therefore, any number $x$, where $0 \le x < A = \prod_{i=1}^{k} a_i$ is uniquely represented by a sequence of positive integers $(x_1, x_2, ..., x_k)$, where $x_i = x \bmod a_i$, $\forall i : 1 \le i \le k$. It is worth observing that, because of the independence of all the elements, in RNS the multiplication, addition, and subtraction operations can be carried out independently and in parallel for each element.

Using the Chinese Remainder Theorem (CRT) [14], it is possible to convert a value $x$ from an RNS base to a radix system, thus achieving a high parallelism. The reconstruction expression is:

Algorithm 1: State-of-the-art RNS Montgomery exponentiation [6], [11]

**Input:** $x$ in $\mathcal{A} \cup \mathcal{B}$ and $e = (e_{g-1}...e_1 e_0)_b$
**Output:** $z = (x^e) \bmod N$ in $\mathcal{A} \cup \mathcal{B}$
**Precomputation:** $B^2 \bmod N$,
   $B \bmod N$ in $\mathcal{A} \cup \mathcal{B}$

1: $\bar{x} \leftarrow$ MM $(x, B^2 \bmod N)$

2: $z_{bi} = (B \bmod N)_{b_i}$ for $i = 1...k$
3: $z_{aj} = (B \bmod N)_{a_j}$ for $j = 1...k$
4: **for** $i$ from $g-1$ down to 0 **do**
5:    $\hat{z} \leftarrow$ MM $(\hat{z}, \hat{z})$
6:    **if** $e_i = 1$ **then**
7:       $z \leftarrow$ MM$(z, \bar{x})$
8:    **end if**
9: **end for**
10: $z \leftarrow$ MM$(z, 1)$

Algorithm 2: Proposed RNS Montgomery exponentiation

**Input:** $x$ in $\mathcal{A} \cup \mathcal{B}$ and $e = (e_{g-1}...e_1 e_0)_b$
**Output:** $z = (x^e) \bmod N$ in $\mathcal{A} \cup \mathcal{B}$
**Precomputation:** $B^2 \bmod N$, $B \bmod N$ in $\mathcal{A} \cup \mathcal{B}$,
   $A_j \bmod a_j$ and $A_j^{-1} \bmod a_j$ for $j = 1...k$

1: $\bar{x} \leftarrow$ MM $(x, B^2 \bmod N)$
2: $\hat{\bar{x}}_{aj} = \bar{x}_j A_j^{-1} \bmod a_j$ for $j = 1...k$
3: $z_{bi} = (B \bmod N)_{b_i}$ for $i = 1...k$
4: $\hat{z}_{aj} = (B \bmod N)_{a_j} A_j^{-1} \bmod a_j$ for $j = 1...k$
5: **for** $i$ from $g-1$ down to 0 **do**
6:    $\hat{z} \leftarrow$ MM $(\hat{z}, \hat{z})$
7:    **if** $e_i = 1$ **then**
8:       $\hat{z} \leftarrow$ MM$(\hat{z}, \hat{\bar{x}})$
9:    **end if**
10: **end for**
11: $\hat{z} \leftarrow$ MM$(\hat{z}, \hat{1})$
12: $z_{aj} = \hat{z}_{a_j} A_j \bmod a_j$ for $j = 1...k$

Fig. 1.   Comparison between the proposed RNS Montgomery exponentiation and the state-of-the-art algorithm

$$x = \left(\sum_{i=1}^{k}((x_i A_i^{-1}) \bmod a_i) A_i\right) \bmod A \tag{1}$$

where $A_i = \frac{A}{a_i}$ and $A_i^{-1}$ is the multiplicative inverse of $A_i$ on $a_i$. The result of $(\sum_{i=1}^{k}((x_i A_i^{-1}) \bmod a_i) A_i$ is equal to $x + \lambda A$, where $\lambda < k$. In order to complete the BE, the modular reductions of $x$ by the elements of the new base must be performed.

### B. Montgomery Exponentiation (ME)

The ME is based on the MM, where MM$(x \times y \bmod N)$ gives $w = xyR^{-1} \bmod N$. ME computes $x^e \bmod N$ at the average cost of $3/2 \log_2 e + 2$ MMs. Let us denote $\bar{x}$ and $\bar{y}$, such that $\bar{x} = xR \bmod N$ and $\bar{y} = yR \bmod N$; then $z = \bar{x}\bar{y}R^{-1} \bmod N = xyR \bmod N$. Therefore, the exponentiation can be executed by iterating MM on $\bar{x}$.

The comparison between the state-of-the-art RNS ME algorithm (the main part of the ME algorithm is common to both [6] and [11]) and the proposed algorithm is shown in Figure 1. Step 1 of both the algorithms calculates $\bar{x}$, as previously described. Steps 2 and 3 in Algorithm 1 [6], [11] initialize the exponentiation process, which is executed in the loop from step 4 to step 9. The proposed RNS ME algorithm (Algorithm 2) executes 3 multiplication steps more than Algorithm 1. Before executing the loop, all the values on base $\mathcal{A}$ are multiplied by $A_j^{-1}$ (steps 2 and 4). The multiplication by $A_j^{-1}$ is used in the state-of-the-art BE algorithm, in order to extend the values from $\mathcal{A}$ to $\mathcal{B}$. In the proposed algorithm both the input values in $\mathcal{A}$ of the RNS MM are pre-multiplied by $A_j^{-1}$ so a multiplication by $A_j$ in

Algorithm 3: State-of-the-art RNS Montgomery multiplication [6], [9]

**Input:** $x$, $y$, and $N$ in $\mathcal{A} \cup \mathcal{B} \cup a_r$, such that[I] $A = \prod_{j=1}^{k} a_j$, $B = \prod_{i=1}^{k} b_i$, and $\gcd(A, B) = 1$, $\gcd(N, B) = 1$, $0 \le xy < NB$, $B > 4N$, and $A > 2N$

**Output:** $w \equiv xy B_N^{-1} \pmod{N}$, $w < 2N$ in $\mathcal{A} \cup \mathcal{B} \cup a_r$

**Precomputation:** $B_A^{-1}$, $N$ in $\mathcal{A} \cup a_r$; $N^{-1}$ in $\mathcal{B}$

1: $s = xy$ in $\mathcal{B}$
2: $s = xy$ in $\mathcal{A} \cup a_r$
3: $u = s(-N^{-1})$ in $\mathcal{B}$
4: $u$ in $\mathcal{A} \cup a_r \Leftarrow \text{BE1}(u \text{ in } \mathcal{B})$
5: $t = uN$ in $\mathcal{A} \cup a_r$
6: $v = s + t$ in $\mathcal{A} \cup a_r$
7: $w = v B_A^{-1}$ in $\mathcal{A} \cup a_r$
8: $w$ in $\mathcal{B} \Leftarrow \text{BE2}(w \text{ in } \mathcal{A} \cup a_r)$

