Journal of Cryptographic Engineering (2019) 9:313–331
https://doi.org/10.1007/s13389-018-0195-8

REGULAR PAPER

RNS Montgomery reduction algorithms using quadratic residuosity

Shinichi Kawamura[1,2] · Yuichi Komano[1] · Hideo Shimizu[1] · Tomoko Yonemura[1]

Received: 10 September 2017 / Accepted: 22 August 2018 / Published online: 3 September 2018

Abstract

The residue number system (RNS) is a method for representing an integer as an n-tuple of its residues with respect to a given base. Since RNS has inherent parallelism, it is actively researched to implement a faster processing system for public-key cryptography. This paper proposes new RNS Montgomery reduction algorithms, Q-RNSs, the main part of which is twice a matrix multiplication. Letting n be the size of a base set, the number of unit modular multiplications in the proposed algorithms is evaluated as $(2n^2+n)$. This is achieved by posing a new restriction on the RNS base, namely, that its elements should have a certain quadratic residuosity. This makes it possible to remove some multiplication steps from conventional algorithms, and thus the new algorithms are simpler and have higher regularity compared with conventional ones. From our experiments, it is confirmed that there are sufficient candidates for RNS bases meeting the quadratic residuosity requirements.

Keywords Residue number system · Montgomery reduction · Quadratic residuosity · Cryptography

1 Introduction

The residue number system (RNS) is a method for representing an integer in which a given integer x is represented by its residues divided by a base of integers, which are pairwise co-prime. If we denote the base by $B = \{m_1, m_2, \ldots, m_n\}$ and the RNS representation of x as $[x_1, x_2, \ldots, x_n]$, it holds that $x_i = x \bmod m_i$. The main feature of RNS is that addition, subtraction, and multiplication are carried out by independent addition, subtraction, and multiplication with respect to each base element. The operation flow at each base element is called a channel. If each channel has a processing unit, an n-fold speed increase can be achieved, as compared with the case with a single processing unit. This parallelism seems attractive in pursuing efficient computation of public-key cryptography, which is constructed by integer operations of several hundred or several thousand bits with a modular reduction. However, modular reduction in RNS was not easy to carry out before it was replaced with Montgomery reduction (M-red) in [1]. Following proposal of the RNS M-red, promising results have been obtained for the RSA algorithm [2–6], Elliptic Curve Cryptosystem [7–14], Pairing-based Cryptosystem [15,16], modular inversion [17], Lattice-based Cryptosystem [18], and an architectural study[19]. In parallel with these applications, improvements in the RNS M-red algorithm have been proposed [3,5,6,9,13–15]. An overview of these researches is presented in [20].

This paper proposes improved RNS M-red algorithms, Q-RNS M-reds, which by posing quadratic residuosity constraints on the RNS base achieves the least number of multiplications. Past improvements in RNS M-red algorithms, with exceptions such as [6], were optimizations within one round of M-red execution, whereas our optimization for Q-RNS M-red is novel in that it transfers the square root of a constant from the current round to the previous round. Q-RNS includes two concrete algorithms called sQ-RNS and dQ-RNS, depending on the difference of the multiplication unit used.

Part of this paper is based on results obtained from a project commissioned by the New Energy and Industrial Technology Development Organization (NEDO).

✉ Shinichi Kawamura
shinichi2.kawamura@toshiba.co.jp; skwmr@ieee.org

Yuichi Komano
yuichi1.komano@toshiba.co.jp

Hideo Shimizu
hideo.shimizu@toshiba.co.jp

Tomoko Yonemura
tomoko.yonemura@toshiba.co.jp

[1] Toshiba Corporation, Kawasaki, Japan

[2] ECSEC Technical Research Association, Tokyo, Japan

This paper is organized as follows. Section 2 introduces used notation and some basic concepts. Section 3 explains the conventional RNS M-red algorithms. In Sect. 4, we introduce the new idea to use quadratic residuosity in order to simplify the M-red algorithm. The first variant, sQ-RNS M-red, is a direct combination of a new idea and a conventional algorithm. The second variant, dQ-RNS M-red, relaxes the constraint to RNS base choice by introducing the double-level Montgomery technique from [13]. Other procedures necessary to implement public-key cryptography, such as Initialize, are discussed in Sect. 5. Section 6 compares RNS M-red algorithms including FPGA implementations, and Sect. 7 concludes this paper.

2 Basic concepts

2.1 Notation

The following definitions are applied in this paper.

w : Bit size of a word in a given computer.

$$\langle x\rangle_m = x \bmod m, \text{ where } \langle x\rangle_m \in [0, m).$$
$$\overline{\langle x\rangle_m} \equiv x \pmod{m} \text{ and } \overline{\langle x\rangle_m} \in [0, 2m).$$

Base $B = \{m_1, \ldots, m_n\}$, where $\gcd(m_i, m_j) = 1$ for $i \neq j$.
Base $B' = \{m'_1, \ldots, m'_n\}$, where $\gcd(m'_i, m'_j) = 1$ for $i \neq j$.
$|B|$: Size of a set B.

$$M = \prod_{i=1}^{n} m_i,\ M' = \prod_{i=1}^{n} m'_i, \text{ where } \gcd(M, M') = 1.$$
$$M_i = M/m_i,\ M'_i = M'/m'_i.$$

$\left\langle x^{-1}\right\rangle_m$: A multiplicative inverse of x if $\gcd(x, m) = 1$.

$$\{x\}_B = \left[\langle x\rangle_{m_1}, \ldots, \langle x\rangle_{m_n}\right].$$
$$\{x\}_{B'} = \left[\langle x\rangle_{m'_1}, \ldots, \langle x\rangle_{m'_n}\right].$$
$$\{x\}_{BB'} = [\{x\}_B, \{x\}_{B'}].$$

Transpose T:

$$\{x\}_B^T = \begin{bmatrix} \langle x\rangle_{m_1} \\ \vdots \\ \langle x\rangle_{m_n} \end{bmatrix}.$$
$$\langle x\rangle_m \otimes \langle y\rangle_m = \langle xy\rangle_m .$$
$$\{x\}_B \otimes \{y\}_B = \{xy\}_B = \left[\langle xy\rangle_{m_1}, \ldots, \langle xy\rangle_{m_n}\right].$$

$\otimes_M$: Single-word Montgomery multiplication.

$$\langle x\rangle_m \otimes_M \langle y\rangle_m = \left\langle xy2^{-w}\right\rangle_m.$$

$$\{x\}_B \otimes_M \{y\}_B = \{xy2^{-w}\}_B = \left[\left\langle xy2^{-w}\right\rangle_{m_1}, \ldots, \left\langle xy2^{-w}\right\rangle_{m_n}\right].$$

In this paper, matrix expressions are used to describe parallel processing using RNS. If the matrix is diagonal, no substantive mixture of B and B' in an operation occurs. In such cases, definitions above are sufficient to carry out the matrix operations. If different bases appear in an operation, which occurs for the base extension operation or the ToBin transformation, the following computation rules will apply.

$$\left[\langle a_1\rangle_{m'_i}, \ldots, \langle a_n\rangle_{m'_i}\right] \otimes \{x\}_B^T + \hat{L} \otimes \langle a\rangle_{m'_i} = \left\langle \sum_{k=1}^{n} \langle a_k\rangle_{m'_i} \langle x\rangle_{m_k} + \hat{L}\, \langle a\rangle_{m'_i} \right\rangle_{m'_i} .$$

If the multiplication unit is the Montgomery one, then

$$\left[\langle a_1\rangle_{m'_i}, \ldots, \langle a_n\rangle_{m'_i}\right] \otimes_M \{x\}_B^T + \hat{L} \otimes_M \langle a\rangle_{m'_i} = \left\langle \sum_{k=1}^{n} \langle a_k\rangle_{m'_i} \langle x\rangle_{m_k} 2^{-w} + \hat{L}\, \langle a\rangle_{m'_i} 2^{-w} \right\rangle_{m'_i} .$$

These definitions suffice to carry out the matrix computations appearing in this paper. Note that the matrix computation in this paper is different from standard matrix computation in that the result in each line is reduced by a modulus unique to that line. Therefore, no inverse matrix can be defined here. However, this representation effectively simplifies the algorithm representation and makes it easy to count the number of operations.

2.2 Modular multiplication

Most public-key cryptosystems are implemented by repetition of a modular multiplication with a large modulus p, that is,

$$z = xy \bmod p, \tag{1}$$

where $p < 2^l$. l ranges from several hundreds to several thousands and p is usually a large prime or a product of two large primes.

Suppose that we need to add the results of two modular multiplications. If we run a modular multiplication twice, at least two multiplications and two modular reductions are necessary. If we instead use the equation

$$z = (x_1 y_1 + x_2 y_2) \bmod p, \tag{2}$$

only a single reduction is sufficient to obtain the result after the two multiplications and an addition. This technique is called a lazy reduction, and it effectively reduces the number

```
Input $x, y, p$
where $x, y \in [0, p)$, $\gcd(R, p) = 1$ and $R > p$
Output $v = \langle xyR^{-1} \rangle_p$
1.  $c \leftarrow xy$
2.  $q \leftarrow c(-p)^{-1} \bmod R$
3.  $r \leftarrow c + qp$
4.  $v \leftarrow r/R$
5.  if $v \geq p$, $v \leftarrow v - p$        ■
```

Fig. 1 Montgomery multiplication $MM(x, y)$

of reduction operations when summation of several modular multiplications is to be computed. Such a case frequently occurs in the implementation of Elliptic Curve Cryptography. We define the number of terms ν as a degree of laziness. For example, Eq. (2) has degree $\nu = 2$.

2.3 Montgomery reduction

When implementing modular multiplication, one option is to simply use Eq. (1), However, to avoid conditional branches inherent to division operation, another popular option is to implement it using the Montgomery multiplication below [21].

$$v = xyR^{-1} \bmod p \tag{3}$$

Figure 1 shows details of the procedure. Step 1 is a multiplication followed by Montgomery reduction in steps 2–5, which here is called M-red. For correct results, it suffices that $\gcd(R, p) = 1$ and $R > p$. Since p is usually an odd number, choosing $R = 2^l$ satisfies these conditions. In this setting, step 4 is carried out simply by a shifting operation. Step 5 is called the final subtraction, which makes the computation result less than p.

Let $MM(x, y)$ be the right-hand side of Eq. (3). Using $MM(x, y)$, the procedure to compute a modular multiplication is described as follows.
Initialize:

$$\begin{aligned} x' &= MM(x, R^2) = xR^2R^{-1} \bmod p \\ &= xR \bmod p \\ y' &= MM(y, R^2) = yR^2R^{-1} \bmod p \\ &= yR \bmod p \end{aligned}$$

Main body:

$$\begin{aligned} z' &= MM(x', y') = (xR)(yR)R^{-1} \bmod p \\ &= xyR \bmod p \end{aligned}$$

Finalize:

$$\begin{aligned} z &= MM(z', 1) = (xyR) \cdot 1 \cdot R^{-1} \bmod p \\ &= xy \bmod p \end{aligned}$$

If our goal is single execution of a modular multiplication, calling $MM(x, y)$ four times is not efficient. However, when computing a modular multiplication many times, the overheads of Initialize and Finalize become negligible because they are called once at the beginning and at the end, respectively.

The goal of this paper is to propose an efficient algorithm to compute the Montgomery reduction in RNS, that is, an RNS M-red algorithm.

2.4 RNS

Let $B = \{m_1, \ldots, m_n\}$ be a base for RNS representation, where $\gcd(m_i, m_j) = 1$ holds for $i \neq j$. RNS representation of an integer x is given by

$$\{x\}_B = [\langle x \rangle_{m_1}, \ldots, \langle x \rangle_{m_n}].$$

The symbol $\langle x \rangle_m$ is defined as $\langle x \rangle_m = x \bmod m$, and thus $\langle x \rangle_m \in [0, m)$ holds. The n-tuple on the right is called the RNS representation of x in base B. The representation is unique if $0 \leq x < M$, where $M = \prod_{i=1}^{n} m_i$. This representation allows fast arithmetic in Z/MZ since

$$\{x\}_B \odot \{y\}_B = [\langle x \odot y \rangle_{m_1}, \ldots, \langle x \odot y \rangle_{m_n}],$$

where $\odot \in \{+, -, \times, /\}$. '/' applies only if y is co-prime to M.

In RNS, a large integer can be processed with independent parallel operations in each channel. If we could use small factors of public-key p as an RNS base, a very efficient implementation would be realized. It is, however, difficult to employ such an approach since the public-key p is usually a large prime or a product of two large primes.

2.5 Chinese remainder theorem

According to the Chinese remainder theorem, the integer x represented in RNS is recovered by

$$x = \sum_{i=1}^{n} \langle x \rangle_{m_i} \left\langle M_i^{-1} \right\rangle_{m_i} M_i \bmod M.$$

Let us consider a method avoiding the modulo M operation in evaluating the right-hand side. Since we can replace $\langle x \rangle_{m_i} \left\langle M_i^{-1} \right\rangle_{m_i}$ by $\xi_i(x) = \left\langle xM_i^{-1} \right\rangle_{m_i}$ without affecting the equality, we can rewrite the equation with a new unknown integer L as

$$x = \sum_{i=1}^{n} \xi_i(x) M_i - LM.$$

Considering $0 \le x/M < 1$, we obtain the following equation.

$$L = \left\lfloor \sum_{i=1}^{n} \frac{\xi_i(x)}{m_i} \right\rfloor$$

An approximation $\hat{L}$ of L is proposed in [3] for an appropriate offset $\alpha \in [0, 1)$,

$$\hat{L} = \left\lfloor \alpha + \sum_{i=1}^{n} \frac{\mathrm{trunc}(t, \xi_i(x))}{2^w} \right\rfloor.$$

$\mathrm{trunc}(t, x)$ is a function to force the lower $w - t$ bits of x to zero.

$$\mathrm{trunc}(t, x) = \left\lfloor x 2^{-(w-t)} \right\rfloor \cdot 2^{(w-t)},$$

where $0 < t \le w$. The difference between $\hat{L}$ and L can be at most 1 if appropriate t and α are selected. This approximation is the most important part of the base extension process in the computation of RNS M-red.

