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.
RNS Arithmetic Approach in Lattice-based Cryptography

Jean-Claude Bajard, Julien Eynard*
Sorbonne Universités, UPMC Univ. Paris 6, UMR 7606, LIP6, F-75005, Paris, France
CNRS, UMR 7606, LIP6, F-75005, Paris
{jean-claude.bajard,julien.eynard}@lip6.fr

Nabil Merkiche
DGA/MI
Rennes, France
nabil.merkiche@intradef.gouv.fr

Thomas Plantard
Univ. of Wollongong
CCISR
Wollongong, Australia
thomaspl@uow.edu.au

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

I. 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 a further 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 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.

II. 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 $B = \{m_1, \ldots, m_n\}$ (which is a set of $n$ pairwise coprime numbers called moduli) there exists a ring isomorphism $\mathbb{Z} / M \mathbb{Z} \cong \mathbb{Z}/m_1 \mathbb{Z} \times \ldots \times \mathbb{Z}/m_n \mathbb{Z}$ where $M = \prod_{i=1}^{n} m_i$. Each integer $x$ belonging to the so-called dynamic range $[0, M]_B$ of $B$ is bijectively associated to a tuple of residues $(\{x\}_{m_i} = x \mod m_i)_{1 \leq i \leq n}$. 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 \( \lfloor z^{-1} \rfloor_M \).

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 \( Bex(B, B', x) \) aims at computing residues in a base \( B' \) of an integer \( x \in [0, M[ \) given in a base \( B = \{m_1, ..., m_n\} \). To achieve it, the constructive proof of CRT leads to:

\[
x = \sum_{i=1}^{n} \lfloor x_i M_i^{-1} \rfloor_{m_i} M_i \mod M.
\]

(1)

The efficiency of such conversion depends on how reduction modulo \( M \) is performed, or in other words how the integer \( \alpha_x = \lfloor \sum_{i=1}^{n} \lfloor x_i M_i^{-1} \rfloor_{m_i} M_i \rfloor \) 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 \( \sum_{i=1}^{n} \lfloor x_i M_i^{-1} \rfloor_{m_i} M_i \) 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 \( q = \lfloor x/p \rfloor_M \) we compute \( s = \frac{x + pq}{p} \). By construction, \( s \equiv xM^{-1} \mod p \). \( q \) is easily computed from residues of \( x \) and \( \lfloor 1/p \rfloor_M \) 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 \( \lfloor xM^{-1} \rfloor_{B'} + e \times p \) 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 \( \lfloor x \rfloor < Mp \) and \( Bex(B, B', q) \) gives residues of \( q + \delta M \) with \( \delta \geq 0 \), then the result satisfies \( -p < \frac{x + pq + \delta MP}{M} < (2 + \delta) p \). 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 \( \{ -p, (n + 1)p \} \). It can be sufficient for certain situations, as it will be for ours. The general scheme of this reduction is summarized in Alg. 1.

\[\text{Algorithm 1 RNSModRed}(x, p, B, B')\]

Require: \( x \in B \cup B' \) with \( |x| < Mp, n \in \mathbb{Z} \).
Ensure: \( s = \lfloor xM^{-1} \rfloor_{B'} + ep \in B' \), with \( e \in [-1, n] \).
1: \( q \leftarrow (-xp^{-1}) \mod M \) \#in parallel in \( B \)
2: \( q' \leftarrow \sum_{i=1}^{n} \lfloor q_iM^{-1} \rfloor_{m_i} M_i \)
3: \( s \leftarrow \frac{x + q'p}{p} \mod M' \) \#in parallel in \( B' \)
4: \( \text{return} s \)

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 \( \mathcal{L} \) for any given vector \( c \in \mathbb{Z}^\ell \). Here and further \( \ell \) denotes the dimension of \( \mathcal{L} \). If one knows a nearly orthogonal basis \( R \in \mathbb{Z}^{\ell \times \ell} \) of \( \mathcal{L} \), an obvious way to exhibit such a close vector is to compute coordinates of \( c \) 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 \( \lfloor cR^{-1} \rfloor \) 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 \( \mathcal{P}_\sigma = [-\sigma, \sigma]^{\ell} \). The secret key is a basis \( R \) of \( \mathcal{L} \) verifying the following ”Babai’s condition”:

\[
\exists \varepsilon \in ]0, \frac{1}{2}[, 0 < \sigma R 1 < \frac{1}{2} - \varepsilon, (2)
\]

where \( \sigma R \) denotes the maximum \( L_1 \)-norm of columns of \( R^{-1} \) and is defined as \( \sigma R = \max \{ \sum_{i=1}^{\ell} \lfloor \lfloor R^{-1} \rfloor_{i,j} \rfloor \mid 1 \leq j \leq \ell \} \).