[I] With BBEs $B > (k+1)^2 N$ and $A > (k+1)N$

Algorithm 4: Proposed RNS Montgomery multiplication

**Input:** $\hat{x}$, $\hat{y}$, and $N$ in $\mathcal{A} \cup \mathcal{B} \cup a_r$, such that[I] $A = \prod_{j=1}^{k} a_j$, $B = \prod_{i=1}^{k} b_i$, and $\gcd(A, B) = 1$, $\gcd(N, B) = 1$, $0 \le xy < NB$, $B > 4N$, and $A > 2N$

**Output:** $w \equiv xy B_N^{-1} \pmod{N}$, $w < 2N$ in $\mathcal{B}$, $\hat{w} \equiv xy B_N^{-1} A_j^{-1} \pmod{N}$ in $\mathcal{A} \cup a_r$

**Precomputation:**

1: $s = xy$ in $\mathcal{B}$
2: $\hat{s} = \hat{x}\hat{y}$ in $\mathcal{A} \cup a_r$
-
3: $\hat{w}$ in $\mathcal{A} \cup a_r \Leftarrow \text{BE1}(s \text{ in } \mathcal{B}, \hat{s} \text{ in } \mathcal{A})$
-
-
-
4: $w$ in $\mathcal{B} \Leftarrow \text{BE2}(\hat{w} \text{ in } \mathcal{A} \cup a_r)$

Fig. 2. Comparison between the proposed RNS Montgomery multiplication and the state-of-the-art algorithm

the RNS MM is required to reach the value for that BE. In contrast to the original multiplication, the correction can be merged to another multiplication, decreasing the number of modular multiplications in the loop. The values that are multiplied by $A_j^{-1}$ in $\mathcal{A}$ are represented with a $\widehat{hat}$ accent. In order to reach the correct result of the exponentiation, another multiplication by $A_j$ is required after the loop of RNS MMs (step 12).

## C. RNS Montgomery Multiplication

In general, in RNS the MM [6], [9] is performed on two RNS bases, $\mathcal{A} = (a_1, ..., a_k)$ and $\mathcal{B} = (b_1, ..., b_k)$, such that $A = \prod_{j=1}^{k} a_j$, $B = \prod_{i=1}^{k} b_i$, and $\gcd(A, B) = 1$. $B$ is used as the Montgomery constant, so $B_A^{-1}$ must be pre-computed on the base $\mathcal{A}$, where $B_A^{-1}$ is the multiplicative inverse on $A$ of $B$. The input values of each RNS MM, Algorithm 3 in Fig. 2, are $x$ and $y$ in $\mathcal{A} \cup \mathcal{B}$, and the output is $w = xy B_N^{-1} \bmod N$, in $\mathcal{A} \cup \mathcal{B}$, where $B_N^{-1}$ is the multiplicative inverse of $B$ modulo $N$.

The relevant characteristics of MM implementation in RNS are described in the following. Step 3 is only performed on $\mathcal{B}$, so that the modular reduction by $B$ does not require additional operations. After the modular reduction, the BE to $\mathcal{A}$ of step 4 is required, since a subsequent step requires a different base in order to perform a division by $B$ and $\gcd(A, B) = 1$. The multiplication in step 5 and the addition in step 6 are only performed on $\mathcal{A}$, since the result of the multiplication in $\mathcal{B}$ is equal to the additive inverse on $\mathcal{B}$ of the result of step 1 and so the result of the addition in $\mathcal{B}$ is 0. The multiplication by $B^{-1}$ in step 7, which corresponds to the division in MM step 6, is only performed on $\mathcal{A}$, since $\gcd(B, B) \ne 1$. The last operation is the BE to $\mathcal{B}$, so that the result can be used as input for other MMs.

In [9] and [6], the respective authors have proposed two different techniques to perform the BE. Both techniques are based on (1), but avoid the last modular reduction by $A$ in order to save computational effort. In [6], the final modular reduction of both BEs required by the RNS MM are replaced by an approximation and a correction. In [9], the final modular reduction of the first BE is not executed, so all the subsequent results are obtained modulo a multiple of the base. However, the second BE gives the exact value due to a correction executed at the end of the summation of multiplications. The adopted correction technique has been presented in [10], and it requires that all the values calculated on $\mathcal{A}$ are also calculated on an additional base element $a_r$.

This paper presents a new RNS MM, Algorithm 4, which requires less modular multiplication steps. In order to reach this result, some values are pre-computed and the algorithm is modified accordingly:

1) The multiplication by $-N^{-1}$ in $\mathcal{B}$ of Algorithm 3 step 3 can be moved into the subsequent BE, and merged with the multiplication by $B_i^{-1}$ in $\mathcal{B}$, which corresponds to $x_i A_j^{-1}$ in (1). Therefore, $-N^{-1}B_i^{-1}$ must be pre-computed in order to avoid a multiplication step.

2) The multiplication by $N$ in $\mathcal{A}$ of Algorithm 3 step 5 can be moved into the previous BE, and merged with the summation of multiplications by $A_j$ in $\mathcal{A}$ of (1). Therefore, $A_j N$ must be pre-computed in order to avoid a multiplication step.

3) The multiplication by $B_A^{-1}$ can be split in two parts. One part can be moved in the first BE, and merged with the summation of multiplications by $A_j N$ in $\mathcal{A}$ of (1). Therefore, $A_j N B_A^{-1}$ (instead of $A_j N$) must be pre-computed. Also $s$ in $\mathcal{A}$ must be multiplied by $B_A^{-1}$, so the number of multiplication steps is not reduced, but the new multiplication can be moved in the first BE.

4) The input values in $\mathcal{A}$ are pre-multiplied by $A_j^{-1}$ (Algorithm 2, steps 2 and 4), so a multiplication by $A_j$ is required to reach the correct value for the second BE. This correction can be merged to the multiplication by $B_A^{-1}$ and no additional execution steps are required. For this, $B_A^{-1}A_j^{-1}$ must be pre-computed.

As already said, the proposed approach is presented in two versions, which employ different BE algorithms. The only difference in the MM algorithm is the presence of the RNS base element $a_r$, since this special correction base is required only by the version based on the approach proposed in [9], which uses the BE of [10] for the second one. According to this version, Algorithm 3 and 4 include the base element $a_r$.