2.6 Special modulus for fast reduction

Usually, arbitrary moduli can be selected as RNS bases so long as they are mutually prime. It is well-known that the pseudo-Mersenne prime number $m_i = 2^w - \mu_i$ is useful for fast reduction. Actually, if

$$\mu_i < 2^{\lfloor w/2 \rfloor} \tag{4}$$

holds, $x \bmod m_i$ can be computed efficiently with the following procedure. First, repeat the operation below twice:

$$x \leftarrow (x \bmod 2^w) + \mu_i \cdot \lfloor x/2^w \rfloor.$$

If $x < 2^{2w}$ holds for the initial value, the result is in $[0, 2^{w+1})$. The final result is obtained by subtracting m_i at most once. In addition, if the Hamming weight of μ_i is small, multiplication by μ_i can be replaced by several additions.

Let $\bar{\mu} = (1/w) \log_2 \mu_i$. Equation (4) is satisfied if $\bar{\mu} < 0.5$.

2.7 Quadratic residuosity

An integer a is called a quadratic residue modulo m if there exists a solution for the congruence equation of x,

$$x^2 \equiv a \bmod m.$$

In other words, a is called a quadratic residue if a has a square root, and a quadratic non-residue otherwise. Unlike real numbers, not every integer has its square roots for a given modulus m. An integer a can be a quadratic residue or a quadratic non-residue, depending on the value of modulus m.

Let a function $\mathrm{QR}(a, m)$ be defined by

$$\mathrm{QR}(a, m) = \begin{cases} 1 \text{ (if } \exists x \text{ s.t. } x^2 \equiv a \pmod{m}) \\ 0 \text{ (if } \forall x \text{ s.t. } x^2 \not\equiv a \pmod{m}) \end{cases}$$

This function will be used as a distinguisher of a quadratic residue. If $\mathrm{QR}(a, m) = 1$, let $\langle a^{1/2} \rangle_m$ denote one of the square roots of a modulo m.

Quadratic residuosity has been used for RNS in signal processing applications to represent a complex signal (for instance, refer to section 8.1 of [22]), whereas no previously proposed RNS M-red algorithm has used quadratic residuosity. This paper applies quadratic residuosity to the RNS M-red algorithm for the first time to construct algorithms that consist of the least number of unit multiplications.

3 Conventional algorithms

3.1 Basic RNS M-red algorithm

3.1.1 Algorithm

Figure 2 shows the Montgomery reduction algorithm corresponding to steps 2–5 of Fig. 1. By relaxing the range of the output to less than $2p$, the final subtraction has been removed. In addition, the upper bound of the input is also relaxed from p^2 to βp^2 with $\beta \ge 4$. The condition $R > \beta p$ ensures that the output is less than $2p$.[1] All RNS M-red algorithms in this paper can be regarded as RNS variants of this M-red algorithm.

Figure 3 shows the RNS M-red algorithm derived straightforwardly from M-red in Fig. 2. A description of each step is on the same line as the step number, followed by the actual specification in matrix form. Steps 1, 4, and 5 correspond to steps 1, 2, and 3 in Fig. 2, respectively. Steps 2 and 3 derive approximation $\{q'\}_{B'}$ from $\{q\}_B$, which is a technique called the base extension step. Similarly, steps 6 and 7 are also the

[1] $s = (x + pq)/R < (\beta p^2 + pR)/R = (\beta p/R + 1)p < 2p$.

```
Input $x, p$
where $0 \le x \le \beta p^2$, $\gcd(R, p) = 1$ and $R > \beta p$.
Output $s = \overline{\langle xR^{-1} \rangle}_p$ where $s \in [0, 2p)$
1.  $q = x(-p)^{-1} \bmod R$
2.  $r = (x + pq)$
3.  $s = R^{-1} r$  ■
```

Fig. 2 Montgomery reduction (M-red)

base extension, deriving $\{s\}_B$ from $\{s\}_{B'}$. A constant R is set $R = M = \prod_{i=1}^{n} m_i$ for RNS M-red, while $R = 2^l$ is a common setting for the binary M-red.

Since step 1 is carried out in base B, modulo M is automatically applied to the computation and the result is equivalent to that of step 1 in Fig. 2. It is in base B' that steps 4 and 5 should be carried out. The reason for this is as follows: As for step 4, it is of no use computing $(x + pq)$ in B because the result is always a multiple of M and thus always 0 in base B. The computation in step 5 is to multiply M^{-1} by r. This can be carried out in base B' but not in base B, since M^{-1} does not exist in base B. Although the final result s is computed at step 5, it is only represented in base B'. In order to complete the representation in base B, steps 6 and 7 extend $\{s\}_{B'}$ to $\{s\}_B$. This ensures compatibility between output and input of the RNS M-red algorithm.

The matrix elements in each step are defined as follows:

Step 1:[2] $d_i = \left\langle -p^{-1} \right\rangle_{m_i}$.

Step 2:[3] $w_{ii} = \left\langle M_i^{-1} \right\rangle_{m_i}$.

Step 3: The first base extension.

$$a_{ij} = \left\langle M_j \right\rangle_{m'_i},$$
$$a_i = \langle -M \rangle_{m'_i}.$$

Step 4: $p_i = \langle p \rangle_{m'_i}$.

Step 5: $w_i = \left\langle M^{-1} \right\rangle_{m'_i}$.

Step 6: $w'_{ii} = \left\langle M_i'^{-1} \right\rangle_{m'_i}$.

Step 7: The second base extension.

$$b_{ij} = \left\langle M'_j \right\rangle_{m_i},$$
$$b_i = \left\langle -M' \right\rangle_{m_i}.$$

This algorithm includes $(2n^2 + 5n)$ unit multiplications. We exclude the multiplications by $\hat{L}$ at steps 3 and 7 because a technique in [3] shows how to carry out each of these by less than n additions.

Input: $\{x\}_{BB'}$, where $0 \le x < \beta p^2$ with conditions (i)-(v).
Output: $\{s\}_{BB'}$ where $s = \overline{\langle xM^{-1} \rangle}_p$
Procedure:

1. $q \leftarrow x(-p^{-1}) \bmod M$ (in *base* B):

$$\begin{bmatrix} \langle q \rangle_{m_1} \\ \langle q \rangle_{m_2} \\ \vdots \\ \langle q \rangle_{m_n} \end{bmatrix} \leftarrow \begin{bmatrix} d_1 & 0 \cdots & 0 \\ 0 & d_2 \cdots & 0 \\ \vdots & \vdots \ddots & \vdots \\ 0 & 0 \cdots & d_n \end{bmatrix} \otimes \begin{bmatrix} \langle x \rangle_{m_1} \\ \langle x \rangle_{m_2} \\ \vdots \\ \langle x \rangle_{m_n} \end{bmatrix}$$

2. $\theta_i \leftarrow \langle q M_i^{-1} \rangle_{m_i}$ (in *base* B):

$$\begin{bmatrix} \langle \theta \rangle_{m_1} \\ \langle \theta \rangle_{m_2} \\ \vdots \\ \langle \theta \rangle_{m_n} \end{bmatrix} \leftarrow \begin{bmatrix} w_{11} & 0 \cdots & 0 \\ 0 & w_{22} \cdots & 0 \\ \vdots & \vdots \ddots & \vdots \\ 0 & 0 \cdots & w_{nn} \end{bmatrix} \otimes \begin{bmatrix} \langle q \rangle_{m_1} \\ \langle q \rangle_{m_2} \\ \vdots \\ \langle q \rangle_{m_n} \end{bmatrix}$$

3. $q \leftarrow \sum M_i \theta_i + L(-M)$ (in *base* B'):

$\hat{L} \leftarrow \left\lfloor 0 + \sum_{i=1}^{n} \mathrm{trunc}(t, \langle \theta \rangle_{m_i})/2^w \right\rfloor$

$$\begin{bmatrix} \langle q' \rangle_{m'_1} \\ \langle q' \rangle_{m'_2} \\ \vdots \\ \langle q' \rangle_{m'_n} \end{bmatrix} \leftarrow \begin{bmatrix} a_{11} & a_{12} \ldots a_{1n} \\ a_{21} & a_{22} \cdots a_{2n} \\ \vdots & \vdots \ddots \vdots \\ a_{n1} & a_{n2} \cdots a_{nn} \end{bmatrix} \otimes \begin{bmatrix} \langle \theta \rangle_{m_1} \\ \langle \theta \rangle_{m_2} \\ \vdots \\ \langle \theta \rangle_{m_n} \end{bmatrix} + \hat{L} \otimes \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$$

4. $r \leftarrow x + pq$ (in *base* B'):

$$\begin{bmatrix} \langle r' \rangle_{m'_1} \\ \langle r' \rangle_{m'_2} \\ \vdots \\ \langle r' \rangle_{m'_n} \end{bmatrix} \leftarrow \begin{bmatrix} \langle x \rangle_{m'_1} \\ \langle x \rangle_{m'_2} \\ \vdots \\ \langle x \rangle_{m'_n} \end{bmatrix} + \begin{bmatrix} p_1 & 0 \cdots & 0 \\ 0 & p_2 \cdots & 0 \\ \vdots & \vdots \ddots & \vdots \\ 0 & 0 \cdots & p_n \end{bmatrix} \otimes \begin{bmatrix} \langle q' \rangle_{m'_1} \\ \langle q' \rangle_{m'_2} \\ \vdots \\ \langle q' \rangle_{m'_n} \end{bmatrix}$$

5. $s \leftarrow M^{-1} r$ (in *base* B'):

$$\begin{bmatrix} \langle s \rangle_{m'_1} \\ \langle s \rangle_{m'_2} \\ \vdots \\ \langle s \rangle_{m'_n} \end{bmatrix} \leftarrow \begin{bmatrix} w_1 & 0 \cdots & 0 \\ 0 & w_2 \cdots & 0 \\ \vdots & \vdots \ddots & \vdots \\ 0 & 0 \cdots & w_n \end{bmatrix} \otimes \begin{bmatrix} \langle r' \rangle_{m'_1} \\ \langle r' \rangle_{m'_2} \\ \vdots \\ \langle r' \rangle_{m'_n} \end{bmatrix}$$

6. $\sigma_i \leftarrow \langle M_i'^{-1} s \rangle_{m'_i}$ (in *base* B'):

$$\begin{bmatrix} \langle \sigma \rangle_{m'_1} \\ \langle \sigma \rangle_{m'_2} \\ \vdots \\ \langle \sigma \rangle_{m'_n} \end{bmatrix} \leftarrow \begin{bmatrix} w'_{11} & 0 \cdots & 0 \\ 0 & w'_{22} \cdots & 0 \\ \vdots & \vdots \ddots & \vdots \\ 0 & 0 \cdots & w'_{nn} \end{bmatrix} \otimes \begin{bmatrix} \langle s \rangle_{m'_1} \\ \langle s \rangle_{m'_2} \\ \vdots \\ \langle s \rangle_{m'_n} \end{bmatrix}$$

7. $s \leftarrow \sum M'_i \sigma_i + L(-M')$(in *base* B):

$\hat{L} \leftarrow \left\lfloor \alpha + \sum_{i=1}^{n} \mathrm{trunc}(t, \langle \sigma \rangle_{m'_i})/2^w \right\rfloor$

$$\begin{bmatrix} \langle s \rangle_{m_1} \\ \langle s \rangle_{m_2} \\ \vdots \\ \langle s \rangle_{m_n} \end{bmatrix} \leftarrow \begin{bmatrix} b_{11} & b_{12} \ldots b_{1n} \\ b_{21} & b_{22} \cdots b_{2n} \\ \vdots & \vdots \ddots \vdots \\ b_{n1} & b_{n2} \cdots b_{nn} \end{bmatrix} \otimes \begin{bmatrix} \langle \sigma \rangle_{m'_1} \\ \langle \sigma \rangle_{m'_2} \\ \vdots \\ \langle \sigma \rangle_{m'_n} \end{bmatrix} + \hat{L} \otimes \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

■

Fig. 3 Basic RNS M-red algorithm[3]

3.1.2 Requirement for parameters

Throughout this paper, we assume that the bit length w is common to base elements m_i and m'_i. Let m_i have a form of the pseudo-Mersenne prime,

$$m_i = 2^w - \mu_i \in B,$$

where μ_i is a relatively small positive integer. Similarly,

$$m'_i = 2^w - \mu'_i \in B'.$$

[2] d is used because it looks like an inverted p, which makes it easier to relate to p^{-1}.

[3] w is similarly used because it looks like an inverted M.

Such a modulus has two important properties:

(a) m_i is a special modulus for fast reduction.
(b) $1/m_i$ can be well approximated as $1/2^w$.

Property (b) applies to the computation of $\hat{L}$ in steps 3 and 7 in Fig. 3. Let $\xi_i(q) = \langle\theta\rangle_{m_i}$ be the results of step 2. To estimate the approximation error between

$$L = \left\lfloor \sum_{i=1}^{n} \xi_i(q)/m_i \right\rfloor$$

and

$$\hat{L} = \left\lfloor \sum_{i=1}^{n} \mathrm{trunc}(t, \xi_i(q))/2^w \right\rfloor,$$

let us consider a function $f(q) = \sum_{i=1}^{n} \xi_i(q)/m_i$ and its approximate function $\tilde{f}(q) = \sum_{i=1}^{n} \mathrm{trunc}(t, \xi_i(q))/2^w$. The following equations hold for f and $\tilde{f}$ [23].

$$f(q) - e_1 \le \tilde{f}(q) \le f(q)$$
$$e_1 = n(2^{-t} - 2^{-w}) + \frac{1}{2^w}\sum_{i=1}^{n}\left(1 - \frac{1}{m_i}\right)\mu_i \qquad (5)$$

Let q' be the extended value of q as computed in step 3. Using the above equations, we can show the relationship between q' and q as $q' = q + uM$ with $u \in \{0, 1\}$ if an offset $\alpha = 0$. This means that q is transformed to q' at step 3 with an error term uM. This error is absorbed in the relaxed range of the output $\overline{\langle xM^{-1}\rangle_p}$ until the end of step 5. A similar analysis can be applied to the second base extension at step 7. In this case, an approximation error e_1 is replaced by e_2 with parameters (m'_i, μ'_i), and an offset α is positive. A typical offset at the second base extension is $\alpha = 0.5$ [3]. The second base extension is error-free if $e_2 < \alpha$ and $2p \le (1 - \alpha)M'$.

