RNS Arithmetic Approach in Lattice-based Cryptography Accelerating the ” Rounding-off ” Core Procedure

Jean-Claude Bajard, Julien Eynard, Nabil Merkiche, Thomas Plantard

To cite this version:

HAL Id: hal-01170577
https://hal.sorbonne-universite.fr/hal-01170577
Submitted on 1 Jul 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Abstract—Residue Number Systems (RNS) are naturally considered as an interesting candidate to provide efficient arithmetic for implementations of cryptosystems such as RSA, ECC (Elliptic Curve Cryptography), pairings, etc. More recently, RNS have been used to accelerate fully homomorphic encryption as lattice-based cryptography. In this paper, we present an RNS algorithm resolving the Closest Vector Problem (CVP). This algorithm is particularly efficient for a certain class of lattice basis. It provides a full RNS Babai round-off procedure without any costly conversion into alternative positional number system such as Mixed Radix System (MRS). An optimized Cox-Rower architecture adapted to the proposed algorithm is also presented. The main modifications reside in the Rower unit whose feature is to use only one multiplier. This allows to free two out of three multipliers from the Rower unit by reusing the same one with an overhead of 3 more cycles per inner reduction. An analysis of feasibility of implementation within FPGA is also given.

Index Terms—Residue Number System, Lattices, CVP, Round-off, Hardware Implementation, FPGA

INTRODUCTION

On the one hand Residue Number Systems (RNS) are supplied with efficient arithmetic. Computations over big data are distributed across small units concurrently and independantly. On the other hand cryptography area deals with lots of computations over huge numbers. Thus, RNS is naturally interesting to provide efficient implementations of some cryptosystems. There have already been conducted works on RSA [1], ECC [2], and pairings [3], [4] procedures acceleration. Advantages of RNS are exploited so as to optimize modular multiplication, which is a core operation for some cryptoprimitives.

Besides these considerations, modern cryptography makes more and more room for lattice-based cryptography which appears to remain secure in a post-quantum paradigm. In this area, protocols are based on some problems related to Euclidean lattices, such as the Closest Vector Problem (CVP) [5]. For instance, it has been used to create GGH [6] and NTRU [7] cryptosystems. Moreover, interest of lattice-based cryptography has significantly increased since discovery of fully homomorphic encryption scheme by Gentry [8]. Expected for more than 30 years, this property is based on ideal lattices used within GGH-like cryptosystems. Despite all those powerful results, huge computational cost of underlying operations over lattices restrains feasibility of practical implementations. A recent result [9] dealing with this drawback contributes to arithmetical enhancement of lattice-based cryptography by giving a specific RNS-MRS algorithm which computes Babai’s round-off procedure. However, it has the inconvenience to require a conversion into a positional system, like MRS [10], in order to obtain an exact RNS modular reduction.

The present work aims at constructing a full RNS round-off procedure, which will avoid an inconvenient MRS conversion. In particular, the problem of possible inaccuracy of RNS modular reduction operation is addressed by using some of geometrical properties of the involved lattice basis. Finally, a full RNS CVP algorithm is analysed and considerations about hardware implementation are given.

Preliminary definitions on RNS and lattice-based cryptography are given in Section I, along with RNS modular reduction inaccuracy problem. Section II presents a solution to this problem along with an optimised algorithm solving the CVP. Section III presents a dedicated architecture and practical considerations are provided in section V before conclusion.

Writing convention: in further discussions, matrices (resp. vectors) will be denoted by uppercase (resp. lowercase) bold-faced letters.

I. BACKGROUND OVERVIEW

A. RNS and modular reduction

1) Recalls and notations: RNS are based on the Chinese Remainder Theorem (CRT) [10]. This theorem states that given an RNS base \( \mathcal{B} = \{m_1, \ldots, m_n\} \) (which is a set of \( n \) pairwise coprime numbers called moduli) there exists a ring isomorphism \( \mathbb{Z}/M \mathbb{Z} \rightarrow \mathbb{Z}/m_1 \mathbb{Z} \times \ldots \times \mathbb{Z}/m_n \mathbb{Z} \) where \( M = \prod_{i=1}^{n} m_i \). Each integer \( x \) belonging to the so-called dynamic range \( [0, M] \) of \( \mathcal{B} \) is bijectively associated to a tuple of residues \( (\lfloor x \rfloor_{m_i} = x \mod m_i)_{1 \leq i \leq n} \). Moreover, additions,
subtractions and multiplications can be performed concurrently and independently on the residues. Exact divisions by an integer $z$ are also possible as long as $\gcd(z, M) = 1$. In this case, it boils down to the multiplication by $\left\lfloor z^{-1} \right\rfloor_M$.

The main drawback of RNS is that they are non-positional numerical systems. Thus, comparisons require conversion within a positional system, such as Mixed Radix System (MRS) [10]. Modular reduction is also quite a difficult task. State-of-art algorithms [11], [12] are based on an adaptation of classical Montgomery’s modular reduction [13]. Such techniques use conversion operations between two RNS bases.

2) Base conversions: A base conversion denoted $\text{Bex}(B, B', x)$ aims at computing residues in a base $B'$ of an integer $x \in [0, M]$ given in a base $B = \{m_1, \ldots, m_n\}$. To achieve it, the constructive proof of CRT leads to:

$$x = \sum_{i=1}^{n} [x_i M_i^{-1}] m_i \mod M.$$  \hspace{1cm} (1)

The efficiency of such conversion depends on how reduction modulo $M$ is performed, or in other words how the integer $\alpha_x = \left\lfloor \sum_{i=1}^{n} [x_i M_i^{-1}] m_i \right\rfloor M$ is computed. It can be done by adding a redundant modulus to $B$ [14], or by computing an approximation [15]. A particularly efficient but not accurate conversion consists in not computing $\alpha_x$ at all [12]. That way, a conversion is a single matrix multiplication. But one only obtains residues of $\sum_{i=1}^{n} [x_i M_i^{-1}] m_i$ in $B'$.

Another approach is to reconstruct $x$ into a positional mixed radix system [10] and to reduce it into $B'$. The problem is that computation of MRS coefficients is not absolutely parallel. Therefore it breaks the efficiency brought by RNS.

