# Programmable RNS Lattice-Based Parallel Cryptographic Decryption

Paulo Martins[†], Leonel Sousa[†]
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
Rua Alves Redol, 9, 1000-029 Lisboa, Portugal
Email: paulo.sergio@netcabo.pt; las@inesc-id.pt

Julien Eynard[*], Jean-Claude Bajard[*]
Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6 UMR 7606, 4 place Jussieu 75005 Paris
Email: {julien.eynard, jean-claude.bajard}@lip6.fr

***Abstract*—Should quantum computing become viable, current public-key cryptographic schemes will no longer be valid. Since cryptosystems take many years to mature, research on post-quantum cryptography is now more important than ever. Herein, lattice-based cryptography is focused on, as an alternative post-quantum cryptosystem, to improve its efficiency. We put together several theoretical developments so as to produce an efficient implementation that solves the Closest Vector Problem (CVP) on Goldreich-Goldwasser-Halevi (GGH)-like cryptosystems based on the Residue Number System (RNS). We were able to produce speed-ups of up to 5.9 and 11.2 on the GTX 780 Ti and i7 4770K devices, respectively, when compared to a single-core optimized implementation. Finally, we show that the proposed implementation is a competitive alternative to the Rivest-Shamir-Adleman (RSA).**

## I. Introduction

Most public-key cryptosystems currently in use, such as RSA, rely on the intractability of factoring and computing discrete logarithms. However, in 1994, Shor proposed efficient quantum algorithms to solve these problems [1]. Hence, should quantum computing become viable, currently-in-use cryptosystems will be broken. As such, research on efficient post-quantum public-key cryptosystems is most valuable.

Lattice-Based Cryptosystems (LBCs) hold a great promise for post-quantum computing since lattice-based problems are thought to be hard even in a quantum computing setting. This type of cryptography appeared during the 90s, with proposals such as GGH [2] and NTRU [3]. In GGH-like approaches, a plaintext is a small "error" added to a vector of a lattice. The public key corresponds to a "bad basis" of the lattice, with which solving the CVP is hard, and the private key is a "good basis", enabling to compute the closest vector for small errors. The decryption just consists on solving the CVP.

The security of GGH and NTRU is assumed to rely on the hardness of CVP, but none reduction proof exists yet. In practice, the size of parameters depends on the complexity of best known attacks such as lattice reductions. In their first form, GGH signature and encryption schemes were severely broken [4], [5] and thwarting these attacks is still a current concern [6]–[8]. Recently, GGH has got many improvements [9]–[11] to make it competitive under secure parameters. This article deals with efficient implementation of GGH decryption function, via an arithmetical approach as suggested in [12].

The RNS allows to split large integer arithmetic over many small finite fields, enabling the exploitation of the integer-arithmetic architectural optimizations of most programmable platforms, while removing the overhead associated to working with multi-precision integers. Further, since each small finite field operates independently, the RNS is prone to data parallelism. Since both multi-core Central Processing Units (CPUs) and Graphic Processing Units (GPUs) have become widely available during the past decades [13], herein, the aforementioned RNS lattice-based cryptographic approaches are analyzed, further developed, and their performance is evaluated for multi-core CPUs and GPUs.

The rest of this paper is organized as follows. In Section II, lattice theory and the RNS are reviewed. In Section III, GGH decryption is analyzed, with a focus on how RNS may be exploited. This procedure is parallelized and implemented; and evaluated and compared to related work in Sections IV, and V, respectively. Then, its performance is compared to that of RSA. Finally, conclusions are drawn.

## II. Background

A full-rank lattice $\mathcal{L}$ is defined as the set of all integral combinations of $n$-linearly independent vectors $r_0, \ldots, r_{n-1} \in \mathbb{R}^n$. If the basis is represented as a matrix $R$, having the basis vectors as rows, the lattice generated by $R$ can be defined as $\mathcal{L}(R) = \{xR : x \in \mathbb{Z}^n\}$, where $x$ is represented as a row-vector, and $xR$ denotes the usual vector-matrix multiplication. Herein, vectors $r_0, \ldots, r_{n-1}$ are restricted to $\mathbb{Z}^n$.

