

Combining Residue Arithmetic to Design Efficient Cryptographic Circuits and Systems

Leonel Sousa, Samuel Antão,
and Paulo Martins

Abstract

Cryptography plays a major role assuring security in computation and communication. In particular, public-key cryptography enables the asymmetrical ciphering of data along with the authentication of the parties that are attempting to share data. The computation of asymmetrical encryption is costly, thus it has motivated extensive research to efficiently accelerate the execution of the most relevant algorithms and improve resistance against Side-Channel Attacks (SCAs), which leverage exposed features by the cryptographic systems, such as power consumption and execution timings, to gain access to private information. Herein, we present a state-of-the-art overview of the use of the Residue Number System (RNS) to exploit parallelism in the computation of the most important public-key algorithms. We also address how it can be exploited to prevent side-channel attacks. The experimental results presented in the literature show that not only the currently used RSA and Elliptic Curve Cryptographic (ECC) algorithms but also emerging post-quantum algorithms, namely the ones supporting Lattice-based Cryptosystems (LBCs), can take advantage of the RNS. It enables the design of more efficient cryptographic systems and also reinforces the prevention of side-channel attacks, improving their security. Finally, we also present the characteristics of the *Computing with the Residue Number System Framework* (CRNS), which aims to automatize the design of fully functional cryptographic accelerators based on RNS.

I. Introduction

The Residue Number System (RNS) was initially proposed back in the 1950s [1]. The RNS allows large bit-width integer arithmetic to be split into small computation channels. Small residues can be processed in parallel avoiding the lengthy carry propagation chains. The direct application of these properties to addition and multiplication widely disseminated the usage of RNS to design efficient systems for linear signal and image processing [2]. However,

since the word-length of most programmable accelerators and processors is sufficient to perform fixed-point signal processing, and most provide sub-word processing capabilities, usually tailored for multimedia applications [3], the use of RNS for signal processing has been mostly directed to the design of ASICs. The word-length of the data-path of those ASICs is adjusted to the width of the RNS channels that are processed in parallel. Recently, RNS has found renewed and extensive application in cryptography, namely for public-key cryptography, not only because of efficiency but also to reinforce the security of cryptosystems as discussed later in this paper.



Public-key cryptosystems underpin the establishment of secure sessions, when no secret information has been previously shared (apart from certificates of trusted authorities), and provide reliable mechanisms to generate and verify signatures [4]. The typical operations of public-key cryptosystems are depicted in Figure 1.

The typical application of public-key cryptography is as follows. A user, Alice, generates a pair of keys, as shown in Figure 1(a). Whereas one of these keys is private, k_A , and should be safely stored, the other is public, K_A , and can be widely distributed. The public-key is bound

to Alice through trusted authorities, which gives confidence to Bob that K_A is effectively the public-key of Alice and not one of an attacker. When Bob wants to send a message m_B to Alice, he can encrypt the message with Alice's public-key, producing c , as depicted in Figure 1(b). c can only be decrypted with Alice's private-key. Finally, some public-key cryptosystems allow Alice to produce a signature using her private-key, as shown in Figure 1(c). To increase the performance of this procedure, signatures are usually applied to shorter forms of data, called hashes or message digests, instead of the entire set of data. This feature is useful, for instance, when distributing trusted software. Bob can verify that the software remains untampered after receiving it through Alice's signature, and only install it if it passes the verification tests.

The most known public-key algorithms, the Rivest-Shamir-Adleman (RSA) [5] and the ones based on the Elliptic Curve (EC) cryptography, proposed simultaneously by Koblitz [6] and Miller [7] in 1985, rely on the intractability of factoring and computing discrete logarithms using the computing capabilities of today. At the time of its proposal, EC cryptography suggested computing efficiency improvements of about 20% when compared with the RSA protocol, which were expected to increase as the computing capabilities of the target

Leonel Sousa and Paulo Martins are with INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal (e-mail: las@inesc-id.pt/paulo.sergio@netcabo.pt). Samuel Antão is with IBM T.J. Watson Research Center (e-mail: samuellantao@gmail.com).

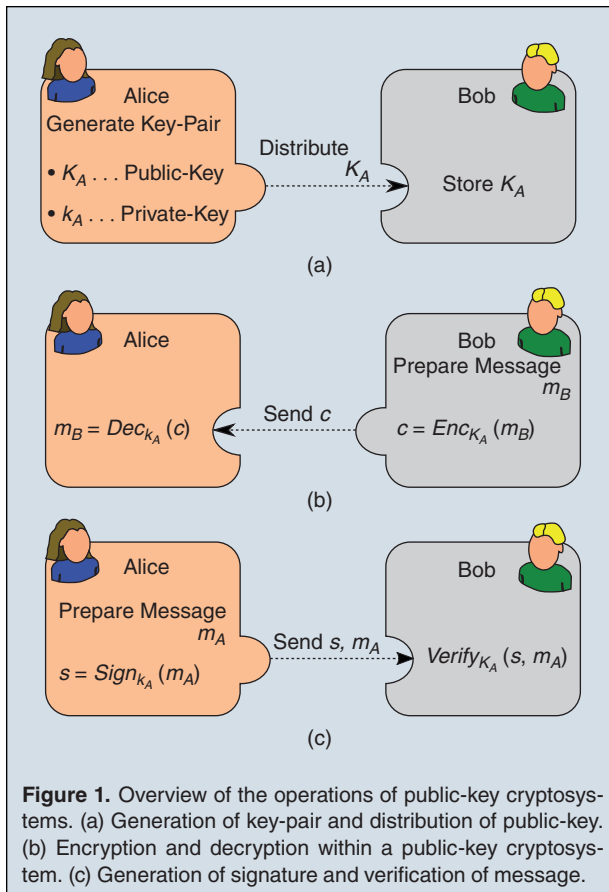


Figure 1. Overview of the operations of public-key cryptosystems. (a) Generation of key-pair and distribution of public-key. (b) Encryption and decryption within a public-key cryptosystem. (c) Generation of signature and verification of message.

devices improve. More recently, Lattice-Based Cryptosystems (LBCs) have attracted considerable attention [8] since lattice-based problems, as opposed to the currently-in-use cryptosystems, have no known systematic attacks even in a quantum computing setting. This type of cryptography appeared with proposals such as Goldreich-Goldwasser-Halevi (GGH) [9], which represents the plain-text as a small “error” that is added to a vector of a lattice. The public key corresponds to a “bad basis” of the lattice, with which solving the Closest Vector Problem (CVP) is hard, and by opposition the private key is a “good basis”, enabling the computation of the closest lattice vector for small errors. However, in their first form, GGH encryption schemes were severely broken [10] and thwarting these attacks is still a current concern [11], [12].

The implementation of public-key cryptography requires processing keys with large sizes to assure security. As an example, RSA public-keys being currently used have up to 4096 bits, while Elliptic Curve Cryptography (ECC) may vary from smaller keys up to 448-bit keys, and thousands of bits are used for the not yet standardized LBC. Given that the operand bit-length is significantly larger than the word-length of the processors, multi-precision arithmetic is required.

Several multiple precision arithmetic libraries have been developed [13], [14] employing generic arithmetic algorithms. In contrast, RNS naturally maps the large operands into smaller integers that match the width of the data-paths, and provides data parallelism in order to efficiently compute multi-precision arithmetic. Therefore, RNS has been a strong candidate to support the design of efficient public-key cryptosystems based on the RSA and ECC, not only for ASICs [15] but also for general-purpose multi-cores [16] and for accelerators, such as Graphical Processing Units (GPUs) [17]. Recently, GGH has got many improvements [18], [19], one of which exploiting RNS [19], to make it competitive under secure parameters. In a parallel effort to improve GGH’s performance, several implementations using RNS were proposed, namely for Central Processing Units (CPUs) [20], Field Programmable Gate Arrays (FPGAs) [21], and GPUs and multi-core CPUs with SIMD extensions [22].

