Improved RNS Montgomery Modular Multiplication with Residue Recovery

Tao Wu, Shuguo Li and Litian Liu

Abstract Finite field arithmetic in residue number system (RNS) necessitates modular reductions, which can be carried out with RNS Montgomery algorithm. By transforming long-precision modular multiplications into modular multiplications with small moduli, the computational complexity has decreased much. In this work, two implementation methods of RNS Montgomery algorithm, *residue recovery* as well as *parallel base conversion*, are reviewed and compared. Then, we propose a new residue recovery method that directly employs binary system rather than mixed radix system to perform RNS modular multiplications. This improvement is appropriate for a series of long-precision modular multiplications with variant operands, in which it is more efficient than parallel base conversion method.

Keywords Montgomery algorithm · Residue number system · Binary system

T. Wu (✉)
Department of Microelectronics and Nanoelectronics, Tsinghua University,
Beijing 100084, People's Republic of China
e-mail: twu03ster@gmail.com

S. Li · L. Liu
Institute of Microelectronics, Tsinghua University,
Beijing 100084, People's Republic of China
e-mail: lisg@tsinghua.edu.cn

L. Liu
e-mail: liulitian@tsinghua.edu.cn

S. Patnaik and X. Li (eds.), *Proceedings of International Conference on Soft Computing Techniques and Engineering Application*, Advances in Intelligent Systems and Computing 250, DOI: 10.1007/978-81-322-1695-7_27,

1 Introduction

Modular multiplication is essential for popular Public Key cryptography that is defined in a finite field: RSA, Diffie-Hellman, and elliptic curve cryptography. In fact, a cryptosystem with enough security level often necessitates frequent large-operand modular multiplications.

As is known, additions, subtractions, and multiplications are parallel and carry-free in residue number system (RNS) [1–4]. Therefore, RNS arithmetic accelerates long-precision operations. However, modular reduction in RNS is harder than that in binary system.

An effective way to tackle RNS modular multiplication is to extend Montgomery algorithm from binary system to RNS [5–12]. Regular Montgomery algorithm breaks modular reductions in binary system into sequential additions and right shifts [13, 14], while the RNS Montgomery modular multiplication utilizes parallelism of RNS. As far as cryptography is concerned, the parallelism also provides natural immunity to side attacks, when the integral information is distributed into several channels. Besides, Phillips et al. [15] have proposed an algorithm directly based on the chinese remainder theorem, in which the constants are also represented in RNS.

In this paper, we will discuss the less-efficient RNS Montgomery algorithm with residue recovery [7, 10, 16], after an overview of the parallel base conversion method. It has been found that the usual residue recovery method can be built on binary system and RNS, without transformation into mixed radix system.

The remaining part of this paper is organized as follows: Sect. 2 reviews the RNS Montgomery algorithm with parallel base conversion; Sect. 3 discusses the RNS Montgomery algorithm with residue recovery; in Sect. 4, we propose our residue recovery method with binary system; and the last section concludes this paper.

2 RNS Montgomery Algorithm with Parallel Base Conversion

There are several approaches to implement RNS Montgomery modular multiplications, all of which obtain the quotients in parallel and then perform modular reduction of $M^{-1} \bmod P$ in an auxiliary RNS [5–9, 11, 17–19]. Among these methods, the parallel base conversion algorithm is the most efficient in computation.

2.1 Montgomery Algorithm

Usually, Montgomery algorithm is implemented in an interleaved form, and the multiplier is counted bit by bit while the multiplicand enters as a whole [14].

Algorithm 1 Montgomery algorithm [13]

Input: Integers A, B, and P satisfy $0 \le A < 2^n$, $0 \le B < P$, $2^{n-1} < P < 2^n$, and $\mathrm{GCD}(P, 2) = 1$. Let $R = 2^n$ and R^{-1} be the modular inverse of R modulo P. Then, $RR^{-1} - P\rho = 1$, or $\rho = (-P^{-1}) \bmod R$.

Output: $S \equiv A \cdot B \cdot R^{-1} \pmod{P}$, where $0 \le S < 2P$.

1: $T = A \cdot B$;
2: $Q = T\rho \bmod R$;
3: $S = (T + PQ)/R$;
4: **return** S.

2.2 RNS Montgomery Algorithm with Parallel Base Conversion

At Line 1, $P^{-1} = \mathrm{RNS}\left((p_1)_{m_1}^{-1}, (p_2)_{m_2}^{-1}, \ldots, (p_n)_{m_n}^{-1}\right)$.

