Novel RNS Parameter Selection for Fast Modular Multiplication

Gavin Xiaoxu Yao, Student Member, IEEE, Ray C.C. Cheung, Member, IEEE, Junfeng Fan, Student Member, IEEE, and Ingrid Verbauwhede, Fellow, IEEE

Abstract—The parameter selection of Residue Number Systems (RNS) has a great impact on its computational efficiency. This paper shows that a base extension, the most costly operation in RNS Montgomery multiplication, can be more efficient when the intervals between the RNS moduli are small. We propose a systematic RNS parameter selection procedure and two methods to select RNS moduli that lead to a reduced complexity. Our experimental results confirm the advantages of the selected moduli.

Index Terms—Modular multiplication, Residue Number System (RNS), base extension

1 INTRODUCTION

MODULAR multiplication of long integers is a basic operation in Public-Key Cryptography (PKC) [1]. The size of the operands ranges from a few hundred bits (e.g., 256-bit Elliptic Curve Cryptography, ECC [2], [3]) to several hundred thousand bits (e.g., 741455-bit fully homomorphic encryption [4]). Building a high-throughput long integer modular multiplier is thus the key step towards a high-performance PKC implementation.

Parallelization has been one of the most powerful methods to accelerate arithmetic operations in both hardware and software implementations [5]. The operations in a Residue Number System (RNS) are distributed into a group of small integers, and are naturally suitable for parallel implementations. Thus, the Montgomery modular algorithms [6] were proposed in the RNS context for cryptographic applications [7], [8], and the parallelism provides the capability of high speed modular multiplication [9], [10], [11], [12].

However, the complexity of the RNS Montgomery is slightly higher than the conventional Montgomery. Digit-serial Montgomery algorithm [13] only uses $2n^2+n$ $w\times w$-bit multiplications, while RNS version requires $2n^2+4n$ [14] for computing $AB \mod P$, where $A$, $B$ and $P$ have $n$ $w$-bit digits. The most computationally expensive step of the RNS Montgomery algorithm is the Base Extension (BE). Out of the $2n^2+4n$ digit multiplications, $2n^2+2n$ multiplications are due to the BEs.

In this paper, we first show that if the difference between any two moduli of the RNS bases are small, i.e., all the moduli are close, the operand size of the $2n^2$ multiplications in BEs can be reduced from $w \times w$-bit to $v \times w$-bit, where $v$ could be much smaller than $w$. Consequently, the $2n^2+4n$ $w \times w$-bit multiplications can be substituted by $2n^2$ $v \times w$-bit and $5n$ $w \times w$-bit multiplications. Two methods are provided to search for such kind of close moduli, and a systematic RNS parameter selection procedure is described to achieve the minimum complexity.

The rest of the paper is organized as follows. Section 2 gives a recap of the mathematical background. Section 3 describes the specifications of bases to achieve faster base extensions. The complexity of the refined RNS algorithm is analyzed in Section 4. We provide two methods to select RNS moduli in Section 5. Section 6 discusses briefly the related work and the implementation results. We conclude this work in Section 7.

Due to the nature of this subject, we need a great amount of variables. For the ease of reading, we summarize all the notations in Table 1. The definition is given when appropriate in the context.

2 BACKGROUND

In this section, we give a brief introduction to RNS arithmetic and RNS Montgomery algorithm.

2.1 Residue Number System

A Residue Number System represents a large integer with several smaller integers. An RNS is defined by a set of $n$ pairwise coprime integer constants: $\mathbb{B} = \{b_1, b_2, \ldots, b_n\}$. The set $\mathbb{B}$ is also known as a base, and the element $b_i$, $1 \leq i \leq n$, is called an RNS modulus, and each modulus forms an RNS channel. Let $M_\mathbb{B} = \prod_{i=1}^{n} b_i$. Let $|a|_b$ be a modulo $b$, then any integer $X$, $0 \leq X < M_\mathbb{B}$, can
be uniquely represented by a set of smaller integers:

\[ \{X\}_{28} = \{X|B_1|X|B_2|\ldots|X|B_n\}. \]

We denote \( w \) the bit-length of \( b_i \).

The original value of \( X \) can be restored from \( \{X\}_{28} \) using the Chinese Remainder Theorem (CRT):

\[
X = \sum_{i=1}^{n} \xi_i \cdot B_i \mod M_{28} := \left[ \sum_{i=1}^{n} \left[ x_i \cdot B_i^{-1} \right] b_i \right]_M, \quad (1)
\]

\[
B_i = M_{28}/b_i = \prod_{j=1, j \neq i}^{n} b_j, \quad 1 \leq i \leq n. \quad (2)
\]

Using RNS, arithmetic operations in \( \mathbb{Z}/M_{28}\mathbb{Z} \) can be efficiently performed. Consider two integers \( X, Y \) and their RNS representations \( \{X\}_{28} = \{x_1, x_2, \ldots, x_n\} \) and \( \{Y\}_{28} = \{y_1, y_2, \ldots, y_n\} \), then

\[
\{X \odot Y\}_{M_{28}} = \{x_1 \odot y_1|b_1, \ldots, x_n \odot y_n|b_n\}.
\]

for \( \odot \in \{+, -, \times, /\} \). The division is available only if the multiplicative inverse of \( Y \) exists in \( \mathcal{B} \), i.e. \( Y \) is coprime with \( M_{28} \).