Conditions (i)–(v) below are typical requirements for ensuring correct results [3,23].

(i) $\gcd(M, M') = 1$
(ii) $\gcd(p, M) = 1$
(iii) $\max(e_1, e_2) \le \alpha < 1$
(iv) $\beta p \le (1 - \alpha)M$
(v) $2p \le (1 - \alpha)M'$

From condition (iii) and Eq. (5), we can derive the lower bound of t, the effective number of bits for approximation, as

$$t \ge t_0 = \left\lceil -\log_2\{(\alpha - \max(e_0, e'_0))n^{-1} + 2^{-w}\} \right\rceil,$$

where e_0 and e'_0 represent the summation parts of e_1 and e_2, respectively.

Input: $\{x\}_{BB'}$, where $0 \le x < \beta p^2$ with conditions (i)-(v).
Output: $\{s\}_{BB'}$ where $s = \overline{\langle xM^{-1}\rangle_p}$
Procedure:
1.
$$\begin{bmatrix}\langle\theta\rangle_{m_1}\\ \langle\theta\rangle_{m_2}\\ \vdots\\ \langle\theta\rangle_{m_n}\end{bmatrix} \leftarrow \begin{bmatrix} d_{11} & 0 \cdots & 0\\ 0 & d_{22} \cdots & 0\\ \vdots & \vdots \ddots & \vdots\\ 0 & 0 & \cdots d_{nn}\end{bmatrix} \otimes \begin{bmatrix}\langle x\rangle_{m_1}\\ \langle x\rangle_{m_2}\\ \vdots\\ \langle x\rangle_{m_n}\end{bmatrix}$$
2.
$$\begin{bmatrix}\langle\xi\rangle_{m'_1}\\ \langle\xi\rangle_{m'_2}\\ \vdots\\ \langle\xi\rangle_{m'_n}\end{bmatrix} \leftarrow \begin{bmatrix} e_{11} & 0 \cdots & 0\\ 0 & e_{22} \cdots & 0\\ \vdots & \vdots \ddots & \vdots\\ 0 & 0 & \cdots e_{nn}\end{bmatrix} \otimes \begin{bmatrix}\langle x\rangle_{m'_1}\\ \langle x\rangle_{m'_2}\\ \vdots\\ \langle x\rangle_{m'_n}\end{bmatrix}$$
3.
$$\hat{L} \leftarrow \left\lfloor 0 + \textstyle\sum_{i=1}^{n} \mathrm{trunc}(t,\ \langle\theta\rangle_{m_i})/2^w \right\rfloor$$
$$\begin{bmatrix}\langle\sigma\rangle_{m'_1}\\ \langle\sigma\rangle_{m'_2}\\ \vdots\\ \langle\sigma\rangle_{m'_n}\end{bmatrix} \leftarrow \begin{bmatrix}\langle\xi\rangle_{m'_1}\\ \langle\xi\rangle_{m'_2}\\ \vdots\\ \langle\xi\rangle_{m'_n}\end{bmatrix} + \begin{bmatrix} a'_{11} & a'_{12} \cdots a'_{1n}\\ a'_{21} & a'_{22} \cdots a'_{2n}\\ \vdots & \vdots \ddots \vdots\\ a'_{n1} & a'_{n2} \cdots a_{nn}\end{bmatrix} \otimes \begin{bmatrix}\langle\theta\rangle_{m_1}\\ \langle\theta\rangle_{m_2}\\ \vdots\\ \langle\theta\rangle_{m_n}\end{bmatrix} + \hat{L} \otimes \begin{bmatrix} a'_1\\ a'_2\\ \vdots\\ a'_n\end{bmatrix}$$
4.
$$\hat{L} \leftarrow \left\lfloor \alpha + \textstyle\sum_{i=1}^{n} \mathrm{trunc}(t,\ \langle\sigma\rangle_{m'_i})/2^w \right\rfloor$$
$$\begin{bmatrix}\langle s\rangle_{m_1}\\ \langle s\rangle_{m_2}\\ \vdots\\ \langle s\rangle_{m_n}\end{bmatrix} \leftarrow \begin{bmatrix} b_{11} & b_{12} \cdots b_{1n}\\ b_{21} & b_{22} \cdots b_{2n}\\ \vdots & \vdots \ddots \vdots\\ b_{n1} & b_{n2} \cdots b_{nn}\end{bmatrix} \otimes \begin{bmatrix}\langle\sigma\rangle_{m'_1}\\ \langle\sigma\rangle_{m'_2}\\ \vdots\\ \langle\sigma\rangle_{m'_n}\end{bmatrix} + \hat{L} \otimes \begin{bmatrix} b_1\\ b_2\\ \vdots\\ b_n\end{bmatrix}$$
5.
$$\begin{bmatrix}\langle s\rangle_{m'_1}\\ \langle s\rangle_{m'_2}\\ \vdots\\ \langle s\rangle_{m'_n}\end{bmatrix} \leftarrow \begin{bmatrix} c_{11} & 0 \cdots & 0\\ 0 & c_{22} \cdots & 0\\ \vdots & \vdots \ddots & \vdots\\ 0 & 0 & \cdots c_{nn}\end{bmatrix} \otimes \begin{bmatrix}\langle\sigma\rangle_{m'_1}\\ \langle\sigma\rangle_{m'_2}\\ \vdots\\ \langle\sigma\rangle_{m'_n}\end{bmatrix}$$
■

Fig. 4 G-RNS M-red algorithm [9]

3.2 G-RNS algorithm

Guillermin proposed an algorithm that at the time achieved the minimum number of unit multiplications [9]. We call this algorithm G-RNS (Fig. 4). Step 1 is the integration of steps 1 and 2 in Fig. 3. Step 2 is from the first term of step 4 combined with steps 5 and 6 of Fig. 3. Step 3 is derived from steps 3–6 of Fig. 3. Step 4 corresponds to step 7 of the basic algorithm. Step 5 is new in Fig. 4.

Elements of the matrices of G-RNS are defined as follows:
Step 1: $d_{ii} = \langle w_{ii} \cdot d_i\rangle_{m_i} = \left\langle M_i^{-1}(-p^{-1})\right\rangle_{m_i}$.
Step 2: $e_{ii} = \langle w'_{ii} \cdot w_i\rangle_{m'_i} = \left\langle M_i'^{-1} M^{-1}\right\rangle_{m'_i}$.
Step 3:

$$a'_{ij} = \langle w'_{ii} \cdot w_i \cdot p_i \cdot a_{ij}\rangle_{m'_i} = \left\langle M_i'^{-1} m_j^{-1} p\right\rangle_{m'_i},$$
$$a'_i = \langle w'_{ii} \cdot w_i \cdot p_i \cdot a_i\rangle_{m'_i} = \left\langle -M_i'^{-1} p\right\rangle_{m'_i}.$$

```
Input: $\{x\}_{BB'}$ , where $0 \le x < \beta p^2$ with conditions (i)-(v).
Output: $\{s\}_{BB'}$ where $s = \langle xM^{-1}\rangle_p$
Procedure:
1.
    $\begin{bmatrix}\langle\theta\rangle_{m_1}\\ \langle\theta\rangle_{m_2}\\ \vdots\\ \langle\theta\rangle_{m_n}\end{bmatrix} \leftarrow \begin{bmatrix}d_{11} & 0\cdots & 0\\ 0 & d_{22}\cdots & 0\\ \vdots & \vdots\ \ddots & \vdots\\ 0 & 0 & \cdots d_{nn}\end{bmatrix} \otimes \begin{bmatrix}\langle x\rangle_{m_1}\\ \langle x\rangle_{m_2}\\ \vdots\\ \langle x\rangle_{m_n}\end{bmatrix}$
2.
    $\begin{bmatrix}\langle\xi\rangle_{m'_1}\\ \langle\xi\rangle_{m'_2}\\ \vdots\\ \langle\xi\rangle_{m'_n}\end{bmatrix} \leftarrow \begin{bmatrix}e_{11} & 0\cdots & 0\\ 0 & e_{22}\cdots & 0\\ \vdots & \vdots\ \ddots & \vdots\\ 0 & 0 & \cdots e_{nn}\end{bmatrix} \otimes \begin{bmatrix}\langle x\rangle_{m'_1}\\ \langle x\rangle_{m'_2}\\ \vdots\\ \langle x\rangle_{m'_n}\end{bmatrix}$
3.
    $\hat{L} \leftarrow \lfloor 0 + \sum_{i=1}^{n} \mathrm{trunc}(t,\ \langle\theta\rangle_{m_i})/2^w \rfloor$
    $\begin{bmatrix}\langle q'\rangle_{m'_1}\\ \langle q'\rangle_{m'_2}\\ \vdots\\ \langle q'\rangle_{m'_n}\end{bmatrix} \leftarrow \begin{bmatrix}a_{11} & a_{12}\ldots a_{1n}\\ a_{21} & a_{22}\cdots a_{2n}\\ \vdots & \vdots\ \ddots\ \vdots\\ a_{n1} & a_{n2}\cdots a_{nn}\end{bmatrix} \otimes \begin{bmatrix}\langle\theta\rangle_{m_1}\\ \langle\theta\rangle_{m_2}\\ \vdots\\ \langle\theta\rangle_{m_n}\end{bmatrix} + \hat{L} \otimes \begin{bmatrix}a_1\\ a_2\\ \vdots\\ a_n\end{bmatrix}$
4.
    $\begin{bmatrix}\langle\sigma\rangle_{m'_1}\\ \langle\sigma\rangle_{m'_2}\\ \vdots\\ \langle\sigma\rangle_{m'_n}\end{bmatrix} \leftarrow \begin{bmatrix}\langle\xi\rangle_{m'_1}\\ \langle\xi\rangle_{m'_2}\\ \vdots\\ \langle\xi\rangle_{m'_n}\end{bmatrix} + \begin{bmatrix}f_{11} & 0\cdots & 0\\ 0 & f_{22}\cdots & 0\\ \vdots & \vdots\ \ddots & \vdots\\ 0 & 0 & \cdots f_{nn}\end{bmatrix} \otimes \begin{bmatrix}\langle q'\rangle_{m'_1}\\ \langle q'\rangle_{m'_2}\\ \vdots\\ \langle q'\rangle_{m'_n}\end{bmatrix}$
5.
    $\hat{L} \leftarrow \lfloor \alpha + \sum_{i=1}^{n} \mathrm{trunc}(t,\ \langle\sigma\rangle_{m'_i})/2^w \rfloor$
    $\begin{bmatrix}\langle s\rangle_{m_1}\\ \langle s\rangle_{m_2}\\ \vdots\\ \langle s\rangle_{m_n}\end{bmatrix} \leftarrow \begin{bmatrix}b_{11} & b_{12}\ldots b_{1n}\\ b_{21} & b_{22}\cdots b_{2n}\\ \vdots & \vdots\ \ddots\ \vdots\\ b_{n1} & b_{n2}\cdots b_{nn}\end{bmatrix} \otimes \begin{bmatrix}\langle\sigma\rangle_{m'_1}\\ \langle\sigma\rangle_{m'_2}\\ \vdots\\ \langle\sigma\rangle_{m'_n}\end{bmatrix} + \hat{L} \otimes \begin{bmatrix}b_1\\ b_2\\ \vdots\\ b_n\end{bmatrix}$
6.
    $\begin{bmatrix}\langle s\rangle_{m'_1}\\ \langle s\rangle_{m'_2}\\ \vdots\\ \langle s\rangle_{m'_n}\end{bmatrix} \leftarrow \begin{bmatrix}c_{11} & 0\cdots & 0\\ 0 & c_{22}\cdots & 0\\ \vdots & \vdots\ \ddots & \vdots\\ 0 & 0 & \cdots c_{nn}\end{bmatrix} \otimes \begin{bmatrix}\langle\sigma\rangle_{m'_1}\\ \langle\sigma\rangle_{m'_2}\\ \vdots\\ \langle\sigma\rangle_{m'_n}\end{bmatrix}$
■
```

Fig. 5 C-RNS M-red algorithm[15]

Step 4:

$$b_{ij} = \left\langle M'_j \right\rangle_{m_i},$$
$$b_i = \left\langle -M' \right\rangle_{m_i}.$$

Step 5: $c_{ii} = \langle M'_i \rangle_{m'_i}$.

The necessary number of unit multiplications for G-RNS is $(2n^2 + 3n)$.

3.3 C-RNS algorithm

Figure 5 shows an algorithm proposed by Cheung et al.[15]. Elements appear in each step are defined as follows:

Step 1: $d_{ii} = \left\langle M_i^{-1}(-p^{-1}) \right\rangle_{m_i}$.

Step 2: $e_{ii} = \left\langle M_i'^{-1} M^{-1} \right\rangle_{m'_i}$.

Step 3:

$$a_{ij} = \left\langle M_j \right\rangle_{m'_i},$$
$$a_i = \langle -M \rangle_{m'_i}.$$

Step 4: $f_{ii} = \left\langle w'_{ii} \cdot w_i \cdot p_i \right\rangle_{m'_i} = \left\langle M_i'^{-1} M^{-1} p \right\rangle_{m'_i}$.

Step 5:

$$b_{ij} = \left\langle M'_j \right\rangle_{m_i},$$
$$b_i = \left\langle -M' \right\rangle_{m_i}.$$

Step 6: $c_{ii} = \langle M'_i \rangle_{m'_i}$.

The difference from G-RNS is that C-RNS restores the original base extension matrix at step 3. The number of multiplications is $(2n^2 + 4n)$ in general cases, but computation of the base extension can be reduced drastically in the special case when n is small. As discussed in [15,16], it follows that

$$a_{ij} = \left\langle M_j \right\rangle_{m'_i} = \prod_{k \ne j} m_k \bmod m'_i$$
$$= \prod_{k \ne j} (m_k - m'_i) \bmod m'_i,$$

and that $|m_k - m'_i| = |\mu'_i - \mu_k|$ is a small number. Therefore, if n is not so large, we can expect that a_{ij} and b_{ij} are close to 2^w. This makes it possible to reduce the computation amount at the base extensions. In hindsight, this property can be applied to the basic RNS M-red as well. It is shown in [16] that efficient parameters exist for $n = 4$ and 258-bit modulus p.