3) RNS modular reduction: Implementation of efficient RNS modular reduction is a critical question because of non-positional character of RNS. Optimized algorithms are adapted from Montgomery’s reduction [13] and use base conversions. The principle is the following. Let $B$ and $B'$ be two coprime RNS bases and $x$ be an integer expressed in those two bases and which has to be reduced modulo $p$. Then by denoting $q = \lceil x/p \rceil_M$ we compute $s = x M^{-1}$ mod $p$. $q$ is easily computed from residues of $x$ and $\left\lfloor 1/p \right\rfloor_M$ directly into $B$. However, division by $M$ cannot be performed in $B$ as is. Hence, $q$ is converted into an auxiliary base $B'$ by using a base conversion. Thus the exact division by $M$ is done in $B'$. However, a problem emphasized in Remark 1.1 rises.

**Remark 1.1:** As for binary Montgomery reduction, RNS modular reduction gives $[x.M^{-1}]_p + e \times p$ where $e$ is integer. The value of $e$ can depend on the sign of $x$ and on the chosen base conversion technique used to convert $q \geq 0$. If $|x| < Mp$ and $\text{Bex}(B, B', q)$ gives residues of $q + \delta M$ with $\delta \geq 0$, then the result satisfies $-p < \frac{x + p + \delta \cdot M}{M} < (2 + \delta) \cdot p$. Thus $e \in [-1, \delta + 1]$. Even if $\delta = 0$, $e$ may still be equal to $\pm 1$. This error can be detected by comparing the result to $p$ and $0$. But such a comparison requires a costly RNS-to-MRS conversion.

The computational cost of RNS modular reduction mainly depends on chosen base conversion technique. Because reduction can be inaccurate whatever base conversion is (cf. Rem. 1.1), a new approach is proposed in [12] to accelerate the reduction by not computing reduction modulo $M$ in (1) at all. In this case, the final result belongs to $\left\{ -p, (n+1)p \right\}$. It can be sufficient for certain situations, as it will be for ours. The general scheme of this reduction is summarized in Alg. 1.

**Algorithm 1 RNSModRed($x, p, B, B'$)**

**Require:** $x \in B \cup B'$ with $|x| < Mp$, $n \in [0, n]$.  

**Ensure:** $s = [x M^{-1}]_p + ep \in B$, with $e \in [-1, n]$.  

1: $q \leftarrow \left\lfloor -x p^{-1} \right\rfloor \mod M$ \#in parallel in $B$
2: $d' \leftarrow \sum_{i=1}^{n}[q_i M_i^{-1}] m_i$ 
3: $s \leftarrow \frac{x + d'}{M} \mod M'$ \#in parallel in $B'$
4: return $s$

B. Round-off operation in lattice-based cryptography

1) **GGH-like cryptosystems:** Some classical lattice-based cryptographic protocols rely on the complexity of computation of a close vector of a lattice $L$ for any given vector $e \in Z^\ell$. Here and further $\ell$ denotes the dimension of $L$. If one knows a nearly orthogonal basis $R \in Z^{\ell \times \ell}$ of $L$, an obvious way to exhibit such a close vector is to compute coordinates of $e$ relatively to $R$, and to select the closest integer vector by applying a round-off operation. This approach was introduced by Babai [5]. Consequently, computations of the form $\lfloor ceR^{-1} \rfloor$ are core operations in lattice-based cryptography. For instance, this approach forges GGH cryptosystem principles [6]. Given a fixed integer parameter $\sigma$ the plaintext space is $P_{\sigma} = \{ -\sigma, \sigma \}^{\ell}$. The secret key is a basis $R$ of $L$ verifying the following "Babai’s condition”:

$$\exists \varepsilon \in [0, 1/2], 0 < \sigma R_{\|} \leq 1/2 - \varepsilon,$$  \hspace{1cm} (2)

where $\sigma R_{\|}$ denotes the maximum $L_1$-norm of columns of $R^{-1}$ and is defined as $\sigma R_{\|} = \max \{|\Sigma_{j=1}^{\ell} |(R^{-1})_{ij}| \mid 1 \leq j \leq \ell \}$. This condition ensures that $| p R^{-1} | = 0$ for any $p \in P_{\sigma}$. More precisely, for any $p \in P_{\sigma}$ and any $k R \in L$ (for $k \in Z^\ell$), then the Babai condition implies the following equalities:

$$| (p + k R) R^{-1} | = | p R^{-1} | + k = k.$$  \hspace{1cm} (3)

The encryption function requires a public basis $B = UR$ (with U unimodular: $U \in GL_n(Z)$). Consequently, an encryption (4) consists in adding a vector of $L$ to $p$. This vector is obtained by using $B$. Decryption (5) performs the round-off procedure.

$$\text{Enc}(p) = p + kB = c \quad (k \in Z^\ell)$$  \hspace{1cm} (4)

$$\text{Dec}(c) = c - \lfloor ceR^{-1} \rfloor R = p$$  \hspace{1cm} (5)

Correctness of decryption (i.e. $\text{Dec}(\text{Enc}(p)) = p$ for $p \in P_{\sigma}$) is due to (3). Indeed, the vector $k R R^{-1} = kU$ is integer. So it can be ruled out from the round-off operation.

2) **RNS Babai’s round-off:** In [9] it is suggested to reduce computational cost of lattice-based cryptoprimitives by using RNS as an efficient arithmetic. More precisely an RNS-MRS algorithm implementing the round-off is given.