For every operation, there is a channel reduction followed to reduce the result to the range \([0, b_i]\). In order to accelerate the channel reduction, pseudomesenne numbers of the form \( b_i = 2^w - d_i \), where \( d_i < 2^w/2 \), are selected as moduli. The channel reduction modulo \( b_i \) can then be performed by Algorithm 1.

If the Hamming weight of \( d_i \) is small, multiplications by \( d_i \) can also be replaced by a few additions.

The RNS is appealing because it is naturally suitable for parallel implementations. For all the basic operations (+, -, \times, /), computations between \( x_i \) and \( y_i \) have no dependency on other channels, which largely simplifies the parallelization of the operations. Fig. 1 shows the top level structure of an RNS multiplier. Note that each channel uses only local memory once the data is loaded, which also simplifies the communication and the scheduling.

Unfortunately, RNS multiplications are always modulo a composite number \( M_{28} \), and it can hardly be used immediately in cryptographic algorithms. For instance, ECC over \( \text{GF}(p) \) uses multiplications modulo a prime number. RSA implementations can make use of RNS when the factors of \( N \) are known. In order to work with the modulus which is prime or infeasible to factorize, one can combine the RNS representation with the Montgomery reduction algorithm [7], [8].

### 2.2 RNS Montgomery Algorithm

Algorithm 2 shows the Montgomery modular multiplication algorithm in RNS representation context. A new base, \( \mathcal{C} = \{c_1, c_2, \ldots, c_n\} \), where \( M_{\mathcal{C}} = \prod_{i=1}^{n} c_i \) is coprime with \( M_{28} \), is introduced to perform the division. Note that all the moduli from both \( \mathcal{B} \) and \( \mathcal{C} \) are pairwise coprime as \( M_{28} \) and \( M_{\mathcal{C}} \) are coprime. The overhead is two Base Extensions (BEs).

**Base Extension** To compute \( \{Q\}_{\mathcal{C}} = \{q_p, q_0, \ldots, q_0\} \) from \( \{Q\}_{28} = \{q_1, q_2, \ldots, q_n\} \), there are two main streams in literature: using CRT [7] and using Mixed Radix number System (MRS) [8]. Since the MRS method is difficult to implement in a full-parallel fashion (c.f. Section 6), we choose the CRT method [15]. Given \( \{X\}_{28} \), for (1), there must exist an integer \( \lambda < n \) such that

\[
\lambda \equiv \sum_{i=1}^{n} x_i \mod b_i.
\]

**Algorithm 1 Reduction modulo \((2^w - d_i)\)**

Require: \( x < 2^w \), \( w \), \( 0 < d_i < 2^w/2 \)

Ensure: \( r \equiv x \mod (2^w - d_i), \ r < 2^w \)

1. \( r \leftarrow (x \mod 2^w) + d_i \cdot \lfloor x/2^w \rfloor \)
2. \( r \leftarrow (r \mod 2^w) + d_i \cdot \lfloor r/2^w \rfloor \)
3. If \( r > 2^w \) then
4. \( r \leftarrow (r \mod 2^w) + d_i \)
5. end if
6. return \( r \)

**Algorithm 2 Montgomery Algorithm**

```
Algorithm 2
Reduction modulo \((2^w - d_i)\)

Require: \( x < 2^w \), \( w \), \( 0 < d_i < 2^w/2 \)
Ensure: \( r \equiv x \mod (2^w - d_i), \ r < 2^w \)
1: \( r \leftarrow (x \mod 2^w) + d_i \cdot \lfloor x/2^w \rfloor \)
2: \( r \leftarrow (r \mod 2^w) + d_i \cdot \lfloor r/2^w \rfloor \)
3: If \( r > 2^w \) then
4: \( r \leftarrow (r \mod 2^w) + d_i \)
5: end if
6: return \( r \)
```

### Table 1: Notations used in RNS Montgomery Algorithm

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>([b] \mod y )</td>
<td>The modulus w</td>
</tr>
<tr>
<td>([b]_w )</td>
<td>The bit-length of RNS moduli</td>
</tr>
<tr>
<td>(\mathcal{B} )</td>
<td>The first RNS base: ( {b_1, b_2, \ldots, b_n} )</td>
</tr>
<tr>
<td>(\mathcal{C} )</td>
<td>The second RNS base: ( {c_1, c_2, \ldots, c_n} )</td>
</tr>
<tr>
<td>(n )</td>
<td>The number of RNS moduli of ( \mathcal{B} ) or ( \mathcal{C} )</td>
</tr>
<tr>
<td>(B_i )</td>
<td>The constant defined by ( \mathcal{B} ): ( B_i = \prod_{j=1, j \neq i}^{n} b_j )</td>
</tr>
<tr>
<td>(C_i )</td>
<td>The constant defined by ( \mathcal{C} ): ( C_i = \prod_{j=1, j \neq i}^{n} c_j )</td>
</tr>
<tr>
<td>(B_{i}^{-1} )</td>
<td>( B_{i}^{-1} \mod b_i )</td>
</tr>
<tr>
<td>(C_{i}^{-1} )</td>
<td>( C_{i}^{-1} \mod c_i )</td>
</tr>
<tr>
<td>(B_{i,j} )</td>
<td>The constant to substitute ( B_i/C_i )</td>
</tr>
<tr>
<td>(B_{i,j}^{-1} )</td>
<td>( B_{i,j}^{-1} \mod b_i )</td>
</tr>
</tbody>
</table>