At Line 3, $M^{-1} = \left(|M^{-1}|_{m_{n+1}}, |M^{-1}|_{m_{n+2}}, \ldots, |M^{-1}|_{m_{2n}}\right)$. There are two base conversions in such an RNS Montgomery modular multiplication: (1) Conversion of $Q = (q_1, q_2, \ldots, q_n)$ from Ω to $Q' = (q_{n+1}, q_{n+2}, \ldots, q_{2n})$ in Γ; (2) Conversion of $R' = (r_{n+1}, r_{n+2}, \ldots, r_{2n})$ from Γ to $R = (r_1, r_2, \ldots, r_n)$ in Ω.

Algorithm 2 RNS Montgomery algorithm with parallel base conversion [5, 7]

Input: The moduli for residue number system Ω are $\{m_1, m_2, \ldots, m_n\}$, with $M = \prod_{i=1}^{n} m_i$. Meanwhile, an auxiliary residue number system Γ are defined by the other moduli $\{m_{n+1}, m_{n+2}, \ldots, m_{2n}\}$, with $N = \prod_{i=n+1}^{2n} m_i$. Integers A, B, and P are represented in $\Omega \bigcup \Gamma$ as $A = (a_1, a_2, \ldots, a_{2n})$, $B = (b_1, b_2, \ldots, b_{2n})$, $P = (p_1, p_2, \ldots, p_{2n})$. Additionally, $A \cdot B < M \cdot P$, $2P < M < N$.

Output: $R \equiv A \cdot B \cdot M^{-1} (\bmod P)$ with $R < 2P$.

1: $Q = (A \times B) \times (-p^{-1}) : \Omega$;
2: $Q' \leftarrow Q : \Gamma \leftarrow \Omega$;
3: $R' = (A \times B + P \times Q') \times M^{-1} : \Gamma$;
4: $R \leftarrow R' : \Omega \leftarrow \Gamma$.
5: **return** $R \bigcup R' : \Omega \bigcup \Gamma$.

Set $M_j = M/m_j$, $\sigma_i = \left|q_i \left|M_i^{-1}\right|_{m_i}\right|_{m_i}$, and $G_{i,j} = \left|M_j\right|_{m_{n+i}}$, then

$$\begin{pmatrix} q_{n+1} \\ q_{n+2} \\ \vdots \\ q_{2n} \end{pmatrix} = \begin{pmatrix} G_{11} & G_{12} & \ldots & G_{1n} \\ G_{21} & G_{22} & \ldots & \vdots \\ \vdots & \vdots & \ddots & G_{n-1,n} \\ G_{n1} & G_{n2} & \ldots & G_{n,n} \end{pmatrix} \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_n \end{pmatrix} - \alpha \cdot \begin{pmatrix} |M|_{m_{n+1}} \\ |M|_{m_{n+2}} \\ \vdots \\ |M|_{m_{2n}} \end{pmatrix}, \quad (1)$$

where q_j is obtained as modulo m_j, and the integer factor α comes from the improved chinese remainder theorem [20]. Suppose that the moduli m_i have k binary bits, i.e., $2^{k-1} < m_i < 2^k$, for $i = 1, 2, \ldots, n$. With $M_i = M/m_i$, the chinese remainder theorem can be written as

$$\begin{aligned} x &= \left| \sum_{i=1}^{n} M_i \left| x_i M_i^{-1} \right|_{m_i} \right|_M = \sum_{i=1}^{n} M_i \left| x_i M_i^{-1} \right|_{m_i} - \alpha M \\ &= \sum_{i=1}^{n} M_i \sigma_i - \alpha M, \end{aligned} \tag{2}$$

where $\sigma_i = \left| x_i M_i^{-1} \right|_{m_i}$, and the indefinite factor α can be fixed by an approximation method [5, 8] or by an extra modulus [7, 20].

3 RNS Montgomery Algorithm with Residue Recovery

Although a direct map of Montgomery algorithm to RNS suffers from a problem in losing residues, it is a good guide or bridge for subsequent algorithms.