If $U$ is a unimodular matrix (i.e., an integer square matrix with determinant $\pm$ 1), the basis $R$ and $UR$ generate the same lattice. In fact, any lattice admits an infinite number of bases as soon as $n \geq 2$. Additionally, due to its periodicity, every lattice $\mathcal{L}$ induces an equivalence relation over $\mathbb{Z}^n$ defined as follows: $v \equiv_{\mathcal{L}} w$ if and only if $v - w \in \mathcal{L}$. Reducing a vector

[†]This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013.
[*]This work has been supported in part by the European Unions H2020 Programme under grant agreement number ICT-644209.

```
1: Input: $p \in \mathbb{Z}^n$, $B0 \in \mathbb{Z}^n$
2: Output: $c \in \mathbb{Z}$
3: $c \leftarrow p[0]$
4: for $i \leftarrow n-1$ to 1 do
5:     $c \leftarrow c - p[i] \times B0[i]$
6: end for
7: $c \leftarrow c \bmod B0[0]$
8: return $c$
```

Fig. 1. Encryption Algorithm

```
1: Input: $c \in \mathbb{Z}$, $R_0^{-1} \in \mathbb{Q}^n$, the first row of $R^{-1}$
2: Output: $p \in \mathbb{Z}^n$
3: $p \leftarrow (c, 0, \ldots, 0) - \lfloor cR_0^{-1} \rceil R$
4: return $p$
```

Fig. 2. Decryption Algorithm

$v$ modulo a basis $R$ corresponds to finding the unique point $w$ in $\{\sum_{i=0}^{n-1} x_i r_i : 0 \leq x_i < 1\}$ such that $v - w \in \mathcal{L}$.

### A. Lattice-Based Cryptography

A lattice-based encryption scheme will now be described, using Rose's approach [10]. The private basis, $R$, is produced as a rotated nearly-orthogonal basis, such that Babai's Round-Off procedure [14] may be used to compute the closest vector. Moreover, the public basis is of a Hermite Normal Form (HNF). The HNF is a basis of $\mathcal{L}(R)$, $B \in \mathbb{Z}^{n \times n}$, such that $B_{i,j} = 0$ if $i < j$, $B_{i,j} \geq 1$ if $i = j$, and $B_{i,j} < B_{j,j}$ if $i > j$. This unique form can be computed from any basis of $\mathcal{L}$. Therefore, it is the worst possible basis for $\mathcal{L}$ from a cryptanalyst's point of view. In particular, Rose's cryptosystem uses bases of an Optimal Hermite Normal Form (OHNF) as the public-key. OHNFs form a subclass of HNFs, where all but the first column are trivial. Concretely, $B$ is an OHNF basis of $\mathcal{L}$ if and only if $B$ is an HNF basis and $\forall_{0<i<n}, B_{i,i} = 1$.

Since the public-key is of an OHNF, it is representable by a single column, denoted as $B0$. A plain-text is then represented as a vector $p \in \mathbb{Z}^n$. To encrypt it, $p$ is reduced modulo the public basis, by applying the algorithm of Figure 1. Due to the public basis structure, the cryptogram corresponds to a vector, where all the entries but the first are zero. Thus, it suffices a scalar $c$ to store its value. In order to decipher $c$, Babai's Round-Off algorithm is applied. This procedure, represented in Figure 2, gives an approximation to the CVP. If

$$p \in \left[ -\left\lceil \frac{\sqrt{n}}{2} \right\rceil + 1, \left\lceil \frac{\sqrt{n}}{2} \right\rceil - 1 \right]^n \tag{1}$$

then the algorithm produces the correct closest vector [10].

### B. Residue Number System

The computation of multi-precision arithmetic may be split among several channels by exploiting the RNS. Under this system, numbers are represented as their remainders when divided by the set of co-primes $m_0, m_1, \ldots, m_{s-1}$ that form the RNS basis. Additions, subtractions and multiplications modulo $M = \prod_{i=0}^{s-1} m_i$ can then be performed independently for each modulo of the set. Furthermore, the Chinese Remainder Theorem (CRT) provides a formula to recover the value of $A$ from $a_i = A \bmod m_i$ for $0 \leq i < s$, for $0 \leq A < M$:

$$\begin{aligned} A &= \textstyle\sum_{i=0}^{s-1} (a_i \left(\frac{M}{m_i}\right)^{-1} \bmod m_i) \frac{M}{m_i} \bmod M \\ &= \textstyle\sum_{i=0}^{s-1} (a_i \left(\frac{M}{m_i}\right)^{-1} \bmod m_i) \frac{M}{m_i} - kM \end{aligned} \tag{2}$$

where $\left(\frac{M}{m_i}\right)^{-1}$ is the multiplicative inverse of $\frac{M}{m_i}$ modulo $m_i$.