The input values of the proposed RNS MM (Algorithm 4) are $\hat{x}$ and $\hat{y}$ in $\mathcal{A} \cup \mathcal{B} \cup a_r$ (which correspond

Algorithm 5: First Base Extension proposed Kawamura et al. [6]

**Input:** $u$ in $\mathcal{B}$, $\alpha = 0$
**Output:** $u$ in $\mathcal{A}$
**Precomputation:** $B_i^{-1} \bmod b_i$ for $i = 1...k$,
  $B_i$ in $\mathcal{A}$ for $i = 1...k$,
  $-B$ in $\mathcal{A}$.
1: $q_i = u_{bi} B_i^{-1} \bmod b_i$ for $i = 1...k$
2: $c = \alpha$
3: $u_{aj} = 0$ for $j = 1...k$
4: **for** $i = 1$ to $k$ **do**
5:     $c = c + \dfrac{\mathrm{trunc}(q_i)}{2^r}$
6:     $f^{\mathrm{I}} = \lfloor c \rfloor$
7:     $c = c - f$
8:     $u_{aj} = (u_{aj} + q_i B_i + f(-B)) \bmod a_j$
  for $j = 1...k$
9: **end for**
$^{\mathrm{I}}$ $f \in \{0, 1\}$

Algorithm 6: First Base Extension based on the algorithm by Kawamura et al. (KBE1)

**Input:** $s$ in $\mathcal{B}$, $\alpha = 0$, $\hat{\hat{s}}$ in $\mathcal{A}$
**Output:** $\hat{w}$ in $\mathcal{A}$
**Precomputation:** $-N^{-1} B_i^{-1} \bmod b_i$ for $i = 1...k$,
  $B_i N B_A^{-1} A_j^{-1}$ in $\mathcal{A}$ for $i = 1...k$, $-BN B_A^{-1} A_j^{-1}$,
  $B_A^{-1} A_j$ in $\mathcal{A}$.
1: $q_i = (s_{bi}(-N^{-1} B_i^{-1})) \bmod b_i$ for $i = 1...k$
2: $c = \alpha$
3: $\hat{w}_{aj} = \hat{\hat{s}}_{aj} B_A^{-1} A_j$ for $j = 1...k$
4: **for** $i = 1$ to $k$ **do**
5:     $c = c + \dfrac{\mathrm{trunc}(q_i)}{2^r}$
6:     $f^{\mathrm{I}} = \lfloor c \rfloor$
7:     $c = c - f$
8:     $\hat{w}_{aj} = (\hat{w}_{aj} + q_i B_i N B_A^{-1} A_j^{-1} + f(-BN B_A^{-1} A_j^{-1})) \bmod a_j$ for $j = 1...k$
9: **end for**

Fig. 3. Comparison between the first BE presented in [6] and the proposed BE (KBE1)

Algorithm 7: Second Base Extension proposed Kawamura et al. [6]

**Input:** $w$ in $\mathcal{A}$, $\alpha = 0.5$
**Output:** $w$ in $\mathcal{B}$
**Precomputation:** $A_j^{-1} \bmod a_j$ for $j = 1...k$,
  $A_j$ in $\mathcal{B}$ for $j = 1...k$, $-A$ in $\mathcal{B}$.
1: $q_j = w_{aj} A_j^{-1} \bmod b_j$ for $i = 1...k$
2: $c = \alpha$
3: $w_{bi} = 0$ for $i = 1...k$
4: **for** $j = 1$ to $k$ **do**
5:     $c = c + \dfrac{\mathrm{trunc}(q_j)}{2^r}$
6:     $f^{\mathrm{I}} = \lfloor c \rfloor$
7:     $c = c - f$
8:     $w_{bi} = (w_{bi} + q_j A_j + f(-A)) \bmod b_i$ for $i = 1...k$
9: **end for**
$^{\mathrm{I}}$ $f \in \{0, 1\}$

Algorithm 8: Second Base Extension based on the algorithm by Kawamura et al. (KBE2)

**Input:** $\hat{w} = q$ in $\mathcal{A}$, $\alpha = 0.5$
**Output:** $w$ in $\mathcal{B}$
**Precomputation:** $A_j$ in $\mathcal{B}$ for $j = 1...k$,
  $-A$ in $\mathcal{B}$.
-
1: $c = \alpha$
2: $w_{bi} = 0$ for $i = 1...k$
3: **for** $j = 1$ to $k$ **do**
4:     $c = c + \dfrac{\mathrm{trunc}(q_j)}{2^r}$
5:     $f^{\mathrm{I}} = \lfloor c \rfloor$
6:     $c = c - f$
7:     $w_{bi} = (w_{bi} + q_j A_j + f(-A)) \bmod b_i$ for $j = 1...k$
8: **end for**

Fig. 4. Comparison between the second BE presented in [6] and the proposed BE (KBE2)

to $x$ and $y$ in $\mathcal{B}$ and $x\mathcal{A}_j^{-1}$ and $y\mathcal{A}_j^{-1}$ in $\mathcal{A} \cup a_r$). The output is $\hat{w}$, which is equivalent to $(xyB_N^{-1})$ $(\bmod\ N) < 2N$ in $\mathcal{B}$, and to $((xyB_N^{-1})\ (\bmod\ N))\mathcal{A}_j^{-1}$ in $\mathcal{A} \cup a_r$.

In the following, the steps of the algorithms in Fig. 2 are analyzed in details. Step 1 of Algorithm 3 corresponds to step 1 of the proposed algorithm (Algorithm 4). Step 2 of Algorithm 4 is the multiplication of two values pre-multiplied by $A_j^{-1}$; thus, it provides the same results of step 2 in Algorithm 3, multiplied by $A_j^{-2}$. In Algorithm 4, steps 3, 5, 6, and 7 of Algorithm 3 are moved into the first BE. Moreover, the multiplication by $A_j$, which is required to correct the input, is also moved into the first BE.

*D. BE based on the approach by Kawamura et al. (KBE)*

Fig. 3 shows the first BE proposed by Kawamura et al. in [6] (Algorithm 5), and the proposed BE based on Kawamura et al. approach (KBE1) (Algorithm 6). KBE1 includes all the operations of Algorithm 5 as well as the operations of Algorithm 3 steps 3, 5, 6, and 7.