Cryptographic algorithms’ implementations have been often targeted by side-channel attacks. Manifestations of processing secret data, such as power and timing can be exploited to predict the private-keys being used. By exploiting the RNS parallel nature, and randomizing the underlying moduli, the level of unpredictability increases to the point where it is no longer possible to retrieve the private-key using the most sophisticated attacks [23]. Further improvements can be obtained by combining RNS with the use of the Montgomery Ladder and randomizing RAM addresses [24].

The main goal of this paper is to provide the mathematical tools, present the algorithms and discuss the architectures and circuits to design efficient and secure public-key cryptosystems supported on RNS. We follow a tutorial style on how the use of RNS can be generalized to design and implement modular arithmetic units for the RSA, ECC, and LBC on platforms and devices with very different characteristics. There have been recent efforts to give a holistic view of RNS, for example in [25], mainly for the application of RNS to hardware implementations of ECC and RSA. This paper aims at expanding these efforts by providing a more comprehensive view and a more hands-on approach on a broader range of applications of RNS on cryptography. Furthermore, we also replicate and evaluate the experimental results in the literature, which enables a more sensible analysis of the benefits of RNS.

II. Mathematical Background

This section presents the mathematical background that underlies the RNS and modular operations that support public-key cryptography. Commutative rings are the mathematical entity supporting RNS.

Definition 1 (Commutative Ring). A commutative ring R is a set with two operations, addition and multiplication:

- R is an abelian group under addition, which in particular means that there exists an element $0 \in R$ such that for any $a \in R$ the following holds, $0 + a = a + 0 = a$;
- $ab = ba$ for all $a, b \in R$ (commutative property);
- $a(bc) = (ab)c$ for any $a, b, c \in R$ (associative property);
- there is an element $1 \in R$ with $1 \neq 0$ and $a = a \cdot 1 = a$ for any $a \in R$;
- $a(b + c) = ab + ac$ for any $a, b, c \in R$ (distributive property).

One example of a ring \mathbb{Z}_n is the set of integers modulo n with the standard binary addition and multiplication (mod n) operations, with 0 and 1 as the respective identity values. In particular, in this ring two integers a and b are said to be congruent modulo n if their difference $a - b$ is an integer multiple of n :

$$a \equiv b \pmod{n} \quad (1)$$

Example 1 (Operations over \mathbb{Z}_{19}). The operations over \mathbb{Z}_{19} are accomplished like an ordinary integer operation followed by reduction, an addition or subtraction with a multiple of $n = 19$:

- $(7 + 15) \equiv 22 \equiv 22 - 19 \equiv 3 \pmod{19}$;
- $(7 - 15) \equiv -8 \equiv -8 + 19 \equiv 11 \pmod{19}$;
- $(7 \times 15) \equiv 105 \equiv 105 - (5 \times 19) \equiv 10 \pmod{19}$.

The RNS is based on Theorem 1, the Chinese Remainder Theorem (CRT).

Theorem 1 (Chinese Remainder Theorem). Let m_1, \dots, m_h be h pairwise co-prime integers, and $M = \prod_{i=1}^h m_i$. The CRT asserts that the function

$$\begin{aligned} \mathbb{Z}_M &\rightarrow \mathbb{Z}_{m_1} \times \dots \times \mathbb{Z}_{m_h} \\ a &\rightarrow f(a) = (a_1, \dots, a_h) \\ &= (a \pmod{m_1}, \dots, a \pmod{m_h}) \end{aligned}$$

is a ring isomorphism [26].

This powerful result concretely states that inefficient arithmetic over the ring \mathbb{Z}_M can be split into multiple rings \mathbb{Z}_{m_i} , not only improving the efficiency of the computation but also data parallelism.

Example 2 (Forward Conversion to the RNS with base $\{2, 3, 5\}$). Here, we exemplify how operating modulo 30 is isomorphic to operating on arithmetic modulo 2, 3 and 5 simultaneously.

We take $a = 20$, $f(a) = (20 \pmod{2}, 20 \pmod{3}, 20 \pmod{5}) = (0, 2, 0)$. Similarly, $f(25) = (1, 1, 0)$ and $f(15) = (1, 0, 0)$. As one would expect:

- $f(20 + 25 \pmod{30}) = f(20) + f(25) = (0, 2, 0) + (1, 1, 0) = (0 + 1 \pmod{2}, 2 + 1 \pmod{3}, 0 + 0 \pmod{5}) = (1, 0, 0) = f(15)$

$$\begin{aligned} \blacksquare f(20 \cdot 25 \pmod{30}) &= f(20) \cdot f(25) = (0 \cdot 1 \pmod{2}, 2 \cdot 1 \pmod{3}, 0 \cdot 0 \pmod{5}) \\ &= (0, 2, 0) = f(20) \end{aligned}$$

There are two main approaches to reverse the RNS representation: while one is parallel in nature but requires inefficient computations (i), the other is recursive but computationally more efficient (ii).

i) Let $\{m_1, \dots, m_h\}$ be a set of integers for which the CRT is applicable. Given Bézout's identity [27], there are b_j and d_j , with $m_1 \mid b_j$ and $m_j \mid d_j$ (where $x \mid y$ is used to denote that x divides y), such that:

$$b_j + d_j = 1, \forall j \in \{2, \dots, h-1\} \quad (2)$$

and thus:

$$\prod_{j=2}^n (b_j + d_j) = 1 \quad (3)$$

If we develop the expression on the left hand side of (3), it can be observed that all terms are multiples of m_1 , except for $c_1 = \prod_{j=2}^n d_j$, which satisfies $m_2 \mid c_1, \dots, m_h \mid c_1$. Thus from (3), for $j > 1$:

$$c_1 \equiv 1 \pmod{m_1}, \quad c_1 \equiv 0 \pmod{m_j} \quad (4)$$

More generally, one can find c_i for all i , such that, for $j \neq i$:

$$c_i \equiv 1 \pmod{m_i}, \quad c_i \equiv 0 \pmod{m_j} \quad (5)$$

One can now compute the value of $a \in \mathbb{Z}_M$ from the RNS representation $(a_1, \dots, a_n) \in \mathbb{Z}_{m_1} \times \dots \times \mathbb{Z}_{m_n}$ as:

$$a = (a_1 c_1 + \dots + a_n c_n) \pmod{M} \quad (6)$$

ii) For the other conversion approach, we start by considering a two moduli set RNS, $\{m_1, m_2\}$. From (2), there are $m_1 \mid b_2$ and $m_2 \mid d_2$ such that $b_2 + d_2 = 1$. Therefore, to compute the value of a with RNS representation $(a_1, a_2) \in \mathbb{Z}_{m_1} \times \mathbb{Z}_{m_2}$ one might use the following equation:

$$a \equiv a_1 + b_2(a_2 - a_1) \pmod{(m_1 m_2)} \quad (7)$$

A two-step procedure is adopted to convert a representation in the three moduli set $\{m_1, m_2, m_3\}$ to $\mathbb{Z}_{m_1 m_2 m_3}$. We first use the previous procedure to convert the representation to $\{m_1 \times m_2, m_3\}$, and afterwards apply the same procedure to get the value in $\mathbb{Z}_{m_1 m_2 m_3}$. The same rationale can be applied repeatedly to extend the procedure for a generic moduli set size.