3.4 R-RNS algorithm

Gandino et al. proposed a reorganized version of the RNS Montgomery multiplication algorithm [6], which we call the R-RNS algorithm here. As shown in Fig. 6, we can describe the R-RNS algorithm using almost the same notation as G-RNS. Let us explain the difference between Figs. 4 and 6.

1. Input and output are changed from $\{x\}_{BB'}$ and $\{s\}_{BB'}$ to $\{x\}_B \cup \{\hat{\hat{x}}\}_{B'}$ and $\{s\}_B \cup \{\hat{s}\}_{B'}$, respectively, where elements of $\{\hat{\hat{x}}\}_{B'}$ and $\{\hat{s}\}_{B'}$ are defined as

   $$\left\langle \hat{\hat{x}} \right\rangle_{m'_i} = \left\langle x M_i'^{-2} \right\rangle_{m'_i},$$
   $$\langle \hat{s} \rangle_{m'_i} = \left\langle s M_i'^{-1} \right\rangle_{m'_i}.$$

2. In step 2, elements of the matrix are changed from e_{ij} to e'_{ij}, where the latter is defined as

   $$e'_{ij} = \left\langle e_{ij} M_i'^2 \right\rangle_{m'_i} = \left\langle M'_i M^{-1} \right\rangle_{m'_i}.$$

Input: $\{x\}_B \cup \{\hat{\hat{x}}\}_{B'}$, where $0 \le x < \beta p^2$ with conditions (i)-(v).
Output: $\{s\}_B \cup \{\hat{s}\}_{B'}$ where $s = \overline{\langle xM^{-1}\rangle_p}$
Procedure:

1.

$$\begin{bmatrix} \langle\theta\rangle_{m_1} \\ \langle\theta\rangle_{m_2} \\ \vdots \\ \langle\theta\rangle_{m_n} \end{bmatrix} \leftarrow \begin{bmatrix} d_{11} & 0 \cdots & 0 \\ 0 & d_{22} \cdots & 0 \\ \vdots & \vdots \ddots & \vdots \\ 0 & 0 & \cdots d_{nn} \end{bmatrix} \otimes \begin{bmatrix} \langle x\rangle_{m_1} \\ \langle x\rangle_{m_2} \\ \vdots \\ \langle x\rangle_{m_n} \end{bmatrix}$$

2.

$$\begin{bmatrix} \langle\xi\rangle_{m'_1} \\ \langle\xi\rangle_{m'_2} \\ \vdots \\ \langle\xi\rangle_{m'_n} \end{bmatrix} \leftarrow \begin{bmatrix} e'_{11} & 0 \cdots & 0 \\ 0 & e'_{22} \cdots & 0 \\ \vdots & \vdots \ddots & \vdots \\ 0 & 0 & \cdots e'_{nn} \end{bmatrix} \otimes \begin{bmatrix} \langle\hat{\hat{x}}\rangle_{m'_1} \\ \langle\hat{\hat{x}}\rangle_{m'_2} \\ \vdots \\ \langle\hat{\hat{x}}\rangle_{m'_n} \end{bmatrix}$$

3.

$$\hat{L} \leftarrow \left\lfloor 0 + \textstyle\sum_{i=1}^{n} \mathrm{trunc}(t,\ \langle\theta\rangle_{m_i})/2^w \right\rfloor$$

$$\begin{bmatrix} \langle\hat{s}\rangle_{m'_1} \\ \langle\hat{s}\rangle_{m'_2} \\ \vdots \\ \langle\hat{s}\rangle_{m'_n} \end{bmatrix} \leftarrow \begin{bmatrix} \langle\xi\rangle_{m'_1} \\ \langle\xi\rangle_{m'_2} \\ \vdots \\ \langle\xi\rangle_{m'_n} \end{bmatrix} + \begin{bmatrix} a'_{11} & a'_{12} \ldots a'_{1n} \\ a'_{21} & a'_{22} \cdots a'_{2n} \\ \vdots & \vdots \ddots \vdots \\ a'_{n1} & a'_{n2} \cdots a_{nn} \end{bmatrix} \otimes \begin{bmatrix} \langle\theta\rangle_{m_1} \\ \langle\theta\rangle_{m_2} \\ \vdots \\ \langle\theta\rangle_{m_n} \end{bmatrix} + \hat{L} \otimes \begin{bmatrix} a'_1 \\ a'_2 \\ \vdots \\ a'_n \end{bmatrix}$$

4.

$$\hat{L} \leftarrow \left\lfloor \alpha + \textstyle\sum_{i=1}^{n} \mathrm{trunc}(t,\ \langle\hat{s}\rangle_{m'_i})/2^w \right\rfloor$$

$$\begin{bmatrix} \langle s\rangle_{m_1} \\ \langle s\rangle_{m_2} \\ \vdots \\ \langle s\rangle_{m_n} \end{bmatrix} \leftarrow \begin{bmatrix} b_{11} & b_{12} \ldots b_{1n} \\ b_{21} & b_{22} \cdots b_{2n} \\ \vdots & \vdots \ddots \vdots \\ b_{n1} & b_{n2} \cdots b_{nn} \end{bmatrix} \otimes \begin{bmatrix} \langle\hat{s}\rangle_{m'_1} \\ \langle\hat{s}\rangle_{m'_2} \\ \vdots \\ \langle\hat{s}\rangle_{m'_n} \end{bmatrix} + \hat{L} \otimes \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

■

Fig. 6 R-RNS M-red algorithm

Due to this definition, the following relationship holds.

$$\left\langle e_{ij} \cdot x \right\rangle_{m'_i} = \left\langle e'_{ij} \cdot \hat{\hat{x}} \right\rangle_{m'_i}$$

Thus, the result of step 2 in Fig. 6 is identical to that in Fig. 4.

3. Notation of the result in step 3 is changed to $\langle\hat{s}\rangle_{m'_i}$, although its value is identical to $\langle\sigma\rangle_{m'_i}$, the result of step 3 of Fig. 4.
4. Since the new output includes $\langle\hat{s}\rangle_{m'_i}$ instead of $\langle s\rangle_{m'_i}$, step 5 of Fig. 4 is omitted in Fig. 6. This reduces the number of unit multiplications by n from Fig. 4.

The number of unit multiplications is $(2n^2+2n)$ in this case.

4 New algorithms

4.1 Derivation of Q-RNS

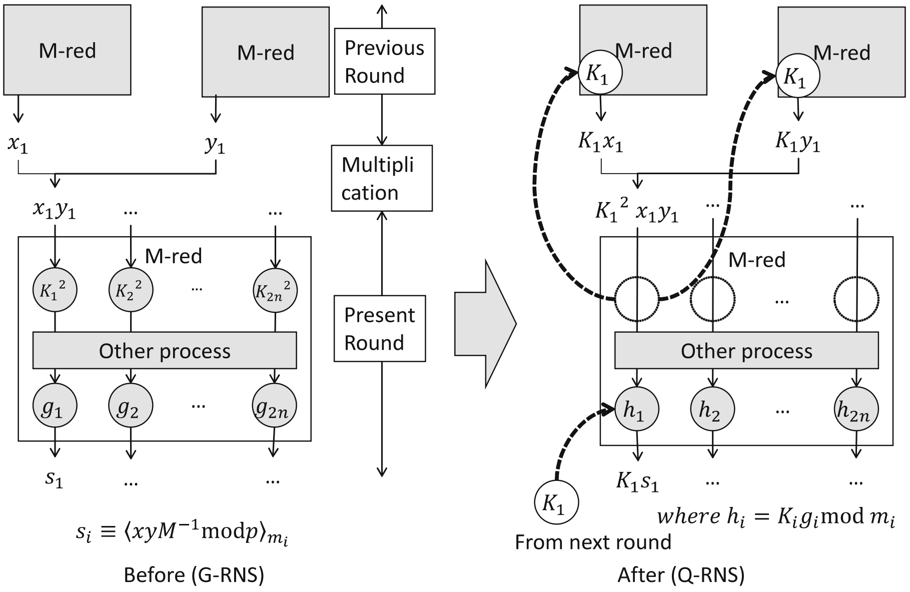
We introduce an idea to pose quadratic residuosity to the RNS base so as to make steps 1 and 2 in the G-RNS algorithm unnecessary. Figure 7 (left) shows part of a long sequence of operations in which a multiplication and G-RNS M-red are repeated. It consists of three phases: the previous M-red, a multiplication, and the present M-red. The input of the present M-red is $\{xy\}_{BB'}$. For simplicity, elements of RNS representation are uniformly numbered from 1 to $2n$ only in Fig. 7. From the definition of G-RNS, the input is multiplied by the constants

$$d_{ii} = \left\langle M_i^{-1}(-p^{-1}) \right\rangle_{m_i}, \tag{6}$$

$$e_{ii} = \left\langle M_i'^{-1} M^{-1} \right\rangle_{m'_i} \tag{7}$$

in base $B = \{m_1, \ldots, m_n\}$ and $B' = \{m'_1, \ldots, m'_n\}$, respectively. If the bases, B and B', are selected so that these

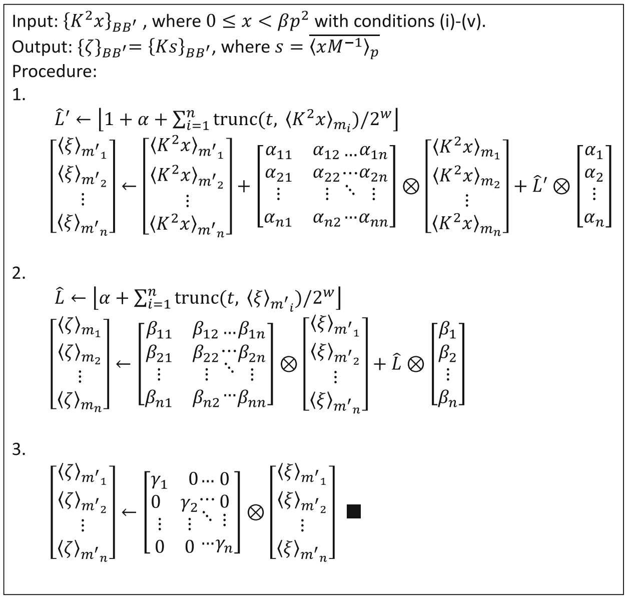
Fig. 7 From G-RNS to Q-RNS



constants are quadratic residues, each constant can be represented as a square of a constant K, as shown in Fig. 7 (left). We will, then, transfer the square root K from the present M-red to the previous M-red, integrating K onto the coefficient of the multiplication at the last steps (Fig. 7, right). As a result, outputs of previous M-reds are modified to Kx and Ky and their product is K^2xy, which is the same as the value immediately after steps 1 and 2 of Fig. 4. We call this new algorithm as Q-RNS M-red or simply as Q-RNS, using initials for quadratic residuosity. Q-RNS includes sQ-RNS which is directly derived from G-RNS and dQ-RNS in which a unit multiplication is replaced by the Montgomery multiplication.

Most past improvements in RNS M-red algorithms except [6] were optimization within one round of M-red execution. Our optimization for Q-RNS M-red is unique in that it transfers a square root[4] of a constant from a present round to a previous round.

As seen in Fig. 7, Q-RNS assumes that multiplication is carried out as preprocessing for the next M-red. This assumption ensures that the degree of K is 2. Let us consider possible degrees of K. All RNS M-red algorithms discussed in this paper use two bases, B and B', with the intent that these M-reds accommodate a number twice the length of what a single base can represent. This means M-red is designed not to accommodate a number with a degree more than or equal to 3. If the degree of K is 1 or 0, we could cope with such cases by multiplying K or K^2 by the input. Even if such cases should occur, the computation amount would be the same as Fig. 7 (left).



```
Input: ${K^2x}_{BB'}$ , where $0 \le x < \beta p^2$ with conditions (i)-(v).
Output: ${\zeta}_{BB'} = {Ks}_{BB'}$, where $s = \overline{\langle xM^{-1}\rangle_p}$
Procedure:
1.
    $\hat{L}' \leftarrow \lfloor 1 + \alpha + \sum_{i=1}^{n} \mathrm{trunc}(t,\ \langle K^2x\rangle_{m_i})/2^w \rfloor$
    $\begin{bmatrix} \langle\xi\rangle_{m'_1} \\ \langle\xi\rangle_{m'_2} \\ \vdots \\ \langle\xi\rangle_{m'_n} \end{bmatrix} \leftarrow \begin{bmatrix} \langle K^2x\rangle_{m'_1} \\ \langle K^2x\rangle_{m'_2} \\ \vdots \\ \langle K^2x\rangle_{m'_n} \end{bmatrix} + \begin{bmatrix} \alpha_{11} & \alpha_{12} \dots \alpha_{1n} \\ \alpha_{21} & \alpha_{22} \cdots \alpha_{2n} \\ \vdots & \vdots \ddots \vdots \\ \alpha_{n1} & \alpha_{n2} \cdots \alpha_{nn} \end{bmatrix} \otimes \begin{bmatrix} \langle K^2x\rangle_{m_1} \\ \langle K^2x\rangle_{m_2} \\ \vdots \\ \langle K^2x\rangle_{m_n} \end{bmatrix} + \hat{L}' \otimes \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix}$
2.
    $\hat{L} \leftarrow \lfloor \alpha + \sum_{i=1}^{n} \mathrm{trunc}(t,\ \langle\xi\rangle_{m'_i})/2^w \rfloor$
    $\begin{bmatrix} \langle\zeta\rangle_{m_1} \\ \langle\zeta\rangle_{m_2} \\ \vdots \\ \langle\zeta\rangle_{m_n} \end{bmatrix} \leftarrow \begin{bmatrix} \beta_{11} & \beta_{12} \dots \beta_{1n} \\ \beta_{21} & \beta_{22} \cdots \beta_{2n} \\ \vdots & \vdots \ddots \vdots \\ \beta_{n1} & \beta_{n2} \cdots \beta_{nn} \end{bmatrix} \otimes \begin{bmatrix} \langle\xi\rangle_{m'_1} \\ \langle\xi\rangle_{m'_2} \\ \vdots \\ \langle\xi\rangle_{m'_n} \end{bmatrix} + \hat{L} \otimes \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix}$
3.
    $\begin{bmatrix} \langle\zeta\rangle_{m'_1} \\ \langle\zeta\rangle_{m'_2} \\ \vdots \\ \langle\zeta\rangle_{m'_n} \end{bmatrix} \leftarrow \begin{bmatrix} \gamma_1 & 0 \dots 0 \\ 0 & \gamma_2 \cdots 0 \\ \vdots & \vdots \ddots \vdots \\ 0 & 0 \cdots \gamma_n \end{bmatrix} \otimes \begin{bmatrix} \langle\xi\rangle_{m'_1} \\ \langle\xi\rangle_{m'_2} \\ \vdots \\ \langle\xi\rangle_{m'_n} \end{bmatrix}$ ■
```

Fig. 8 sQ-RNS M-red algorithm

4.2 sQ-RNS algorithm

Figure 8 shows the sQ-RNS algorithm—the initial "s" indicating a single-level rather than double-level Montgomery—a technique proposed in [13]. sQ-RNS is basically derived according to the procedure shown in Fig. 7 with a small extra optimization.