The modular reductions by $A$ in the first BE and by $B$ in the second BE would require a great computational effort, so they are replaced by an approximation and a correction, that are expected to be able to reduce the delay. The result of the summation in the BE of $x$ from $\mathcal{A}$ to $\mathcal{B}$ is $\tilde{x} = x + \lambda A$, where $\lambda \in \mathbb{Z}$ and $0 \leq \lambda < k$. Instead of performing the modular reduction in order to reach $x$, the algorithm proposed by Kawamura et al. calculates the approximate value of $\lambda$, $\tilde{\lambda} = \lfloor \alpha + \sum_0^k trunc(q_i)/2^r \rfloor \simeq \lfloor \sum_0^k q_i/b_i \rfloor$. The value of $\tilde{\lambda}$ is estimated accumulating the $\varrho$ most significant bits of $q_i$, cut by $trunc()$ and divided by $2^r$. Moreover, $trunc(q_i) = q_i \wedge (1_{(r)}...1_{(r-\varrho+1)}0_{(r-\varrho)}...0_{(1)})_{(2)}$ , and $\wedge$ means a bitwise AND operation. Kawamura et al. introduced a further variable ($\alpha$), which represents the starting value of the parameter that is used to correct the error introduced by the approximation. In [6], two theorems prove that with correct values of $\alpha$, with a low $\varrho$, and by selecting the base elements so that $2^r$ is close to $b_i$, the approximation does not introduce errors. During the first BE $\alpha = 0$ and the input is unknown; thus, according to Theorem 2 in [6], the result of the BE is $\tilde{x} < 2B$, which is approximate. During the second BE $\alpha = 0.5$ and the input is lower then $2N$; hence, according to Theorem 1 in [6], the result of the BE is correct. With $r = 32$, $k = 33$, and $\max(2^r - a_i; 2^r - b_i) < 2^{16}$, $\forall i$, it is possible to choose $\varrho \geq 7$. This approach requires that $A \geq 2N$ and $B \geq 4N$. The two BEs have the same algorithm with exchanged bases, but according to the theorems presented in [6], the first BE produces an approximate result, which is corrected by the second BE.

Step 1 of Algorithm 5 is the multiplication of each $x_i$ by $B_i^{-1}$, modulo $b_i$. In KBE1, this multiplication is merged to the multiplication by $-N^{-1}$ of Algorithm 3 step 3; thus, instead of $B_i^{-1}$, $-N^{-1}B_i^{-1}$ is pre-computed. It can be easily demonstrated that the result of this operation is the same as the result of the corresponding operation in the Kawamura et al. BE, for the associative property.

Step 3 of Algorithm 5 is a simple initialization. In KBE1, step 3 executes the multiplication in step 7 of Algorithm 3 merged with the multiplication by $A_j$, used to correct the $A_j^{-2}$ factor. Therefore, $A_j B_A^{-1}$ must be pre-computed.

Step 8 of Algorithm 5 requires, for each base element $a_j$, the summation of the results of the multiplications of each result of step 1 by the corresponding $B_i \bmod a_j$, and of $-B$, when the floor $f$ of the

Algorithm 9: First Base Extension employed by Bajard et al. [9], [11]

**Input:** $u$ in $\mathcal{B}$
**Output:** $\tilde{u}$ in $\mathcal{A} \cup a_r$
**Precomputation:** $B_i^{-1} \bmod b_i$ for $i = 1...k$,
 $B_i$ in $\mathcal{A} \cup a_r$ for $i = 1...k$.
1: $q_i = u_{bi}B_i^{-1} \bmod b_i$ for $i = 1...k$
2: $\tilde{u}_{aj} = \sum_{i=1}^{k} q_i B_i \bmod a_j$ for $j = 1...k$ and $j = r$

Algorithm 10: First BE based on the algorithm by Bajard et al. (BBE1)

**Input:** $s$ in $\mathcal{B}$, $\hat{s}$ in $\mathcal{A} \cup a_r$
**Output:** $\hat{w} = q$ in $\mathcal{A} \cup a_r$
**Precomputation:** $-N^{-1}B_i^{-1} \bmod b_i$ for $i = 1...k$,
 $B_i N B_A^{-1} A_j^{-1}$ in $\mathcal{A}$ for $i = 1...k$, $B_A^{-1} A_j$ in $\mathcal{A}$.
1: $q_i = s_{bi}(-N^{-1}B_i^{-1}) \bmod b_i$ for $i = 1...k$
2: $\hat{w}_{aj} = \hat{s}_{aj} B_A^{-1} A_j + \sum_{i=1}^{k} q_i (B_i N B_A^{-1} A_j^{-1}) \bmod a_j$
 for $j = 1...k$ and $j = r$

Fig. 5. Comparison between the first BE used in [9] and the proposed BE (BBE1)

Algorithm 11: Second Base Extension employed by Bajard et al. [9]–[11]

**Input:** $w$ in $\mathcal{A} \cup a_r$
**Output:** $w$ in $\mathcal{B}$
**Precomputation:** $A_j^{-1} \bmod a_j$ for $j = 1...k$,
 $A_r^{-1} \bmod a_r$, $-A$ in $\mathcal{B}$, $A_j$ in $\mathcal{B}$ for $j = 1...k$,
 $A_j \bmod a_r$ for $i = 1...k$.
1: $q_j = w_{aj}(A_j^{-1}) \bmod a_j$ for $j = 1...k$
2: $d_r = (\sum_{j=1}^{k} q_j (A_j \bmod a_r)) \bmod a_r$
3: $\beta = (d_r - x_r)(A_r^{-1}) \bmod a_r$
4: $d_i = (\sum_{j=1}^{k} q_j (A_j)) \bmod b_i$ for $i = 1...k$
5: $w_{bi} = (d_i - (A\beta) \bmod b_i) \bmod b_i$ for $i = 1...k$

Algorithm 12: Second BE Based on the algorithm by Bajard et al. (BBE2)

**Input:** $\hat{w} = q$ in $\mathcal{A}$, $w_r$ in $a_r$
**Output:** $w$ in $\mathcal{B}$
**Precomputation:** $A_r^{-1} \bmod a_r$, $-A$ in $\mathcal{B}$,
 $A_j$ in $\mathcal{B}$ for $j = 1...k$,
 $A_j \bmod a_r$ for $j = 1...k$.
-
1: $d_r = (\sum_{j=1}^{k} q_j (A_j \bmod a_r)) \bmod a_r$
2: $\beta = (d_r - x_r)(A_r^{-1}) \bmod a_r$
3: $d_i = (\sum_{j=1}^{k} q_j (A_j)) \bmod b_i$ for $i = 1...k$
4: $w_{bi} = (d_i - (A\beta) \bmod b_i) \bmod b_i$ for $i = 1...k$

Fig. 6. Comparison between the second BE used in [9] and the proposed BE (BBE2)

approximation is equal to 1. In KBE1, the multiplications in Kawamura et al. step 8 are merged to the multiplication by $N$ in Algorithm 3 step 5, by $B_A^{-1}$ in Algorithm 3 step 7, and by $A_j^{-1}$ in the second BE. Instead of of $B_i \bmod a_j$ and $-B$, $B_i N B_A^{-1} A_j^{-1} \bmod a_j$ and $-BNB_A^{-1}A_j^{-1}$ are pre-computed, respectively

It can also be easily demonstrated that the result of step 8 of KBE1 is the same result of the corresponding operations in Algorithm 5, for the associative property and for the distributive property,

The second BE proposed in [6], Algorithm 7 in Fig 4, uses the same algorithm of the first BE (Algorithm. 5), but with the bases switched. The proposed algorithm (KBE2), Algorithm 8, is the same, but it does not perform step 1 of Algorithm 7, since the input value is already multiplied by $A_j^{-1}$.