* We choose \( n w \geq \lceil \log_2 3P \rceil \). If lazy reduction is used, the dynamic range should be enlarged correspondingly.
such that:

\[
Q = \sum_{i=1}^{n} \xi_{Q,i} \cdot B_i \pmod{M_B} = \sum_{i=1}^{n} \xi_{Q,i} \cdot B_i - \lambda \cdot M_B
\]  

(3)

where \( \lambda \) can be calculated by the following equation:

\[
\lambda = \left[ \sum_{i=1}^{n} \xi_{Q,i} \cdot B_i \right] \div \left( \sum_{i=1}^{n} \xi_{Q,i} \div b_i \right)
\]  

(4)

In [9], \( \xi_{Q,i} / b_i \) is further approximated by \( \xi_{Q,i} / 2^w \) with error correction as \( b_i \) is of the form \( 2^w - d_i, d_i > 0 \). Once \( \lambda \) is obtained, \( \{ Q \} \epsilon = \{ q'_1, \ldots, q'_n \} \) can be computed as follows:

\[
q'_i = \left\lfloor \frac{Q}{\epsilon} \right\rfloor_{c_j} = \sum_{i=1}^{n} \xi_{Q,i} \cdot \left\lfloor \frac{B_i}{c_j} \right\rfloor - \lambda \cdot M_B \pmod{c_j}
\]  

(5)

The \( \left\lfloor \frac{B_i}{c_j} \right\rfloor \) and \( \left\lfloor \frac{M_B}{c_j} \right\rfloor, 1 \leq i, j \leq n \), can be precomputed once \( \mathcal{B} \) and \( \mathcal{E} \) are fixed.

Clearly, we need \( n \cdot w \cdot w \)-bit multiplications to calculate \( \xi_{Q,i} \) and \( n^2 \cdot w \cdot w \)-bit multiplications for all \( \sum_{i=1}^{n} \xi_{Q,i} \cdot \left\lfloor \frac{B_i}{c_j} \right\rfloor, 1 \leq j \leq n \). Therefore, each base extension uses \( n^2 + n \cdot w \cdot w \)-bit multiplications. In total, one modular multiplication uses \( 2n^2 + 4n \cdot w \cdot w \)-bit multiplications with precomputations [14].

3 BASE SELECTION SPECIFICATION

In this section, we provide the base selection specification to ensure faster BE operation.

3.1 The Observation on BE Operations

In our previous work [12], [16], we observe that the complexity of BE can be reduced if the moduli in the two bases are close to each other. The BE step, (5), can be written as a matrix-by-vector multiplication followed by channel reductions.

\[
\begin{pmatrix} [B_{1}]_{c_1} & \cdots & [B_{n}]_{c_1} \\ \vdots & \ddots & \vdots \\ [B_{1}]_{c_n} & \cdots & [B_{n}]_{c_n} \end{pmatrix} \begin{pmatrix} \xi_{Q,1} \\ \vdots \\ \xi_{Q,n} \end{pmatrix} - \lambda \begin{pmatrix} [M_B]_{c_1} \\ \vdots \\ [M_B]_{c_n} \end{pmatrix}
\]  

(6)

\[
\{ Q \} \epsilon \equiv \{ q'_1, \ldots, q'_n \} : = \{ [q'_1]_{c_1}, \ldots, [q'_n]_{c_n} \}
\]  

(7)

Note that the elements in the matrix, \( \left[ B_i \right]_{c_j}, 1 \leq i, j \leq n \), are constants and are generated as follows:

\[
[B_i]_{c_j} = \prod_{k=1,k\neq i}^{n} b_k
\]  

(8)

Define \( \tilde{B}_{i,j} := \prod_{k=1,k\neq i}^{n} (b_k - c_j) \). When \( b_k \) and \( c_j \) are close to each other, the difference \( b_k - c_j \) is small. Therefore, the bit-length of \( B_{i,j} \) can be much shorter than that of \( c_j \) if \( n \) is relatively small. Moreover, using \( \left[ B_i \right]_{c_j} \) or \( \tilde{B}_{i,j} \) makes no difference in the final results due to the channel reduction on the products. We denote by \( v \) the maximal bit-length of \( B_{i,j} \). Now the \( w \times w \) multiplications are substituted by \( v \times w \) multiplications, where \( v < w \).

Further Bit-length Reduction If all the \( 2n \) moduli are odd, \( B_{i,j} \) for \( 1 \leq i, j \leq n \) will predictably have \((n - 1)\) zeros in the Least Significant Bits (LSBs). Since \( b_k, c_j \) are all odd, \((b_k - c_j)\) can be divided by 2, and \( B_{i,j} \) is then divisible by \( 2^{n-1} \). In practice, \( B_{i,j} \) usually has more than \( n - 1 \) zero bits in LSBs.

To further reduce the operand size, we can truncate the least significant zeros of \( B_{i,j} \), and restore the correct multiplication results by a simple left-shift. We denote \( \tilde{B}_{i,j}' \) for the \( B_{i,j} \) after truncation, and \( v' \) for the maximal bit-length of \( \tilde{B}_{i,j}' \).