Modular reduction may be implemented using an adaptation of Montgomery's algorithm [15] to RNS [16]. The RNS Montgomery algorithm replaces the modulo of the reduction $D$, by another more suitable modulo $M_1$. With that purpose, $Q$ is defined to be the value that satisfies $A + QD \equiv 0(\bmod M_1)$, for $M_1 \times D > A$ and $M_1$ co-prime to $D$. The value of $Q$ may be computed using an RNS set $\mathcal{M}_1$ such that $M_1 = \prod_{m \in \mathcal{M}_1} m$, and setting $Q \leftarrow -AD^{-1}(\text{mod} M_1)$. Afterward, $Q$ is extended to another basis $\mathcal{M}_2$, and $Z \leftarrow (A + QD)M_1^{-1}$ is computed. In the end $Z$ satisfies the following: $Z \equiv AM_1^{-1}(\text{mod} D)$ and $Z < 2 \times D$. If a full reduction is required, $Z$ must be compared with $D$, so that $D$ is subtracted from $Z$ when $Z \geq D$. However, the RNS is not of a positional nature, and therefore this comparison cannot be directly and easily computed. Herein, numbers are converted to the Mixed Radix System (MRS) to be compared. If $\bar{z}_0, \bar{z}_1, \ldots, \bar{z}_{s-1}$ corresponds to the MRS representation of $Z$, with respect to the basis $m_0, m_1, \ldots, m_{s-1}$, then $Z$ takes the value of $Z = \bar{z}_0 + \bar{z}_1 m_0 + \ldots + \bar{z}_{s-1} m_0 \ldots m_{s-2}$.

## III. RNS Babai's Round-Off

Exploiting RNS to accelerate Babai's Round-Off algorithm requires the computation of Figure 2 to be converted to integer arithmetic [12]. Hence $R' = \det(R)R^{-1}$ ($R' \in \mathbb{Z}^{n \times n}$) is used instead of $R^{-1}$. Moreover $\lfloor cR_0^{-1} \rceil$ is rewritten as:

$$\lfloor cR_0^{-1} \rceil = \frac{2cR'_0 + \det(R)v_1 - [2cR'_0 + \det(R)v_1 \bmod 2\det(R)]}{2\det(R)} \tag{3}$$

where $v_1 = (1, \ldots, 1)$. Furthermore, since $p$ is restricted as stated in (1), if $\beta > 2\left\lceil \frac{\sqrt{n}}{2} \right\rceil - 1$ then the computation of Figure 2 may be performed modulo $\beta$. In this work, the value $\beta$ was selected to be $m_{2,0}$, that is, the value of the first element of the second RNS Montgomery basis.

A description of the resulting algorithm can be found in Figure 3. The value of $v'_1$ corresponds to $\det(R)$, and is used to compute $a \leftarrow 2cR'_0 + (v'_1, \ldots, v'_1) \bmod m_{2,0}$. Afterward, the value of $a \bmod D_R$ is determined, where $D_R = 2 \times \det(R)$, using RNS. In order to compute this value, $R''_0 \leftarrow 2R'_0 M_1 m_3 \bmod D_R$ and $v''_1 \leftarrow \det(R)M_1 m_3 \bmod D_R$ are precomputed. The value of $M_1$ regards the first Montgomery basis, and $m_3$ is an extra modulo that is introduced to simplify the reduction operation. The extra $M_1 m_3$ term will be eliminated during the Montgomery reduction.

The ReduceModDR function is described in Figure 4. First, $q_1$ is computed such that $cR''_0[i] + v''_1 + q_1 D_R$ is divisible by $M_1$. Since $cR''_0[i] + v''_1 < \det(R)D_R + D_R$, $M_1$ should satisfy $M_1 > \det(R) + 1$. Afterward, $q_1$ is extended to the second basis $\mathcal{M}_2$, by evaluating (2) for each of the moduli of $\mathcal{M}_2$. In the figure, $m_{j,i}$ corresponds to the $(i+1)$th modulo of base $j$. It should be noted that the value of $k$ in (2) is set to zero, and therefore there will be an extension error less than $(s-1)M_1$ [16]. As such, $a'_2 \leftarrow ((cR''_0[i] + v''_1) + D_R \times q_2) \times M_1^{-1}$ is bounded by $a'_2 < (s+1)D_R$. Hence, $a'_2$ should be reduced a second time, by using an extra modulus $m_3$. If $m_3 > (s+1)$

```
1: Input: $c \in \mathbb{Z}$, $R'_0 \bmod m_{2,0} \in \mathbb{Z}^n$, $R''_0 \in \mathbb{Z}^n$, $R \bmod m_{2,0} \in \mathbb{Z}^{n\times n}$, $v'_1 \in \mathbb{Z}$, $v''_1 \in \mathbb{Z}$
2: Output: $p \in \mathbb{Z}^n$
3: $a \leftarrow 2cR'_0 + (v'_1, \ldots, v'_1) \bmod m_{2,0}$
4: $a' \leftarrow \text{ReduceModDR}(c, R''_0, v''_1)$
5: $b \leftarrow \frac{a-a'}{2\det(R)}$
6: $p \leftarrow (c, 0, \ldots, 0) - bR \bmod m_{2,0}$
7: return $p$
```