Algorithm 3 Direct map of Montgomery algorithm in residue number system [16]

Input: The integers A, B, and P are represented with residue number system Ω: $\{m_1, m_2, \ldots, m_n\}$, with $M = \prod_{i=1}^{n} m_i$ and $M_i = M/m_i$. $A = (a_1, a_2, \ldots, a_n)$, $B = (b_1, b_2, \ldots, b_n)$, and $P = (p_1, p_2, \ldots, p_n)$. Meanwhile, A is also expressed in the mixed radix system as $A = a_1 + a_2 \cdot m_1 + \cdots + a_n \cdot \prod_{i=1}^{n-1} m_i$. In addition, $0 \le A < \frac{m_n - 1}{2} \cdot \frac{M}{m_n}$, $0 \le B < 2P$, $0 < P < \frac{M}{\max_{1 \le i \le n}\{m_i\}}$.

Output: $R = A \cdot B \cdot M^{-1} = \mathrm{RNS}(r_1, r_2, \ldots, r_n)$.

1: $R := (0, 0, \ldots, 0)$;
2: **for** $i = 0$ to $n - 1$ **do**
3: $q_i' := (r_i + a_i' \cdot b_i)(m_i - p_i)_i^{-1} \bmod m_i$;
4: $R := R + a_i' \cdot B + q_i' \cdot P$;
5: $R := R \div m_i$;
6: **end for**
7: **return** R.

3.1 Direct Map of Montgomery Algorithm into RNS

At Line 3, we make sure that

$$(R + a_i' \cdot B + q_i' \cdot P) \bmod m_i = 0. \tag{3}$$

Given that $R \bmod m_i = r_i$, $B \bmod m_i = b_i$, and $P \bmod m_i = p_i$, we have

$$\begin{aligned} R + a_i' \cdot B + q_i' \cdot P &\equiv r_i + a_i' \cdot b_i + q_i' \cdot p_i \\ &= (r_i + a_i' b_i) + p_i \cdot (r_i + a_i' b_i)(m_i - p_i)^{-1} \bmod m_i \\ &\equiv (r_i + a_i' b_i)\Big(1 - (m_i - p_i)(m_i - p_i)^{-1}\Big) \\ &= 0 (\bmod\ m_i). \end{aligned}$$

Therefore, at Line 4 of the ith iteration, R is a multiple of m_i. At the end of all iterations, as is shown in [7, 16], we have

$$R = \frac{1}{m_1 \cdot m_2 \cdots m_n} \left(B \cdot \sum_{i=1}^{n} a_i' \prod_{j=1}^{i-1} m_i + P \cdot \sum_{i=1}^{n} q_i' \prod_{j=1}^{i-1} m_i \right). \tag{4}$$

Notice that $M = m_1 \cdot m_2 \cdots m_n$ and $A = \sum_{i=1}^{n} a_i' \prod_{j=1}^{i-1} m_i$, the above equation yields $R \equiv A \cdot B \cdot M^{-1} (\bmod P)$. At the last step of each loop, R will keep below $3P$, which can be examined by induction [16]. Given that $R_{n-1} < 3P$ and $A < \frac{m_n - 1}{2} \cdot \frac{M}{m_n}$, it should yield [16] $R_n < 2P$.

However, there is a problem hidden at line L5: The residue r_i gets lost when $r_i \div m_i$ appears [7]. In RNS, r_i / m_j is equivalent to $r_i \times (m_j)_{m_i}^{-1} \bmod m_i$. However, $(m_i)_{m_i}^{-1}$ does not exist because of $m_i \times (m_i)_{m_i}^{-1} \equiv 0 (\bmod\ m_i)$ (The product of an integer and its inverse modular m_i should be equal 1 modulo m_i). Therefore, one more residue becomes meaningless after each cycle of the loop.

3.2 RNS Montgomery Algorithm with Residue Recovery

With the improved chinese remainder theorem [20], the lost residue can be recovered by the help of the redundant residue r_{n+1}. At Line 4,

Algorithm 4 RNS Montgomery algorithm with residue recovery [16]

Input: The group of moduli: $\{m_1, m_2, \ldots, m_n\} \bigcup \{m_{n+1}\}$ defines the residue number system Ω, with $M = \prod_{i=1}^{n} m_i$ and $M_i = M/m_i$. A is represented in mixed radix system: $A = a_1' + \sum_{i=2}^{n} a_i' \prod_{j=1}^{i-1} m_j$. Meanwhile, $B = (b_1, b_2, \ldots, b_n, b_{n+1})$ and $P = (p_1, p_2, \ldots, p_n, p_{n+1})$. Also, $A < \frac{m_n - 1}{2} \cdot \frac{M}{m_n}$, $B < 2P$.