3.2 Base Selection Specification

A good base thus should satisfy:

- The maximal bit-length of \( B_{i,j} \) or \( \tilde{B}_{i,j}' \) (i.e. \( v \) or \( v' \)) is minimized;
- The channel reduction is efficient. For instance, \( b_i \) is a power of two or a pseudo-Mersenne number.

Previous practices, e.g. [9], [10], [11], tend to consider solely the second condition. The choice of \( n \), the number of moduli in a base, is normally determined after \( w \), the size of integer multipliers in use. For example, it seems natural to choose \( w = 18 \) or \( w = 34 \) for FPGA designs since modern FPGAs have 18-bit or 34-bit dedicated multipliers. In the following sections, we will show that this might not be the best choice.

4 COMPLEXITY ANALYSIS

We first show all the operations involved in the RNS Montgomery algorithm in Algorithm 3. The algorithm is mainly borrowed from [14] which employ precomputations to reduce the steps, and we modify it to make it suitable to utilize the \( v \)-bit operands.

1. The similar method can also apply to \( \left[ M_B \right]_{c_j} \).
2. The mechanism of notation also works for the BE from base \( \mathcal{E} \) to base \( \mathcal{B} \), i.e. the second BE in Algorithm 2.
Algorithm 3 RNS Montgomery modular multiplication with pre-computations

Require: RNS bases $B$ and $C$ with $M_B, M_C > 3P$, and $\epsilon$ determined by $B$ and $C$ [9].

Require: $\{X\}_{B}$ s.t. $X \equiv XM_B$ mod $P, X < 3P$

Require: $\{X\}_{C} = \{X\}_{B} \times C^{-1}_{i}$

Require: $\{Y\}_{B}$ s.t. $Y \equiv YM_B$ mod $P, Y < 3P$

Require: $\{Y\}_{C} = \{Y\}_{B} \times C^{-1}_{i}$

Precompute: $\{P^{-1}\}_{B} \equiv \{P^{-1}\}_{B} \times \{M_B\}_{B} \times \{B^{-1}\}_{B}$

Precompute: $\{C_iM_j\}_{C} \equiv \{M_B\}_{B} \times \{C_i\}_{C} \times \{C^{-1}_{i}\}_{C}$

Precompute: $\{PM^{-1}_{C}\}_{C} \equiv \{P\}_{C} \times \{M_B\}_{B} \times \{C_{i}^{-1}\}_{C}$

Ensure: $\{U\}_{B} \equiv UXM_B$ mod $P, U < 3P$

Ensure: $\{U'\}_{B} \equiv \{U\}_{B} \times \{C^{-1}_{i}\}_{C}$

1. $\{T\}_{B} = \{X\}_{B} \times \{Y\}_{B}$
2. $\{T'\}_{B} = \{X\}_{B} \times \{Y\}_{B}$
3. $\{\xi_Q\}_{B} = \{T\}_{B} \times \{B^{-1}\}_{B}$
4. $\{T''\}_{B} = \{T'\}_{B} \times \{C_i\}_{C}$
5. $\lambda = \sum_{j=0}^{2^n} B_{i,j} \cdot \xi_{Q,j}$
6. $\{Q\}_{j} = \sum_{j=0}^{2^n} B_{i,j} \cdot \xi_{Q,j} - \lambda \cdot \hat{M}_{B_{i,j}}$
7. $\{U\}_{C} = \{T''\}_{B} \times \{Q\}_{j} \times \{P\}_{C}$
8. $\lambda' = \epsilon + \sum_{j=0}^{2^n} \lambda_{j}$
9. $\{U'\}_{j} = \sum_{j=0}^{2^n} \alpha_{i,j} \cdot \lambda_{i,j}$
10. return $\{U\}_{B}$ and $\{U'\}_{j}$

Table 2 provides all the operations involved and their executed times in Algorithm 3. The most expensive operations are the $5n_w x w$-bit multiplications in Steps 1–4 and 7, and the $2n^2 w x w$-bit (or $v' x w$-bit) multiplications in Steps 6 and 9. We also consider the overhead of lazy reduction. Consider $\sum A_iB_i$ mod $P$, the complexity increases by $2n$ $w x w$-bit multiplications for every additional term. For conventional Montgomery, $n^2$ multiplications are added.

Furthermore, Table 2 provides the number of $m$-bit multiplications, where $m$ refers to the bit-length of the machine word. We assume the $w$-bit moduli are multi-precision. Denote $s$ and $t$ the numbers of machine words to represent a $w$-bit value and a $v$-bit one, respectively, i.e. $s = [w/m], t = [v/m]$. For a fair comparison, we provide two methods to perform the multi-precision multiplication: the textbook multiplication, and the Karatsuba multiplication [17].

If one employs the textbook multiplication, and takes the number of $m$-bit multiplications as the complexity metric, the complexity expression of one modulo-$P$ multiplication is:

$$2n^2 ts + 5ns^2 + 5ns + 4nt + 9n$$

$$2n^2 ts + 5ns^2$$

Notice that $\lambda \cdot \hat{M}_{B_{i,j}}$ or $\lambda \cdot \hat{M}_{C_{i,j}}$ can be carried out by an accumulator or a look-up table since $\lambda$ is only a few bits. Moreover, the multiplication by $d_i$ in the channel reduction (c.f. Algorithm 1) can be performed by additions if the Hamming weight of $d_i$ is small. If excluding these multiplications, the total number of $m$-bit multiplications for a modular multiplication is:

For comparison, the complexity of the digit-serial Montgomery algorithm is:

$$2n^2 s^2 + ns$$

Therefore, as long as $t < s - (5s - 1)/2n$, the complexity of the refined RNS Montgomery algorithm is less than that of the digit-serial Montgomery algorithm. If lazy reduction is available, the advantage of the refined RNS Montgomery algorithm will be more obvious. Interested readers can also derive the results for cases which use Karatsuba multiplication and/or count the number of additions.