*E. BE based on the approach by Bajard et al. (BBE)*

In [9], Bajard at al. propose a MM algorithm requiring two different BEs; the former, Algorithm 9 in Fig. 5, trades approximation for speed, whereas the latter, Algorithm 11 in Fig. 6 originally proposed by Shenoy and Kumaresan [10], corrects the result. The result of the approximate BE of $x$ is $\tilde{x} = x + \lambda B$, and no correction steps are performed in order to reach the correct results. Further details are presented in [11], where the algorithm is applied to the ME in the contexts of RSA. The approximation does not affect the final result of the MM, provided that overflows after the BE are avoided through the use of

larger bases guaranteeing:

$$A, B > N(k + 2)^2 \ . \tag{2}$$

The First BE proposed in this work, Algorithm 10 in Fig. 5, is based on the approach by Bajard et al. (BBE1) and includes the operations of Algorithm 9 as well as the operations of Algorithm 3 steps 3, 5, 6, and 7.

In BBE1, the multiplication of Algorithm 9 step 1, is merged to the multiplication by $-N^{-1}$ of Algorithm 3 step 3; hence, instead of $B_i^{-1}$, $-N^{-1}B_i^{-1}$ is pre-computed. It can be easily demonstrated that the result of this operation is the same of the corresponding operation in Algorithm 9.

In step 2 of BBE1, the summation is initialized with the multiplication of Algorithm 3 step 7 merged with the multiplication by $A_j$, used to correct the $A_j^{-2}$ factor. Therefore, $A_j B_A^{-1}$ must be pre-computed. The multiplications in the summation are merged to the multiplication by $N$ in Algorithm 3 step 5, by $B_A^{-1}$ in Algorithm 3 step 7, and by $A_j^{-1}$ in the second BE. Instead of $B_i \bmod a_j$, $B_i N B_A^{-1} A_j^{-1} \bmod a_j$ is pre-computed. It can be easily proved that the result of this operation is the same of the respective operation in Algorithm 5.

The approximation correction is only performed after the second BE, and it requires an additional RNS base element $a_r$, such that $\gcd(a_r, A) = 1$ and $\gcd(a_r, B) = 1$ (where $a_r$ is a power of 2). All the values in $\mathcal{A}$ are also calculated in $a_r$, according to [10].

The second BE employed in [9], Algorithm 11, requires a correction in order to avoid the approximation. Step 2 calculates the difference between the correct value of $x$ in $a_r$ and the result of the approximate BE on $a_r$, which correspond to:

$$t_r = x \bmod a_r - (x + \lambda A) \bmod a_r = \lambda A \bmod a_r. \tag{3}$$

The second proposed BE based on the approach by Bajard et al.(BBE2), Algorithm 12 in Fig. 6, corresponds to Algorithm 11 without step 1; this step can be avoided since the input of the BE is already multiplied by $A_j^{-1}$.

TABLE III

NUMBER OF MODULAR MULTIPLICATIONS REQUIRED BY THE CONSIDERED RNS MM ALGORITHMS

|  | [6] | [9] | Proposed (with KBE) | Proposed (with BBE) |
|---|---|---|---|---|
| Step 1, 3, and 4 of MM | $5k$ | $5k$ | $2k$ | $2k$ |
| First BE without correction | $k^2 + k$ | $k^2 + k$ | $k^2 + 2k$ | $k^2 + 2k$ |
| First BE correction | $k$ | $0$ | $k$ | $0$ |
| Second BE without correction | $k^2 + k$ | $k^2 + k$ | $k^2$ | $k^2$ |
| Second BE correction | $k$ | $k$ | $k$ | $k$ |
| Total without BE correction | $2k^2 + 7k$ | $2k^2 + 7k$ | $2k^2 + 4k$ | $2k^2 + 4k$ |
| Total | $2k^2 + 9k$ | $2k^2 + 8k$ | $2k^2 + 6k$ | $2k^2 + 5k$ |

TABLE IV

CLASSIFICATION OF THE MULTIPLICATION STEPS OF THE MM ALGORITHM (WITHOUT BE CORRECTION)

| Multiplication ID | Operation | Base | Characteristics | Number of multiplications | Number of execution steps | Proposed | [6], [9] |
|---|---|---|---|---|---|---|---|
| 1 | $s = xy$ | $\mathcal{B}$ | Full | $k$ | $p$ | ● | ● |
| 2 | $s = xy$ | $\mathcal{A} \bigcup a_r$ | Full parallelizable | $k$ | $\lfloor \frac{1}{p+M-1} \rfloor$ | ● | ● |
| 3 | $u = s(-N^{-1})$ | $\mathcal{B}$ | Full | $k$ | $p$ | ● | ● |
| 4 | $q = uB_i^{-1}$ | $\mathcal{B}$ | Full | $k$ | $p$ | ○ | ● |
| 5 | $\hat{w} = \hat{s}B_A^{-1}A_j$ | $\mathcal{A} \bigcup a_r$ | Full parallelizable | $k$ | $\lfloor \frac{1}{p+M-1} \rfloor$ | ● | ○ |
| 6...$k$+5 | $q_i B_j$ | $\mathcal{A} \bigcup a_r$ | Parallelizable | $k^2$ | $\lceil \frac{k}{M} \rceil - 1 + p$ | ● | ● |
| $k$+6 | $t = uN$ | $\mathcal{A} \bigcup a_r$ | Full | $k$ | $p$ | ○ | ● |
| $k$+7 | $w = vB_A^{-1}$ | $\mathcal{A} \bigcup a_r$ | Full | $k$ | $p$ | ○ | ● |
| $k$+8 | $q = wA_j^{-1}$ | $\mathcal{A}$ | Full | $k$ | $p$ | ○ | ● |
| $k$+9...$2k$+8 | $q_j A_j$ | $\mathcal{B} \bigcup a_r$ | Parallelizable | $k^2$ | $\lceil \frac{k}{M} \rceil - 1 + p$ | ● | ● |

○ The multiplication is executed by the algorithm ● The multiplication is not executed

## III. ALGORITHM ANALYSIS

In this section, the proposed approach is evaluated and compared with the state-of-the-art algorithms. The analysis is focused on the MM, which requires the majority of the total computational time. In particular, Section III-A performs an analysis taking into account the number of modular multiplications, as in previous works [6], [11]. Section III-B analyzes the characteristics of the operations performed by the algorithms, evaluating their weight. Finally, Section III-C summarizes the conclusions of the analysis.