Output: $R \equiv A \cdot B \cdot M^{-1} (\bmod P)$.

1: $R := (0, 0, \ldots, 0)$;
2: **for** $i = 1$ to n **do**
3: $q_i' := (r_i + a_i' \cdot b_i) \big| (m_i - p_i)^{-1} \big|_{m_i} \bmod m_i$;
4: $R' := R + a_i' \cdot B + q_i' \cdot P$;
5: $R := R' \div m_i$;
6: $r_i := \text{restoreRNS}(r_1, \ldots, r_{i-1}, r_{i+1}, \ldots, r_{n+1})$;
7: **end for**
8: **return** R.

$$R + a_i' \cdot B + q_i' \cdot P = (r_1, \ldots, r_{n+1}) + a_i' \cdot (b_1, \ldots, b_{n+1}) + q_i' \cdot (p_1, \ldots, p_{n+1})$$
$$= \left(\left| r_1 + a_i' \cdot b_1 + q_i' \cdot p_1 \right|_{m_1}, \ldots, \left| r_{n+1} + a_i' \cdot b_{n+1} + q_i' \cdot p_{n+1} \right|_{m_{n+1}} \right).$$

At Line 5, $R \div m_i$ is conducted for $j \neq i, j \in \{1, 2, \ldots, n+1\}$ by multiplying the modular inverse, i.e., $r_j := r_j \cdot |m_i^{-1}|_{m_j} \bmod m_j$ for $j \neq i$.

At Line 6, the function 'restore RNS' just restores the lost residue r_i at Line 5. Set $\sigma_{i,j} = \left| r_j \cdot \left| \left(\frac{M_{i,i}}{m_j} \right)^{-1} \right|_{m_j} \right|_{m_j}$, then r_i is obtained by the improved chinese remainder theorem as follows:

$$r_i = \left| \sum_{j=1, j \neq i}^{n} \left| \sigma_{i,j} \cdot \left| \frac{M_i}{m_j} \right|_{m_i} \right|_{m_i} - \left| \gamma_i \cdot |M_i|_{m_i} \right|_{m_i} \right|_{m_i}, \tag{5}$$

where

$$\gamma_i = \left| |M_i^{-1}|_{m_{n+1}} \cdot \left(\sum_{j=1, j \neq i}^{n} \left| \sigma_{i,j} \cdot \left| \frac{M_i}{m_j} \right|_{m_{n+1}} \right|_{m_{n+1}} - r_{n+1} \right) \right|_{m_{n+1}}. \tag{6}$$

Taking the computation of a basic modular multiplication with m_i as the unit, i.e., $a_i \cdot b_i \bmod m_i$, then there are $(6n + 5)$ modular multiplications at each iteration of the above algorithm. Then, a total Montgomery modular multiplication with residue recovery costs $6n^2 + 5n$ basic modular multiplications.

4 Proposed Algorithm with Residue Recovery

The idea of RNS Montgomery algorithm is to keep the final result below some constants such as $2P$ with parallel or sequential modular reductions. The aforementioned residue recovery method performs sequential k-bit modular multiplications, in which one operand should be represented in mixed radix system. However, it is not convenient for conversions among binary system, RNS, and mixed radix system. Therefore, we propose a new residue recovery algorithm that avoids the use of mixed radix system.

4.1 Initial Algorithm

Algorithm 5 Proposed RNS Montgomery algorithm with residue recovery I

Input: The residue number system Ω has a group of moduli: $\{m_1, m_2, \ldots, m_n\} \bigcup \{m_{n+1}\}$, where $m_1 = 2^t$, $2^{k-1} < m_i < m_j < 2^k$ for $2 \le i < j \le n+1$, and the integer $t \le k-1$. All the moduli are relatively prime, with $M = \prod_{i=1}^{n} m_i$ and $M_i = M/m_i$. Also, $A = \sum_{i=1}^{n} A_i \cdot 2^{(i-1)t}$, where $0 \le A_i \le 2^t - 1$ and $A < 2^{nt-1}$. B and P are represented in residue number system: $B = (b_1, b_2, \ldots, b_n, b_{n+1})$, and $P = (p_1, p_2, \ldots, p_n, p_{n+1})$, with $T = 2^{nt}$, $A < T$, $P < T$, and $B < 2P$.

Output: $R \equiv A \cdot B \cdot T^{-1} (\mathrm{mod}\ P)$, where $0 \le R < 2P$.