5 RNS Parameter Selection

5.1 Selection of $n$, the Number of Moduli

The complexity of RNS Montgomery multiplication thus depends on $n$ and $v$. According to the experiments, we observe that $v$, and consequently $t$, are also mainly determined by $n$. The following second-order polynomial can be used to estimate $v$ and $v'$ (c.f. Section 5.3 and Fig. 3(a)).

$$\hat{v} = 0.1302n^2 + 4.2215n - 7.3003$$

$$\hat{v'} = 0.1174n^2 + 2.4932n - 4.7379$$

Therefore, the complexity can be considered as a function with the solely variable $n$. Given $P$ and $m$, we can easily enumerate all possible values of $n$ and choose the optimal. Fig. 2 shows the best choices of $n$ for different bit-length of $P$ on a 16-bit machine ($m = 16$). We use the number of 16-bit multiplications as the performance indicator, and it is calculated by (9). For each case, the envelope of the curve is convex, and we can always find an optimal $n$ (circled in the figure).

4. The number of machine words to represent an integer less than $P$ is approximately $ns$. 
### 5.2 Multiple-Plus-Prime (MPP)

The following theorem provides a deterministic way to construct base $\mathcal{B}$ and base $\mathcal{C}$. The proof is not included due to the paper length limitation.

**Theorem 1:** Let $\mathbb{P} = \{1, 2, 3, 5, 7, 11, 13, \ldots, \delta\}$, i.e. a set that includes 1 and the first $(2n - 1)$ primes. Let $\Theta := \prod_{i=1}^{2n} p_i$, $p_i \in \mathbb{P}$, and $M$ a multiple of $\Theta$, then all the $M + p_i$ are coprime pairwise.

Given $w$ and $n$, define $M := 2^w - [2^w]_\Theta$, then $M$ is a multiple of $\Theta$. If $[2^w]_\Theta - p_i < 2^{w/2}, \forall p_i \in \mathbb{P}$, then $M + p_i$ are pseudo-Mersenne numbers $(d_i = [2^w]_\Theta - p_i)$. Hence, the elements of $\mathcal{M} = \{M + p_i : p_i \in \mathbb{P}\}$ can then serve as the moduli in $\mathcal{B}$ and $\mathcal{C}$. This method is named Multiple-Plus-Prime (MPP) approach.

MPP approach is deterministic and the value of $B_{i,j}$ is determined by $n$ alone. Hence, it takes no intensive computations to generate bases. More importantly, MPP proves that there must exist a set of coprimes $\mathcal{M}$ consisting of $2n$ coprime moduli and $\max(\mathcal{M}) - \min(\mathcal{M}) < \varphi(2n)$, where $\varphi(2n)$ is the $2n$-th prime. However, $d_i = [2^w]_\Theta - p_i$ is not likely to be a small integer or with small Hamming weight when $n$ is large, and consequently, the channel reduction (Algorithm 1) cannot be performed efficiently. Therefore, MPP is only attractive when $n$ is small, and multiplications by $d_i$ are not performed by additions. A more practical method that leads to good RNS bases for larger $n$ is provided in the following section.

### 5.3 First Come, First Selected Improved (FCFS$^+$)

This method only selects pseudo-Mersenne moduli, thus guarantees an efficient channel reduction.

**Algorithm 4 First Come, First Selected Improved (FCFS$^+$) base selection algorithm**

1. $blacklist \leftarrow \emptyset$
2. repeat
3. $previous\_blacklist \leftarrow blacklist$
4. $i \leftarrow 1; T \leftarrow \varphi; n$
5. repeat
6. if $(2^w - i)$ is co-prime to all elements in $T$ and $(2^w - i) \notin blacklist$ then
7. $T \leftarrow T \cup \{2^w - i\}$
8. end if
9. $i \leftarrow i + 2; \backslash \backslash \$ even integers are excluded.
10. until $\#T = 2n$
11. for $j = 1 \rightarrow 2n$ do
12. $f_2 \leftarrow$ the second least prime factor of $T_j$ \footnote{If $T_j$ has no second least factor, $f_2 \leftarrow \infty$}
13. if $f_2 < (T_j - T_m)/2$ then
14. $blacklist \leftarrow blacklist \cup \{T_j\}$
15. end if
16. end for
17. until $previous\_blacklist = blacklist$
18. return $T$

Steps 4–10 of Algorithm 4 describes one straightforward way to select $2n$ odd coprime moduli by the “First Come, First Selected (FCFS)” principle: we search down from $2^w - 1$ until we get $2n$ coprimes, as the numbers near $2^w$ are pseudo-Mersenne numbers and often yield low Hamming weight.