Example 3 (Reverse Conversion for the RNS with base $\{2, 3, 5\}$). Herein, we will convert $(1, 1, 0) \in \{\mathbb{Z}_2, \mathbb{Z}_3, \mathbb{Z}_5\}$ to \mathbb{Z}_{30} using both aforementioned conversion methods.

An arithmetic formula for the values c_i of the conversion method (i) is $c_i = (M/m_i) \times ((M/m_i)^{-1} \pmod{m_i})$, where

$m_1 = 2, m_2 = 3, m_3 = 5, M = 2 \times 3 \times 5 = 30$ and the expression $x^{-1} \bmod m_i$ refers to the multiplicative inverse of x modulo m_i , i.e. $xx^{-1} \equiv 1 \bmod m_i$. This inverse exists since $\text{GCD}(m_i, m_j) = 1$ for $i \neq j$. For the considered set:

$$\begin{aligned} c_1 &= 3 \times 5 \times ((3 \times 5)^{-1} \bmod 2) = 15 \\ c_2 &= 2 \times 5 \times ((2 \times 5)^{-1} \bmod 3) = 10 \\ c_3 &= 2 \times 3 \times ((2 \times 3)^{-1} \bmod 5) = 6 \end{aligned}$$

Therefore,

$$a \equiv 1 \times 15 + 1 \times 10 + 0 \times 6 \equiv 25 \bmod 30$$

For the conversion method (ii), starting with $(1, 1) \in \mathbb{Z}_2 \times \mathbb{Z}_3$, b_2 can be computed as $m_1 \times (m_1^{-1} \bmod m_2)$, $b_2 = 2 \times (2^{-1} \bmod 3) = 4$. For the first iteration of the algorithm:

$$a^{(1)} \equiv 1 + 4 \times (1 - 1) \equiv 1 \bmod 6$$

On the second iteration, we consider $(1, 0) \in \mathbb{Z}_6 \times \mathbb{Z}_5$. b_2 changes to $b_2 = 6 \times (6^{-1} \bmod 5) = 6$, and we get:

$$a^{(2)} \equiv 1 + 6 \times (0 - 1) \equiv -5 \equiv 25 \bmod 30$$

We are now able to provide an exact and detailed definition for the RNS.

Definition 2 (Residue Number System). An RNS is a representation system for elements of a ring $a \in \mathbb{Z}_M$, where M is the product of h pairwise co-prime integers m_i . We call $\mathcal{B} = \{m_1, \dots, m_h\}$ the basis of the RNS, and $M = \prod_{i=1}^h m_i$ the dynamic range of the RNS. Residues $(a_1, a_2, \dots, a_h) \in \mathbb{Z}_{m_1} \times \dots \times \mathbb{Z}_{m_h}$ in the RNS are produced with respect to a ring element a by computing:

$$f(a) = (a \bmod m_1, \dots, a \bmod m_h)$$

The RNS representation is converted back to the ring \mathbb{Z}_M by applying one of the two reverse conversion methods presented below. The first reverse conversion method, denoted CRT-based, corresponds to the instantiation of (6), with $M_i = (M/m_i)$:

$$a = \left(\sum_{i=1}^n (a_i (M_i)^{-1} \bmod m_i) M_i \right) \bmod M \quad (8)$$

The other conversion method is designated Mixed-Radix Conversion (MRC), and corresponds to the recursive instantiation of (7):

$$\begin{aligned} \hat{a}_1 &= a_1 \bmod m_1 \\ \hat{a}_2 &= (a_2 - \hat{a}_1) m_1^{-1} \bmod m_2 \\ \hat{a}_3 &= ((a_3 - \hat{a}_1) m_1^{-1} - \hat{a}_2) m_2^{-1} \bmod m_3 \\ &\vdots \\ \hat{a}_n &= (((a_n - \hat{a}_1) m_1^{-1} - \hat{a}_2) m_2^{-1} \dots) m_{n-1}^{-1} \bmod m_n \\ a &= \hat{a}_1 + m_1(\hat{a}_2 + m_2(\hat{a}_3 + \dots + m_{n-1}\hat{a}_n)) \end{aligned} \quad (9)$$

As a side note, the values of \hat{a}_i are themselves digits of another radix representation system called Mixed-Radix System (MRS).

Example 4 (Image processing with RNS). In Figure 2, one can find an example of image processing, where an edge detection kernel is convoluted with a picture of a Rubik's cube. In the first row, pictures are represented using a matrix of integers ranging from 0 to 209, where 0 represents black pixels and 209 represents white pixels, while pixels in between are gray. Arithmetic is performed modulo 210, and the kernel

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 209 & 0 \\ 209 & 4 & 209 \\ 0 & 209 & 0 \end{bmatrix} \bmod 210 \quad (10)$$

is convoluted with the image. If a pixel shares a tone with its neighbors, the pixel that will result from the convolution at those coordinates will be black. In contrast, one will get a shade of gray or white if this does not happen.

In the bottom three rows, one can find a graphic representation of the same computation but using the RNS. Moduli $\{5, 6, 7\}$ were chosen since $5 \times 6 \times 7 = 210$. Pictures in these rows are represented by matrices with integers in the range $\{0, \dots, m_i - 1\}$, where 0 corresponds to black, whereas $m_i - 1$ corresponds to white, with $m_i \in \{5, 6, 7\}$. The pictures on the left side of the figure correspond to the m_i residues of original matrix. Afterwards, they are convoluted with the residues modulo m_i of the kernel in (10). Finally, one can apply an RNS reverse conversion to the convoluted pictures, to obtain the convoluted picture modulo 210.

We conclude this Section by noting that finite fields are used for some cryptographic constructs.

Definition 3 (Finite Field). A finite field $\text{GF}(q)^{\dagger}$ is a ring whose nonzero elements form an abelian group under multiplication (i.e. all nonzero elements in the ring have multiplicative inverses).

Herein, we will focus on finite fields of the form $\text{GF}(p)$ with p prime, which are composed of all integers $\{0, 1, \dots, p - 1\}$ along with addition, multiplication and the corresponding inverses and identities. Additions and multiplications are computed using modular arithmetic. The multiplicative inverse of $x \in \text{GF}(p)$ corresponds to the value x^{-1} such that $x \times x^{-1} \equiv 1 \bmod p$, and can be computed using the Extended Euclidean Algorithm (EEA) or Euler's theorem [28].

^{\dagger} $\text{GF}()$, the acronym of Galois Field, is an alternative designation for a finite field that is named after the mathematician Évariste Galois.