This condition ensures that \( \lfloor pR^{-1} \rfloor = 0 \) for any \( p \in \mathcal{P}_\sigma \). More precisely, for any \( p \in \mathcal{P}_\sigma \) and any \( kR \in \mathcal{L} \) (for \( k \in \mathbb{Z}^\ell \)), then the Babai condition implies the following equalities:

\[
\lfloor (p + kR)R^{-1} \rfloor = \lfloor pR^{-1} \rfloor + k = k. (3)
\]

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

\[
\text{Enc}(p) = p + kB = c \quad (k \in \mathbb{Z}^\ell) \quad (4)
\]

\[
\text{Dec}(c) = c - \lfloor cR^{-1} \rfloor R = p \quad (5)
\]

Correctness of decryption (i.e. \( \text{Dec}(\text{Enc}(p)) = p \) for \( p \in \mathcal{P}_\sigma \)) is due to (3). Indeed, the vector \( kBR^{-1} = kU \) 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 \( \lfloor cR^{-1} \rfloor \) is turned into an integer formula. If \( d \) denotes det \( R \) then \( R' \equiv dR^{-1} \) is
an integer matrix. Consequently one can rewrite $[e \mathbf{R}^{-1}] = \left[ \frac{e \mathbf{R}^{-1}}{d} + \frac{1}{2} \mathbf{v}_1 \right]$ where $\mathbf{v}_1 \overset{\text{def}}{=} (1, \ldots, 1) \in \mathbb{Z}^\ell$. This leads to (6), where $\mathbf{d}$ denotes the vector $d \times \mathbf{v}_1$.

$$[e \mathbf{R}^{-1}] = \frac{2e \mathbf{R'} - \mathbf{d} - [(2e \mathbf{R'} + \mathbf{d}) \text{ mod } (2d)]}{2d} \quad (6)$$

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”:

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

Since our goal is to provide a full RNS implementation of the Dec function (5) and because $\mathbf{p}$ is in $\mathcal{P}_\sigma = [-\sigma, \sigma]^f$, 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 $\mathcal{B} = \{m_1, \ldots, m_n\}$ in order to use 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 uncompletely reduced value.

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

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

Finally, this allows to recover and to correct $e$.

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 $e$. 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 $e$ in order to correct it. It will allow to compute $[\mathbf{R}^{-1}] = e$ 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 $[\mathbf{R}^{-1}] = 0 \text{ mod } \mathcal{P}$ for any ciphertext $\mathbf{c}$ and such that $e \neq 0 \text{ mod } \gamma$ as soon as $e \neq 0$, then $\mathbf{c}$ 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 $[\gamma \mathbf{R}^{-1}]$. Since this round-off is still computed via (6) and Alg. 1, we obtain $[\gamma \mathbf{R}^{-1}] - e$, where $e$ belongs to $[-1/n]^f$. Moreover, we have $[\gamma \mathbf{R}^{-1}] = [\gamma \mathbf{p}^{-1}] + [\gamma \mathbf{e}]$ (cf. (3)). But now Babai’s condition (2) for $\mathbf{R}$ does not guarantee anymore that $[\gamma \mathbf{p}^{-1}] = 0$. Thus by computing $[\gamma \mathbf{e}]$ we introduce a global error which is $[\gamma \mathbf{p}^{-1}] - e$. Finally, the result obtained at the end of computation is (8).

$$[\gamma \mathbf{e}] - e = [\gamma \mathbf{e}] + [\gamma \mathbf{p}^{-1}] - e \quad (8)$$

$$\Rightarrow ([\gamma \mathbf{e}] - e) \text{ mod } \gamma = ([\gamma \mathbf{p}^{-1}] - e) \text{ mod } \gamma \quad (9)$$