As a first step, the rational formula $\lfloor ceR^{-1} \rfloor$ is turned into an integer formula. If $d$ denotes det$R$ then $R' \overset{df}{=} dR^{-1}$ is
an integer matrix. Consequently one can rewrite $[\text{cR}^{-1}] = \left[ \frac{\text{dR}'}{d} + \frac{1}{2} \text{v}_1 \right]$ where $\text{v}_1 \leftarrow (1, \ldots, 1) \in \mathbb{Z}^d$. This leads to (6), where $\text{d}$ denotes the vector $d \times \text{v}_1$.

$$[\text{cR}^{-1}] = \frac{2\text{cR}' + d - [(2\text{cR} + d) \mod (2d)]}{2d} \tag{6}$$

The modular reduction is feasible through RNS Montgomery reduction (Alg. 1) with a main base $\mathcal{B}$ whose product of moduli is denoted $M$. To do so, one has to compute the following "Montgomery representations":

$$\mathbf{R} = (2M\text{R}') \mod (2d), \text{ ~} \mathbf{d} = (M\text{d}) \mod (2d). \tag{7}$$

Since our goal is to provide a full RNS implementation of the Dec function (5) and because $\mathbf{p}$ is in $\mathcal{P}_\sigma = [-\sigma, \sigma]^d$, computation of (6) can be performed within a single modulus RNS base called $m_\sigma$, $m_\sigma$ has to be greater than $2\sigma + 1$. Thus $\mathbf{p}$ is recovered from its centered remainder mod $m_\sigma$. The reduction mod $2d$ requires a large base $\mathcal{B} = \{m_1, \ldots, m_n\}$ to prevent RNS reduction (in our case, $\mathcal{B}'$ is just $\{m_\sigma\}$). As mentioned in Remark 1.1, a problem is that we can possibly obtain an uncompletedly reduced value.

Remark 1.2: When the modular reduction in (6) is performed by using RNSModRed(\text{cR} + \text{d}, 2d, \mathcal{B}, m_\sigma), one obtains $(2\text{cR}' + \text{d}) \mod (2d) + 2\text{de}$ with an error vector $\mathbf{e} \in [-1, n]^d$. Thus, (6) computed together with Alg. 1 gives $[\text{cR}^{-1}] - \text{e} \mod m_\sigma$.

In [9] this error is corrected thanks to a comparison. A modified version of Alg. 1 is used. A redundant modulus $m_\sigma$ is chosen such that $\mathcal{B}'$ allows to perform a second reduction directly into $\mathcal{B}'$. That way, the final error vector $\mathbf{e}$ belongs to $[0, 1]^d$. However, it does not completely solve the problem. The solution in [9] is to use a large base $\mathcal{B}'$ which verifies $M' > 4d > (2\text{cR}' + \text{d}) \mod (2d) + 2\text{de}$. Then a conversion into MRS is performed. It allows the comparison with 2d. Finally, this allows to recover and to correct $\mathbf{e}$.

Our purpose is to avoid expensive conversion into any other number system than RNS. So we choose to compute the modular reduction in (6) via Alg. 1 as is. The next section aim at finding another type of approach to correct the error vector $\mathbf{e}$. To achieve it, some properties of $\mathbf{R}$ will be used.

II. ACCELERATING THE ROUND-OFF

A. Correcting modular reduction in round-off computation

We aim at recovering $\mathbf{e}$ in order to correct it. It will allow to compute $[\text{cR}^{-1}] - \mathbf{e}$ thanks to (6) used together with Alg. 1.

1) Establishing a strategy: Following Remark 2.1 will be a guideline for the new strategy.

Remark 2.1: If one can find an integer $\gamma$ such that $[\text{cR}^{-1}] = 0 \mod \gamma$ for any ciphertext $\text{c}$ and such that $\mathbf{e} \neq 0 \mod \gamma$ as soon as $\mathbf{e} \neq 0$, then $\mathbf{e}$ can easily be detected.

A priori there is no reason that such a $\gamma$ exists. Thus, we consider some integer $\gamma > 0$ and we compute $[\gamma \text{cR}^{-1}]$. Since this round-off is still computed via (6) and Alg. 1, we obtain $[\gamma \text{cR}^{-1}] - \mathbf{e}$, where $\mathbf{e}$ belongs to $[-1, n]^d$. Moreover, we have $[\gamma \text{cR}^{-1}] = [\gamma \text{pR}^{-1}] + [\gamma \text{e}]$ (cf. (3)). But now Babai’s condition (2) for $\mathbf{R}$ does not guarantee anymore that $[\gamma \text{pR}^{-1}] = 0$. Thus by computing $[\gamma \text{cR}^{-1}]$ we introduce a global error which is $[\gamma \text{pR}^{-1}] - \mathbf{e}$. Finally, the result obtained at the end of computation is (8).

$$[\gamma \text{cR}^{-1}] - \mathbf{e} = [\gamma \text{cR}^{-1}] + [\gamma \text{pR}^{-1}] - \mathbf{e} \tag{8}$$

$$\Rightarrow (\gamma \text{cR}^{-1} - \mathbf{e}) \mod \gamma = (\gamma \text{pR}^{-1} - \mathbf{e}) \mod \gamma \tag{9}$$

Consequently, if there exists an integer $\gamma$ large enough so that $[\gamma \text{pR}^{-1}] - \mathbf{e}$ is computable from its residue mod $\gamma$, we will be able to correct it. Then the exact round-off result can be obtained. This is the purpose of following remark.

Remark 2.2: Finding an adequate $\gamma$: We need to find an acceptable odd $\gamma$ for which the bounds described in Remark 2.2 are verified for any $\mathbf{p} \in \mathcal{P}_\sigma$ and any $\mathbf{e} \in [-1, n]^d$. For that purpose, we use Remark 2.3.

Remark 2.3: If $-\frac{2^{e_i} - 1}{2} < (\gamma \text{pR}^{-1})_i - \mathbf{e}_i < \frac{2^{e_i}}{2} + \frac{1}{2}$ for all $i \in [1, \ell]$, then Remark 2.2 and (10) hold.

From Remark 2.3 we deduce that $\gamma$ will depend on $\mathbf{e}$ in Babai’s condition (2). Theorem 2.4 clarifies this dependency.