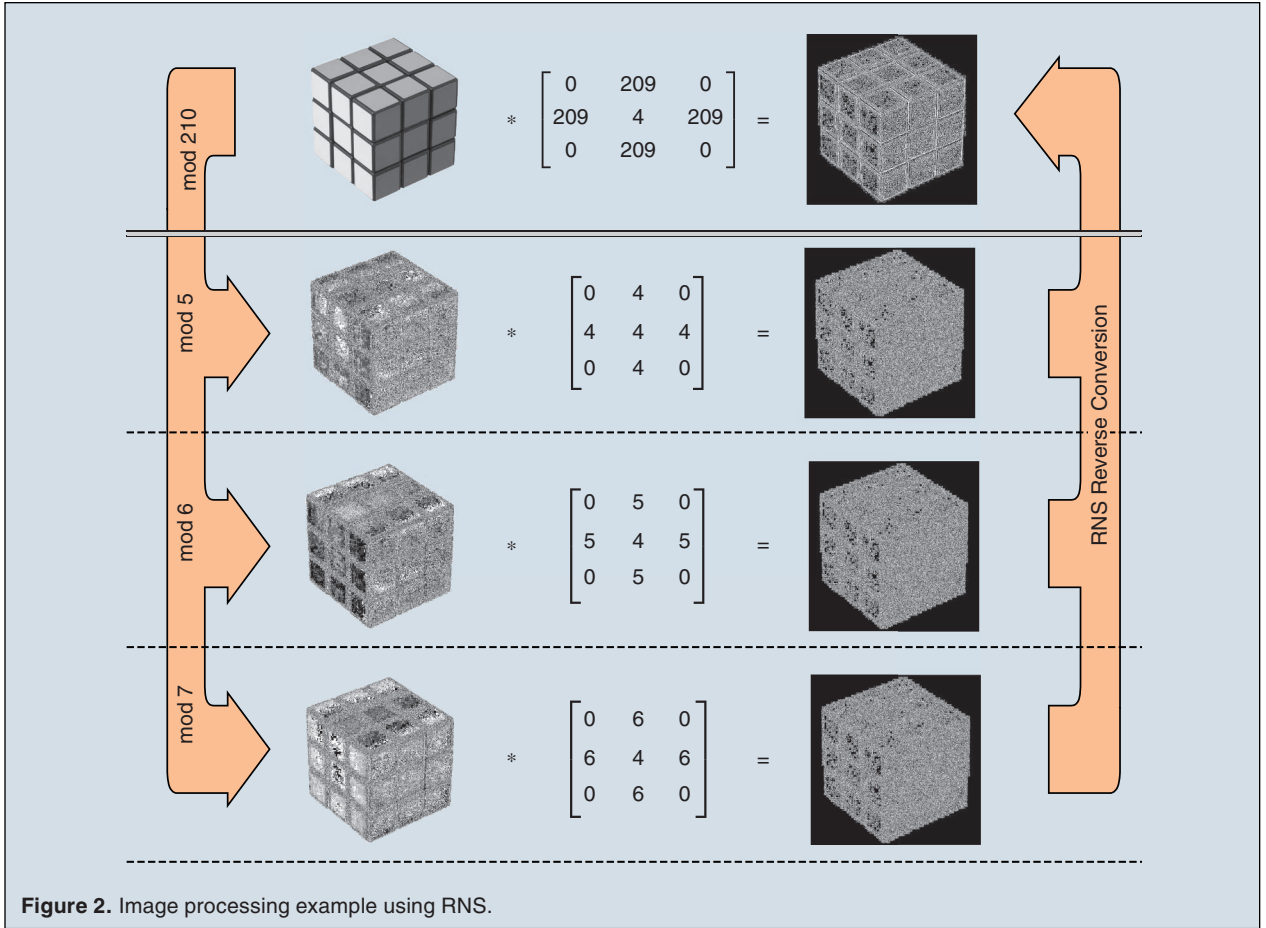


Figure 2. Image processing example using RNS.

III. The RSA and the Essential of RNS Arithmetic Applied To Cryptography

In this section we will show how RNS can be used to perform modular arithmetic and implement public-key cryptographic algorithms. The mathematical tools and techniques for RNS presented in this section have a wide range of applicability. They can be applied not only to the RSA cryptosystem but also to ECC and LBCs, as it will be discussed in the next sections.

A. The RSA Cryptographic Algorithm

In the RSA algorithm [5], a user A (Alice in Figure 1) sets his public key $K_A = (N_A, e_A)$ and private key $k_A = (N_A, d_A)$, where N_A , e_A and d_A are integers. With K_A the user can compute a one-way trapdoor permutation of an integer m in the range $[0, N_A - 1]$ as:

$$c = f_A(m) = m^{e_A} \bmod N_A. \quad (11)$$

Definition 4 (One-way trapdoor permutation). A one-way trapdoor permutation corresponds to a function that can be efficiently computed but can only be inverted with extra information. In the case of (11) that extra information is the private key of the user A .

Only the user A knows his private key k_A , thus he is the only one capable of inverting the permutation in (11) and obtain m from c as:

$$m = f_A^{-1}(c) = y^{d_A} \bmod N_A. \quad (12)$$

In order for the permutation to be secure, the keys must comply with some constraints. To compute his keys, A has to choose two prime numbers p_A and q_A , computing $N_A = p_A q_A$, according to the methods defined in the RSA-based digital signature standards [29].

Example 5 (RSA operations). Herein, we exemplify the encryption and decryption operations of RSA, choosing as the secret prime numbers $p = 5$ and $q = 7$. In this setting, operations are performed modulo $N = 35$. Furthermore, we have selected as the public exponent the value $e = 5$. In order to encrypt $m = 2$, the exponentiation operation $c = m^e \bmod N = 2^5 \bmod 35$ is performed as $c = 2^5 \bmod 35 = 32$. Since we know the factorization of N , we can compute the value of $\phi(N) = (p - 1)(q - 1) = 24$. According to Euler's theorem, $2^{\phi(35)} = 2^{24} = 1 \bmod 35$. To decrypt the value of $c = 32$, we choose as private-key the value $d = 5$, since $5 \times 5 = d \times e = 1 + \phi(N) = 25$,

Algorithm 1:
***k*-bit modular exponentiation.**

Require: a a k -bit integer, the base
Require: b a k -bit integer, the exponent
Require: N a k -bit integer that is the product of two primes ($N = p \times q$)
Ensure: $r = a^b \bmod N$

```

1:  $r = a$ ;
2: /* Assume the bit  $b_{(k-1)} = 1$ 
3: for  $i = k-2$ ;  $i \geq 0$ ;  $i--$  do
4:    $r = r \times r \bmod N$ ;
5:   /* if the  $i^{\text{th}}$  bit of  $b$  is different from 0 */
6:   if  $b_i \neq 0$  then
7:      $r = r \times a \bmod N$ ;
8:   end if
9: end for
10: return  $r$ .
```

and therefore we will be able to get the message back: $m = (2^5)^5 = 2^{25} = 2^{24} \times 2 = 1 \times 2 = 2 \bmod 35$.

The main operation to be computed in the RSA algorithm is the modular exponentiation (11) and (12), which can be accomplished by a square-and-multiply algorithm as presented in Algorithm 1. Since N is the product of two random primes, it cannot benefit from the improved arithmetic efficiency of, e.g., powers of two. Therefore, other alternative algorithms have to be used to optimize the modular exponentiation. Herein, we show how to apply the RNS to the Montgomery modular multiplication so as to improve the performance of modular multiplication and squaring.

B. RNS Reduction

Typically, after each RSA multiplication or squaring, a modular reduction maps the result back to the set of representatives $\{0, \dots, N-1\}$. The reduction modulo N is herein discussed as an individual operation and also as part of a modular multiplication algorithm known as Montgomery modular multiplication [30]. The

Montgomery algorithm replaces the costly reduction $\bmod N$ by a reduction $\bmod L$ easier to compute. In the original proposal [30], the target was a binary system, hence a good choice for L was a power of 2. However, in RNS the reduction modulo a power of 2 is inefficient. Therefore an RNS variation of the Montgomery modular multiplication that relies in a Basis Extension (BE) method [31] has been proposed to address this problem. This method extends the representation of an RNS basis to another one during the computation. These two bases are herein referred to as $\mathfrak{B}_1 = \{m_{1,1}, \dots, m_{h_1,1}\}$ and $\mathfrak{B}_2 = \{m_{1,2}, \dots, m_{h_2,2}\}$ with dynamic ranges M_1 and M_2 , respectively. Reduction modulo the dynamic range M_1 is an easy task since it relies on the modulo operation in each one of the RNS channels of the basis \mathfrak{B}_1 :

$$Y = X \bmod M_1 \Leftrightarrow y_{e,1} = x_{e,1} \bmod m_{e,1} \\ 1 < e < h_1, M_1 = \prod_{e=1}^{h_1} m_{e,1} \quad (13)$$

This property is exploited in the BE method by replacing the arithmetic modulo N by arithmetic modulo $L = M_1$. This replacement requires the use of precomputed constants that require $|N^{-1}|_{M_1}$ to exist, which is guaranteed if $\text{GCD}(M_1, N) = 1$. Moreover, $|M_1^{-1}|_{M_2}$ must also exist which requires $\text{GCD}(M_1, M_2) = 1$.