### A. Number of modular multiplications

Both the approaches in [6] and [11] have been evaluated by the respective authors according to the number of modular multiplications required. Table III reports the comparison of the proposed RNS MM algorithm with the previous ones. It can be easily seen that the algorithm presented in [11] achieves a reduction of $k$ modular multiplications with respect to [6], whereas the proposed algorithm allows a further saving of $3k$ modular multiplications.

*B. Analysis and classification of the required multiplications*

As shown in Table III, the described algorithms require $2k^2$ and between $5k$ to $9k$ modular multiplications. All the necessary multiplications (not considering the BE correction) are listed and classified in Table IV. It can be observed that $2k^2$ modular multiplications correspond to $2k$ multiplications required for the BEs (IDs from 6 to $k+5$ and from $k+9$ to $2k+8$) performed on $k$ base elements. Moreover, $8k$ modular multiplications correspond to 6 multiplications (ID 3, 4, 5, $k+6$, $k+7$, and $k+8$) performed on $k$ base elements, and to 1 multiplication (ID 1 and 2), performed on $2k$ base elements.

Since each operation is performed on $k$ base elements and only one multiplication is performed on a larger base, up to $k$ cells can perform in parallel the required operations. Thus, considering that $k$ cells can work in parallel, the multiplications shown in Table IV are organized in $2k+8$ multiplication steps, composed by $k$ parallel multiplications. The multiplication steps are classified according to the opportuniy of parallelization and pipelining. These aspects are of paramount importance, since different multiplication steps can require a different number of execution steps. The identified types of multiplication steps are:

- $full$, where the beginning of the operation must wait for the completion of the previous operation that calculates an input value; the number of required execution steps is $p$, where $p$ corresponds to the number of pipeline stages;

- $parallelizable$, where a group of operations can be executed in parallel, or the first operation can be executed as full, and the subsequent ones can be pipelined;

- $fullparallelizable$, where an operation can be executed in parallel to the previous and/or to the subsequent one; when the operation is executed by a parallel or pipelined cell the number of required execution steps is 0.

*C. Remarks*

Considering that $k$ cells can perform $k$ modular multiplications in one multiplication step, by analyzing Table IV it can be observed that, without considering the BE correction, the RNS MM involves: 6 full multiplication steps (IDs 1, 3, 4, $k+6$, $k+7$, and $k+8$), two groups of $k$ parallelizable multiplication steps (IDs from 6 to $k+5$ and from $k+9$ to $2k+8$), and 2 fully parallelizable multiplication steps (IDs 2 and 5).

When the RNS MM algorithm used in the previous approaches is analyzed, by considering $M$ as the number of parallel multipliers per cell, it can be observed that the fully parallelizable step does not need the result of the previous operation. Hence, it can be fully parallelized by any pipelined architecture and

it does not affects the overall delay. This step requires $\lfloor\frac{1}{p+M-1}\rfloor$ execution steps.

When executed by an architecture with $M \leq k$, the two groups of consecutive parallelizable multiplications require $p$ execution steps for the first multiplication step, and $1/M$ execution steps for each other multiplication step, corresponding to $2\lceil\frac{k}{M}\rceil - 2 + 2p$ execution steps.

Each full multiplication step requires $p$ execution steps, which correspond to $6p$ execution steps.

Considering an architecture with $k$ cells, the number of execution steps required by the RNS MM used in [6], [11], without the BE correction, is $2\lceil\frac{k}{M}\rceil - 2 + \lfloor\frac{1}{p+M-1}\rfloor + 8p$.

As shown in Table IV, the proposed algorithm allows achieving a reduction of $4p$ steps (IDs 4, $k$+6, $k$+7, and $k$+8), and requires $\lfloor\frac{1}{p+M-1}\rfloor$ additional steps (ID 5). Therefore, the improvement due to the proposed modification is directly matched to the number of pipeline stages and of parallel multipliers. Considering $p = 3$, $M = 1$ and $k = 33$ as in [6], without the error correction a delay reduction of $13.63\%$ is obtained. With a higher degree of pipelining a larger reduction is achieved, e.g. $16.66\%$ with $p = 4$ and $M = 1$, $19.23\%$ with $p = 5$ and $M = 1$, $25\%$ with $p = 4$ and $M = 2$, etc.

The proposed exponentiation algorithm requires $2p + 1$ additional multiplication steps, but their impact on the total delay is negligible since it is equal to $(2p + 1)/(iteration \times (2\lceil\frac{k}{M}\rceil - 2 + \lfloor\frac{1}{p+M-1}\rfloor + 8p))$, e.g. $< 0.01\%$ with $p = 3$, $M = 1$, and $iteration > 1024$.

## IV. Implementation and Results

In this section the state-of-the-art architectures exploited in [6] and [11] are described and analyzed. Kawamura et al. presented some details about their architecture in [7]. This implementation is composed by a set of identical cells, where each is matched to one base element for each RNS base, or to a set of elements for each base. The cells are made up of a Modular Multiplier and Accumulator Unit (MMAU) with three stages of pipeline, a Cox Unit (CU) for the correction of the BE, and some memory. The details regarding the implementation based on the algorithm in [9] have not been presented, so in this section a new architecture tailored to this particular case is proposed. Except for the correction unit, the architecture adopted by Kawamura et al. is also suitable for the approach proposed by Bajard et al., which nonetheless requires a separate cell for the redundant base element, instead of the Cox correction unit.

In order to reach an efficient implementation, the multiplications are performed through reduction trees of Carry Save Adders (CSAs), whereas the addition is achieved by means of Carry Look Ahead Adders.

## A. Modular reduction

In the following, the operations used by the cells to perform the modular reduction are shown. The approach is the same used in [7].