The constants d_{ii} and e_{ii}, defined by Eqs. (6) and (7), are the diagonal elements in steps 1 and 2 of G-RNS. Quadratic residuosity of these constants is key to the design of Q-RNS. The square root of d_{ii} would yield $\left\langle M_i^{-1/2}(-1)^{1/2}p^{-1/2}\right\rangle_{m_i}$. The factor $(-1)^{1/2}$ requires that (-1) should be a quadratic residue modulo m_i. To reduce constraints on base B even a bit, a factor $\langle -1\rangle_{m_i}$ in d_{ii} is moved to the base extension matrix a'_{ij}. As a result, the constant K^2 is defined by

$$\left\langle K^2\right\rangle_{m_i} = \langle -d_{ii}\rangle_{m_i} = \left\langle M_i^{-1}p^{-1}\right\rangle_{m_i},$$

$$\left\langle K^2\right\rangle_{m'_i} = e_{ii} = \left\langle M_i'^{-1}M^{-1}\right\rangle_{m'_i}.$$

The new requirement for base B is that values on the right-hand side of the above equations must be quadratic residues. For a given p and $i, j \in [1, n]$, we can describe the requirement using the function QR as follows:

$$\prod_{i \neq j} \mathrm{QR}(m_j, m_i) \prod_{\forall i} \mathrm{QR}(p, m_i) \times \prod_{i \neq j} \mathrm{QR}(m'_j, m'_i) \prod_{\forall i, j} \mathrm{QR}(m_j, m'_i) = 1 \tag{8}$$

If Eq. (8) holds, there exists a coefficient K defined by the following equations and Q-RNS is properly defined.

$$\langle K\rangle_{m_i} = \left\langle M_i^{-1/2}p^{-1/2}\right\rangle_{m_i}$$
$$\langle K\rangle_{m'_i} = \left\langle M_i'^{-1/2}M^{-1/2}\right\rangle_{m'_i}$$

The computation of $\hat{L}'$ at step 1 is also modified due to the transfer of the factor $\langle -1\rangle_{m_i}$. Before the transfer, it was $\hat{L} \leftarrow \lfloor 0 + \sum_{i=1}^{n} \mathrm{trunc}(t, \left\langle K^2x\right\rangle_{m_i})/2^w\rfloor$. This is replaced by

$$\hat{L}' \leftarrow \left\lfloor 1 + \alpha + \sum_{i=1}^{n} \mathrm{trunc}(t, \left\langle K^2x\right\rangle_{m_i})/2^w \right\rfloor.$$

Note here that the offset value changes from 0 to $(1 + \alpha)$, which compensates for the effect of transfer of the factor $\langle -1\rangle_{m_i}$. Derivation of the new formula is explained in

[4] If more than one square root of the constant exists, either is useful to construct Q-RNS M-Red.

"Appendix A". This also makes the constant a'_i negative, as shown in the next paragraph.

In Fig. 8, input and output of the algorithm are $\left\{K^2x\right\}_{BB'}$ and $\{Ks\}_{BB'}$, respectively, and the elements in each matrix are defined from those of G-RNS as follows:

$$\begin{aligned}
\alpha_{ij} &= \left\langle -a'_{ij} \right\rangle_{m'_i} = \left\langle -M_i'^{-1} m_j^{-1} p \right\rangle_{m'_i} \\
\alpha_i &= \left\langle -a'_i \right\rangle_{m'_i} = \left\langle M_i'^{-1} p \right\rangle_{m'_i} \\
\beta_{ij} &= \left\langle K b_{ij} \right\rangle_{m_i} = \left\langle K M'_j \right\rangle_{m_i} = \left\langle M_i^{-1/2} p^{-1/2} M'_j \right\rangle_{m_i} \\
\beta_i &= \left\langle K b_i \right\rangle_{m_i} = \left\langle K(-M') \right\rangle_{m_i} = \left\langle -M_i^{-1/2} p^{-1/2} M' \right\rangle_{m_i} \\
\gamma_i &= \left\langle K c_{ii} \right\rangle_{m'_i} = \left\langle K M'_i \right\rangle_{m'_i} = \left\langle M_i'^{1/2} M^{-1/2} \right\rangle_{m'_i}
\end{aligned}$$

The constants β_{ij}, β_i, and γ_i are each multiplied by K, because they correspond to the explanatory constants h_1 through h_{2n} in Fig. 7 (right).

Note that steps 2 and 3 in Fig. 8 can be carried out simultaneously. Therefore, the computation time of sQ-RNS can be estimated as comparable with twice that of the matrix multiplication. The number of unit multiplications is $(2n^2 + n)$, which is the minimum among all previously proposed RNS M-red algorithms.

4.3 dQ-RNS algorithm

The double-level Montgomery is a technique proposed in [13]. It replaces the standard modular unit multiplication in RNS M-red with single-word Montgomery multiplication. dQ-RNS in Fig. 9 is derived by applying this technique to sQ-RNS. Using Montgomery multiplication $\otimes_M$ removes the requirement $\bar{\mu} < 0.5$, due to its special modulus for fast reduction. Without this requirement, we can take square numbers $m_i = \sigma_i^2$, as base elements which may violate the condition, $\bar{\mu} < 0.5$. In this approach, the condition for quadratic residue becomes very simple as

$$\prod_{\forall i} \mathrm{QR}(p, m_i) = 1 \tag{9}$$

since $m_i = {\sigma_i}^2$ automatically satisfies $\mathrm{QR}(m_i, m_j) = 1$. Therefore, it is expected that bases can be found efficiently for a wider range of base size n.

The elements of matrices in dQ-RNS are defined from those of sQ-RNS with the modification by coefficient $2^{kw/2}$ (for $k = 1, 2, 3$), which is represented by a symbol with a dot.

$$\begin{aligned}
\dot{\alpha}_{ij} &= \left\langle 2^w \alpha_{ij} \right\rangle_{m'_i} \\
\dot{\alpha}_i &= \left\langle 2^w \alpha_i \right\rangle_{m'_i}
\end{aligned}$$

Input: $\{K^2x\}_{BB'}$, where $0 \le x < \beta p^2$ with conditions (i)-(v).
Output: $\{\dot{\zeta}\}_{BB'} = \left\{2^{w/2}Ks\right\}_{BB'}$, where $s = \overline{\langle xM^{-1}\rangle_p}$
Procedure:
1.

$$\hat{L}' \leftarrow \left\lfloor 1 + \alpha + \textstyle\sum_{i=1}^{n} \mathrm{trunc}(t,\ \langle K^2x\rangle_{m_i})/2^w \right\rfloor$$

$$\begin{bmatrix} \langle\xi\rangle_{m'_1} \\ \langle\xi\rangle_{m'_2} \\ \vdots \\ \langle\xi\rangle_{m'_n} \end{bmatrix} \leftarrow \begin{bmatrix} \langle K^2x\rangle_{m'_1} \\ \langle K^2x\rangle_{m'_2} \\ \vdots \\ \langle K^2x\rangle_{m'_n} \end{bmatrix} + \begin{bmatrix} \dot{\alpha}_{11} & \dot{\alpha}_{12} \dots \dot{\alpha}_{1n} \\ \dot{\alpha}_{21} & \dot{\alpha}_{22} \cdots \dot{\alpha}_{2n} \\ \vdots & \vdots \ \ddots \ \vdots \\ \dot{\alpha}_{n1} & \dot{\alpha}_{n2} \cdots \dot{\alpha}_{nn} \end{bmatrix} \otimes_M \begin{bmatrix} \langle K^2x\rangle_{m_1} \\ \langle K^2x\rangle_{m_2} \\ \vdots \\ \langle K^2x\rangle_{m_n} \end{bmatrix} + \hat{L}' \otimes_M \begin{bmatrix} \dot{\alpha}_1 \\ \dot{\alpha}_2 \\ \vdots \\ \dot{\alpha}_n \end{bmatrix}$$

2.

$$\hat{L} \leftarrow \left\lfloor \alpha + \textstyle\sum_{i=1}^{n} \mathrm{trunc}(t,\ \langle\xi\rangle_{m'_i})/2^w \right\rfloor$$

$$\begin{bmatrix} \langle\dot{\zeta}\rangle_{m_1} \\ \langle\dot{\zeta}\rangle_{m_2} \\ \vdots \\ \langle\dot{\zeta}\rangle_{m_n} \end{bmatrix} \leftarrow \begin{bmatrix} \dot{\beta}_{11} & \dot{\beta}_{12} \dots \dot{\beta}_{1n} \\ \dot{\beta}_{21} & \dot{\beta}_{22} \cdots \dot{\beta}_{2n} \\ \vdots & \vdots \ \ddots \ \vdots \\ \dot{\beta}_{n1} & \dot{\beta}_{n2} \cdots \dot{\beta}_{nn} \end{bmatrix} \otimes_M \begin{bmatrix} \langle\xi\rangle_{m'_1} \\ \langle\xi\rangle_{m'_2} \\ \vdots \\ \langle\xi\rangle_{m'_n} \end{bmatrix} + \hat{L} \otimes_M \begin{bmatrix} \dot{\beta}_1 \\ \dot{\beta}_2 \\ \vdots \\ \dot{\beta}_n \end{bmatrix}$$

3.

$$\begin{bmatrix} \langle\dot{\zeta}\rangle_{m'_1} \\ \langle\dot{\zeta}\rangle_{m'_2} \\ \vdots \\ \langle\dot{\zeta}\rangle_{m'_n} \end{bmatrix} \leftarrow \begin{bmatrix} \dot{\gamma}_1 & 0 \dots 0 \\ 0 & \dot{\gamma}_2 \cdots 0 \\ \vdots & \vdots \ \ddots \ \vdots \\ 0 & 0 \cdots \dot{\gamma}_n \end{bmatrix} \otimes_M \begin{bmatrix} \langle\xi\rangle_{m'_1} \\ \langle\xi\rangle_{m'_2} \\ \vdots \\ \langle\xi\rangle_{m'_n} \end{bmatrix} \blacksquare$$

Fig. 9 dQ-RNS M-red algorithm

$$\begin{aligned}
\dot{\beta}_{ij} &= \left\langle 2^{3w/2} \beta_{ij} \right\rangle_{m_i} \\
\dot{\beta}_i &= \left\langle 2^{3w/2} \beta_i \right\rangle_{m_i} \\
\dot{\gamma}_i &= \left\langle 2^{3w/2} \gamma_i \right\rangle_{m'_i}
\end{aligned}$$

A variable $\{\zeta\}_{BB'}$ is also modified as

$$\{\dot{\zeta}\}_{BB'} = \left\{2^{w/2}\zeta\right\}_{BB'}.$$

If we take an even-valued w, these elements are always well-defined. Constants, $\left\{K^2\right\}_{BB'}$ and $\{K\}_{BB'}$ are the same as in sQ-RNS. Consequently, variables $\left\{K^2x\right\}_{BB'}$ and $\{\xi\}_{B'}$ are unchanged from sQ-RNS, leaving the formulae for $\hat{L}'$ and $\hat{L}$ unchanged.

The number of unit multiplications of dQ-RNS is also $(2n^2 + n)$. It should be noted that the unit multiplication in this case is the Montgomery multiplication.

4.4 Base search

A base search experiment is carried out for a given modulus p to find RNS bases that satisfy the requirement for quadratic residuosity. To avoid bias, we use five NIST primes [24] and one for Curve25519 [25], which are defined as common moduli for Elliptic Curve Cryptography. The NIST primes are called P-192, P-224, P-256, P-384, and P-521, with numbers representing the bit size of each prime. The prime for Curve25519 is defined as $2^{255} - 19$.

Experiment for sQ-RNS:

We search for bases satisfying Eq. (8) using the following search algorithm.

Table 1 Example base for sQ-RNS

Prime	w	$\mu_i = 2^w - m_i$				$\mu'_i = 2^w - m'_i$				max $\log_2 \mu$	max $\bar{\mu}$
NIST P-192	50	27	117	351	951	1163	2567	2855	8543	13.0	0.26
NIST P-224	58	57	63	147	447	27	731	3807	7403	12.9	0.22
NIST P-256	65	535	751	3219	8031	49	979	2191	11,335	13.5	0.21
NIST P-384	98	51	855	4343	52,155	117	831	1571	1827	15.7	0.16
NIST P-521	132	347	363	527	38,835	725	6647	11,535	38,679	15.2	0.12
Curve25519	65	535	2191	3219	8031	49	751	979	11,335	13.5	0.21

Search algorithm 1:

1. Let candidates be an ordered sequence of prime numbers in the form $c_i = 2^w - \mu_i$, where $\mu_i > \mu_j > 0$ for $i > j$. The search is done in a smaller-index-first manner.
2. Pool $= \{c_i | \mathrm{QR}(c_i, c_j) \cdot \mathrm{QR}(c_j, c_i) = 1$ for $i \neq j\}$.
3. $B = \{c_i | c_i \in \text{Pool} \wedge \mathrm{QR}(p, c_i) = 1 \wedge |B| = n\}$.
4. $B' = \{c_i | c_i \in \text{Pool} \wedge c_i \notin B \wedge |B'| = n\}$.

Since we choose the candidates from among prime numbers, they all satisfy the condition that they must be mutually prime. This also makes it easier to determine the quadratic residuosity.