The input data to be reduced is represented in the Montgomery Domain (MD) by ${}_MA = A(M_1 \bmod N) \bmod N$, as depicted in Figure 3. For the addition and subtraction of two elements in the MD there is a direct correspondence with the Original Domain (OD) since ${}_MA + {}_MB \equiv (A + B)(M_1 \bmod N) \bmod N$. For the multiplication of two elements in the MD, the result is given by ${}_MR = {}_MA {}_MB \equiv (AB)(M_1^2 \bmod N)$, which requires a reduction ${}_MU = {}_MR(M_1^{-1} \bmod N) \bmod N$. To keep a unified reduction method in RNS, each time a reduction has to be performed after an addition or subtraction the value of the input should be multiplied by $M_1 \bmod N$. Hence, for a value R to be reduced in the OD, the input of the reduction in the MD has the form $R(M_1^2 \bmod N)$. The reduction result ${}_MU$ is obtained by computing:

$${}_MU = ({}_MR + QN)/M_1 \quad (14)$$

where Q is an integer such that M_1 divides $({}_MR + QN)$. (14) corresponds to ${}_MU \equiv {}_RM_1 \bmod N$ as desired, since $QN \equiv 0 \bmod N$. A condition for $({}_MR + QN)$ to be a multiple of M_1 is $({}_MR + QN) \equiv 0 \bmod M_1$, which corresponds to computing ${}_Mr_{e,1} + q_{e,1}n_{e,1} \equiv 0 \bmod m_{e,1}$ for all the RNS channels $m_{e,1}$ of \mathfrak{B}_1 . With the aforementioned property, Q is efficiently computed in the channels as:

$$q_{e,1} = -{}_Mr_{e,1}n_{e,1}^{-1} \bmod m_{e,1}, \forall m_{e,1} \in \mathfrak{B}_1$$

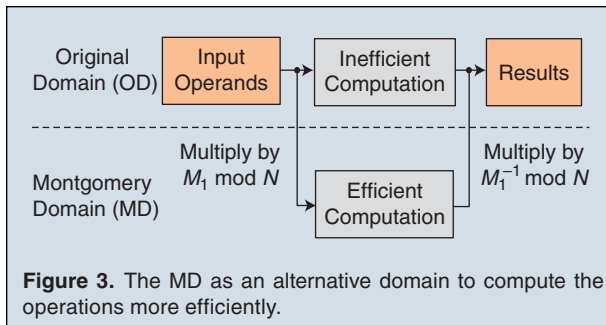


Figure 3. The MD as an alternative domain to compute the operations more efficiently.

Although the resulting value ${}_M U$ is supposed to be reduced modulo N , for ${}_M R = {}_M A {}_M B$, with ${}_M A, {}_M B < N$ and $M_1 > N$, the following holds:

$${}_M U = ({}_M A {}_M B + QN)/M_1 < (N^2 + M_1 N)/M_1 < 2N \quad (15)$$

which means that the identity ${}_M U \bmod N = {}_M U - N$ may hold. Nevertheless, the utilization of the values ${}_M U < 2N$ enables the computation of bounded and correct results modulo N as long as $M_1 > 4N$, which can be verified by considering ${}_M A, {}_M B < 2N$ and the inequality:

$$\begin{aligned} {}_M U &= ({}_M A {}_M B + QN)/M_1 < ((2N)^2 + M_1 N)/M_1 \\ &< (M_1 N + M_1 N)/M_1 = 2N \end{aligned} \quad (16)$$

This method can not be directly applied to \mathfrak{B}_1 since M_1 cannot be represented in \mathfrak{B}_1 . Therefore, an extra basis \mathfrak{B}_2 is also used to compute (14), with dynamic range M_2 higher than M_1 , with $\text{GCD}(M_2, M_1) = 1$. Data to be reduced (${}_M R$) has also to be represented in \mathfrak{B}_2 and the value of Q to be converted from \mathfrak{B}_1 to \mathfrak{B}_2 . This conversion can be performed with any of the methods described in Definition 2 for the RNS reverse conversion, with the result computed modulo $m_{e,2}$ for the channels of \mathfrak{B}_2 . After computing ${}_M U$ in \mathfrak{B}_2 , it has to be converted to \mathfrak{B}_1 and the reduction algorithm is complete.

Algorithm 2 presents in detail the steps of the described reduction algorithm for each RNS channel $m_{e,i}$, and Figure 4 summarizes the correspondence between the steps in Algorithm 2 and the two bases. In Algorithm 2, two conversions are performed using (8): from \mathfrak{B}_1 to \mathfrak{B}_2 and \mathfrak{B}_2 to \mathfrak{B}_1 , by using α_1 and α_2 as the correction factors to subtract the appropriate multiple of M_1 and M_2 , respectively, so that the result is bounded. Shenoy and Kumaresan [32] proposed a technique supported on a redundant modulus to compute the value of α_2 . This method was later superseded by Kawamura et al., who proposed a technique supported on fixed-point arithmetic [33].

The Kawamura et al. [33] method observes that $X < M_i \Leftrightarrow (X/M_i) < 1$ (where M_i corresponds to the dynamic range of the base B_i), and hence (8) can be rewritten as:

$$\sum_{e=1}^{h_i} \frac{\xi_{e,i}}{m_{e,i}} = \alpha + \frac{X}{M_{e,i}} \Rightarrow \alpha = \left\lfloor \sum_{e=1}^{h_i} \frac{\xi_{e,i}}{m_{e,i}} \right\rfloor \quad (17)$$

Since (17) requires costly divisions by $m_{e,i}$, an approximation ($\hat{\alpha}$) is suggested to this expression:

$$\begin{aligned} \hat{\alpha} &= \left\lfloor \sum_{e=1}^{h_i} \frac{\text{truncH}_t(\xi_{e,i})}{2^k} + \beta \right\rfloor \\ &= \left\lfloor \sum_{e=1}^{h_i} \frac{\text{truncH}_t(\xi_{e,i})}{2^{k-t}} + \Phi \right\rfloor / 2^t \end{aligned} \quad (18)$$

where $\text{truncH}_t(\xi_{e,i})$ sets the $k-t$ least significant bits of $\xi_{e,i}$ to zero, with $t < k$, and $\Phi = \lfloor 2^t \beta \rfloor$. The corrective parameter β (and consequently Φ) should be carefully chosen such that $\alpha = \hat{\alpha}$. A set of inequalities that support the selection of appropriate values for β were stated in [33], based on the maximum approximation errors due to the replacement of the denominator and numerator in (17) by the ones in (18). These inequalities are expressed by Theorems 2 and 3 that establish the computation of $\hat{\alpha}$, its relationship with β , the number of channels h_i , and the errors:

$$\begin{aligned} \epsilon &= \max_e \left(\frac{2^k - m_{e,i}}{2^k} \right) \\ \delta &= \max_e \left(\frac{\xi_{e,i} - \text{truncH}_t(\xi_{e,i})}{m_{e,i}} \right) \end{aligned} \quad (19)$$

Theorem 3 refers to the computation of a non-exact, although bounded, value of $\hat{\alpha}$ that enables a non-exact RNS to binary conversion, while Theorem 2 refers to an exact conversion.