Fig. 3. Improved Decryption Algorithm

```
1: Input: $c \in \mathbb{Z}$, $R''_0 \in \mathbb{Z}^n$, $v''_1 \in \mathbb{Z}$, RNS constants
2: Output: $a' \in \mathbb{Z}^n$
3: for $i \leftarrow 0$ to $n-1$ do
4:     $q_1 \leftarrow (-D_R^{-1})(cR''_0[i] + v''_1)$ in $\mathcal{M}_1$
5:     $q_2 \leftarrow \sum_{i=0}^{s-1}(q_{1,i}\left(\frac{M_1}{m_{1,i}}\right)^{-1} \bmod m_{1,i}) \times \frac{M_1}{m_{1,i}}$ in $\mathcal{M}_2$
6:     $a'_2 \leftarrow ((cR''_0[i] + v''_1) + D_R \times q_2) \times M_1^{-1}$ in $\mathcal{M}_2$
7:     $q_3 \leftarrow \sum_{i=0}^{s-1}(q_{1,i}\left(\frac{M_1}{m_{1,i}}\right) \bmod m_{1,i})\frac{M_1}{m_{1,i}} \bmod m_3$
8:     $a'_3 \leftarrow ((cR''_0[i] + v''_1) + D_R \times q_3) \times M_1^{-1} \bmod m_3$
9:     $\hat{q} \leftarrow (D_R^{-1})a'_3 \bmod m_3$
10:    $a'_2 \leftarrow (a'_2 - D_R \times \hat{q}) \times m_3^{-1}$ in $\mathcal{M}_2$
11:    for $j \leftarrow 0$ to $s-1$ do
12:        for $k \leftarrow j+1$ to $s-1$ do
13:            $a'_{2,k} \leftarrow (a'_{2,k} - a'_{2,j})m_{2,j}^{-1} \bmod m_{2,k}$
14:        end for
15:    end for
16:    if $a'_{2,s-1} < \lfloor \frac{m_{2,s-1}}{2} \rfloor$ then
17:        $a'[i] = a'_{2,0}$
18:    else
19:        $a'[i] = a'_{2,0} + D_R \bmod m_{2,0}$
20:    end if
21: end for
22: return $a'$
```

Fig. 4. RNS Modular Reduction (ReduceModDR)

$(\forall_{0\leq i<s}, m_3 < m_{2,i})$, $a'_2 \leftarrow (a'_2 - D_R \times \hat{q}) \times m_3^{-1}$ is computed, and $-D_R < a'_2 < D_R$, where $\hat{q} \leftarrow D_R^{-1}a'_3 \bmod m_3$.

If $a'_2 < 0$, $D_R$ must be addded so that the result is fully reduced. When $a'_2 < 0$, its RNS representation corresponds to $M_2 + a'_2$. If $M_2$ is chosen such that $D_R < \lfloor \frac{m_{2,s-1}}{2} \rfloor \frac{M_2}{m_{2,s-1}} < M_2 - D_R$, checking if $a'_2 < 0$ is equivalent to testing if $\bar{a}'_{2,s-1} \geq \lfloor \frac{m_{2,s-1}}{2} \rfloor$, where $\bar{a}'_{2,s-1}$ denotes the $s^{th}$ MRS digit of $a'_2$. In the algorithm, the RNS digits are overwritten with the MRS digits, and computed in the most inner loop. Afterward, $D_R$ is added to the result modulo $m_{2,0}$ if $a'_2 < 0$. Subsequent to the addition, the returned value from the function in figure 4 corresponds to $a' \leftarrow a \bmod D_R \bmod m_{2,0}$. The plain-text $p$ is then evaluated as $p \leftarrow (c, 0, \ldots, 0) - \frac{a-a'}{2\det(R)}R \bmod m_{2,0}$.