Consequently, if there exists an integer $\gamma$ large enough so that $[\gamma \mathbf{p}^{-1}] - e$ 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:**: mode denotes the centered remainders mod $\gamma$. If for any $i \in [1, f]$ the bounds $[-2^{f-2}, 2^{f-2}, \varepsilon_i < \gamma \mathbf{p}^{-1}]$ hold, then one can retrieve $[\gamma \mathbf{p}^{-1}] - e$ by:

$$[\gamma \mathbf{p}^{-1}] - e = ([\gamma \mathbf{p}^{-1}] - e) \text{ mod } \gamma \quad (10)$$

Therefore, using (8), (9) and (10), $[\mathbf{R}^{-1}]$ can be exactly computed via following equality:

$$[\mathbf{R}^{-1}] = \frac{1}{\gamma} \left([\gamma \mathbf{e}] + e - ([\gamma \mathbf{p}^{-1}] - e) \text{ mod } \gamma \right). \quad (11)$$

From now on, the goal is to find a condition so that $\gamma$ satisfies the bounds emphasized in Remark 2.2. Then, after computation of $[\gamma \mathbf{e}]$ through (6), we will be able to deduce the exact round-off $[\mathbf{R}^{-1}]$ using (11). Furthermore, it may be noticed that since we need to compute $[\gamma \mathbf{R}]$ mod $\gamma$ through (6) in order to obtain $\gamma$, $\gamma$ must be coprime to $2d$.

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 $\mathbf{p} \in \mathcal{P}$ and any $e \in [-1/n]^f$. For that purpose, we use Remark 2.3.

**Remark 2.3:** If $[-2^{f-2} + \varepsilon, \varepsilon] < ([\gamma \mathbf{p}^{-1}] - e)$ holds for any $\varepsilon \in [1, f]$, then Remark 2.2 and (10) hold.

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

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

$$[\gamma \mathbf{p}^{-1}] - e = ([\gamma \mathbf{e}] + e - ([\gamma \mathbf{p}^{-1}] - e) \text{ mod } \gamma)$$

**Proof:** Due to Babai’s condition, $\mathbf{p}^{-1} \mathbf{e} \in [-1/2, 1/2]$. Consequently, any $i$th component of $([\gamma \mathbf{p}^{-1}] - e)$ verifies:

$$\frac{1}{\gamma} \left(-2^{-f+1} - \frac{e_i}{\gamma} \right) \leq \left([\gamma \mathbf{p}^{-1}] - e\right)_i \leq \left(\frac{2}{\gamma} \right) - \frac{e_i}{\gamma} + 1$$

Hence, by using Remark 2.3, the bounds $[\gamma \mathbf{p}^{-1}] - e \in ([\gamma \mathbf{p}^{-1}] - e + \varepsilon]$ hold as soon as $\gamma$ verifies the following set of conditions:

$$\left\{ \begin{array}{c} \left[\frac{1}{\gamma} - \frac{1}{2}\right] - \frac{1}{2} \leq \frac{e_i}{\gamma} + \varepsilon \leq \frac{1}{\gamma} - \frac{1}{2} \\ \frac{1}{\gamma} - \frac{1}{2} \leq \frac{e_i}{\gamma} + n \leq \frac{1}{\gamma} - \frac{1}{2} \end{array} \right. \quad \Rightarrow \left\{ \begin{array}{c} \gamma \varepsilon \geq n \\ \gamma \varepsilon \geq 1 \\ \gamma \varepsilon \geq n \end{array} \right. \quad \Rightarrow \gamma \varepsilon \geq n \quad (12)$$

Then Remark 2.2 and (9) allow to conclude the proof.

Theorem 2.4 provides a lower bound for acceptable $\gamma$’s for any basis $\mathbf{R}$ verifying $\sigma \mathbf{R} \leq \frac{1}{2} - \varepsilon$. This enables to directly
2\gamma R' + d 
\mod m_\sigma \text{ and } \mod \gamma

\times |(2d)^{-1}|_{m_\sigma} 
\mod \gamma

= \gamma [cR']^{-1} + [\gamma pR']^{-1} - e \mod m_\sigma = [\gamma pR']^{-1} - e

\times |[\gamma^{-1}_{m_\sigma}] = [cR']^{-1} \mod m_\sigma

Fig. 1: Computing \([cR']^{-1} \mod m_\sigma\) by formula (6), together with strategy described in Corollary 2.5

declare the following corollary, which expresses this bound no more in terms of \(\varepsilon\) but by directly using the value of \(p_m\). It also details how to use previous theorem to perform an exact round-off through formula (6).