1: $R := (0, 0, \ldots, 0)$;
2: **for** $i = 1$ to n **do**
3: $q_i' := (r_1 + A_i \cdot b_i) \left| (m_1 - p_1)^{-1} \right|_{m_1} \mathrm{mod}\ m_1$;
4: $R' := R + A_i \cdot B + q_i' \cdot P$;
5: $R := R' \div m_1$;
6: $r_1 := \mathrm{restoreRNS}(r_2, r_3, \ldots, r_{n+1})$;
7: **end for**
8: **return** R.

With $i = 1$,

$$\begin{aligned} R &= (A_1 \cdot B + q_1' \cdot P)/m_1 \\ &< (2^t - 1)B + (2^t - 1)P/2^t \\ &< 3P. \end{aligned}$$

Assuming $R < 3P$ at the end of the ith iteration, then the $(i+1)$-th iteration yields

$$R < (3P + (2^t - 1)2P + (2^t - 1)P)/2^t = 3P.$$

Then, the induction shows that $R < 3P$ at last. However, we expect to get the final result within a smaller range $[0, 2P)$. As

$$\begin{aligned} R &< (3P + A_n \cdot B + q_n' \cdot P)/2^t \\ &\le (3P + A_n \cdot 2P + (2^t - 1)P)/2^t, \end{aligned}$$

we can set

$$(3P + A_n \cdot 2P + (2^t - 1)P)/2^t < 2P.$$

The above equation yields $A_n < 2^{t-1} - 1$. Notice that $A - A_n \cdot 2^{(n-1)t} < 1 \cdot 2^{(n-1)t}$, then as long as $A < (A_n + 1) \cdot 2^{(n-1)t} < 2^{nt-1}$, there will be $R < 2P$.

At Line 6, we have

$$r_1 = \left| \sum_{j=2}^{n} \left| \sigma_{1,j} \cdot \left| \frac{M_1}{m_j} \right|_{m_1} \right|_{m_1} - \left| \gamma_1 \cdot |M_1|_{m_1} \right|_{m_1} \right|_{m_1}, \tag{7}$$

where $\sigma_{1,j} = \left| r_j \cdot \left| \left(\frac{M_1}{m_j} \right)^{-1} \right|_{m_j} \right|_{m_j}$, and

$$\gamma_1 = \left| |M_1^{-1}|_{m_{n+1}} \cdot \left(\sum_{j=2}^{n} \left| \sigma_{1,j} \cdot \left| \frac{M_1}{m_j} \right|_{m_{n+1}} \right|_{m_{n+1}} - r_{n+1} \right) \right|_{m_{n+1}}. \tag{8}$$

4.2 Improved Algorithm

While Algorithm 5 is more efficient than Algorithm 4, it still requires more sequential steps to compute one RNS Montgomery modular multiplication than parallel base conversion method with Algorithm 2. However, we found out that a little revision will double the efficiency of our proposal, which is shown in Algorithm 6.

At first glance, Algorithm 6 differs from Algorithm 5 in the choice of t and m_{n+1}. By setting $t = 2k$ rather than $t = k$, the number of loops is reduced by a half, while the computational complexity only increases a little. The range of m_1 then gets much larger than 2^k and other RNS moduli, but it does not impact the validity of the algorithm.

By setting $m_{n+1} = 2^s - 1$, then the modular reduction in $x \cdot y \mod m_{n+1}$ can be simplified. Take Eq. 6 as example, set $x = \sigma_{i,j} < 2^k$, $y = \left| \frac{M_i}{m_j} \right|_{m_{n+1}} < 2^s$. Since $k \gg s$, one can set $l = \lceil k/s \rceil$, and there will be $x = x_{l \cdot s-1..0}$, with $x_i = 0$ for $i \geq k$. As $2^{j \cdot s} \bmod (2^s - 1) = 1$, then

$$x \mod m_{n+1} = \sum_{j=1}^{l} x_{j \cdot s-1 \ldots j \cdot s-s} \cdot 2^s \bmod (2^s - 1) = \sum_{j=1}^{l} x_{j \cdot s-1 \ldots j \cdot s-s}.$$

Furthermore, the additions modulo $2^s - 1$ can be simplified by setting the highest carry out as the lowest carry in. Assuming that $x' = x \bmod m_{n+1}$, then $x \cdot y \bmod m_{n+1} = x' \cdot y \bmod (2^s - 1)$ can be computed by one $s \times s$-bit multiplication and one s-bit addition.