Theorem 2.4: Let $\mathbf{R} \in \mathbb{Z}^{d \times \ell}$ be a basis of lattice $\mathcal{L}$ verifying Condition (2) for a certain $\mathbf{e} \in [0, \frac{1}{2}]^\ell$. Let $\mathbf{e} \in [-1, n]^d$ be an integer vector (with $n \geq 1$), and $\mathbf{p} = k\mathbf{R}$ with $\mathbf{p} \in \mathcal{P}_\sigma$ and $k \in \mathbb{Z}^{d \times \ell}$. Then for any odd integer $\gamma$ verifying $\gamma \mathbf{e} \geq n$, the following equality holds:

$$[\gamma \text{pR}^{-1}] - \mathbf{e} = (\gamma \text{cR}^{-1} - \mathbf{e}) \mod \gamma. \tag{11}$$

Proof: Due to Babai’s condition, $\text{pR}^{-1} \mathbf{e} = -\frac{1}{2} + \mathbf{e}, \frac{1}{2} - \mathbf{e} \ell$. Consequently, any $i$th component of $(\gamma \text{pR}^{-1} - \mathbf{e})$ verifies:

$$\frac{1}{2} + \gamma \mathbf{e} - n < (\gamma \text{pR}^{-1} - \mathbf{e})_i < \frac{1}{2} - \gamma \mathbf{e} + 1. \tag{12}$$

Hence, by using Remark 2.3, the bounds $-\frac{2^{e_i} - 1}{2} < (\gamma \text{pR}^{-1} - \mathbf{e})_i < \frac{2^{e_i}}{2} + \frac{1}{2}$ hold as soon as $\gamma$ verifies the following set of conditions:

$$\left\{ \begin{array}{l}
-\frac{2^{e_i} - 1}{2} - \frac{1}{2} \leq -\frac{2^{e_i}}{2} + \gamma \mathbf{e} - n \\
\frac{2^{e_i}}{2} - \gamma \mathbf{e} + 1 \leq \frac{2^{e_i}}{2} + \frac{1}{2}
\end{array} \right. \Rightarrow \left\{ \begin{array}{l}
\gamma \mathbf{e} \geq n \\
\gamma \mathbf{e} \geq 1
\end{array} \right. \Rightarrow \gamma \mathbf{e} \geq n$$

(12)

Then Remark 2.2 and (9) allow to conclude the proof.
deduce the following corollary, which expresses this bound no more in terms of ε but by directly using the value of ρₘ. It also details how to use previous theorem to perform an exact round-off within formula (6).

**Corollary 2.5:** Let R be a basis of lattice L such that σρₘ < 1/2. n ≥ 1 an integer (e.g., for our purpose, n is size of RNS base B involved in the modular reduction, Alg. 1), and mₘ a modulus coprime to 2d. Let’s denote γₘ,n = [2n⁻¹(2nρₘ)], and r = (2γcR' + d) mod (2d) + 2d × e mod mₘ and s = (γpR⁻¹ - e) mod mₘ. For any γ coprime to 2dmₘ and s.t. γ ≥ γₘ,n, we have:

\[
[\text{cR}^{-1}] \mod mₘ = \left(\frac{2\gamma cR' + d - r}{2d} - \frac{2\gamma cR' + d - r}{2d} \mod \gamma\right) \gamma^{-1} mₘ
\]

**Proof:** We fix ε = 1 - σρₘ. It follows that γₘ,n = [2n⁻¹(2nρₘ)] is a lower bound for acceptable odd γ’s with respect to theorem 2.4. Moreover, by using (6) and (8), we can write:

\[
\frac{2\gamma cR' + d - r}{2d} = [\gamma pR^{-1}] - e = \gamma [\text{cR}^{-1}] + [\gamma pR^{-1}] - e
\]

Furthermore, previous theorem enables to write [γpR⁻¹] - e = 2\gamma cR' + d - r mod γ. Hence (13) immediately follows.

Given adequate γ with respect to Corollary 2.5, the strategy is to compute the whole formula (6) mod mₘ and mod γ with RNSModRed \((cR + d, 2d, B, \{γ, mₘ\})\) where Montgomery representations are now the following ones:

\[
\tilde{R} \overset{\text{def}}{=} (2γM/R') \mod (2d), \quad \tilde{d} \overset{\text{def}}{=} (Md) \mod (2d).
\]

Then, residues modulo γ enable to correct residues modulo mₘ of rounded value \([\text{cR}^{-1}]\). Fig. 1 illustrates this approach.

**B. About the size of γ**

GGH-like protocols with parameter σ ∈ N are based on the matrices \(R \in \mathbb{Z}^{k \times ℓ}\) verifying σρ₂ < 1/2. By definition the lower bound γₘ,n exists for any such matrix R and any n. It is clear that closer to 1/2 the σρ₂ is, the bigger γₘ,n will be.

For implementing an efficient RNS CVP round-off procedure which uses new error correction technique, γₘ,n should be “reasonably” small, meaning comparatively to word size β of other moduli (i.e. \(m < β\) for all \(m \in B\)). Indeed when γₘ,n is not larger than β, γ can be a single modulus. Then the base \(B'\) in Alg. 1 is just the set of two moduli \(\{γ, mₘ\}\). In this case the needed comparison to recover a centered remainder modulo γ is directly performed on the residue mod γ. Hence the present technique is particularly efficient for matrices R such that 0 < σρ₂ < 1/2 - β/2 (i.e. γₘ,n < β).

In practice, the constraint γₘ,n < β does not seem to be so restrictive for usual values of β = 2² (i.e. \(r ≥ 16\)). By analyzing basis R obtained through LLL-reduction of matrices randomly picked up in \([−ℓ, ℓ]^{k \times ℓ}\) (as suggested in [16]) for some dimensions \(ℓ\) from 128 to 768 with respect to σ = 2 or 3, it appears (cf. Tab. 1) that γₘ,n is most of the time quite “small” (i.e. comparatively to β). Then choosing a lattice basis R with γₘ,n < β should not weaken the protocol.

**C. Full RNS Rounding-off CVP algorithm**

The next part presents a detailed algorithm computing \(\text{c} - [\text{cR}^{-1}]R\) in Alg. 1 together with (6), and it performs the acceleration technique. To provide an optimized algorithm, all possible precomputations for Alg. 2 are provided in a first part. A radix β = 2² as word size is fixed from now on. All moduli used are constrained to own only one β-digit.

So any involved matrix R is assumed to verify γₘ,n < β and \(n = |B|\). Next, the proposed algorithm is detailed. Finally, a discussion on computational and space costs is provided.