Theorem 2. If $0 \leq h_i(\epsilon + \delta) \leq \beta < 1$ and $0 \leq X < (1 - \beta)M_i$, then $\alpha = \hat{\alpha}$;

Theorem 3. If $\beta = 0$, $0 \leq h_i(\epsilon + \delta) \leq 1$ and $0 \leq X \leq M_i$, then $\alpha = \hat{\alpha}$ or $\alpha = \hat{\alpha} + 1$.

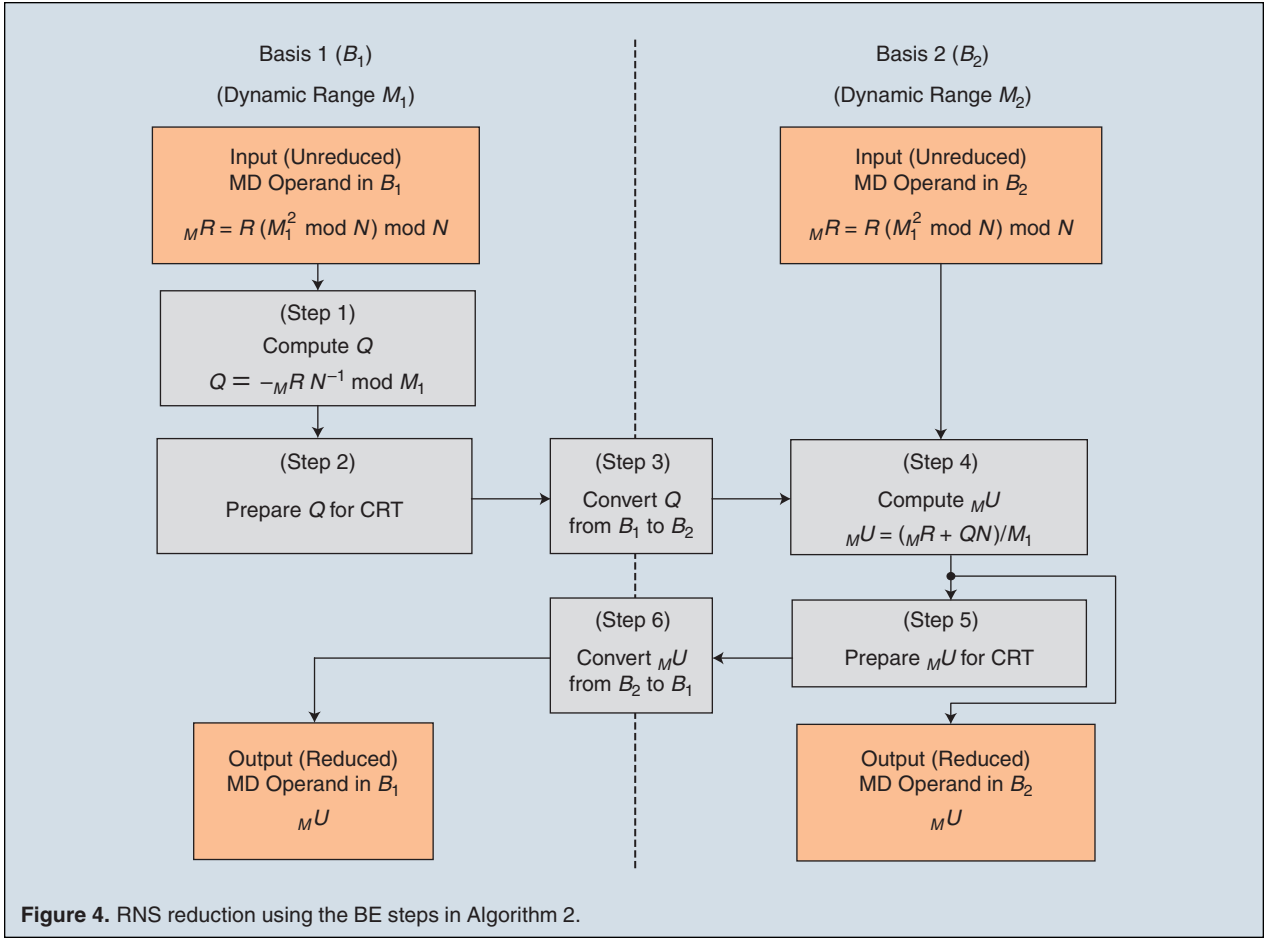
Given the range of Q , the conditions of Theorem 2 cannot be verified for step 3 of Figure 4, because of the value $\beta = 0$. Thus, α_1 should be obtained from (18) in the conditions of Theorem 3, with the associated error that can impose an extra term M_1 in the final conversion result. Therefore, for the computation of α_1 the value Φ in (18) should be set to zero and the minimum value of t (the number of bits used in the accumulation) should be used such that $0 \leq \Delta = h_i(\epsilon + \delta) \leq 1$ (see Theorem 3). For ${}_M U$, it is possible to obtain α_2 under the conditions of Theorem 2, i.e., a value β such that $0 \leq {}_M U < (1 - \beta)M_2$

Algorithm 2: RNS reduction.

Require: ${}_M R_{e,1}, {}_M R_{e,2}$

Ensure: ${}_M U_{e,1} = ({}_M R_{e,1} \bmod N)_{e,1}$ and ${}_M U_{e,2} = ({}_M R_{e,1} \bmod N)_{e,2}$

- 1: In ch. $e, 1$: $q_{e,1} = \lfloor -{}_M R_{e,1} n_{e,1}^{-1} \rfloor_{m_{e,1}}$
- 2: In ch. $e, 1$: $\xi_{e,1} = \lfloor q_{e,1} \rfloor_{m_{e,1}} \lfloor M_{e,1}^{-1} \rfloor_{m_{e,1}}$
- 3: In ch. $e, 2$: $q_{e,2} = \lfloor \sum_{j=1}^{h_1} \xi_{j,1} \rfloor_{m_{e,2}} - \alpha_1 M_1 \rfloor_{m_{e,2}}$
- 4: In ch. $e, 2$: ${}_M U_{e,2} = \lfloor ({}_M R_{e,2} + q_{e,2} n_{e,2}) M_1^{-1} \rfloor_{m_{e,2}}$
- 5: In ch. $e, 2$: $\xi_{e,2} = \lfloor {}_M U_{e,2} \rfloor_{m_{e,2}} \lfloor M_{e,2}^{-1} \rfloor_{m_{e,2}}$
- 6: In ch. $e, 1$: ${}_M U_{e,1} = \lfloor \sum_{j=1}^{h_2} \xi_{j,2} \rfloor_{m_{e,1}} \lfloor M_{j,2} \rfloor_{m_{e,1}} - \alpha_2 M_2 \rfloor_{m_{e,1}}$
- 7: **return** ${}_M U_{e,1}, {}_M U_{e,2}$.



exists as long as the difference between the dynamic range M_2 and the value to be converted is large enough. Therefore, the minimum value of t such that $\max(MU) < (1 - \Delta)M_2$ should be computed and α_2 obtained from (18) using $\Phi = \lceil 2^t \Delta \rceil$.