The sequential steps and computational complexity of Algorithm 6 can be measured as follows:

Algorithm 6 Proposed RNS Montgomery algorithm with residue recovery II

Input: RNS $\Omega : \{m_1, m_2, \ldots, m_n\} \bigcup \{m_{n+1}\}$, where $t = 2k$, $m_1 = 2^t$, $2^{k-1} < m_i < m_j < 2^k$ **for** $2 \le i < j \le n$, $m_{n+1} = 2^s - 1$, $s \ll k$. $\mathrm{GCD}(m_i, m_j) = 1$ for $i \ne j$. $M = \prod_{i=1}^{n} m_i$, $M_i = M/m_i$. $n' = \lceil n/2 \rceil$, $A = \sum_{i=1}^{n'} A_i \cdot 2^{(i-1)t}$, where $0 \le A_i \le 2^t - 1$ and $A < 2^{n't-1}$. B and P are represented in residue number system: $B = (b_1, b_2, \ldots, b_n, b_{n+1})$, and $P = (p_1, p_2, \ldots, p_n, p_{n+1})$, with $T = 2^{2n't}$, $A < T$, $P < T$, and $B < 2P$.

Output: $R \equiv A \cdot B \cdot T^{-1} (\mathrm{mod}\ P)$, where $0 \le R < 2P$.

1: $R := (0, 0, \ldots, 0)$;
2: for $i = 1$ to n' **do**
3: $R' := R + A_i \cdot B$;
4: $q_i' := q_1' \cdot \left|(m_1 - p_1)^{-1}\right|_{m_1} \mod m_1$;
5: $R'' := R' + q_i' \cdot P$;
6: $R := R'' \div m_1$;
7: $r_1 := \mathrm{restoreRNS}(r_2, r_3, \ldots, r_{n+1})$;
8: **end for**
9: **return** R.

1. Assuming that the complexity of a k-bit modular multiplication is 1, and the complexity of a $k \times k$ multiplication is 1/3. As is well known, modular multiplication by Montgomery algorithm and Barrett modular reduction requires three multiplications. In addition, the computational complexity of a $k \times 2k$ multiplication is 2/3, and the complexity of an s-bit modular multiplication is $\varepsilon, \varepsilon \ll 1$.
2. Neglecting the complexity of modular additions and modular subtractions, since it is small compared with modular multiplications.
3. The computation of $A_i \cdot B$ requires one sequential modular multiplication, and the total complexity is $(n + \varepsilon)$ k-bit modular multiplications.
4. The determination of q_i' requires 2/3 sequential modular multiplication, and the total complexity is just 2/3. At Line 4, the value $(m_1 - p_1)^{-1}$ can be precomputed.
5. The computation of $q_i' \cdot P$ requires 5/3 sequential modular multiplications, and the total complexity is $(5n/3 + 2\varepsilon)$ k-bit modular multiplications. The increased computational complexity of 2/3 and ε corresponds to the modular reduction in q_i' to k-bit and s-bit moduli m_i, $i = 2, 3, \ldots, n + 1$.
6. The computation of $R_i'' \div m_1$ can be performed by modular multiplications of $m_1^{-1} = 2^{-t}$ modulo m_i. It requires one sequential modular multiplication, and the total complexity is $(n + \varepsilon)$ k-bit modular multiplications.
7. As are shown in Eqs. (5) and (6), the recovery of residue r_1 after division of m_1 requires $(2/3 + \varepsilon)$ sequential modular multiplication, and the total complexity is $(2n/3 + 2\varepsilon)$ k-bit modular multiplications. With $i = 1$ in Eq. 5, modular reduction over m_1 can be performed by truncation instead of multiplications, which leads to the complexity of 2/3 rather than 1.

In total, Algorithm 6 merely needs

$$\begin{aligned} K &= (1 + 2/3 + 5/3 + 1 + (2/3 + \varepsilon)) \cdot n' \\ &= (15/3 + \varepsilon) \cdot n' \\ &\approx (2.5 + \varepsilon)n \end{aligned} \tag{9}$$

sequential modular multiplications, and the total computational complexity is about

$$\begin{aligned} N &= ((n + \varepsilon) + 2/3 + (5n/3 + 2\varepsilon) + (n + \varepsilon) + (2n/3 + 2\varepsilon)) \cdot n' \\ &= (13n/3 + (2/3 + 6\varepsilon))n' \\ &\approx (13n/6 + 1/3 + 3\varepsilon)n \end{aligned} \tag{10}$$

k-bit modular multiplications.