**1) Precomputations:** Precomputable data (22) are found by expanding involved expressions. We aim at computing \(\text{c} - [\text{cR}^{-1}]R\mod mₘ\), c is the only input variable. Due to our previous discussions, the strategy exploits following quantities:

\[
q'_i = \sum_{i=1}^{n} q_m, i = M_i, \quad q_m = \left\lfloor (\text{cR} + d)(2dM'_i)^{-1}\right\rfloor_{m_i}, \quad \left(15\right)
\]

\[
r = [2\gamma cR' + d]_{2d} + 2d = \frac{cR + d + 2dq'}{M}
\]

\[
s = [\gamma pR^{-1}] - e = \frac{1}{2d} (2\gamma cR' + d - r)
\]

First, (15) is obtained in RNS base B. Second, (16) is computed in \(\{γ, mₘ\}\). Then, (17) is obtained modulo γ and mₘ. Finally centered remainder of s mod γ enables to correct residues in RNS channel mₘ so that \([\text{cR}^{-1}]\mod mₘ\) can be obtained (cf. (13) in Corollary 2.5).

In B: By using \(R_m\) and \(d_m\), in (22) we can re-write (15) with \(q_{m_i} = [cR_m + d_m]_{m_i}\) for all \(m_i \in B\). For base conversion, it will be useful to write the set of n vectors \(q_{m_i}\), as a matrix Q whose \(i^{th}\) row is the vector \(q_{m_i}\).
In \{\gamma\}: If we develop right member of (17) modulo \gamma by using (15) and (16) we have:

$$\left[ s \right]_\gamma = \left[ e^{-\tilde{R}} + \frac{d}{2dM} - \tilde{d} \frac{n}{2dM} \right] \mod \gamma \quad \text{where} \quad \gamma = \left[ \gamma_p R^{-1} \right] - e. \quad \text{(19)}$$

Due to precomputations (22) the required result modulo \gamma is:

$$\left[ s \right]_\gamma = \left( c - \tilde{R} \tilde{R}_\gamma + \tilde{d} \tilde{d}_\gamma + h_\gamma Q \right) \mod \gamma \quad \text{(19)}$$

In \{m_\sigma\}: The one modulus-base \{m_\sigma\} is dedicated to the full computation of \(c - \lfloor c R^{-1} \rfloor R\). First, by using (13) from corollary and (17) and (19), it comes that:

$$\left[ c - \lfloor c R^{-1}\rfloor R \right]_{m_\sigma} = \left[ c - \frac{1}{\gamma} (s - s_\gamma) R \right]_{m_\sigma} \quad \text{(20)}$$

where \(\left[ s \right]_{m_\sigma} = \frac{1}{\gamma} (s - s_\gamma) R \mod m_\sigma\). Thus precomputed data (22) allow to write:

$$\left[ c - \lfloor c R^{-1}\rfloor R \right]_{m_\sigma} = \left[ c R_{\tilde{R}_\gamma} + \tilde{d}_\sigma + (h_\sigma Q + s_\gamma) R_{\sigma} \right]_{m_\sigma} \quad \text{(21)}$$

2) Main Algorithm 2: RNS base \(B\) used for the modular reduction (16) verifies \(2dM > \lfloor c R + \tilde{d} \rfloor\) for any input \(c\). Thus we assume that the set of all possible \(\lfloor c R \rfloor\) is bounded by some integer \(c_x\). \(c\) can own negative coefficients. Finally \(B\) is assumed to satisfy \(M > \ell c_x + 1\).

Complexity analysis: Complexity of RNS algorithms is generally expressed in terms of elementary modular multiplications with respect to some moduli \(m\) such that \(m < \beta = 2^\ell\). To simplify complexity analysis, \(\text{EMM}_m\) denotes such modular multiplication. The \(n + 1\) moduli of \(B \cup \{\gamma\}\) are assumed to own only one \(\beta\)-digit. Since bit-size of \(m_\sigma\) and \(\beta\) can differ by orders of magnitude it is useful to denote \(\beta_\sigma\) the quantity of \(\beta\)’s. Binary-to-RNS conversions are based on decomposition of coefficients of \(c\) in base \(\beta\). Residue of powers of \(\beta\) can be precomputed. That way the conversion requires \((n + 1) \ell \times \lceil \log_\beta (c_x) \rceil \text{EMM}_\beta + \ell \times \lceil \log_\beta (c_x) \rceil \text{EMM}_\beta_\sigma\). When this conversion is done, the cost of steps 2 to 11 of RNS CVP algorithm 2 is \((n + 1) \ell^2 + n \ell^2 \text{EMM}_\beta + 2 \ell^2 + n \ell^2 \text{EMM}_\beta_\sigma\).

Space efficiency: Precomputed powers of \(\beta\) for binary-to-RNS conversion cost no more than \((n + 1) \lceil \log_\beta (c_x) \rceil - 1\) \(\beta\)-words and \((\log_\beta (c_x)) - 1\) \(\beta\)-words. Precomputations (22) represent \((n + 1)(\ell^2 + \ell) + n \beta\)-words and \(2 \ell^2 + \ell + n n \beta\)-words. For example, precomputations \(\text{mod} m_\sigma\) is almost identical.

Algorithm 2 Full RNS Rounding-off CVP algorithm

Require: input \(c\); data: RNS base \(B\) with \(M > (\ell c_x + 1)\) and \(n = [B], \gamma, \gamma_\sigma R, m_\sigma \geq 2\sigma + 1, M, \gamma, m_\sigma\) and \(2d\) all pairwise coprime, and precomputations (22).

Ensure: \((c - \lfloor c R^{-1}\rfloor \times R) \mod m_\sigma\).