In [15], the authors showed that the modulo reduction of $x < 2^{2r}$ requires two multiplications and three additions, where $a_i = 2^r - c_i$, with $c_i < 2^h$ and $h < \frac{r-1}{2}$. The partial modular reduction $y \equiv x \pmod{a_i}$ can be calculated by:

$$y = x \bmod 2^r + (x << r) \cdot c_i, \tag{4}$$

where $x < 2^z$, $z > r$, and $c_i < 2^h$. Thus, it is $y < \max\left(2^{r+1}, 2^{z-r+h+1}\right)$ and each iteration of this method can reach a reduction of $r-h-1$ bits. In order to reach a larger reduction per step with $2^{2r} < x < 2^{4r-2h-1}$, it is possible to calculate:

$$y = x \bmod 2^r + ((x << r) \bmod 2^r) \cdot c_i + (x << 2r) \cdot c_i^2, \tag{5}$$

where $x < 2^z$, $z > 2r$, and $c_i < 2^h$. Hence, it is $y < \max\left(2^{r+h+1}, 2^{z-2r+2h+1}\right)$. Each iteration of this equation can therefore reach a reduction of $2r - 2h - 1$ bits.

## B. Cell architecture without BE correction

Without the error correction, a cell basically corresponds to a MMAU opportunely controlled. The starting point for the design is represented by the architecture proposed in [7], that is shown in Fig. 7 ($p = 3$ and $M = 1$). The MMAU is divided in three pipelined units:

- *Multiplier Adder Unit (MAU)*, which performs the multiplications and the additions;
- *First Modular Reduction Unit (FMRU)*, which performs the partial modular reduction (5);
- *Second Modular Reduction Unit (SMRU)*, which calculates the final result of the modular reduction performing (4) and an addition.

## C. Analysis of the Cell without BE Correction

In order to evaluate the area and delay of the analyzed architecture, the number of gates that compose the arithmetic cells, and that represent the critical path, have been counted. In order to present a more clear comparison, the delay and area cost of each logic gate have been also converted in the equivalent
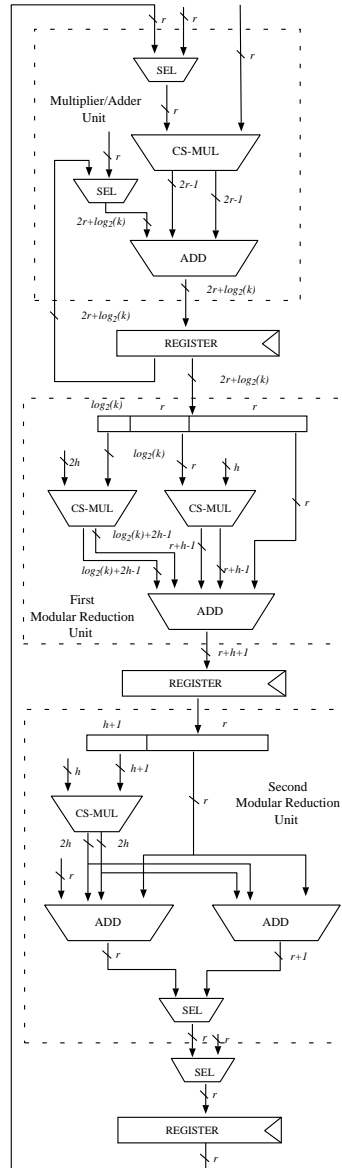
Fig. 7. MMAU architecture with three stages of pipelining

TABLE V

BASIC LOGIC LIBRARY IN CMOS TECHNOLOGY (MODEL FROM [16])

| Gate | Area (transistors) | Delay (Inverter) |
|---|---|---|
| Inverter | 2 | 1 |
| NAND | 4 | 1.4 |
| NOR | 4 | 1.4 |
| XNOR | 12 | 3.2 |
| NAND3 | 8 | 1.8 |
| NAND4 | 10 | 2.2 |
| REGISTER | 15 | 4.8 |

TABLE VI

AREA COST FOR THE CONSIDERED UNITS WITH $r = 32, k = 33, h = 11$

| Unit | Input | Reduction tree | Adder | Output | Reg. | Tot. (transistors) | |
|------|-------|----------------|-------|--------|------|--------------------|---|
| MAU | $1254NAND+$ $1414NOT$ | $960FA$ + $32HA$ | $140XNOR+69NAND4+92NAND3+$ $369NAND+70NOR+370NOT$ | 0 | 70 | 57248 | |
| FMRU | $484NAND+$ $630NOT$ | $431FA$ + $31\{29\}HA$ | $86XNOR+42NAND4+56NAND3+$ $225NAND+43NOR+226NOT$ | 0 | 44 | 26708 | 97692 |
| SMRU | $132NAND+$ $165NOT$ | $131FA$ + $20HA$ | $128XNOR+60NAND4+80NAND3+$ $324NAND+64NOR+328NOT$ | $192NAND+$ $41NOT$ | 64 | 13736 | |

The values between curly brackets are obtained using merged reduction trees

TABLE VII

DELAY OF THE CONSIDERED UNITS WITH $r = 32, k = 33, h = 11$

| Unit | Input | Red. tree | Adder | Output | Reg. | Step[I] (inverters) | MM (inverters) [6], [9] | Prop. |
|------|-------|-----------|-------|--------|------|---------------------|-------------------------|-------|
| MAU | $3NAND+3NOT$ | $14XNOR$ | $XNOR$ + $4NAND4$ + $12NAND+NOT$ | 0 | 1 | **86.6** | | |
| FMRU | $NAND+3NOT$ | $12XNOR$ | $XNOR$ + $4NAND4$ + $8NAND+NOT$ | 0 | 1 | 71.8 | 7620 | 7101 |
| SMRU | $NAND+2NOT$ | $10XNOR$ | $2XNOR$ + $XNOR$ + $4NAND4+6NAND+2NOT$ | $3NOT+$ $4NAND$ | 1 | 77.6 | | |

[I] Bold numbers represent the longest delay, which is equivalent to critical path of the cell

inverter delay, and in the equivalent number of transistors, respectively. For the conversion, the metric in [16], which is summarized in Table V, was selected.

Table VI and VII show the area and delay characteristics of the described cells considering $r = 32, k = 33$, and $h = 11$, as in [7], [11]. In order to evaluate the delay of the described architecture, the time requested by the longest critical path must be multiplied by the number of steps required. According to the theoretical analysis, the considered architecture needs to execute $2k + 22$ steps.

By contrast, the proposed RNS MM algorithm requires only $2k + 10$ steps. Considering $k = 33$, with the described cell the proposed algorithm reaches a time saving of $13.63\%$ compared with the previous ones, due to the smaller number of steps. These data confirm the results of the analysis in Section III-C.

## D. Error correction with the approach by Kawamura et al.

The error correction significantly affects the architecture of the cells, and it represents the main architectural difference between the approaches by Kawamura et al. and by Bajard et al.