Table 1 presents the search results for $n = 4$. The rightmost column shows that these bases satisfy the condition $\bar{\mu} < 0.5$.

Experiment for dQ-RNS:

We apply the following search algorithm, which generates bases satisfying the condition given by Eq. (9).

Search algorithm 2:

1. Let seeds be an ordered sequence of odd numbers in the form $\sigma_i = 2^{w/2} - \nu_i$, where $\nu_i > \nu_j > 0$ for $i > j$. The search is done in a smaller-index-first manner.
2. Pool $= \{\sigma_i |\, \gcd(\sigma_i, \sigma_j) = 1$ for $i \neq j\}$.
3. $B = \{\sigma_i^2 | \sigma_i \in \text{Pool} \wedge \mathrm{QR}(p, \sigma_i) = 1 \wedge |B| = n\}$.
4. $B' = \{\sigma_i^2 | \sigma_i \in \text{Pool} \wedge \sigma_i^2 \notin B \wedge |B'| = n\}$.

As a lemma, if $\mathrm{QR}(p, \sigma_i) = 1$, then $\mathrm{QR}(p, {\sigma_i}^2) = 1$ (see "Appendix B"). For the base elements found by algorithm 2, it holds that $\bar{\mu} > 0.5$, since

$$\sigma_i^2 = (2^{w/2} - \nu_i)^2 = 2^w - \nu_i 2^{w/2+1} + \nu_i^2.$$

Figure 10 shows the search results for $\alpha = 0.5$ and $\nu \geq 2$, the degree of laziness. The search succeeds for (n, w) plotted in the figure, although the graph P-256 is almost hidden behind that of C25519. The search fails when $\max(e_1, e_2)$ exceeds 0.5 and violates condition (iii) $\max(e_1, e_2) \leq \alpha$ in Sect. 3.1.2. The lower bounds of word length w for success are 22, 24, 24, 26, 28, and 24 bits for P-192, P-224, P-256, P-384, P-512, and Curve25519, respectively. The lower bound t_0 for necessary bit length for approximation ranges from 3 to 8. Therefore, it is possible to realize a compact computation circuit for $\hat{L}$ and $\hat{L}'$. Let N_1 be a number of seeds satisfying $\mathrm{QR}(p, \sigma_i) = 1$, and let N_0 be the number of all seeds generated until the algorithm halts. In our experiment, N_1/N_0 ranges from 0.30 to 0.43, which implies that the probability that $\mathrm{QR}(p, \sigma_i) = 1$ is near 0.3 for these primes. We also confirm that bases are efficiently found for some randomly chosen non-NIST primes.

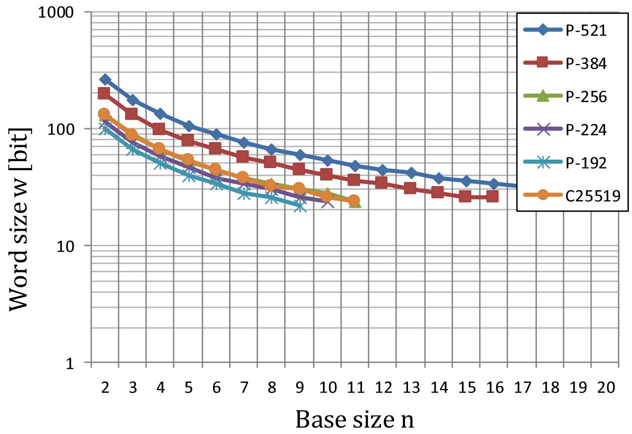


Fig. 10 Base search results for dQ-RNS

Experiments show that bases for dQ-RNS can be found unless the word size w is too small. For instance, $w \geq 22$ suffices for P-192. Since values less than 22 do not seem to be promising parameters for efficient hardware implementation, dQ-RNS has a sufficient range of word size selection. It is up to hardware designers to determine optimum sizes for specific Q-RNS applications.

5 Application to cryptography

We discuss several procedures necessary for RNS implementation of public-key cryptography, including Initialize, Finalize, transform to RNS representation (ToRNS, hereafter), and transform to Binary representations (ToBin, hereafter). We also provide formulae for bounds on degree

Table 2 Comparison of basic representations

	Algorithm	Representation	Note
(a)	Orthodox, Eq. (1)	$\langle x\rangle_p$	Typically, p is a large prime or a product of two primes.
(b)	M-red, Eq. (3)	$\overline{\langle xM\rangle_p}$	Setting $M = 2^l$ is efficient for a binary computer.
(c)	RNS M-red	$\{\overline{\langle xM\rangle_p}\}_{BB'}$	M is a product of elements in B
(d)	sQ-RNS M-red	$\{K\overline{\langle xM\rangle_p}\}_{BB'}$	Constant K is a function of (p, B, B').
(e)	dQ-RNS M-red	$\{2^{w/2}K\overline{\langle xM\rangle_p}\}_{BB'}$	The double-level Montgomery variant of (d)

of laziness and for relaxation of reduction within a channel. Although these issues were discussed in previous work, we are interested in the case of Q-RNS and exact expressions of bounds.

5.1 Basic representation

Table 2 shows the representations of computation result of each algorithm. Row (a) corresponds to the orthodox modular multiplication described by Eq. (1). Row (b) is for the standard Montgomery multiplication defined by Eq. (3), in which a constant $M = 2^l$ is multiplied by x. The bar symbol in row (b) means relaxation of the upper bound of reduction from p to $2p$. Row (c) represents conventional RNS M-reds other than R-RNS. This is an immediate transformation of (b) into an RNS representation with base B and B'. Rows (d) and (e) are for Q-RNS, derived from (c) by multiplying constant K and $2^{w/2}K$, respectively. The representation for the R-RNS algorithm is derived from (d) if we replace coefficient $\{K\}_{BB'}$ with $\{1\}_B \cup \{X\}_{B'}$, where $\langle X\rangle_{m'_i} = \left\langle M_i'^{-1}\right\rangle_{m'_i}$.

Montgomery developed an efficient reduction algorithm (b) by multiplying a constant R (here, M) by representation (a), whereas this paper proposes efficient RNS M-red algorithms (d) and (e) by multiplying constants K and $2^{w/2}K$ by representation (c). As a result, (d) and (e) are realized with fewer unit multiplications, and their structures are much simpler. As will be explained in the next subsection, we can embed multiplication by K or $2^{w/2}K$ into the Initialized process. We can also carry out the removal process of K or $2^{w/2}K$ in parallel with the Finalized process.

5.2 ToRNS and Initialize

If $\{\overline{\langle x\rangle_p}\}_{BB'}$ is given, Initialize for conventional RNS M-red is carried out as follows: First, a product

$$\{\overline{\langle x\rangle_p}\}_{BB'} \otimes \left\{\left\langle M^2\right\rangle_p\right\}_{BB'} \tag{10}$$

is computed, then the product is input to RNS M-red to obtain $\{\overline{\langle xM\rangle_p}\}_{BB'}$. A similar Initialize process can be defined for Q-RNS and applied to the result of ToRNS, which denotes the transformation from binary to RNS representation. To describe the concrete procedure, we assume the input is represented in binary as $\overline{\langle x\rangle_p} = \sum_{j=0}^{n-1} x_j 2^{jw}$ with $x_j \in [0, 2^w - 1]$.

For sQ-RNS:

$$X_0 \leftarrow x_{n-1}$$
$$X_j \leftarrow \langle X_{j-1}2^w + x_{n-1-j}\rangle_{m_i} \ (j = 1, \ldots, n-1).$$

This procedure outputs $X_{n-1} = \left\langle\overline{\langle x\rangle_p}\right\rangle_{m_i}$. The second step can be implemented with a single-word modular reduction. This matches the special modulus for fast reduction. By running this procedure with all moduli, we obtain $\{\overline{\langle x\rangle_p}\}_{BB'}$.

To initialize this variable, we first multiply it by a constant.

$$\{\overline{\langle x\rangle_p}\}_{BB'} \otimes \left\{K^2\left\langle M^2\right\rangle_p\right\}_{BB'} \tag{11}$$

Then, we input the product to sQ-RNS and obtain the basic representation $\{K\overline{\langle xM\rangle_p}\}_{BB'}$.

For dQ-RNS:

$$Y_0 \leftarrow x_0$$
$$Y_j \leftarrow \overline{\langle (x_j 2^w + Y_{j-1})2^{-w}\rangle_{m_i}} \ (j = 1, \ldots, n-1).$$

This procedure outputs $Y_{n-1} = \left\langle 2^{-(n-1)w}\overline{\langle x\rangle_p}\right\rangle_{m_i}$. The second step can be implemented with a single-word Montgomery reduction, which matches well with the double-level Montgomery. By running this procedure with all moduli, we obtain $\{2^{-(n-1)w}\overline{\langle x\rangle_p}\}_{BB'}$. It is possible to prepare the following lookup table for the single-word Montgomery reduction.

$$\left\langle -m_i{}^{-1}\right\rangle_{2^w}, \left\langle -m_i'^{-1}\right\rangle_{2^w} \ (i = 1, \ldots, n) \tag{12}$$

These constants are used in a similar way to the constant $(-p^{-1})$ mod R at step 1 in Fig. 2.

For Initialize, we first multiply a constant as follows.

$$\left\{2^{-(n-1)w}\overline{\langle x\rangle_p}\right\}_{BB'} \otimes_M \left\{2^{nw}K^2\left\langle M^2\right\rangle_p\right\}_{BB'} \tag{13}$$

Then, we input this product to dQ-RNS and obtain the basic representation $\left\{2^{w/2}K\overline{\langle xM\rangle_p}\right\}_{BB'}$.

5.3 Finalize and ToBin

In the conventional RNS M-red, the Finalize of the Montgomery reduction is carried out by inputting the following value to RNS M-red.

$$\left\{\overline{\langle xM\rangle_p}\right\}_{BB'} = \left\{\overline{\langle xM\rangle_p}\right\}_{BB'} \otimes \{1\}_{BB'}$$

Similarly, the Finalize for sQ-RNS and dQ-RNS is carried out by inputting the following values to the respective M-red algorithms.

sQ-RNS algorithm:

$$\left\{K\overline{\langle xM\rangle_p}\right\}_{BB'} \otimes \{K\}_{BB'}\,. \tag{14}$$

dQ-RNS algorithm:

$$\left\{2^{w/2}K\overline{\langle xM\rangle_p}\right\}_{BB'} \otimes_M \left\{2^{w/2}K\right\}_{BB'}\,. \tag{15}$$

Regardless of sQ-RNS or dQ-RNS, the products above are both $\left\{K^2\overline{\langle xM\rangle_p}\right\}_{BB'}$. Similarly, the intermediate results at step 1 of the corresponding Q-RNS M-reds are the same, namely,

$$\begin{bmatrix} \langle \xi\rangle_{m'_1} \\ \vdots \\ \langle \xi\rangle_{m'_n} \end{bmatrix} = \begin{bmatrix} \left\langle \overline{\langle x\rangle_p} M_1'^{-1}\right\rangle_{m'_1} \\ \vdots \\ \left\langle \overline{\langle x\rangle_p} M_n'^{-1}\right\rangle_{m'_n} \end{bmatrix}.$$

To compute binary representation, we need an additional subroutine, ToBin, shown in Fig. 11, the input of which is the intermediate result of step 1 above. From condition (v) in Sect. 3.1.2, it follows that

$$\overline{\langle x\rangle_p} < 2p \leq (1-\alpha)M'.$$

This meets the input requirement of ToBin. If the return value Z of ToBin is not less than p, it is best to carry out the final subtraction in the binary representation.

ToBin in Fig. 11 is derived for dQ-RNS from the one proposed in [9]. In [9], one of the moduli in base B is chosen as $m_1 = 2^w$, which is used as 2^w in Fig. 11. On the other hand, due to quadratic residuosity, it is not possible to use $m_1 = 2^w$ for Q-RNS. Therefore, ToBin needs $(n+1)$ more words in its lookup table in step 2 than were used in [9]. In addition, step 4 needs an n-word table, while step 3 needs no table.

In Fig. 11, unit multiplication is basically the Montgomery multiplication, although step 2 is an exception. Typical implementation of step 2 is to apply multiplication without reduction and take the lower w bits. Since steps 2 and 3 can be carried out at the same time, efficient implementation is possible.

```
ToBin($\{\xi\}_{B'}, B', \alpha$)
Required: $\langle\xi\rangle_{m'_i} = \langle sM_i'^{-1}\rangle_{m'_i}$, $0 \leq s < (1-\alpha)M'$
Ensure: $s = \sum_{j=0}^{n-1} z_j 2^{jw}$.
1. For j=0, n-1 do
2.     $\hat{L} \leftarrow \lfloor \alpha + \sum_{l=1}^{n} \mathrm{trunc}(t,\ \xi_l)/2^w \rfloor$
       $z_j \leftarrow [\langle M'_1\rangle_{2^w} \quad \ldots \quad \langle M'_n\rangle_{2^w}] \begin{bmatrix}\xi_1\\ \xi_2\\ \vdots\\ \xi_n\end{bmatrix} + \hat{L}\langle -M'\rangle_{2^w}$
3.
       $\begin{bmatrix}\zeta_1\\ \zeta_2\\ \vdots\\ \zeta_n\end{bmatrix} \leftarrow \mathrm{diag}[\langle 1\rangle_{m'_1} \quad \cdots \quad \langle 1\rangle_{m'_n}] \otimes_M \begin{bmatrix}\xi_1\\ \xi_2\\ \vdots\\ \xi_n\end{bmatrix}$
4.
       $\begin{bmatrix}\xi_1\\ \xi_2\\ \vdots\\ \xi_n\end{bmatrix} \leftarrow \begin{bmatrix}\zeta_1\\ \zeta_2\\ \vdots\\ \zeta_n\end{bmatrix} + \begin{bmatrix}\langle -M_1'^{-1}\rangle_{m'_1}\\ \vdots\\ \langle -M_n'^{-1}\rangle_{m'_n}\end{bmatrix} \otimes_M z_j$
5. End for
6. Return $(z_0, z_1, \ldots, z_{n-1})$ ■
```

Fig. 11 ToBin transform for dQ-RNS

To obtain ToBin for sQ-RNS, steps 3 and 4 in Fig. 11 should be respectively modified by the following parts of the equation. These need a $2n$-word table for sQ-RNS in addition to the conventional RNS M-reds.

$$\mathrm{diag}\left[\left\langle 2^{-w}\right\rangle_{m'_1}, \ldots, \left\langle 2^{-w}\right\rangle_{m'_n}\right] \otimes,$$

$$\begin{bmatrix} \left\langle -M_1'^{-1}2^{-w}\right\rangle_{m'_1} \\ \vdots \\ \left\langle -M_n'^{-1}2^{-w}\right\rangle_{m'_n} \end{bmatrix} \otimes\,.$$

5.4 Degree of laziness

We will represent the upper bound for degree of laziness ν described with Q-RNS parameters. As a typical lazy reduction, we consider the product sum.