4.3 Comparison with Parallel Conversion Method

By contrast, the parallel base conversion method in Algorithm 2 will need about $2n + 8$ sequential k-bit modular multiplications [7, 15], and the total computational complexity is about $(2n^2 + 10n + 4)$ k-bit modular multiplications. While this result seems better than our proposals, it should be noticed that all the inputs of Algorithm 2 should be in residue number system. If only one input lies in RNS, the other should be transformed from binary number system to RNS [8], then the parallel base conversion method will require $3n + 8$ sequential k-bit modular multiplications and $(3n^2 + 9n + 4)$ k-bit modular multiplications. Therefore, if one input of RNS modular multiplication keeps in binary number system, our proposed Algorithm 6 will be better than parallel base conversion method. The above comparison is shown in Table 1, where 'Seq. Num. Mod-mult' denotes sequential number of k-bit modular multiplications and 'Tot. Num. Mod-mult' denotes the total number of k-bit modular multiplications.

If there are a series of modular multiplications by variant long-precision integers, i.e., $\tau_1, \tau_2, \ldots, \tau_h$, with h be some regular index. The modulus is denoted as P. All τ_i and P are long-precision binary numbers, e.g., between $(2^{L-1}, 2^L]$, with $L \gg 1$. Then, we intend to compute $R = \prod_{i=1}^{h} \tau_i \bmod P$.

We just need τ_1 and P in RNS $\Omega : \{m_1, m_2, \ldots, m_n\}$, where $\tau_1 = U_{\text{rns}}^{(0)} = (u_1, u_2, \ldots, u_n)$, $P = (p_1, p_2, \ldots, p_n)$. By the new residue recovery method 'MR'(Algorithm 6), one gets

$$U_{\text{rns}}^{(1)} = \tau_1 \cdot \tau_2 \cdot T^{-1} \bmod P \equiv \text{MR}(U_{\text{rns}}^{(0)}, \tau_2)(\bmod P), \tag{11}$$

Table 1 Comparison of proposed residue recovery method to parallel base conversion method, with one input in RNS and the other in binary number system

Method	Residue recovery (this work)	Parallel conversion [7]
Seq. num. mod-mult	$(2.5+\varepsilon)n$	$3n+8$
Tot. num. mod-mult	$(13n/6+1/3+3\varepsilon)n$	$3n^2+9n+4$

$$U_{\text{rns}}^{(2)} = \prod_{i=1}^{3} \tau_i \cdot T^{-2} \bmod P \equiv \text{MR}(U_{\text{rns}}^{(1)}, \tau_3)(\text{mod}P), \tag{12}$$

$$\ldots, \quad \ldots,$$

$$U_{\text{rns}}^{(h-1)} = \prod_{i=1}^{h} \tau_i \cdot T^{-(h-1)} \bmod P \equiv \text{MR}(U_{\text{rns}}^{(h-2)}, \tau_h)(\text{mod } P). \tag{13}$$

At the last step, there is

$$U_{\text{rns}}^{(h-1)} = \left(\prod_{i=1}^{h} \tau_i \cdot T^{-(h-1)} \right) \bmod P.$$

Suppose that $c = T^h \bmod P$ has been precomputed, then

$$R = \prod_{i=1}^{h} \tau_i = \text{MR}(U_{\text{rns}}^{(h-1)}, c).$$

If it is necessary, R can also be converted from RNS to binary system by the chinese remainder theorem:

$$R = \sum_{i=1}^{n} M_i \left| r_i M_i^{-1} \right|_{m_i} - \alpha M, \tag{14}$$

where r_i is the ith component of R, and α is obtained with the aforementioned approximation method [8].

Nevertheless, the proposed method is not less efficient than parallel conversion method [7] for modular exponentiation, in which the computational complexity can be greatly decreased in the latter since only one RNS-to-binary conversion with the base number is enough.

5 Conclusion

Long-precision modular multiplications can be performed in RNS with RNS Montgomery algorithm, for which there are mainly two implementation methods: parallel base conversion as well as residue recovery. The first method performs modular reductions with respect to the dynamic range in parallel, and base

conversions between two RNSs are required. The second method, however, performs modular reduction in a number of sequential steps and requires more sequential steps.

Then, we propose a new residue recovery method that is based on binary system rather than mixed radix system, which is supposed to suit modular multiplications of many various large integers. In this case, it is more efficient than parallel base conversion method since little binary-to-RNS conversion is required.