Algorithm 2 can be further optimized by merging constants so as to reduce the number of precomputed constants and temporary results. Algorithm 3 is an

optimized version of Algorithm 2 that comprises the merging of the constants as:

$$\begin{aligned} \lambda_{1e} &= \lceil -n_{e,1}^{-1} M_{e,1}^{-1} \rceil_{m_{e,1}} & \lambda_{2ej} &= \lceil -M_{j,1} M_1^{-1} \rceil_{m_{e,2}} \\ \lambda_{3e} &= \lceil -M_1 n_{e,2} \rceil_{m_{e,2}} & \lambda_{4e} &= \lceil M_1^{-1} \rceil_{m_{e,2}} & \lambda_{5e} &= \lceil M_{e,2}^{-1} \rceil_{m_{e,2}} \\ \lambda_{6ej} &= \lceil -M_{j,2} M_2^{-1} \rceil_{m_{e,1}} & \lambda_{7e} &= \lceil -M_2 \rceil_{m_{e,1}} \end{aligned}$$

Note that the methodology for computing the values α_i in Algorithm 3 is similar to the one adopted in Algorithm 2, with the difference that the values $q_{e,1}$ in steps 1 and 4 of Algorithm 3 are used instead of the values ξ in steps 2 and 5 of Algorithm 2.

The optimization of this operation has a major impact in the overall performance of an RNS implementation. An optimized implementation of the BE method is presented in Algorithm 3. All the steps in this algorithm correspond to operations that are accomplished in parallel for each RNS basis \mathfrak{B}_1 and \mathfrak{B}_2 . Only the computation of the values α_i requires the serialization of the computation given that each of the channels computing steps 2 and 5 of Algorithm 3 require the values α_1 and α_2 , respectively, to proceed with the computation.

Algorithm 3: Optimized RNS reduction.

Require: $M r_{e,1}, M r_{e,2}$.

Ensure: $M u_{e,1} = (R M_1 \bmod N)_{e,1}$ and $M u_{e,2} = (R M_1 \bmod N)_{e,2}$.

- 1: In ch. $e, 1$: $q_{e,1} = \lceil M r_{e,1} \lambda_{e,1} \rceil_{m_{e,1}}$
- 2: In ch. $e, 2$: $q_{e,2} = \lceil (\sum_{j=1}^{h_1} q_{j,1} \lambda_{2ej} + \alpha_1) \lambda_{3e} \rceil_{m_{e,2}}$
- 3: In ch. $e, 2$: $M u_{e,2} = \lceil (M r_{e,2} + q_{e,2}) \lambda_{4e} \rceil_{m_{e,2}}$
- 4: In ch. $e, 2$: $q_{e,1} = \lceil M u_{e,2} \lambda_{5e} \rceil_{m_{e,2}}$
- 5: In ch. $e, 1$: $M u_{e,1} = \lceil (\sum_{j=1}^{h_2} q_{j,1} \lambda_{6ej} + \alpha_2) \lambda_{7e} \rceil_{m_{e,1}}$
- 6: **return** $M u_{e,1}, M u_{e,2}$.

Addressing this problem, Bajard et al. [31] proposed to fix the value of α_1 to $\alpha_1 = 0$ at the expense of a higher dynamic range M_1 and a larger multiple of N offsetting the resulting values ${}_M U$. Without the computation of α_1 and knowing that $\alpha_1 < h_1$, an extra offset $(h_1 - 1)M_1$ needs to be accommodated during the computation, after the conversion from \mathfrak{B}_1 to \mathfrak{B}_2 , of the value Q . Hence, the upper bounds obtained in (15) and (16) change. Considering ${}_M A, {}_M B < N$ and $\alpha_1 = 0$, the following holds:

$$\begin{aligned} {}_M U &= ({}_M A {}_M B + (Q + \alpha_1 M_1)N) / M_1 \\ &= ({}_M A {}_M B + QN) / M_1 + \alpha_1 N \\ &< (M_1 N + M_1 N) / M_1 + h_1 N \\ &= (h_1 + 2)N. \end{aligned} \quad (20)$$

In order to keep the value ${}_M U$ always bounded including when ${}_M A, {}_M B < (h_1 + 2)N$, the value M_1 has to comply with $M_1 > (h_1 + 2)^2 N$ [31].

Example 6 (Basis extension). *This example addresses the computation of a modular operation using the BE method to perform the modular reduction. The conversion of an operand $A = 123$ to the MD associated with the modulus $N = 167$ is accomplished by computing ${}_M U = A(M_1 \bmod N) \bmod N$, a multiplication and a reduction modulo N . In order to use the BE method the following operation will be computed instead:*

$${}_M U = (AB + QN) / M_1, \quad B = (M_1^2 \bmod N)$$

In this example, the basis $\mathfrak{B}_1 = \{31, 27, 23\}$ and $\mathfrak{B}_2 = \{32, 29, 25\}$ are used with dynamic ranges $M_1 = 19251$ and $M_2 = 23200$. We will be using $\alpha_1 = 0$ for the first conversion and α_2 will be computed using Kawamura's method. One can verify that $M_1 > (h_1 + 2)^2 N$, which bounds the output to ${}_M U < (h_1 + 2)M_1$, and also that with $\beta = 0.95$, $t = 0$ and $k = 5$, given that $\epsilon = 0.28125$, $0 \leq h_i(\epsilon + \delta) \leq \beta < 1$ and $0 \leq X < (h_1 + 2)N < (1 - \beta)M_2$. Therefore, Theorem 2 is applicable and the second conversion is exact. With the bases set we obtain the residues of the operands of the multiplication:

$$\begin{aligned} a_{1,1} &= 30; a_{2,1} = 15; a_{3,1} = 8; a_{1,2} = 27; a_{2,2} = 7; a_{3,2} = 23; \\ b_{1,1} &= 19; b_{2,1} = 4; b_{3,1} = 20; b_{1,2} = 16; b_{2,2} = 25; b_{3,2} = 12; \end{aligned}$$

and obtain the residues of ${}_M R = AB$ to be reduced:

$$\begin{aligned} {}_M r_{1,1} &= 12; {}_M r_{2,1} = 6; {}_M r_{3,1} = 22; \\ {}_M r_{1,2} &= 16; {}_M r_{2,2} = 1; {}_M r_{3,2} = 1; \end{aligned}$$

Before proceeding to the BE computation with Algorithm 3 the precomputation of the constants λ should be accomplished:

m_e	1	2	3
λ_{1e}	18	26	20
λ_{2e1}	1	14	4
λ_{2e2}	13	15	12

m_e	1	2	3
λ_{2e3}	25	5	13
λ_{3e}	27	23	8
λ_{4e}	27	23	1
λ_{5e}	29	12	17
λ_{6e1}	30	16	5
λ_{6e2}	16	13	19
λ_{6e3}	26	14	11
λ_{7e}	19	20	7

The first step in the computation of Algorithm 3 is the calculus of $q_{e,1} = \lfloor {}_M r_{e,1} \lambda_{e,1} \rfloor_{m_e,1}$:

$$\begin{aligned} q_{1,1} &= 12 \times 18 \bmod 31 = 30; \\ q_{2,1} &= 6 \times 26 \bmod 27 = 21; \\ q_{3,1} &= 22 \times 20 \bmod 23 = 3. \end{aligned}$$

We now set $\alpha_1 = 0$, and convert Q to the second basis, in step 2 of Algorithm 3:

$$\begin{aligned} q_{1,2} &= (30 \times 1 + 21 \times 13 + 3 \times 25) \times 27 \bmod 32 = 30; \\ q_{2,2} &= (30 \times 14 + 21 \times 15 + 3 \times 5) \times 23 \bmod 29 = 24; \\ q_{3,2} &= (30 \times 4 + 21 \times 12 + 3 \times 13) \times 8 \bmod 25 = 13. \end{aligned}$$