**Corollary 2.5:** Let \(R\) be a basis of lattice \(L\) such that \(\sigma p_R < \frac{1}{2}, n \geq 1\) an integer (e.g. for our purpose, \(n\) is size of RNS base \(B\) involved in the modular reduction, Alg. 1), and \(m_\sigma\) a modulus coprime to \(2d\). Let’s denote \(\gamma_{R,n} = [\frac{2n}{2n - 2\sigma p_R}]\) and \(r = (2\gamma R' + d) \mod (2d) + 2d \times e \mod m_\sigma\) with \(e \in [-1, n/\ell]^\ell\). For any \(\gamma\) coprime to \(2d m_\sigma\) and s.t. \(\gamma \geq \gamma_{R,n}\), we have:

\[\gamma [cR']^{-1} \mod m_\sigma = \left(\frac{2\gamma R' + d - r}{2d} - \frac{2\gamma R' + d - r}{2d} \mod \gamma\right) \left[\gamma^{-1}_{m_\sigma}\right].\]  

**Proof:** We fix \(\varepsilon = \frac{1}{2} - \sigma p_R\). It follows that \(\gamma_{R,n} = [\frac{n}{\ell}] = \lfloor \frac{2n}{2n - 2\sigma p_R} \rfloor\) is a lower bound for acceptable odd \(\gamma\)’s with respect to theorem 2.4. Moreover, by using (6) and (8), we can write:

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

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

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

\[\tilde{R} \overset{\text{def}}{=} (2\gamma M' R') \mod (2d), \tilde{d} \overset{\text{def}}{=} (Mr) \mod (2d).\]  

Then, residues modulo \(\gamma\) enable to correct residues modulo \(m_\sigma\) of rounded value \([cR']^{-1}\). Fig. 1 illustrates this approach.

**B. About the size of \(\gamma\)**

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

For implementing an efficient RNS CVP round-off procedure which uses new correction technique, \(\gamma_{R,n}\) should be “reasonably” small, meaning comparatively to word size \(\beta\) of other moduli (i.e. \(m < \beta\) for all \(m \in B\)). Indeed when \(\gamma_{R,n}\) is not larger than \(\beta\), \(\gamma\) can be a single modulus. Then the base \(B'\) in Alg. 1 is just the set of two moduli \(\{\gamma, m_\sigma\}\). In this case the needed comparison to recover a centered remainder modulo \(\gamma\) is directly performed on the residue mod \(\gamma\). Hence the present technique is particularly efficient for matrices \(R\) such that \(0 < \sigma p_R < \frac{1}{2} - \frac{1}{3}\) (i.e. \(\gamma_{R,n} < \beta\)).

In practice, the constraint \(\gamma_{R,n} < \beta\) does not seem to be so restrictive for usual values of \(\beta = 2^\varepsilon\) (i.e. \(\varepsilon \leq 16\)). By analysing basis \(R\) obtained through LLL-reduction of matrices randomly picked up in \([-\ell, \ell]^\ell\) (as suggested in [7]) for some dimensions \(\ell\) from 128 to 768 with respect to \(\sigma = 2\) or 3, it appears (cf. Tab. 1) that \(\gamma_{R,n} < \beta\). As \(\varepsilon\) increases, the bigger \(\gamma_{R,n}\) is, the bigger \(\sigma p_R < \frac{1}{2}\) is, the bigger \(\gamma_{R,n}\) will be.

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

The next part presents a detailed algorithm computing \(c - [cR']^{-1} \mod R\). It is based on 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 \(\beta = 2^\varepsilon\) as word size is fixed from now on. All moduli used are constrained to own only one \(\beta\)-digit.

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

\[q' = \sum_{i=1}^n q_{m_i}, m_i = \lfloor \frac{cR + d}{(2dM_i)^{-1}} \rfloor_{m_i},\]  

\[r = \lfloor 2\gamma cR' + d \rfloor_{2d} + 2d = \frac{cR + d + 2d} {M}\]  

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

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

In \(B\): By using \(R_{m_i}\) and \(d_{m_i}\) in (22) we can re-write (15) with \(q_{m_i} = [cR_{m_i} + d_{m_i}^{-1}]\), 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 \(q_{m_i}\).