The coprimes provided by the FCFS method are usually not very close to each other, as numbers which come first will affect the recruitment of the following numbers. For instance, $2^w - 7$ and $2^w - 11$, where

<table>
<thead>
<tr>
<th><strong>Steps</strong></th>
<th><strong>Operations</strong></th>
<th><strong>Operations</strong></th>
<th><strong># $m$-bit MUL/operation</strong></th>
<th><strong># $m$-bit MUL</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1–4, 7</td>
<td>$w \times w$-bit MUL</td>
<td>$5n$</td>
<td>$s^2$</td>
<td>$5ns^2$</td>
</tr>
<tr>
<td>6, 9</td>
<td>$u \times u$-bit MUL</td>
<td>$2n^2$</td>
<td>$ts$</td>
<td>$2nt^2 + 2nt + 2n$</td>
</tr>
<tr>
<td>6, 9</td>
<td>$\lfloor \log_2 n \rfloor \times \lfloor \frac{n}{2} \rfloor$-bit MUL $^+$</td>
<td>$2n$</td>
<td>$t + 1$</td>
<td>$2nt + 2n$</td>
</tr>
<tr>
<td>1–4, 7</td>
<td>$2w$-bit Reduction $^+$</td>
<td>$5n$</td>
<td>$s + 1$</td>
<td>$5ns + 5n$</td>
</tr>
<tr>
<td>6, 9</td>
<td>$(w + v + [ \log_2 n ])$-bit Reduction $^+$</td>
<td>$2n$</td>
<td>$t + 1$</td>
<td>$2nt + 2n$</td>
</tr>
<tr>
<td>7</td>
<td>$w$-bit Mod ADD/SUB</td>
<td>$n$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5, 8</td>
<td>$(w + [ \log_2 n ])$-bit ADD $^*$</td>
<td>$2n - 1$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6, 9</td>
<td>$(w + v + [ \log_2 n ])$-bit ADD</td>
<td>$2n^2$</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

$^+$ It is not counted when multiplications by $d_i$ is performed by additions (Algorithm 1), and we assume $d_i$ fits in one word.

$^*$ $[\log_2 n]$ is the bit-length of $\lambda$, and $\lfloor n/(n - 1) \rfloor$ the bit-length of $M_{i,j}$ or $M_j$.

This is for the approximation of $\lambda$, and even smaller bit-length (e.g. 8-bit) will work [9].
w = 64, are coprime, but none of them are coprime with 2^w − 1. Hence, for w = 64, it is better to exclude 2^w − 1 so that the selected coprimes are closer. We propose an improved version of FCFS (hence FCFS+) to perform the refinement.

Steps 12-15 examine whether T_j is blocking other candidates to be recruited. Let f_1, f_2 be the first and second least distinct prime factor of T_j, respectively (hence, f_1 < f_2). Therefore, two odd number, T_j − 2f_1, T_j − 2f_2, are not coprime with T_j, and hence excluded as T_j comes first. If T_j − 2f_2 > T_{2n}, then T_{2n} < T_j − 2f_2 < T_j − 2f_1 < T_j ≤ T_1. We can then blacklist T_j. In the next round (Steps 2–18), since T_j is blacklisted and not considered anymore, we may include two other numbers T_j − 2f_1 and T_j − 2f_2. Consequently, the element selected in the next iteration may get closer.

Note that FCFS+ does not guarantee the best base selection, as T_j − 2f_1 or/and T_j − 2f_2 may still have small factors and go into the blacklist. Nevertheless, FCFS+ generally provides closer moduli than FCFS.

Fig. 3(a) shows the value of v and v’ for moduli selected by FCFS+ and MPP for different n and w. According to it, v and v’ are generally determined by n, the number of moduli, and has little relation with w. Base on this experiment, we obtain (12) and (13) by the second-order polynomial curve-fitting.

Fig. 3(b) shows the average Hamming weight of all d_i. One can also develop the variants of Algorithm 4, e.g. to recruit the moduli from both sides of 2^w on the number axes (i.e. d_i can be 0, positive or negative). The average Hamming weights of the moduli from both sides of 2^w are also shown in Fig. 3(b) using dashed line. Generally speaking, the Hamming weight of d_i also increases with the growth of n.

5.4 RNS Parameter Selection Process

To summarize, we propose the following process to select parameters:

1) Emulate n from 2 to \([\log_2 3P]/m]\;
2) For each n value, compute the w, s, v, t, and the complexity index;
3) Select the optimal n and its corresponding w;
4) Select 2n coprimes using FCFS+ given n and w.

For n > \([\log_2 3P]/m]\, w will be less than m, and the platform will not be utilized efficiently. Therefore, for Step 1), we only need to emulate from 2 to \([\log_2 3P]/m]\.

For Step 2), w = \([\log_2 3P]/n]\, v or v’ is calculated by (12) or (13), s = \([w/m]\), t = \([v/m]\), and the complexity index is computed by (9), (10) or according to the user’s condition. Step 3) is described in Section 5.1, and 4) is discussed in Section 5.3. We use the following example to demonstrate the process.

256-bit Modular Multiplications on a 16-bit CPU

We first enumerate the n ranging from 2 to 16. Then we compute all the w, s, v, t values, and the complexity index values by (9) as shown in Fig. 2. The optimal n is 4 with w = 64, s = 4, t = 1.