Input:

$$\sum_{i=1}^{\nu} \left\{2^{w/2}K\overline{\langle x_i M\rangle_p}\right\}_{BB'} \otimes_M \left\{2^{w/2}K\overline{\langle y_i M\rangle_p}\right\}_{BB'}$$

$$= \left\{K^2 \sum_{i=1}^{\nu} \overline{\langle x_i M\rangle_p} \cdot \overline{\langle y_i M\rangle_p}\right\}_{BB'}\,.$$

Output:

$$\left\{2^{w/2}K\overline{\left\langle M\cdot\sum_{i=1}^{\nu}x_iy_i\right\rangle_p}\right\}_{BB'}.$$

If $4\nu\le\beta$ holds, then

$$\sum_{i=1}^{\nu}\overline{\langle x_iM\rangle_p}\cdot\overline{\langle y_iM\rangle_p}\le\nu(2p)^2=4\nu p^2\le\beta p^2.$$

This satisfies the upper bound on input for dQ-RNS. On the other hand, from condition (iv) in Sect. 3.1.2, it holds that

$$\beta p\le(1-\alpha)M.$$

Combined with $4\nu\le\beta$, this leads to

$$\nu\le\beta/4\le(1-\alpha)M/4p.$$

Thus, we can conclude that

$$\nu_{\max}=\left\lfloor\frac{(1-\alpha)M}{4p}\right\rfloor.$$

Here, α may be replaced by e_1. The same formula can be applied to sQ-RNS.

5.5 Relaxation of reduction within channel

So far, we have assumed that the modular reduction in a unit multiplication is carried out strictly; that is, its result is always less than m_i. It is, however, known for the Montgomery reduction in Fig. 2 that relaxation of reduction is effective toward avoiding a conditional branch due to the final subtraction, thus making the implementation simpler. It may be also possible to apply this idea to modular reduction at a unit operation.

Let $\overline{\overline{\langle\xi\rangle_m}}$ denote the δ-relax of $\langle\xi\rangle_m$ defined as $\overline{\overline{\langle\xi\rangle_m}}\equiv\langle\xi\rangle_m\ (\mathrm{mod} m)$ and $\overline{\overline{\langle\xi\rangle_m}}\in[0,\delta m)$, where $\delta\ge 1$. A special case $\delta=1$ means strict reduction. If we introduce δ-relax to RNS M-red, it affects the representation of error bound e_1, which is modified as

$$\begin{aligned}\widetilde{e_1}&=n(2^{-t}-2^{-w})+\frac{1}{2^w}\sum_{i=1}^{n}\left(\delta-\frac{1}{m_i}\right)\mu_i.\\&=e_1+\frac{\delta-1}{2^w}\sum_{i=1}^{n}\mu_i\end{aligned}$$

Similar modification is required for e_2.

By replacing e_1 and e_2 with $\widetilde{e_1}$ and $\widetilde{e_2}$ in conditions (i)–(v) in Sect. 3.1.2, the basic algorithm in Fig. 3 and all its variants including Q-RNSs output a correct result for δ-relax variables. Note that the ranges of $\hat{L}$ and $\hat{L}'$ change when δ-relax is applied. In the strict reduction case, their ranges are

$$\begin{aligned}&0\le\hat{L}\le n-1\\&1\le\hat{L}'\le n+1,\end{aligned}$$

whereas in the δ-relax case,

$$\begin{aligned}&0\le\widetilde{L}\le\delta n-1,\\&1\le\widetilde{L}'\le\delta n+1.\end{aligned}$$

Since the relaxation requires a wider bit length than w, there is a tradeoff between simple reduction and word size.

6 Comparison

6.1 Number of unit multiplications

Table 3 summarizes comparison of four conventional RNS M-red algorithms and two Q-RNS M-red algorithms. Among these, the proposed ones achieved the least number of unit multiplications. It should be noted that unit multiplication for dQ-RNS is Montgomery's, while other algorithms use standard modular multiplication. Note also that if n is small in C-RNS, there is a possibility that one can find base extension matrices with less computation. As for the requirements for base choice, the basic algorithms G-RNS and R-RNS pose the weakest requirements, while sQ-RNS poses the strongest. C-RNS and dQ-RNS fall somewhere in between. dQ-RNS has weaker requirements on the RNS base than does sQ-RNS, since it is possible to employ square numbers as elements of the bases.

As in conventional RNS M-reds, it is easy to implement sQ-RNS and dQ-RNS in parallel processing architecture due to RNS. Since sQ-RNS and dQ-RNS mostly consist of two matrix multiplications, these algorithms have more regularity and simplicity than do conventional ones. From past work, it is definite that Q-RNS can terminate in $(2n^2+n)/n=(2n+1)$ cycles if n processing units operate in parallel. Since multiplication previous to Q-RNS finishes in $2n/n=2$ cycles, the total cycles of Montgomery multiplication is $(2n+1+2\nu)$, where ν is the degree of laziness. Another possibility, though less likely, is that with (n^2+n) unit multipliers, Q-RNS finishes in two cycles. Although this seems theoretically possible, in practice there are several issues for elaboration, such as feasibility of fan-out n of registers and design of an efficient circuit for summing up the results from unit multipliers.

Bigou et al. proposed a method that consists of fewer unit multiplications than other RNS M-red algorithms,

Table 3 Comparison of RNS M-red

RNS M-red	Feature	# of unit mult.	Requirements to base	
Basic [3] (Fig. 3)	Straightforward	$2n^2 + 5n$	Weak	Mutually prime $\bar{\mu} < 0.5$
G-RNS [9] (Fig. 4)	Integrated lookup tables	$2n^2 + 3n$	Weak	Mutually prime $\bar{\mu} < 0.5$
C-RNS [15] (Fig. 5)	Special form of Base extension matrices	$\leq 2n^2 + 4n$	Medium	Mutually prime $\bar{\mu} < 0.5$ n is smaller
R-RNS [6] (Fig. 6)	Reorganized G-RNS	$2n^2 + 2n$	Weak	Mutually prime $\bar{\mu} < 0.5$
sQ-RNS (Fig. 8)	Quadratic Residuosity (QR)	$2n^2 + n$	Strong	Mutually prime $\bar{\mu} < 0.5$ QR by Eq. (8)
dQ-RNS (Fig. 9)	QR and The double-level Montgomery	$2n^2 + n$	Medium	Mutually prime QR by Eq. (9)

Table 4 Number of memory words

	G-RNS [9]	R-RNS [6]	sQ-RNS	dQ-RNS
Main body[*1]	$2n^2 + 5n$	$2n^2 + 4n$	$2n^2 + 3n$	$2n^2 + 3n$
Base B, B'	$+2n$	$+2n$	$+2n$	$+2n$
ToRNS	$+0$	$+0$	$+0$	$+2n$[*2]
Initialize[*3]	$+2n$	$+2n$	$+2n$	$+2n$
Finalize[*4]	$+0$	$+n$	$+2n$	$+2n$
ToBin[*5]	$+2n$	$+2n$	$+(3n + 1)$	$+(2n + 1)$
Total	$2n^2 + 11n$	$2n^2 + 11n$	$2n^2 + 12n + 1$	$2n^2 + 13n + 1$

[*1] The Main body shows the necessary number of words, and other rows show numbers in addition to the Main body.
[*2] Table for Eq. (12).
[*3] Multipliers in Eqs. (10), (11), and (13), respectively.
[*4] Multipliers in Eqs. (14) and (15), respectively.
[*5] Refer to Sect. 5.3. For G-RNS, $m_1 = 2^w$ is assumed

including Q-RNS, under the hypothesis that the modulus p and the product of base moduli M should satisfy a certain equation [12,14]. Although Q-RNS also poses quadratic residuosity conditions, their hypothesis is much stronger than that of Q-RNS. Actually, no base exists for NIST primes [14]. In their algorithm, it should be preferable to fix the base first and then determine p under the hypothesis. On the other hand, we can find bases with very high probability not only for NIST primes but also for other primes. Therefore, the discussion in this paper does not include their algorithm for comparison.

6.2 Size of lookup table

Table 4 shows comparison of the lookup table size necessary for the four algorithms, G-RNS, R-RNS, sQ-RNS, and dQ-RNS. Compared with G-RNS and R-RNS, sQ-RNS and dQ-RNS need only $(n + 1)$ and $(2n + 1)$ words of extra memory, respectively. With such little additional memory, Q-RNSs provide sufficient merit regarding reduction in the number of multiplications and simplicity of the algorithm. A toy example of parameters is shown in “Appendix C”.

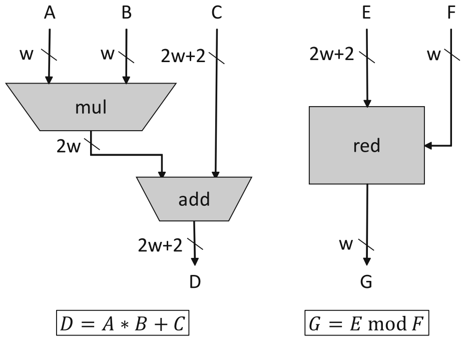


Fig. 12 A set of operation units

6.3 FPGA implementation

We have implemented sQ-RNS on FPGA with parameters $n = 4$, $w = 65$ and P-256 as a modulus. We have also implemented R-RNS for comparison.

Figure 12 shows the main operation units, a multiply-and-add unit and a modular reduction unit, where the latter carries out the fast reduction algorithm presented in Sect. 2.6. Let c_m and c_r be the clock cycles required to carry out these operations, respectively. In our implementation, it follows that $c_m = 1$ and $c_r = 2$. n sets of these operation units are prepared. We use almost the same configuration for both sQ-RNS and R-RNS.

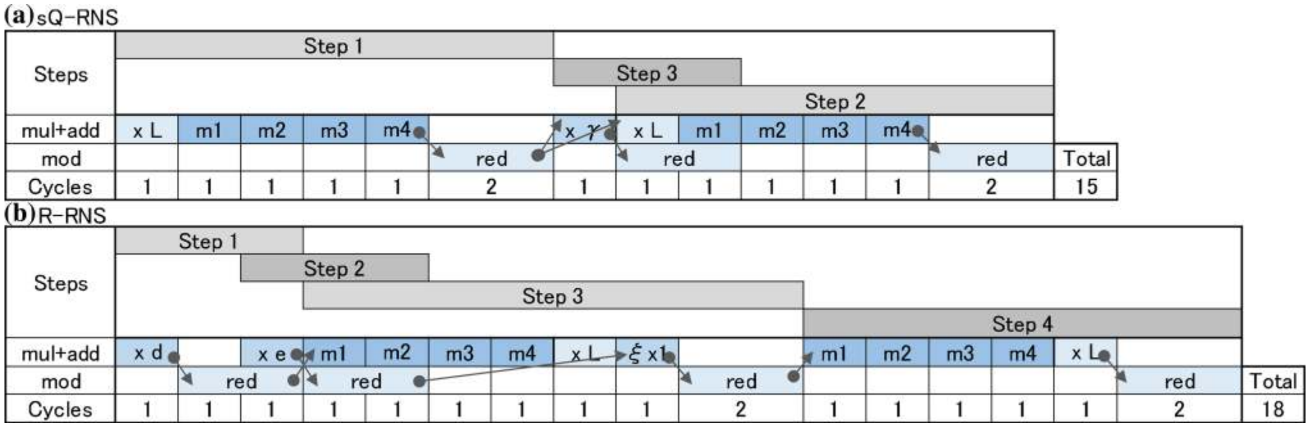


Fig. 13 Operation diagrams for **a** sQ-RNS and **b** R-RNS

Figure 13 shows the operation diagram for both implementations. In each implementation, four sets of operation units run in parallel and the diagram show the operation of each unit. For sQ-RNS, first 7 cycles correspond to step 1 of the algorithm, followed by 3 cycles of step 3. Step 2 overwraps step 3 with 1 cycle of delay. sQ-RNS completes in 15 cycles. A similar diagram is shown for R-RNS with 18 cycles. A 3-cycle difference is caused by steps 1 and 2 of R-RNS, which are unnecessary in sQ-RNS. An approximately 17% reduction in clock cycles is achieved by sQ-RNS compared with R-RNS. If $c_m \leq c_r$ holds, we can derive the equations representing clock cycles from Fig. 13 as:

$$\text{sQ-RNS: } (2n+3)c_m + 2c_r$$
$$\text{R-RNS: } (2n+4)c_m + 3c_r .$$

Table 5 Synthesis results

	(a) sQ-RNS	(b) R-RNS
Word length w	65	65
Base size n	4	4
Number of operation units	4	4
Clock cycles	15	18
Max. frequency(MHz)	139.5	142.7
Hardware components		
LUT	4076	4247
FF	2104	2329
DSP	84	84
FPGA device	Kintex® UltraScale+™	
Compiler	Vivado® 2018.1	

Table 5 summarizes the results of FPGA implementations. Both (a) and (b) consume almost the same hardware resources specific to FPGA, such as look up table (LUT), flip-flop (FF), and digital signal processing (DSP). In the implementation, we did not apply hand tuning to the multiplier and adder. Namely, these components are synthesized automatically by the compiler. Further optimization may be possible.

7 Conclusion

This paper proposed new RNS Montgomery reduction algorithms, namely, sQ-RNS and dQ-RNS, which are derived by posing quadratic residuosity requirements on RNS bases. They achieve fewer number of unit multiplications than all previously proposed algorithms. The size of the lookup tables they use is comparable with conventional ones. Improvement over the R-RNS algorithm was confirmed with FPGA implementations. Since the proposed algorithms have more regularity and symmetry than do conventional ones, it may be worth studying software implementations for multi-core processors. Another topic for future study is improvement to the two base search algorithms proposed in this paper.