**TABLE I:** Statistics about \(\gamma_{R,n}\) for 100 random LLL-reduced matrices \(R\) for each dim. \(\ell\).

<table>
<thead>
<tr>
<th>(\ell)</th>
<th>128</th>
<th>256</th>
<th>384</th>
<th>512</th>
<th>640</th>
<th>768</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\sigma)</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>max</td>
<td>2003</td>
<td>4715</td>
<td>3637</td>
<td>4059</td>
<td>683</td>
<td>334</td>
</tr>
<tr>
<td>min</td>
<td>10</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>median</td>
<td>66.5</td>
<td>37.5</td>
<td>31</td>
<td>28.5</td>
<td>21</td>
<td>9</td>
</tr>
<tr>
<td>mean diff.</td>
<td>193.7</td>
<td>260.9</td>
<td>162.6</td>
<td>193.6</td>
<td>41.6</td>
<td>13.2</td>
</tr>
</tbody>
</table>
In \{\gamma\}: If we develop right member of (17) modulo \(\gamma\) by using (15) and (16) we obtain:

\[
|s|_\gamma = \left[ c \frac{-\tilde{R}}{2dM} + \frac{d}{2dM} - \sum_{i=1}^{n} \frac{q_{m_i}}{m_i} \right]_\gamma.
\]  

(18)

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

\[
|s|_\gamma = \left( \left( c \tilde{R}_\gamma + \tilde{d}_\gamma + h_\gamma Q \right) \right) \mod \gamma = [\gamma p R^{-1}] - e.
\]  

(19)

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

\[
|c - [c R^{-1}] R|_{m_\sigma} = |c - \gamma (s - s_\gamma) R|_{m_\sigma}.
\]  

(20)

where \(|s|_{m_\sigma} = \frac{\tilde{R} - 2\gamma M R}{2dM} + \frac{d - M d}{2dM} + \sum_{i=1}^{n} \frac{q_{m_i}}{m_i} |\rangle_{m_\sigma} \). Thus precomputed data (22) allow to write:

\[
|c - [c R^{-1}] R|_{m_\sigma} = \tilde{c} R_\sigma + \tilde{d}_\sigma + (h_\sigma Q + s_\gamma) R_\sigma |\rangle_{m_\sigma}.
\]  

(21)

2) Main Algorithm 2: 

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

Complexity analysis: 

Complexity of RNS algorithms is generally expressed in terms of elementary modular multiplications with respect to some moduli \(m\) such that \(m < \beta = 2^r\). To simplify complexity analysis, \(\text{EMM}_\beta \) 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 \(2^{[\log_2(m_\sigma)\}]}\). 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 \((n + 1)\ell \times [\log_\beta(c_x)] \text{EMM}_\beta + \ell \times [\log_\beta(c_x)] \text{EMM}_\beta\). When this conversion is done, the cost of steps 2 to 11 of RNS CVP algorithm 2 is \((n + 1)\ell^2 + n\ell) \text{EMM}_\beta + (2\ell^2 + n\ell) \text{EMM}_\beta\).

Space efficiency: 

Precomputed powers of \(\beta\) for binary-to-RNS conversion cost no more than \((n + 1)([\log_\beta(c_x)] - 1)\) \(\beta\)-words and \([\log_\beta(c_x)] - 1)\) \(\beta\)-words. Precomputations (22) represent \((n + 1)(\ell^2 + \ell) + n\) \(\beta\)-words and \(2\ell^2 + \ell + n\)

Algorithm 2: Full RNS Rounding-off CVP algorithm

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

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

1: \((c_m)_{m \in B}; (\gamma, m_\sigma)\) \(-\) Bin_to_RNS(c) \#1st \ parallel \ step

2: for \(i = 1\ to \ n\ do\ #2nd \ parallel \ step

3: \(Q_{i,\sigma} \leftarrow c_m, R_{m_i} + \tilde{d}_m, m_i\) \#3rd \ parallel \ step

4: end \ for

5: \(s_i \leftarrow [c \tilde{R}_\gamma + \tilde{d}_\gamma]_\gamma\ #2nd \ parallel \ step; \ part \ of \ (19)

6: \(p_{m_\sigma} \leftarrow [c_{m_\sigma} R_{\sigma} + d_{m_\sigma}]_\gamma\ #2nd \ parallel \ step; \ part \ of \ (21)