Using Algorithm 4 (FCFS+), two rounds are performed. The details are shown in Table 3. In the 1st round, T_1 = 2^w − 1, which has f_1 = 3 and f_2 = 5, blocks T_1 − 2f_1 (= 2^w − 7) and T_1 − 2f_2 (= 2^w − 11). After blacklisting it, 2^w − 7 and 2^w − 11 are recruited in the 2nd round, and the moduli get closer. After 2nd round, the blacklist stays the same, and Algorithm 4 returns the moduli. We group the two bases and let L_{\mathcal{X}} to show the bit-length of all X_{i,j} in matrix format:

\[\mathcal{B} = \{2^w − 33, 2^w − 15, 2^w − 7, 2^w − 3\},\]
\[\mathcal{C} = \{2^w − 17, 2^w − 11, 2^w − 9, 2^w − 5\} .\]

\[L_\mathcal{B} = \begin{pmatrix} 10 & 10 & 7 & 6 \\ 9 & 10 & 9 & 7 \\ 9 & 10 & 10 & 8 \\ 9 & 9 & 12 & 9 \end{pmatrix}, \]
\[L_\mathcal{C} = \begin{pmatrix} 10 & 8 & 8 & 7 \\ 9 & 7 & 6 & 5 \\ 6 & 7 & 7 & 8 \\ 14 & 14 & 14 & 14 \end{pmatrix} .\]
TABLE 3: Example of base selection using FCFS+ for \( n = 4, w = 64 \)

<table>
<thead>
<tr>
<th>1st round</th>
<th>2nd round</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_1 )</td>
<td>( T_1 )</td>
</tr>
<tr>
<td>( 2^7 - 1 )</td>
<td>( 2^7 - 3 )</td>
</tr>
<tr>
<td>( 2^w - 1 )</td>
<td>( 2^w - 3 )</td>
</tr>
<tr>
<td>( 2^w - 3 )</td>
<td>( 2^w - 5 )</td>
</tr>
<tr>
<td>( 2^w - 9 )</td>
<td>( 2^w - 2 )</td>
</tr>
<tr>
<td>( 2^w - 9 )</td>
<td>( 2^w - 11 )</td>
</tr>
<tr>
<td>( 2^w - 11 )</td>
<td>( 2^w - 15 )</td>
</tr>
<tr>
<td>( 2^w - 15 )</td>
<td>( 2^w - 17 )</td>
</tr>
<tr>
<td>( 2^w - 17 )</td>
<td>( 2^w - 33 )</td>
</tr>
<tr>
<td>( 2^w - 33 )</td>
<td>( 2^w - 39 )</td>
</tr>
</tbody>
</table>

After zero truncation:

\[
L_{q_j} = \left( \begin{array}{cccc}
6 & 6 & 3 & 3 \\
4 & 5 & 4 & 4 \\
4 & 4 & 4 & 1 \\
3 & 3 & 6 & 6 \\
\end{array} \right), \quad L_{q'} = \left( \begin{array}{cccc}
5 & 3 & 5 & 2 \\
3 & 3 & 3 & 1 \\
2 & 3 & 4 & 4 \\
6 & 7 & 5 & 8 \\
\end{array} \right).
\]

Now we have:

\[ v = \max \{ L_{q_j}, L_{q'} \} = 14, \quad v' = \max \{ L_{q_j'}, L_{q'} \} = 8. \]

The size of \(|B_1|_{c_i}\) is reduced from 64-bit to 14-bit or 8-bit (from four 16-bit words to one word). Hence, the original \(64 \times 64\) multiplications can be replaced by \(14 \times 64\) ones or \(8 \times 64\) ones.

6 DISCUSSIONS

6.1 CRT versus MRS

When using MRS-based BE, the computation employs two phases: an RNS-to-MRS transform (Phase I) and an MRS-to-RNS transform (Phase II). For comparison reason, the CRT-based method can also be separated into 2 phases: Phase I to compute \( \xi \); and Phase II the matrix-by-vector multiplication. Table 4 summarizes the complexity of MRS-based BE along with the comparison with the CRT-based BE.

Phase I of MRS-based method is to transform the RNS representation of \( Q \), \( \{ q_n, q_{n-1}, \ldots, q_1 \} \), to \( \{ q_n, q_{n-1}, \ldots, q_1 \} \), where \( q_i < b_1 \) and satisfies the following equation:

\[
Q = \left[ \left( \begin{array}{c}
\bar{q}_n \\
\bar{q}_{n-1} \\
\vdots \\
\bar{q}_1 \\
\end{array} \right) \cdot \left( \begin{array}{c}
b_{n-1} \\
b_{n-2} \\
\vdots \\
b_1 \\
\end{array} \right) + \left( \begin{array}{c}
\bar{q}_n \\
\bar{q}_{n-1} \\
\vdots \\
\bar{q}_1 \\
\end{array} \right) \right] \cdot \tilde{b}_1 + \bar{q}_1 \quad (14)
\]

Fig. 4 shows the data flow of Phase I. Each element in Fig. 4 performs the pattern \( \bar{x}_j \leftarrow \left( \bar{x}_j - \bar{q}_i \right) \cdot \left| b_1^{-1} \right|_{b_j} \). Bajard, Meloni and Plantrat showed the multiplication by \( \left| b_1^{-1} \right|_{b_j} \) can be accelerated when \( b_i \) and \( b_j \) are close [18]. Note that \( \left| b_1^{-1} \right|_{b_j} = \left| b_i - b_j \right|^{-1} \). The main idea is to use the Montgomery reduction, which computes \( \left[ x \cdot \mathcal{R}_{q_i}^{-1} \right]_{b_i} \), to perform \( \left( \bar{x}_j - \bar{q}_i \right) \cdot \left| b_1^{-1} \right|_{b_j} \). The division by \( \left( b_i - b_j \right) \), which is required in the Montgomery reduction, is performed with Barrett’s algorithm [19]. However, the actual efficiency of this algorithm in hardware or software is not clear. The relatively complex structure of the Montgomery reduction algorithm may lead to more clock cycles, even if the size of the operands are small.