## IV. Parallelization and Implementation Details

In this section, parallelism is exploited to speed up the execution of the presented algorithms. Several Application Programming Interfaces (APIs) were used in this work to exploit different levels of data parallelism, namely OpenCL [17] for GPU programming, OpenMP [18] for exploiting multi-threaded CPU parallelism, and AVX2 [19] for Single Instruction Multiple Data (SIMD) parallelism.

```
1: Input: $z \in \mathbb{Z}, m_i \in \mathbb{Z}$; Output: $z \in \mathbb{Z}$
2: while $z \geq 2^l$ do
3:     $z_L = z\&(2^l - 1)$; $z_H = z >> l$; $z = z_L + (2^l - m_i)z_H$
4: end while
5: return $z \leftarrow min(z, z - m_i)$
```

Fig. 5. GPU Modular Reduction

### A. GPU Approach

As a first approach to the implementation of Babai's Round-Off algorithm, a CPU offloaded the execution of the `ReduceModDR` function to the GPU, transferring the required data. During the GPU execution of line 4 of the algorithm in Figure 3, the CPU executes line 3 simultaneously. After a synchronizing the CPU and the GPU operations, which assures that the `ReduceModDR` results were fully transferred to the CPU, the computation of lines 5 and 6 takes place on the CPU.

The computation of $a \leftarrow 2cR'_0 + (v'_1, \ldots, v'_1) \bmod m_{2,0}$ was split among the cores of the CPU. Each core computed a subset of the result.

Modular reductions of $z$ in channel $m_i$ on the GPU were performed using the algorithm in Figure 5. Further, the values of the moduli were selected such that $2^{l-1} < m_i < 2^l$; the operation $min(z, z - r_i)$ was performed using unsigned arithmetic; and the `while` therein was unrolled.

The `ReduceModDR` function was implemented as an OpenCL `kernel`. Each work-group was associated with a single dimension, and each work-item with a modulo of $\mathcal{M}_1$ and another modulo of $\mathcal{M}_2$. The resulting `kernel` only requires 3 barriers: after lines 4, 9 and 14 of Figure 4. Moreover, lines 7 up to 9, and 16 up to 20 are executed on a single thread. Lastly, since the considered GPUs operated on a maximum of 32-bits, the value $l$ of Figure 5 was set to $l = 16$, so that multiplications did not overflow the result.

After the reduction result is transferred to the CPU, $b \leftarrow \frac{a-a'}{2\det(R)} \bmod m_{2,0}$ is co-jointly computed by multiple threads in the multiple available CPU cores. Then the vector-matrix multiplication $bR \bmod m_{2,0}$ takes place: each core multiplied a set of entries of $b$ by the corresponding lines of $R$, and afterward the partial results of each core were added to produce the multiplication result. Finally, the value of $p \leftarrow (c, 0, \ldots, 0) - bR \bmod m_{2,0}$ is determined, and each core computes a subset of the final result.

### B. CPU Approach

The second approach herein presented is similar to the one for the GPU, except that all computation takes place on the CPU. The steps that were executed on the CPU in Section IV-A take place in a similar way. Additionally, the `ReduceModDR` function, which still made use of the RNS, was enhanced with multi-threading, with each core computing part of the loop iterations in line 3 of Figure 4.

The OpenMP `#pragma omp for` directive was used to split the multi-dimensional computation of $a \leftarrow 2cR'_0 + (v'_1, \ldots, v'_1) \bmod m_{2,0}$, $b \leftarrow \frac{a-a'}{2\det(R)} \bmod m_{2,0}$ and $p \leftarrow (c, 0, \ldots, 0) - bR \bmod m_{2,0}$ among the cores.

The vector-matrix multiplication $bM$ required not only the use of `#pragma omp for` but also of `#pragma omp critical` for the sum of the threads partial results. For executing the `ReduceModDR` function on the CPU, a `#pragma omp for` directive was applied to the line 3 of Figure 4. It should be noted that, since the targeted CPUs featured datapaths of 64-bits, the moduli bit-width was changed to 32-bits.