1: \((c_m)_{m \in B \cup \{\gamma, m_\sigma\}} \leftarrow \text{Bin_to_RNS}(c) \quad \#1\text{st parallel step}

2: for \(i = 1\) to \(n\) do \(\#2\text{nd parallel step}

3: \(Q_{i, \sigma} \leftarrow c_m R_{i, \sigma} + \tilde{d}_m \mod m_\sigma \quad \#3\text{rd parallel step; part of (19)}

4: end for

5: \(s_\sigma \leftarrow c R_{\tilde{R}_\gamma} + \tilde{d}_\gamma \mod \gamma \quad \#2\text{nd parallel step; part of (19)}

6: \(p_{\sigma, \gamma} \leftarrow s_\sigma + q_\gamma \mod \gamma \quad \#4\text{th parallel step; part of (19)}

7: \(t_{\sigma, \gamma} \leftarrow s_\sigma + q_\gamma \mod \gamma \quad \#5\text{th parallel step; part of (19)}

8: \(p_{m_\sigma} \leftarrow [p_{\sigma, \gamma} + t_{\sigma, \gamma}] \mod m_\sigma \quad \#6\text{th parallel step; part of (19)}

9: end for

10: \return \(p_{m_\sigma}\) \mod \(m_\sigma\)

11: end for

TABLE II: Memory overhead of RNS precomputations comparatively to binary storage of \(R^{-1}\) for several word-size \(\beta = 2^\ell\) and dimensions \(\ell\).

<table>
<thead>
<tr>
<th>(\ell)</th>
<th>(64)</th>
<th>(128)</th>
<th>(256)</th>
<th>(512)</th>
<th>(1024)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(r = 24)</td>
<td>+6.9%</td>
<td>+3.2%</td>
<td>+1.5%</td>
<td>+0.7%</td>
<td>+0.3%</td>
</tr>
<tr>
<td>(r = 32)</td>
<td>+8.0%</td>
<td>+3.7%</td>
<td>+1.7%</td>
<td>+0.8%</td>
<td>+0.4%</td>
</tr>
<tr>
<td>(r = 64)</td>
<td>+12.4%</td>
<td>+5.7%</td>
<td>+2.6%</td>
<td>+1.2%</td>
<td>+0.6%</td>
</tr>
</tbody>
</table>

\(\beta_\sigma\)-words. To compare with a binary approach, each coefficient of \(R^{-1}\) should be computed with a \(\log_2 (c_x (\frac{1}{2} - \sigma R))^{-1}\) bit precision to ensure no decryption error [6]. We recall that \(\gamma R \equiv \left[ n \times (\frac{1}{2} - \sigma R) \right]^{-1}\) and that our approach is optimal when \(\gamma R_\sigma < \beta\). Thus the binary method requires at most \(\lceil \log_2 (\frac{c_x}{\beta}) \rceil \times \ell^2\) bits of storage. Tab. II shows some memory ratio between binary storage of \(R^{-1}\) and RNS precomputations where we assume that \(c_x = d\) and \(d\) is set to Hadamard’s bound \(d = \ell\) for \(\ell \leq 16\) and \(\ell = 2\) otherwise.

3) Comparison with hybrid RNS-MRS approach [9]: In [9], the final computation is also performed modulo \(m_\sigma\). The same RNS base \(\bar{B}\) than ours (i.e. \(M > \ell c_x + 1\)) is used for computing modular reduction. The main difference in this other approach is that an auxiliary base \(B^*\) with \(M > 4d\) is required. Indeed the result of reduction is in \([0, 4d]\), and it must be compared with \(2d\) through MRS. The first step of reduction is a fast conversion (like in Alg. 1) from \(B^* \cup \{m_\sigma\}\) to \(B^* \cup \{m_\sigma\}\) (simple "duplication" of residues mod \(m_\sigma\) in order to reduce the result in \([0, 4d]\). Montgomery representations differ from (14) but their size is still the same. They are \(R = [2m_\sigma R]_{2d_\theta} + d = [m_\sigma M d]_{2d_\theta}\).

By denoting \(k = [B^*], \text{i.e. } k = [\log_2 (4d)]\), the extra cost (mem. and comp.) of hybrid approach are due to computations in \(B^*\) and to the MRS conversion. In terms of precomputations dedicated to modular reduction, the overhead is \(k (\ell + 1) + (k + 1) n \beta\)-words for adequate matrices and vectors \(\tilde{R}_m, \tilde{d}_m, h_m\) and extra specific values \(\lfloor c_x\rfloor [\gamma, \sigma] \mod \gamma_\sigma\) for each \(m_\sigma \in B^*\).
<table>
<thead>
<tr>
<th>$r$</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
<th>1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r=24$</td>
<td>+97.0%</td>
<td>+98.8%</td>
<td>+99.5%</td>
<td>+99.8%</td>
<td>+99.9%</td>
</tr>
<tr>
<td>$r=32$</td>
<td>+95.6%</td>
<td>+98.1%</td>
<td>+99.2%</td>
<td>+99.7%</td>
<td>+99.9%</td>
</tr>
<tr>
<td>$r=64$</td>
<td>+91.2%</td>
<td>+95.8%</td>
<td>+98.1%</td>
<td>+99.2%</td>
<td>+99.6%</td>
</tr>
</tbody>
</table>

TABLE IV: Memory overhead of [9] comparatively to present full RNS approach for several word-size $\beta=2^\ell$ and dimensions $\ell$.

<table>
<thead>
<tr>
<th>$\ell$</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
<th>1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ell=24$</td>
<td>+133.5%</td>
<td>+141.9%</td>
<td>+149.1%</td>
<td>+156.9%</td>
<td>+162.3%</td>
</tr>
<tr>
<td>$\ell=32$</td>
<td>+121.1%</td>
<td>+130.5%</td>
<td>+136.4%</td>
<td>+141.7%</td>
<td>+146.6%</td>
</tr>
<tr>
<td>$\ell=64$</td>
<td>+105.1%</td>
<td>+112.1%</td>
<td>+116.7%</td>
<td>+120.2%</td>
<td>+123.0%</td>
</tr>
</tbody>
</table>

TABLE III: Memory overhead of [9] comparatively to present full RNS approach for several word-size $\beta=2^\ell$ and dimensions $\ell$.

The only extra cost is due to $|m^{-1}_\ell|_{m_\sigma}$ for second conversion. Next, MRS conversion requires to precompute $\frac{k(k-1)}{2}$ residues $[(m_i^\ell)^{-1}|_{\mu_j}]$ for $1 \leq i < j \leq k$. A similar analysis leads to an extra cost in terms of $\text{EMM}_{\beta}$ given by $(k\ell^2 + k\ell(n + 1) + \frac{k(k-1)}{2})\text{EMM}_{\beta}$. Tab. III and IV show these overheads with bounds $c_\ell = d \leq \ell^2\ell$. In this case, $n \sim k$. Thus the RNS-MRS approach requires around twice more moduli. Ratio in Tab. IV are clearly greater than 2 because of MRS conversions.