TABLE VIII

AREA COST AND DELAY OF THE CONSIDERED UNITS WITH THE COX SUBUNIT

| Unit | Delay | Area (inverters) |
|------|-------|------------------|
| MAU | **93** | |
| FMRU | 62.2 | 99100 |
| SMRU | 68 | |

Fig. 8.  Cox Unit (a) and Redundant Cell Matched to $a_r$ (b)

The algorithm proposed in [6] can be implemented by adding to each cell a Cox Unit. This unit, which is illustrated in Fig. 8, is composed by an adder, a register and a set of AND gates. The delay of this unit corresponds to an adder and one AND gate. According to [7], a suitable value for $\varrho$, which represents the size of the adder in the Cox Unit, is 9, with $r = 32$, $h = 11$, and $k = 33$; in this case, the delay required by the unit can be estimated in 22.6 inverters. Kawamura et al. use an architecture similar to Fig. 8, and they place the Cox Unit in parallel to the reduction tree. The area of the reduction three is $r$ FA larger. The area and the delay of the units with the additional input line are reported in Table VIII.

### E. Error correction with the approach by Bajard et al.

The algorithm employed in [11] can be implemented by using the architectures previously described, but it requires an additional cell matched to a redundant base element. The aim of this cell is to calculate the BE correction, which is used by the other cells as a standard input value. Therefore, from the point of view of the cells matched to the base elements, the only difference is the sequence of operations.

The architecture of the redundant cell is shown in Fig. 8. It is composed by the Multiplier Unit (MU)

TABLE IX

AREA AND DELAY COMPARISON WITH $k = 33, r = 32, h = 11, M = 1, p = 3$

| Algorithm | Correction | # steps | Step delay | MM delay (Inverter) | Area (Transistor) |
|---|---|---|---|---|---|
| [6], [9] | Kawamura et al. [6] | 88 | 93 | 8184(100%) | 99873 × 33 |
| Proposed | Kawamura et al. [6] | 76 | 93 | 7068(86.4%) | 99873 × 33 |
| [6], [9] | Bajard et al. [9] | 89 | 86.6 | 7707(94.2%) | 99840 × 33 |
| Proposed | Bajard et al. [9] | 77 | 86.6 | 6669(81.5%) | 99840 × 33 |

and by the Adder Unit (AU). The number of represented bits is shorter than in other cells, according to the requirements, so two multiplications can be processed in parallel and added in a step. Moreover, $a_r$ can be a power of 2, so no reduction is required. The area overhead is similar to the approach proposed in [6]. The BE correction requires an additional step. However, as suggested in [11], it is possible to avoid a multiplication using tables, but the result from the table should be summed by adding an input line.

### F. Overall comparison and concluding remarks

Table IX summarizes the results of the comparison among the considered approaches. It is possible to observe that the BE correction does not affect noticeably the area required by the cell. The BE correction proposed by Bajard et al. requires an additional step (unless tables are used for the correction multiplication), but the Kawamura et al. correction increases the delay of the MAU, which represents the critical path of the cell. Therefore, the approach proposed by Bajard et al. provides a $5.8\%$ time saving.

The proposed algorithm provides a delay reduction linked to the BE correction algorithm, and it does not affect the area. With the Kawamura et al. correction, the delay reduction is $13.6\%$, while with the Bajard et al. correction it is $13.4\%$.

The most efficient cell is obtained by mixing the BE approach used in [11] with the proposed algorithm, since it does not require additional area and it provides a $18.5\%$ delay reduction.

## V. CONCLUSION

In this paper a novel RNS Montgomery exponentiation algorithm is proposed. The algorithm is presented in two versions, targeted to the BE approaches adopted in [6] and in [11], respectively. The architecture proposed in [7], that is compliant with the approach in [6], has been used as a reference point for the design of a new architecture suitable for the method adopted in [11].

An algorithmic analysis has shown that the proposed approach is capable of providing a reduction of $4p - \lfloor 1/(p + M - 1) \rfloor$ steps over the $2\lceil k/M \rceil - 2 + \lfloor 1/(p + M - 1) \rfloor + 8p$ required by each RNS MM (without considering the BE correction). Then, an architectural analysis has shown that, with the

BE proposed in [6], the total number of steps and the reduction are the same, whereas with the BE approach used in [11], the reduction is the same but the MM requires one additional step. According to the algorithmic characteristics described in [6], [11], and to the architectural features described in [7], the delay reduction is equal to $13.6\%$ or $13.4\%$ depending on whether the BE adopted in [6] or in [11] is used, respectively.

## REFERENCES

[1] N. S. Szab and R. I. Tanaka, *Residue Arithmetic and Its Applications to Computer Technology*. McGraw-Hill, 1967.

[2] R. Szerwinski and T. Gneysu, "Exploiting the power of GPUs for asymmetric cryptography," in *Cryptographic Hardware and Embedded Systems CHES 2008*, ser. LNCS, E. Oswald and P. Rohatgi, Eds. Springer Berlin / Heidelberg, 2008, vol. 5154, pp. 79–99.

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

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

[5] ——, "Base extension using a convolution sum in residue number systems," *Computing*, vol. 50, pp. 93–104, 1993, 10.1007/BF02238608. [Online]. Available: http://dx.doi.org/10.1007/BF02238608

[6] S. Kawamura, M. Koike, F. Sano, and A. Shimbo, "Cox-rower architecture for fast parallel Montgomery multiplication," in *Advances in Cryptology EUROCRYPT 2000*, ser. LNCS. Springer Berlin / Heidelberg, 2000, pp. 523–538.

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

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

[9] ——, "Modular multiplication and base extensions in residue number systems," in *Computer Arithmetic, 2001. Proceedings. 15th IEEE Symposium on*, 2001, pp. 59–65.

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

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

[12] N. Guillermin, "A high speed coprocessor for elliptic curve scalar multiplications over $F_p$," in *Workshop on Cryptographic Hardware and Embedded Systems 2010 (CHES 2010)*, ser. LNCS. Springer Berlin / Heidelberg, 2010.

[13] BLIND, "Pre-processing in RNS Montgomery multiplication," Tech. Rep.

[14] M. Pohst and H. Zassenhaus, Eds., *Algorithmic algebraic number theory*. New York, NY, USA: Cambridge University Press, 1989, ch. 2.2.5.

[15] N. M. J.C. Bajard and T. Plantard, "Efficient RNS bases for cryptography," in *IMACS'05 : World Congress: Scientific Computation, Applied Mathematics and Simulation*, July 2005.

[16] D. Gajski, *Principles of Digital Design*. Prentice-Hall, 1997.