*1) SIMD Parallelism:* SIMD parallelism was used to enhance the execution on the CPU. Another method was implemented, similar to the previous one, but the `ReduceModDR` was modified to exploit SIMD extensions. First, it was possible to process multiple channels at a time for the steps in lines 4, 6, 10 and 13 of Figure 4. Second, it was possible to accelerate all the summations by splitting their computation over multiple summations and perform those in parallel.

In order to perform multiple operations in parallel, data was loaded to the AVX2 registers using the `vmovdqu` instruction, which loads 256 bits from memory to a register. Then, words were rearranged using `vpshufd` so that the 32 most significant bits of each 64-bit word was set to zero. Multiplications and additions may afterward take place without overflowing the result lanes. Modular reductions after multiplications were performed using the algorithm of Figure 5. Finally, when the desired result is obtained, registers are rearranged using `vpshufd` and `vpunpckldq`, and stored with `vmovdqu`.

## V. Experimental Results

The proposed methods were implemented and thoroughly tested. Also, the sequential method of Figure 2, which does not make use of RNS, was implemented using the NTL 6.2.1 library [20] for comparison purposes. They were tested on three systems: *i)* an i7 3930K with 32GB of RAM and 4 cores, operating at 3.2GHz, and a GeForce GTX 680 with 2GB of main memory with 1536 Shader Processing Units (SPUs), operating at 1GHz; *ii)* an i7 4770K with 32GB of RAM and 4 cores, operating at 3.5GHz, and a Tesla K40c with 12GB of main memory and 2888 SPUs, operating at 0.7GHz; *iii)* an i7 4770K with 32GB of RAM and 4 cores, operating at 3.5GHz, and a GeForce GTX 780 Ti with 3GB and 2880 SPUs, operating at 0.9GHz. All code was compiled with gcc 4.7, with the `-O3` flag, and times were measured using the `readtsc` instruction. 512 random messages were encrypted, and the average decryption time was measured, for $n \in \{400, 600, 800, 1000\}$. The performance is reported in Tables I and II. The RNS-GPU label is used for the approach that implements `ReduceModDR` on the GPU, whereas for the 4-core RNS-CPU label this function runs on the CPU.

The results show that it is possible to similarly enhance the performance on all platforms, when SIMD extensions are not used. Further, since when using RNS it is possible to choose channels whose bit-width is smaller than the word-length of the machine, it is expected that the presented techniques work for a wide range of general-purpose platforms.

The graphics show that there is a direct link between the GPU performance and their memory bandwidth, since the GTX 780 Ti has outperformed the remainder GPUs.

| Execution Times [$\times 10^6$ clock cycles] | | |
|---|---|---|
| Method | $n = 500$ | $n = 800$ |
| [21] | 294.42 | 1323 |

TABLE III
Decryption Performance for the Intel Core 2 Duo Platform.

This results from the low arithmetic intensity of the kernels. Moreover, the K40c and the GTX 680 were outperformed by the i7 platforms. There are two aspects that contribute to this behavior. One is related to the memory transfers between the CPU and the GPU, that must take place when the GPU is used. Even though it is possible to hide part of this overhead by executing line 3 of Figure 3 in parallel on the CPU, this step has a small arithmetic complexity. The other is concerned with the different moduli that are used. Since it is possible to work with moduli whose bit-width is twice as large when only using the CPU, the number of arithmetic operations to be performed is approximately halved.

Finally, AVX2 extensions greatly boosted the performance of the decryption operation. This was only possible due to the use of the RNS which, due to its carry-free nature, is very well suited to speed up computation with SIMD extensions.

In [21], a similar cryptosystem to the herein presented was implemented using the NTL 5.5.2 library on an Intel Core 2 Duo platform, running at 2.1 GHz, with a 4 Gb RAM. Even though different platforms were used, the number of clock cycles reported in Table III are in the same order of magnitude to those of Table I for the sequential method. As such, one may conclude that is most beneficial not only to employ RNS for the whole Babai's Round-Off procedure, but also that LBCs are greatly enhanced with data parallelism.

## VI. Performance Comparison with the RSA cryptosystem

Whereas some related art states that safe implementations of GGH-like cryptosystems should be of dimension at least 400 [5], more pessimistic approximations propose dimensions of at least 800 [22]. We compared the performance of the decryption operation for dimensions of this order of magnitude with the performance of the equivalent RSA operation, for typical security parameters. The RSA cryptosystem was tested using OpenSSL 1.1.0-dev [23] on the i7 4770K platform, and its performance, as well as the performance of the AVX2 implementation of the GGH-like decryption is reported in Figure 6. Notably, OpenSSL makes use of the 128-bits SIMD technology SSE2, in order to accelerate multi-precision integer arithmetic. One concludes that the GGH decryption operation takes approximately the same time to execute for dimensions of 400 and 1000, as the equivalent RSA operations for 3072 and 7680 bits, respectively. Taking into account that the proposed implementation has the advantage of post-quantum security, it presents itself as a competitive alternative to RSA.