## III. Towards Hardware Implementation

In order to use concurrency properties of RNS, a feasibility study about efficient and dedicated architecture for hardware implementations on FPGA is led. The chosen architecture is based on Cox-Rower architecture proposed in [15], and in [17] with several modifications. We also explain how the main Algorithm 2 fits with such type of architecture.

### A. Cox-Rower Architecture

In [15], the Cox-Rower architecture was proposed along with a new approach for base conversion. It has been improved several times [2]–[4], [17]–[19]. Its structure consists in a Sequencer, a Cox and a set of Rower units. Rowers deal with the core of RNS computations. They compute modular sum-of-products $\sum_{i=1}^n a_ib_i \mod m_i$. A Rower is divided into 2 stages. The first one consists in multiplication of 2 elements, and the second one is in charge of reducing and accumulating elements from first stage as shown in Fig. 2. Efficient algorithms for reduction inside the Rower are proposed in [2], [3], [17], [18]. They take advantage either by the pseudo-Mersenne form of the moduli, which is $m_i = 2^\ell - \mu_i$ with $\mu_i < 2^{\ell/2}$, allowing a direct reduction, or by using a second level of Montgomery reduction as proposed in [17] with $\gcd(m_i, 2^\ell) = 1$.

### B. Proposed Architecture

Our architecture is based on the one proposed in [2], [17], [18], with several modifications. The Cox unit is unnecessary for our purpose, as Alg. 2 does not reduce the base conversion. So as to decrease the area of a Rower, 2 multipliers out of 3 are removed from the Rower for a cost of 3 extra cycles to execute the reduction inside it. Indeed, one of the most time-consuming operation in RNS is the base conversion procedure which is a computation of the form $\sum_{i=1}^n a_ib_i \mod m_i$. Moreover, the most time-consuming operations in Alg. 2 are the matrix-vector products. By using a second level of Montgomery reduction coupled with lazy-reduction [20], it is possible to reduce a sum of products only once. Hence the second stage in charge of reduction can be deleted by reusing first stage. Fig. 3 shows the proposed implementation for Rower Unit.

From lines 5 to 11 of Alg. 2, only 2 Rower units are required. To compute the $\modc \gamma$ function (line 8), two tricks are used to make it easy in hardware without using signed numbers. First, one can see that $s_{\gamma,i} \modc \gamma = \begin{cases} s_{\gamma,i} & \text{if } s_{\gamma,i} \leq \lfloor \gamma/2 \rfloor \\ s_{\gamma,i} - \gamma & \text{otherwise} \end{cases}$. This can be used if the comparison is easy to perform. Thus, our second trick is to choose $\gamma = 2\alpha - 1$ such that $\alpha = 2^\theta$. It makes the comparison easy to be done in hardware (we just need to check the $\theta$th bit). Moreover, these 2 Rower units are used to compute respectively $m_\sigma$ and $\modc \gamma$. Because $m_\sigma$ is small ($\leq 7$ as $s \in \{2, 3\}$), and because of the trick for $\gamma$, we dedicate two Rower units to computations $\mod m_\sigma$ and $\modc \gamma$. The $\gamma$ Rower unit remains the same than the one in Fig. 3 but with the test of the $\theta$th bit in order to subtract or not $\gamma$.

Alg. 3 details the computation of an inner reduction by using the proposed Rower implementation. From line 1 to 4, we multiply and accumulate the result. Then, the reduction is performed at steps 5 and 6. Because of the Rower unit pipeline, the computation cannot be done directly. Thus we need to wait 2 depth of the pipeline. However, as usually in cryptography, most of computations can be pipelined. Thus no idle cycles are induced. We can show that the computations in...
Alg. 2 can be done with a 100% occupancy of the pipeline. In order to fully compute Alg. 2 with the proposed architecture, the residues have to be reduced from $2m_i$ to $m_i$ before they go to the main bus (as we need the exact value during the base conversion procedure). This last reduction can be done by a module added in the main bus. It reduces the residues going through the bus.

**Algorithm 3 InnerReduction($a, b, m$)**

Require: A set of $a, b \in (\mathbb{Z}/2m\mathbb{Z})^k$, $s$ parameter for lazy-reduction, $-m^{-1}\mod 2^{r+s}$ and $d$ the pipeline depth

Ensure: $t \equiv \sum_{i=1}^{r+s} a_i b_i 2^{-i} \mod m$, $0 \leq t < 2m$

1: $t \leftarrow a_1 \times b_1$
2: for $i \leftarrow 2$ to $k$ do
3: $t \leftarrow t + a_i \times b_i$
4: end for
5: $q \leftarrow (t \times m^{-1}) \mod 2^{r+s}$
6: $r \leftarrow \frac{t}{2^{r+s}}$
7: return $t$

**C. Full RNS CVP Algorithm Implementation**

Alg. 4 gives the procedure to compute lines 2 to 4 of Alg. 2 by using the Rower unit detailed previously. Most of the precomputations have to be done using the inner Montgomery representation in the Rower unit (i.e. precomputed values have to be multiplied by $2^{r+s}$ where $r$ the radix size and $s$ enables the use of lazy reduction). Other steps of Alg. 2 are not detailed but are executed in a similar way. The point is that all computations are modular sums of products, fitting well with Cox-Rower architecture. As emphasized in comments of main Alg. 2, there are 6 main steps which do not overlap.

Features of the Cox-Rower architecture presented here match GPGPU and multi-core CPU architecture purposes. More generally, practical considerations that we make are essentially due to the well flexibility of main algorithm. Since it involves only RNS and fast base conversions, parallelization is easily achieved. In next section, we analyse feasibility of FPGA implementation. The efficiency of such platforms has been proven for RNS implementations.