Bajard, Kaihara and Plantrat showed how to select moduli such that \( \left| b_i^{-1} \right|_{b_j} \) has low Hamming weight in Non-Adjacent Form (NAF) [20]. They suggested to choose moduli that have the form \( 2^k \), \( 2^k - 1 \) and \( 2^k - 2^l \pm 1 \). However, the Hamming weight of \( \left| b_1^{-1} \right|_{c_j} \) increases quickly with the growth of \( n \). Using the suggested parameters by [20], the maximal Hamming weight is 5 for \( n = 4 \), while it is 17 for \( n = 6 \). Note that for FCFS+ method, the maximal Hamming weight of \( B_{i,j}^\prime \) is 4 and 6 for \( n = 4 \) and \( n = 6 \), respectively.

Phase I of MRS-based BE, as shown in Fig. 4, has a triangle data-flow. As a result, it is difficult to parallelize. While for CRT-based BE, the computation of \( \xi \) for \( 1 \leq i \leq n \) can be performed in parallel. Note that one of the main reasons for using RNS is that it’s easy to parallelize.

Phase II is to compute \( \{ |Q|_{c_i}, |Q|_{c_{i+1}}, \ldots, |Q|_{c_n} \} \) from the MRS representation of \( Q \). Note that \( |Q|_{c_i} \) can be computed by involving \( (n-1) \) times the following operation: \( x_i' \leftarrow \left( x_i' \cdot b_j + \bar{q}_i \right)_{c_j} \). It can also be computed as an inner product of two vectors:

\[
|Q|_{c_i} = \left[ q_n, q_{n-1}, \ldots, q_1 \right] \cdot \left( \prod_{k=1}^{n-1} b_k \right)_{c_i} \cdot \left( \prod_{k=1}^{n-2} b_k \right)_{c_{i-1}} \cdot \ldots \cdot \left( \prod_{k=1}^{1} b_k \right)_{c_1} \cdot \left[ \begin{array}{c}
1 \\
1 \\
\vdots \\
1 \\
\end{array} \right]_{c_i} \quad (15)
\]
Note that the close moduli can also reduce the bitlength of of $|b_i|_{c_i}$ or $\prod_{k=1}^{n-1} b_k|_{c_k}$ in (15).

6.2 Implementations using Proposed Method

We used the proposed method to select parameters for pairing implementations on FPGA [12], [16]. The Cox-Rower architecture have been used in these works, and a 256-bit modulo operation is reduced from 19 cycles to 5 cycles on the similar hardware architecture.

There are two designs in [12] and they both choose \( n = 8 \). Design I uses the full-length multiplication for BE, and it needs 19 cycles for one modular reduction. Design II uses $B'_{i,j}$ (25-bit) to substitute $|B_i|_{c_i}$ (34-bit), and a modular reduction only takes 12 cycles. However, the optimal parameter is \( n = 4 \) for $[\log_2 P] = 256$ and \( m = 18 \) (the FPGA has pre-mounted 18-bit multipliers). Therefore, our following work [16] chooses \( n = 4 \) and \( w = 67 \), and one modular reduction only takes 6 cycles. With precomputations as Algorithm 3, it can be further reduced to 5 cycles on the same architecture as [16].

The implementation details are out of the scope of this paper. Nonetheless, the acceleration of pairing computations in [12], [16] has demonstrated the advantage of the refined parameter selection methods.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we observe that when moduli are close to each other, the complexity of base extensions using CRT can be reduced, and consequently, RNS Montgomery algorithm is accelerated. Based on the analysis of the refined RNS algorithm, we propose a systematic parameter selection procedure, which emphasizes on the selection of the optimal \( n \), the number of moduli. Subsequently, MPP and FCFS* methods are proposed to select the bases consisting of \( 2n \) close moduli. Our complexity analysis and the experimental results prove the advantage of the refined RNS base selection methods.

For the future work, we will investigate theoretical lower bound of the complexity. One possible way to find the optimal bases is considering the base selection as a weighted clique problem [21]. All pseudo-Mersenne numbers in range $[2^{w} - 2^{w/2}, 2^{w} + 2^{w/2}]$ are the vertices, and two vertices, say \( x_i \) and \( x_j \), are adjacent if they are coprime. The problem is thus to find a fixed-size clique with the minimum $\prod_{i \neq j} |x_i - x_j|$. We believe that MPP method provides a hint (or even a benchmark) on the distribution of the coprime moduli.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for the constructive comments. We also would like to thank Frederik Vercauteren, Nicolas Guillermin, and Hongbing Fan for the helpful discussions.

This work is partly supported by the Research Grant Council of the Hong Kong Special Administrative Region, China (Project No. CityU 123612) and CityU Strategic Research Grant (Project No. 7008185), and partly by the Flemish Government through FWO G.0550.12N and G.0130.13, and the Hercules Foundation AKUL/11/19.

REFERENCES