## VII. Conclusions and Future Work

In this work, the proposals [11], [12] for using RNS to enhance the decryption procedure of GGH-like cryptosystems

| Execution Times [$\times 10^6$ clock cycles] (Speed-up) | | | | |
|---|---|---|---|---|
| Method | $n = 400$ | $n = 600$ | $n = 800$ | $n = 1000$ |
| Sequential (i7 3930K) | 104.4 | 350.2 | 748.4 | 1350 |
| RNS-GPU (GTX 680) | 22.02 (4.7) | 79.68 (4.4) | 262.5 (2.9) | 348.4 (3.9) |
| 4-core RNS-CPU (i7 3930K) | 18.87 (5.5) | 68.42 (5.1) | 169.9 (4.4) | 384.4 (3.5) |

TABLE I
DECRYPTION PERFORMANCE FOR THE *i)* I7 3930K AND GTX 680 PLATFORM.

| Execution Times [$\times 10^6$ clock cycles] (Speed-up) | | | | |
|---|---|---|---|---|
| Method | $n = 400$ | $n = 600$ | $n = 800$ | $n = 1000$ |
| Sequential (i7 4770K) | 97.51 | 283.8 | 619.4 | 1222 |
| RNS-GPU (K40c) | 22.97 (4.2) | 79.87 (3.6) | 248.9 (2.5) | 512.4 (2.4) |
| RNS-GPU (GTX 780 Ti) | 16.55 (5.9) | 59.73 (4.8) | 148.2 (4.2) | 349.6 (3.5) |
| 4-core RNS-CPU (i7 4770K) | 21.05 (4.6) | 75.48 (3.8) | 189.9 (3.3) | 369.7 (3.3) |
| 4-core RNS-CPU (with AVX2) (i7 4770K) | 8.668 (11.2) | 29.05 (9.8) | 74.78 (8.3) | 148.5 (8.2) |

TABLE II
DECRYPTION PERFORMANCE FOR THE *ii)* I7 4770K AND K40C AND *iii)* I7 4770K AND GTX 780 TI PLATFORMS.

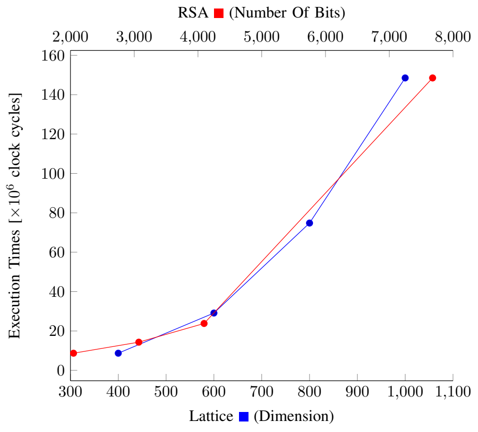

Fig. 6. Execution times for RSA and LBC decryption on the i7 4770K Platform.

were considered and concretized. They were applied not only to GPU accelerators, but also to CPU devices. Maximum speed-ups of 5.9 and 11.2 were obtained for the GTX 780 Ti and i7 4770K devices, respectively, in comparison with a sequential multi-precision floating point approach.

One concludes that due to the burdensome memory transfers between the CPU and the GPU, it is often best to execute the whole decryption procedure on the CPU. Furthermore, since the RNS lends itself to SIMD parallelism, it was possible to greatly boost the performance using the AVX2 extensions. Moreover, it was concluded that LBCs present a competitive post-quantum alternative to RSA.

Future works should focus on arithmetically optimized implementations of alternative cryptographic primitives relying on ideal lattices and Learning With Error problems, which are core components of current homomorphic schemes [24], [25].

## REFERENCES

[1] P.W. Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *IEEE Symp. on Found. of Comp. Sci.*, p.124-134, 1994.

[2] O. Goldreich, S. Goldwasser, and S. Halevi. Public-key cryptosystems from lattice reduction problems. In *Proc. of the 17th Annual Int. Cryptology Conf. on Adv. in Cryptology*, pages 112–131, London, 1997.