**IV. ABOUT FEASIBILITY OF IMPLEMENTATION ON FPGA**

Estimations about performances and area for some dimensions will be given. Parameters are determined by a radix size $\beta = 2^r$, and by Hadamard’s bound for $c_p$ and $d$.

**A. Performance and Memory Analysis**

With $n + 2$ parallel Rowers, binary-to-RNS conversion requires $\lceil \log_2(c_p) \rceil\lceil 1 < n \rceil$ cycles when residues of powers of $\beta$ are precomputed. Table V details the number of cycles needed to execute Alg. 2, as well as the number of precomputed elements required for each of 6 main steps of Alg. 2. For instance, we can notice that lines 5 and 6 completely overlap with computation of $Q$ detailed in Alg. 4.

**Algorithm 4 Full RNS CVP Detailed: $Q$ computation step**

Require: Residues of $c$ in $B$ and precomputations among (22)

Ensure: $Q \in \mathbb{Z}^{n\times n}$ s.t. $Q \cdot a = [c \ell R + d \ell_m]$ for $1 \leq i \leq n$

1: for $i \leftarrow 1$ to $n$ do
2: $Q_{i,a} \leftarrow (0)^i$
3: for $y \leftarrow 1$ to $\ell$ do
4: $Q_{x,a} \leftarrow Q_{x,a} + c_{m,x} \times (\tilde{R}_{m,x})_{x,y}$
5: for $x \leftarrow 1$ to $\ell$ do
6: $Q_{x,y} \leftarrow Q_{x,y} + (\tilde{a}_{m,y})_{x,y}$
7: end for
8: end for
9: for $x \leftarrow 1$ to $\ell$ do
10: $q_{m,x} \leftarrow Q_{x,x} \times m^{-1}\mod 2^{r+s}$
11: end for
12: for $x \leftarrow 1$ to $\ell$ do
13: $Q_{x,x} \leftarrow Q_{x,x} + q_{m,x} \cdot Z_{m,x}$
14: end for
15: end for
16: return $Q$

**TABLE V: Performance and Memory Size**

<table>
<thead>
<tr>
<th>($\ell$, $n$)</th>
<th>Mem./Row (Els)</th>
<th>Area/Row (LUTs+DSP+RAM)</th>
<th>Freq. (MHz)</th>
<th>Tot. Area (estimation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(64,16)</td>
<td>4162</td>
<td>19 (30/17/2)</td>
<td>400</td>
<td>304 (480/272/240)</td>
</tr>
<tr>
<td>(128,38)</td>
<td>16514</td>
<td>19 (30/17/2)</td>
<td>300</td>
<td>722 (1140/4676/2128)</td>
</tr>
</tbody>
</table>

**TABLE VI: Results of P&R for ML510 Eval. board (Virtex5)**

<table>
<thead>
<tr>
<th>($\ell$, $n$)</th>
<th>Mem./Row (Els)</th>
<th>Area/Row (LUTs+DSP+RAM)</th>
<th>Freq. (MHz)</th>
<th>Tot. Area (estimation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(64,16)</td>
<td>4162</td>
<td>20 (60/17/2)</td>
<td>468</td>
<td>320 (960/272/240)</td>
</tr>
<tr>
<td>(128,38)</td>
<td>16514</td>
<td>20 (60/17/2)</td>
<td>415</td>
<td>760 (2280/4676/2128)</td>
</tr>
</tbody>
</table>

**TABLE VII: Results of P&R for KC705 Eval. board (Kintex7)**

**B. Implementation**

We focus on FPGA SRAM devices. Indeed, FPGA are good candidates for hardware implementation as they embed DSP blocks as well as Block RAM. Moreover, the Rower unit proposed in previous section easily fits in Block RAM and DSP blocks, without adding too many LUTs or registers from the FPGA (except for multiplexers and control). Thus good performances can be reached in terms of frequency. We implement the Rower unit in Xilinx Virtex-5 FPGA family (Tab. VI) as well as Kintex-7 (Tab. VII) and estimate overall area and performance. We take $r = 24$ as bit-size of radix by cascading 2 DSP blocks in order to have few more bits for lazy reduction (10 bits). The results show good perspective to
implement lattice on FPGA, but a memory bottleneck rapidly appears. Indeed, for \( \ell = 128 \), there are not enough BRAM in both devices (Virtex-5 and Kintex-7) to store precomputed data.

C. Discussion

In previous part, we have shown that hardware implementation of lattices can be done using RNS and Algorithm 2. Adding a margin error of 25% to the maximal frequency of the design, one can easily deduce that we need around 20\( \mu \)s to compute a close lattice vector in dimension \( \ell = 64 \), which can be implemented on FPGA as is. For dimension 128, too much precomputations have to be stored. Thus it could be interesting to work on specific architectures such as clusters of FPGA, or with a DMA for quick changes of memorised values. Nevertheless, our hardware implementation shows a real good trade-off for all existing RNS architecture in terms of footprint or reachable frequency [2]–[4], [17].

V. CONCLUSION

A method for implementing the Babai round-off procedure in RNS has been proposed. We provide the first full RNS implementation of an algorithm solving the CVP (i.e., without using MRS). The efficiency of this technique depends on the secret lattice basis \( \mathbf{R} \) which must verify some condition connected to RNS word size. Nevertheless, by analysing several random matrices verifying Babai’s condition, the new requirement does not seem restrictive when considering practical RNS word sizes. An architecture for hardware implementation has been detailed. It is based on a Cox-Rower like architecture which is traditionally used for RNS implementations. Some modifications are suggested in order to fit with specific features of lattice-based algorithms. We have given an analysis of FPGA implementation for dimensions 64 and 128. Despite some memory bottlenecks, the results highlight the viability of our approach in terms of flexibility, because the principles of our implementation match GPGPU and multi-core CPU features. It reinforces the fact that the RNS is an excellent candidate for practical implementation of lattice-based protocols.

We fully realize that 64 and 128 are not dimensions high enough to allow lattice-based cryptography with secure parameters. Nevertheless, this implementation should be seen as a proof of concept. It opens the door to multiple directions such as approaches by blocks and pipeline, or by using FPGA clusters or eventually multi-core CPU’s or GPU’s. All those future works will use as their core this new proposition.

REFERENCES