Acknowledgments The authors would like to thank the editor and the reviewers for their comments. This work was partly supported by the National High Technology Research and Development Program of China (No.2012AA012402), the National Natural Science foundation of China (No.61073173), and the Independent Research and Development Program of Tsinghua University (No. 2011Z05116).

References

1. Soderstrand, M., Jenkins, W., Jullien, G., Taylor, F. (eds.): Residue Number System Arithmetic: Modern Applications in Signal Processing, pp. 1–185, IEEE Press, New York (1986)
2. Mohan, P.A.: Residue Number Systems: Algorithms and Architectures. Kluwer Academic Publishers, Boston (2002)
3. Wu, A.: Overview of Residue Number Systems. National Taiwan University, Taipei (2002)
4. Taylor, F.: Residue arithmetic: a tutorial with examples. Computer **17**(5), 50–62 (1984)
5. Posch, K., Posch, R.: Modulo reduction in residue number system. IEEE Trans. Parallel Distrib. Syst. **6**, 449–454 (1995)
6. Ciet, M., Neve, M., Peeters, E., Quisquater, J.: Parallel FPGA implementation of RSA with residue number systems-can side-channel threats be avoided? In: 46th IEEE International Midwest Symposium on Circuits and Systems, vol. 2. pp. 806–810 (2003)
7. Bajard, J., Didier, L., Kornerup, P.: An RNS Montgomery modular multiplication algorithm. IEEE Trans. Comput. **47**, 766–776 (1998)
8. Kawamura, S., Koike, M., Sano, F., Shimbo, A.: Cox-rower architecture for fast parallel Montgomery multiplication. In Preneel, B. (ed.) Advances in Cryptology-EuroCrypt'00. Volume 1807 of Lecture Notes in Computer Science, pp. 523–538, Springer-Verlag, Berlin (2000)
9. Nozaki, H., Motoyama, M., Shimbo, A., Kawamura, S.: Implementation of RSA algorithm based on RNS Montgomery modular multiplication. In: Third International Workshop on Cryptographic Hardware and embedded systems. Volume 2162 of Lecture Notes in Computer Science, pp. 364–376. Springer, Berlin (2001)
10. Bajard, J., Didier, L., Kornerup, P.: Modular multiplication and base extensions in residue number systems. In: 15th IEEE Symposium on Computer Arithmetic, pp. 59–65 (2001)
11. Bajard, J., Imbert, L.: A full RNS implementation of RSA. IEEE Trans. Comput. **53**, 769–774 (2004)
12. Bajard, J., Imbert, L., Liardet, P., Yannick, T.: Leak resistant arithmetic. In: Cryptographic Hardware and Embedded Systems (CHES 2004). Volume 3156 of Lecture Notes in Computer Science, pp. 62–75 Springer, Berlin (2004)
13. Montgomery, P.: Modular multiplication without trial division. Math. Comput. **44**, 519–521 (1985)
14. Orup, H.: Simplifying quotient determination in high-radix modular multiplication. In: 12th IEEE Symposium on Computer Arithmetic, pp. 193–199 (1995)

15. Phillips, B., Kong, Y., Lim, Z.: Highly parallel modular multiplication in the residue number system using sum of residues reduction. Appl. Algebra Eng. Commun. Comput. **21**, 249–255 (2010)
16. Bajard, J., Didier, L., Kornerup, P.: An RNS Montgomery modular multiplication algorithm. In: 13th IEEE Sympsoium on Computer Arithmetic, pp. 234–239 (1997)
17. Yang, T., Dai, Z., Yang, X., Zhao, Q.: An improved RNS Montgomery modular multiplier. In: 2010 International Conference on Computer Application and System Modeling (ICCASM 2010), Vol. 10, pp. 144–147
18. Guillermin, N.: A high speed coprocessor for elliptic curve scalar multiplications over $\mathbb{F}_p$. In: Cryptographic Hardware and Embedded Systems, CHES 2010. Vol. 6225 Lecture Notes in Computer Science, pp. 48–64 (2010)
19. Wu, T., Liu, L.: Elliptic curve point multiplication by generalized Mersenne numbers. J. Electro. Sci. Technol **10**(3), 199–208 (2012)
20. Shenoy, A., Kumaresan, R.: Fast base extension using a redundant modulus in RNS. IEEE Trans. Comput. **38**, 292–297 (1989)