7: \(1_{\gamma} \leftarrow \langle h_\sigma \rangle_\gamma\ \langle h_\sigma \rangle_\gamma\ Q \in \mathbb{Z}^2 \times \ell\ \#3rd \ parallel \ step

8: \(s_i \leftarrow [s_i + \ell_\gamma]\ mode \gamma\ #4th \ parallel \ step; \ (19)

9: \(t_{m_\sigma} \leftarrow [q_{\sigma} + s_\gamma]\ mode \gamma\ #5th \ parallel \ step; \ part \ of \ (21)

10: \(p_{m_\sigma} \leftarrow [p_{m_\sigma} + t_{m_\sigma}] R_{\sigma}\ #6th \ parallel \ step; \ part \ of \ (21)

11: \(return\ p_{m_\sigma} \mod \sigma\_\sigma\)

\[\begin{array}{cc}
\ell & 64 & 128 & 256 & 512 & 1024 \\
\hline
\frac{r = 24}{6.99} & +6.99 & +3.24 & +1.56 & +0.67 & +0.34 \\
\frac{r = 32}{8.08} & +8.08 & +3.74 & +1.77 & +0.84 & +0.41 \\
\frac{r = 64}{12.48} & +12.48 & +5.77 & +2.67 & +1.72 & +0.60 \\
\end{array}\]

TABLE II: Memory overhead of RNS precomputations comparatively to binary storage of \(R^{-1}\) for several word-size \(\beta = 2^r\) and dimensions \(\ell\).
TABLE IV: Number of $\text{EMM}_3$ overhead of [9] comparatively to present full RNS approach for several word-size $\beta = 2^r$ 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>+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 III: Memory overhead of [9] comparatively to present full RNS approach for several word-size $\beta = 2^r$ 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>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>32</td>
<td>+123.1%</td>
<td>+130.5%</td>
<td>+136.4%</td>
<td>+141.7%</td>
<td>+146.6%</td>
</tr>
<tr>
<td>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>

The only extra cost is due to $|m_i^{-1}|_{m_\sigma}$ for second conversion. Next, MRS conversion requires to precompute $\frac{k(k-1)}{2}$ residues $|(m_i^0)^{-1}|_{m_i}$ for $1 \leq i < j \leq k$. A similar analysis leads to an extra cost in terms of $\text{EMM}_3$ given by $(k\ell^2 + k\ell(n + 1) + \frac{k(k-1)}{2})$ $\text{EMM}_3$. Tab. III and IV show these overheads with bounds $c_\beta = d \leq \ell^2\ell$. 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 an 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 $\sum_{i=1}^{n} a_i b_i \mod m_i$. 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 $m_i = 2^r - \mu_i$ with $\mu_i < 2^{r/2}$, allowing a direct reduction, or by using a second level of Montgomery reduction as proposed in [17] with $\text{gcd}(m_i, 2^r) = 1$.

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 $3\ell$ 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 $\sum_{i=1}^{n} a_i b_i \mod m_i$. 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 $\text{modc}$ function (line 8), two tricks are used to make it easy in hardware without using signed numbers. First, one can see that $s_{\gamma,i} \text{modc } \gamma = \begin{cases} s_{\gamma,i} & \text{if } s_{\gamma,i} \leq \lfloor \gamma/2 \rfloor \\ s_{\gamma,i} - \gamma & \text{otherwise} \end{cases}$. 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 $\text{modc } \gamma$. Because $m_\sigma$ is small ($\leq 7$ as $\sigma \in \{2, 3\}$), and because of the trick for $\gamma$, we dedicate two Rower units to computations $\text{modc } m_\sigma$ and $\text{modc } \gamma$. 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 \( a, b \in \mathbb{Z}/2m \mathbb{Z} \), \( s \) parameter for lazy-reduction, \( -m^{-1} \) mod \( 2^r+s \) and \( d \) the pipeline depth

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

1: \( t \leftarrow a_1 \times b_1 \) #Cycle 1: Multiply
2: for \( i \leftarrow 2 \) to \( k \) do #Cycle 2 to \( k 
3: \quad t \leftarrow t + a_i \times b_i \) #Mult. and get \( r+s \) LSB
4: end for
5: \( q \leftarrow (t \times m^{-1}) \) mod \( 2^r+s \) #Cycle \( k+1+d \): Mult. and get \( r+s \) LSB
6: \( \tilde{q} \leftarrow \frac{q}{2^{q / m}} \) #Cycle \( k+1+2d \): Mult., cycle \( k+2+2d \): add and get \( r+1 \) MSB
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 \( c_x \) and \( d \).