Appendix A: Derivation of $\hat{L}'$

In order to derive the formula for $\hat{L}'$ of Q-RNS, we start from the basic RNS M-red in Fig. 3. From step 1, we obtain

$$\{q\}_B = \left\{x(-p)^{-1}\right\}_B .$$

From step 2, we have

$$\theta_i = \left\langle q {M_i}^{-1} \right\rangle_{m_i} = \left\langle x(-p)^{-1} {M_i}^{-1} \right\rangle_{m_i}.$$

Let $\varphi_i = \langle K^2 x \rangle_{m_i}$, then

$$\varphi_i = \left\langle (p)^{-1} {M_i}^{-1} x \right\rangle_{m_i}.$$

Thus, the following equation holds.

$$\theta_i = m_i - \varphi_i \tag{16}$$

Note that this transformation of variables will change the offset from 0 to $(1+\alpha)$, as described later in this proof. From the Chinese remainder theorem, there exists an integer L satisfying

$$q = \sum_{i=1}^{n} \theta_i M_i - LM.$$

Substituting Eq. (16) into the above equation leads to

$$q = \sum_{i=1}^{n} \varphi_i (-M_i) + (n - L)M.$$

Division by M on both sides results in

$$\sum_{i=1}^{n} \frac{\varphi_i}{m_i} = (n - L) - \frac{q}{M}. \tag{17}$$

Define tr_i as

$$tr_i = \mathrm{trunc}(t, \varphi_i).$$

Then

$$tr_i \le \varphi_i \le tr_i + 2^{w-t} - 1$$

holds. Following Appendix B of [23], we can derive the equation

$$f(q) - e_1 \le \tilde{f}(q) \le f(q), \tag{18}$$

where f and $\tilde{f}$ are defined as

$$f(q) = \sum_{i=1}^{n} \frac{\varphi_i}{m_i}, \; \tilde{f}(q) = \sum_{i=1}^{n} \frac{tr_i}{2^w}.$$

By adding $(1+\alpha)$ on both sides of Eq. (18) and considering Eq. (17), $e_1 \le \alpha < 1$ and $0 \le q/M < 1$, we can obtain the relationship

$$(n - L) < 1 + \alpha + \tilde{f}(q) < (n - L) + 2.$$

If we define $\hat{L}' = \left\lfloor 1 + \alpha + \tilde{f}(q) \right\rfloor$, we can derive the equation

$$\hat{L}' = (n - L) + u,$$

where $u \in \{0, 1\}$. Letting $\hat{q}$ be an approximation of q as computed by $\hat{L}'$, it follows that

$$\begin{aligned}\hat{q} &= \sum_{i=1}^{n} \varphi_i (-M_i) + \hat{L}' M \\ &= q + uM.\end{aligned}$$

This shows that $\hat{q}$ has the same range as q', which is the base extension result at step 3 in Fig. 3. Finally, we obtain the following formula computing $\langle \hat{q} \rangle_{m'_i}$ from $\{\varphi\}_B$.

$$\langle \hat{q} \rangle_{m'_i} = \left\langle \sum_{i=1}^{n} \langle -M_i \rangle_{m'_i} \varphi_i + \hat{L}' \langle M \rangle_{m'_i} \right\rangle_{m'_i}.$$

Appendix B: Proof of lemma

Lemma 1 *For given mutually prime odd numbers p and σ_i, if* $\mathrm{QR}(p, \sigma_i) = 1$, *then* $\mathrm{QR}(p, {\sigma_i}^2) = 1$.

Proof From assumption $\mathrm{QR}(p, \sigma_i) = 1$, there exists an integer $x_0 \in [0, \sigma_i)$ which satisfies $p \equiv {x_0}^2 (\mathrm{mod} \sigma_i)$. In addition, it is true that $\gcd(x_0, \sigma_i) = 1$, because otherwise, $\gcd(p, \sigma_i) \ne 1$, which contradicts the assumption. Letting $p' = (p - {x_0}^2)/\sigma_i$, p' is an integer. Since $\gcd(2, \sigma_i) = 1$, the following x_1 can be defined.

$$x_1 = p'(2x_0)^{-1} \bmod \sigma_i.$$

This means that there is an integer k satisfying

$$2x_1 x_0 = p' + k\sigma_i.$$

This equation will be used later. Now, we define a new integer x as

$$x = x_1 \sigma_i + x_0.$$

Let us evaluate $\Delta = p - x^2$ as follows:

$$\begin{aligned}\Delta &= p - (x_1\sigma_i + x_0)^2 = p - (x_1\sigma_i)^2 - 2x_1x_0\sigma_i - {x_0}^2 \\ &= p - (x_1\sigma_i)^2 - (p' + k\sigma_i)\sigma_i - {x_0}^2\end{aligned}$$

$$= p - (x_1\sigma_i)^2 - \left(\frac{p - {x_0}^2}{\sigma_i} + k\sigma_i\right)\sigma_i - {x_0}^2$$
$$= -({x_1}^2 + k){\sigma_i}^2$$

This shows that $\Delta \equiv 0 \pmod{\sigma_i^2}$, in other words, that $p \equiv x^2 \pmod{\sigma_i^2}$. Therefore, $\mathrm{QR}(p, {\sigma_i}^2) = 1$ holds. □

Appendix C: Toy example for dQ-RNS

We present a toy example of parameters for the dQ-RNS algorithm. Since n is small, we can find these parameters using search algorithm 1 in Sect. 4.4, rather than search algorithm 2. Therefore, every base element is a prime number and not a square number. As a result, $\bar{\mu} < 0.5$ holds and we can use the same bases B and B' for the computation of sQ-RNS.
$p = 288230376151711813$ (58 bits)
$w = 32$ and $n = 2$.

$$\begin{aligned}
B &= \{4294967291,\ 4294967189\} = \{2^{32} - 5,\ 2^{32} - 107\}\\
B' &= \{4294967161,\ 4294966661\} = \{2^{32} - 135,\ 2^{32} - 635\}\\
e_1 &= 2.6 \times 10^{-8}, e_2 = 1.8 \times 10^{-7}\\
t &\geq t_0 = 3 \text{ (bits)}\\
\bar{\mu} &= 0.30\\
\nu_{\max} &= 7\\
\delta &= 1
\end{aligned}$$

ToRNS:

$$\left[\left\langle -{m_1}^{-1}\right\rangle_{2^w}\ \left\langle -{m_2}^{-1}\right\rangle_{2^w}\right] = \left[3435973837\ 2368252995\right]$$
$$\left[\left\langle -{m'_1}^{-1}\right\rangle_{2^w}\ \left\langle -{m'_2}^{-1}\right\rangle_{2^w}\right] = \left[3785934135\ 804883635\right]$$

Initialize:

$$\left\{2^{nw}K^2\left\langle M^2\right\rangle_p\right\}_B = \left[2021288184\ 2297663181\right]$$
$$\left\{2^{nw}K^2\left\langle M^2\right\rangle_p\right\}_{B'} = \left[732735884\ 604521080\right]$$

Main body:

$$\begin{bmatrix}\dot{\alpha}_{11} & \dot{\alpha}_{12}\\ \dot{\alpha}_{21} & \dot{\alpha}_{22}\end{bmatrix} = \begin{bmatrix}2641058936 & 4285691761\\ 354704368 & 1204129832\end{bmatrix}$$
$$\begin{bmatrix}\dot{\alpha}_1\\ \dot{\alpha}_2\end{bmatrix} = \begin{bmatrix}259711200\\ 4169481193\end{bmatrix}$$
$$\begin{bmatrix}\dot{\beta}_{11} & \dot{\beta}_{12}\\ \dot{\beta}_{21} & \dot{\beta}_{22}\end{bmatrix} = \begin{bmatrix}2079821785 & 3360655186\\ 2733482470 & 3593946207\end{bmatrix}$$
$$\begin{bmatrix}\dot{\beta}_1\\ \dot{\beta}_2\end{bmatrix} = \begin{bmatrix}4088860008\\ 3523066947\end{bmatrix}$$
$$\begin{bmatrix}\dot{\gamma}_1 & 0\\ 0 & \dot{\gamma}_2\end{bmatrix} = \begin{bmatrix}1292050203 & 0\\ 0 & 129202610\end{bmatrix}$$

Finalize:

$$\left\{2^{w/2}K\right\}_B = \left[3932984385\ 3650095956\right]$$
$$\left\{2^{w/2}K\right\}_{B'} = \left[351404616\ 967078097\right]$$

ToBin:

$$\left[\left\langle M'_1\right\rangle_{2^w}\ \left\langle M'_2\right\rangle_{2^w}\right] = \left[4294966661\ 4294967161\right]$$
$$\left\langle -M'\right\rangle_{2^w} = 4294881571$$
$$\begin{bmatrix}\left\langle -{M'_1}^{-1}\right\rangle_{m'_1}\\ \left\langle -{M'_2}^{-1}\right\rangle_{m'_2}\end{bmatrix} = \begin{bmatrix}506806125\\ 3788160595\end{bmatrix}$$

References

1. Posch, K.C., Posch, R.: Modulo reduction in residue number systems. IEEE Trans. Parallel Distrib. Syst. **6**(5), 449–454 (1995)
2. Schwemmlein, J., Posch, K.C., Posch, R.: RNS-modulo reduction upon a restricted base value set and its applicability to RSA cryptography. Comput. Secur. **17**(7), 637–650 (1998)
3. Kawamura, S., Koike, M., Sano, F., Shimbo, A.: Cox-rower architecture for fast parallel Montgomery multiplication. In: EUROCRYPT2000, LNCS1807, pp. 523–538. Springer (2000)
4. Nozaki, H., Motoyama, M., Shimbo, A., Kawamura, S.: Implementation of RSA algorithm based on RNS Montgomery multiplication. In: CHES2001, LNCS2162, pp. 364–376. Springer (2001)
5. Bajard, J.-C., Imbert, L.: A full RNS implementation of RSA. IEEE Trans. Comput. (Brief Contrib.) **53**(6), 769–774 (2004)
6. Gandino, F., Lamberti, F., Paravati, G., Bajard, J.-C., Montuschi, P.: An algorithmic and architectural study of Montgomery exponentiation in RNS. IEEE Trans. Comput. **61**(8), 1071–1083 (2012)
7. Schinianakis, D.M., Kakarountas, A.P., Stouraitis, T.: A new approach to elliptic curve cryptography: an RNS architecture. In: Proceedings of IEEE MELECON 2006, May 16–19, Benalmadena (Malaga), Spain, pp. 1241–1245 (2006)
8. Schinianakis, D.M., Fournaris, A.P., Michail, H.E., Kakarountas, A.P., Souraitis, T.: An RNS implementation of an Fp elliptic curve point multiplier. IEEE Trans. Circuits Syst. **56**(6), 1202–1213 (2009)
9. Guillermin, N.: A high speed coprocessor for elliptic curve scalar multiplications over Fp. In: CHES2010, LNCS6225, pp. 48–64. Springer (2010)
10. Antão, S., Bajard, J.-C., Sousa, L.: RNS-based elliptic curve point multiplication for massive parallel architectures. Comput. J. **55**(5), 629–647 (2012)
11. Schinianakis, D.M., Souraitis, T.: Multifunction residue architectures for cryptography. IEEE Trans. Circuits Syst. **61**(4), 1156–1169 (2014)
12. Bigou, K., Tisserand, A.: RNS modular multiplication through reduced base extensions. In: ASAP, pp. 57–62. IEEE (2014)
13. Bajard, J.-C., Merkiche, N.: Double level Montgomery cox-rower architecture, new bounds. In: Smart Card Research and Advanced Applications (CARDIS), LNCS 8968, pp. 139–153. Springer (2015)
14. Bigou, K., Tisserand, A.: Single base modular multiplication for efficient hardware RNS implementations of ECC. In: CHES2015, LNCS9293, pp. 123–140. Springer (2015)
15. Cheung, R., Duquesne, S., Fan, J., Guillermin, N., Verbauwhede, I., Yao, G.: FPGA implementation of pairing using residue number

system and lazy reduction. In: CHES2011, LNCS6917, pp. 421–441. Springer (2011)
16. Yao, G.X., Fan, J., Cheung, R.C.C., Verbauwhede, I.: Faster pairing coprocessor architecture. In: Pairing 2012, LNCS 7708, pp. 160–176. Springer (2012)
17. Bigou, K., Tisserand, A.: Improving modular inversion in RNS using the plus-minus methods. In: CHES 2013, LNCS8086, pp. 233–249. Springer (2013)
18. Bajard, J.-C., Eynard, J., Merkiche, N., Plantard, T.: RNS arithmetic approach in lattice-based cryptography. In: 22nd IEEE Symposium on Computer Arithmetic (2015)
19. Gérard, B., Kammerer, J.-G., Merkiche, N.: Contribution to the design of RNS architecture. In: 22nd IEEE Symposium on Computer Arithmetic (2015)
20. Bajard, J.-C., Eynard, J., Merkiche, N.: Montgomery reduction within the context of residue number system arithmetic. Special Issue on Montgomery Arithmetic. J. Cryptogr. Eng. https://doi.org/10.1007/s13389-017-0154-9
21. Montgomery, P.L.: Modular multiplication without trial division. Math. Comput. **44**(170), 519–521 (1985)
22. Ananda Mohan, P.V.: Residue Number Systems—Theory and Applications. Birkhäuser. ISBN: 978-3-319-41383-9(2016)
23. Kawamura, S., Yonemura, T., Komano, Y., Shimizu, H.: Exact error bound of cox-rower architecture for RNS arithmetic. Cryptology ePrint Archive: Report 2016/266, March (2016). https://eprint.iacr.org/2016/266
24. Federal Information Processing Standards Publication: FIPS186-4 "Digital Signature Standard (DSS)." Appendix D, National Institute of Standards and Technology, July (2013)
25. Bernstein, D.J.: Curve25519: new Diffie-Hellman speed records. In: Public Key Cryptography—PKC 2006, LNCS 3958, pp. 207–228. Springer (2006)