[3] J. Hoffstein, J. Pipher, and J.H. Silverman. NTRU: A ring-based public key cryptosystem. In JoeP. Buhler, editor, *Algorithmic Number Theory*, volume 1423 of *Lecture Notes in Comput. Sci.* p.267-288, 1998.

[4] P. Nguyen and O. Regev. Learning a Parallelepiped: Cryptanalysis of GGH and NTRU Signatures. In *Advances in Cryptology - EUROCRYPT 06*, volume 4004 of *Lecture Notes in Comput. Sci.* p.271-288, 2006.

[5] P. Nguyen. Cryptanalysis of the Goldreich-Goldwasser-Halevi cryptosystem from crypto 97. In *In Proc. of Crypto '99, volume 1666 of LNCS*, pages 288–304. Springer-Verlag, 1999.

[6] C.A. Melchor, X. Boyen, J.-C. Deneuville, and P. Gaborit. Sealing the Leak on Classical NTRU Signatures. In *Post-Quantum Cryptography*, volume 8772 of *Lecture Notes in Computer Science*. p.1-21, 2014.

[7] M. Yoshino and N. Kunihiro. Improving GGH cryptosystem for large error vector. In *Information Theory and its Applications (ISITA), 2012 Int. Symp. on*, pages 416–420, Oct 2012.

[8] C. F. de Barros and L. Menasché Schechter. GGH may not be dead after all. In *Anais do XXXV Congresso Nacional de Matemática Aplicada e Computacional, CNMAC 2014*, 2014.

[9] D. Micciancio. Improving lattice based cryptosystems using the Hermite normal form. In *Cryptography and Lattices*, volume 2146 of *Lecture Notes in Comput. Sci.* p.126-145, 2001.

[10] M. Rose, T. Plantard, and W. Susilo. Improving BDD cryptosystems in general lattices. In F. Bao and J. Weng, editors, *ISPEC*, volume 6672 of *Lecture Notes in Comput. Sci.*, pages 152–167. Springer, 2011.

[11] T. Plantard, M. Rose, and W. Susilo. Improvement of lattice-based cryptography using CRT. In *QuantumComm'09*, pages 275–282, 2009.

[12] J.-C. Bajard, J. Eynard, N. Merkiche, and T. Plantard. Babai round-off CVP method in RNS: Application to lattice based cryptographic protocols. In *Integrated Circuits (ISIC), 2014 14th Int. Symp. on*.

[13] J. L. Hennessy and D. A. Patterson. *Computer Architecture, Fifth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edition, 2011.

[14] L. Babai. On Lovász' lattice reduction and the nearest lattice point problem (shortened version). In *Proc. of the 2Nd Symp. of Theoretical Aspects of Comput. Sci.*, STACS '85, pages 13–20, London, UK, 1985.

[15] P. L. Montgomery. Modular Multiplication Without Trial Division. *Mathematics of Computation*, 44:519–519, 1985.

[16] J.-C. Bajard and L. Imbert. A full RNS implementation of RSA. *Computers, IEEE Transactions on*, 53(6):769–774, June 2004.

[17] J. E. Stone, D. Gohara, and G. Shi. Opencl: A parallel programming standard for heterogeneous computing systems. *IEEE Des. Test*, 12(3):66–73, May 2010.

[18] The OpenMP specification for parallel programming. http://www.openmp.org, 2014.

[19] Intel Corporation. Intel intrinsics guide. https://software.intel.com/sites/landingpage/IntrinsicsGuide/.

[20] V. Shoup. NTL 6.6.1: A library for doing number theory. www.shoup.net/ntl, 2014.

[21] M.l Rose. Lattice-based cryptography: a practical implementation. Master's thesis, School of Computer Science and Software Engineering, University of Wollongong, 2011.

[22] C. Ludwig. The security and efficiency of Micciancio's cryptosystem. *IACR Cryptology ePrint Archive*, 2004:209, 2004.

[23] The OpenSSL Project. OpenSSL: Cryptography and SSL/TLS toolkit. www.openssl.org, March 2015.

[24] C. Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *Proc. of the Forty-first Annual ACM Symp. on Theory of Computing*, STOC '09, pages 169–178.

[25] Z. Brakerski and V. Vaikuntanathan. Efficient Fully Homomorphic Encryption from (Standard) LWE. In *Proc. of the 2011 IEEE 52nd Annu. Symp. on Found. of Comput. Sci.*, pages 97–106.