**A. Performance and Memory Analysis**

With \( n+2 \) parallel Rowers, binary-to-RNS conversion requires \( \log_\beta(c_x) - 1 \leq n \) 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.

<table>
<thead>
<tr>
<th>Lines of Algorithm 2</th>
<th>Cycles</th>
<th>Memory used (nb of words)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( nf )</td>
<td>((n+1)[n+2] )</td>
</tr>
<tr>
<td>2 to 6</td>
<td>((l+4)/l )</td>
<td>((n+2)[(l+1)/l+2] )</td>
</tr>
<tr>
<td>7</td>
<td>((n+3)/l )</td>
<td>(2[n+2] )</td>
</tr>
<tr>
<td>8</td>
<td>(l+3 )</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>(l+3 )</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>((l+4)/l )</td>
<td>((l+1)/l+2 )</td>
</tr>
<tr>
<td>Total</td>
<td>((2+2n+13)/l+6 )</td>
<td>((n+2)[(l+2)+l+1] ) +(l^2+l+2+n )</td>
</tr>
</tbody>
</table>

**TABLE V: Performance and Memory Size**

**Algorithm 4** Full RNS CVP Detailed: \( Q \) computation step

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

Ensure: \( Q \in \mathbb{Z}_{c}^{\times}/R \) s.t. \( Q_i \cdot a = ([cR + d]_{m_i})^l \) for \( 1 \leq i \leq n \)

1: for \( i \leftarrow 1 \) to \( n \) do #concurrently in \( n \) Rowers
2: \( \bar{Q}_i,x \leftarrow (0)^i \)
3: for \( y \leftarrow 1 \) to \( \ell \) do \#first loop: \( \ell \) cycles
4: \( \bar{Q}_i,y \leftarrow \bar{Q}_i,y + c_{m_ix} \times (\bar{R}_{m_i})_x,y \)
5: end for
6: \( Q_i,y \leftarrow \bar{Q}_i,y + (\bar{a}_{m_i})_y \) #last add.: line 4 of Alg. 2
7: end for
8: for \( x \leftarrow 1 \) to \( \ell \) do #this loop: \( \ell \times \ell \) cycles
9: \( Q_{x,y} \leftarrow Q_{x,y} \times -m_i^{-1} \) mod \( 2^r+s \)
10: end for #this loop: \( \ell \) cycles
11: for \( x \leftarrow 1 \) to \( \ell \) do #no idle cyc. when pipeline depth \( \ell \)
12: \( Q_{x,y} \leftarrow Q_{x,y} + q_{m_i,x} \times m_{x,y} \)
13: end for #this loop: \( 2\ell \) cycles
14: end for #this loop: \( \ell \times \ell \) cycles
15: return \( Q \)

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

<table>
<thead>
<tr>
<th>( (\ell, n) )</th>
<th>Mem. Row (Elts)</th>
<th>Area Row (LUT/Rep./DSP/BRAM)</th>
<th>Freq. (MHz)</th>
<th>Tot. Area (estimation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(64,16)</td>
<td>4162</td>
<td>19</td>
<td>400</td>
<td>304</td>
</tr>
<tr>
<td></td>
<td>(30/17/2/15)</td>
<td></td>
<td>(480/272/32/240)</td>
<td>722</td>
</tr>
<tr>
<td>(128,38)</td>
<td>16514</td>
<td>19</td>
<td>300</td>
<td>(1140/646/76/2128)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>( (\ell, n) )</th>
<th>Mem. Row (Elts)</th>
<th>Area Row (LUT/Rep./DSP/BRAM)</th>
<th>Freq. (MHz)</th>
<th>Tot. Area (estimation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(64,16)</td>
<td>4162</td>
<td>20</td>
<td>468</td>
<td>320</td>
</tr>
<tr>
<td></td>
<td>(60/17/2/15)</td>
<td></td>
<td>(960/272/32/240)</td>
<td>760</td>
</tr>
<tr>
<td>(128,38)</td>
<td>16514</td>
<td>20</td>
<td>415</td>
<td>(2280/646/76/2128)</td>
</tr>
</tbody>
</table>

**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 \( R \) 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