Step 3 of Algorithm 3 can be accomplished as:

$$\begin{aligned} {}_M u_{1,2} &= (16 + 30) \times 27 \bmod 32 = 26; \\ {}_M u_{2,2} &= (1 + 24) \times 23 \bmod 29 = 24; \\ {}_M u_{3,2} &= (1 + 13) \times 1 \bmod 25 = 14; \end{aligned}$$

These residues correspond to $147 + 167 = 314$, which is congruent with AM_1 modulo $N = 167$. The value ${}_M U$ can now be converted to the basis \mathfrak{B}_1 by computing the remaining steps of Algorithm 3. In step 4 one can obtain $q_{e,1}$ as:

$$\begin{aligned} q_{1,1} &= 26 \times 29 \bmod 32 = 18; \\ q_{2,1} &= 24 \times 12 \bmod 29 = 27; \\ q_{3,1} &= 14 \times 17 \bmod 25 = 13; \end{aligned}$$

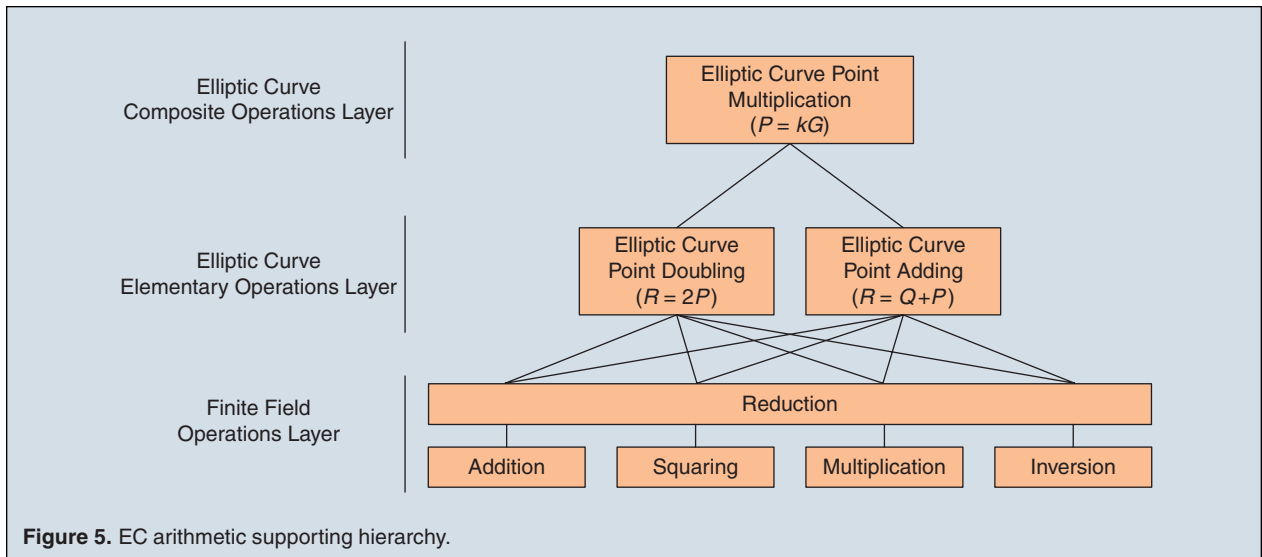
enabling the computation of α_2 with Kawamura's method:

$$\alpha_2 = \left\lfloor \frac{18 + 27 + 13}{32} + 0.95 \right\rfloor = 2$$

The residues of ${}_M U$ in basis B_1 are obtained as:

$$\begin{aligned} {}_M u_{1,2} &= (18 \times 30 + 27 \times 16 + 13 \times 26 + 2) \times 19 \bmod 31 = 4; \\ {}_M u_{2,2} &= (18 \times 16 + 27 \times 13 + 13 \times 14 + 2) \times 20 \bmod 27 = 17; \\ {}_M u_{3,2} &= (18 \times 5 + 27 \times 19 + 13 \times 11 + 2) \times 7 \bmod 23 = 15. \end{aligned}$$

It can be verified that these residues correspond to the value ${}_M U = 314$, the same obtained for the basis B_2 .



Algorithm 1, which comprises the core operation of RSA, can be adjusted to exploit the RNS, by converting the input operands into the Montgomery domain, and replacing modular multiplications by RNS Montgomery multiplications. We close this section by noting that even though the RNS reduction method was motivated herein by the RSA algorithm, we will see it applied several times to other cryptosystems in the following sections.

IV. RNS-Based Elliptic Curve Cryptography

Most public-key cryptography is supported on rings and fields, as we saw in the RSA cryptosystems and we will see in this section for ECC. The RNS will be applied so that

additions and multiplications are performed with a large degree of data parallelism, and reductions are computed using the procedures described in the previous section.

The EC cryptography arithmetic relies in a hierarchical scheme that is depicted in Figure 5. The bottom layer of the hierarchy is the underlying finite field that supports addition, multiplication and the respective additive and multiplicative inverses and identities. A reduction operation is also included in this layer to assure that the output of each of the aforementioned operations remains in the finite field. The middle layer of the hierarchy is composed of the EC group basic operations, namely the addition and doubling. For illustrative purposes, in Figure 6, one can find a graphical representation of an EC defined in \mathbb{R} , as well as the point addition operation.

The top layer of Figure 5 is a composite of the elementary operations in the middle layer and is the most important operation in EC cryptography, granting the security of the system. The following subsection introduces the definition of ECs over $GF(p)$ for a prime p , and introduces the formulae for EC operations.

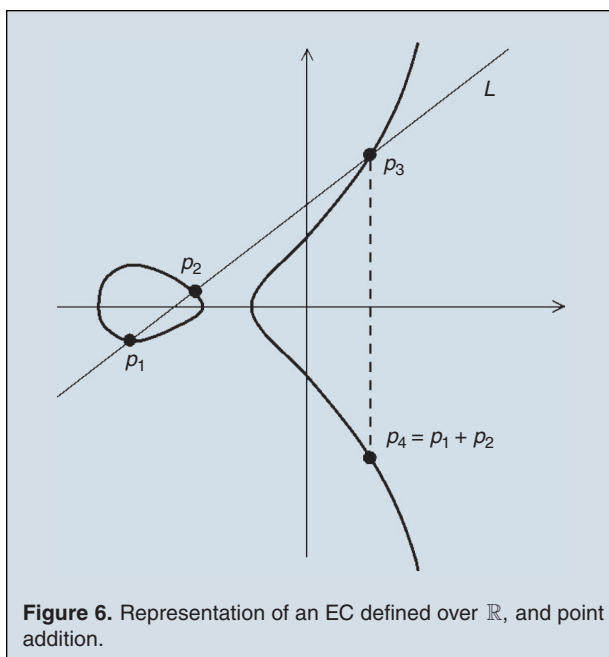
A. Elliptic Curve over $GF(p)$ and EC operations

Without loss of generality, and by a matter of simplicity, ECs are presented as in the standards [29], which are also known as short Weierstrass curves.

Definition 5 (Elliptic Curve over $GF(p)$). A short Weierstrass EC over $GF(p)$, referred as $E(a,b,GF(p))$, consists of the set of two-coordinated points $P_i = (x_i, y_i) \in GF(p) \times GF(p)$ complying with:

$$y_i^2 = x_i^3 + ax_i + b, \quad a, b \in GF(p), \quad (21)$$

together with the condition $4a^3 + 27b^2 \neq 0$.



An additive group with identity O is defined with this curve according to Definition 6.

Definition 6 (Elliptic Curve addition over $GF(p)$). Consider two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ in $E(a, b, GF(p))$ with p prime. The addition in $E(a, b, GF(p))$ is commutative and consists of the following field operations modulo p [34]:

- For $P_1 \neq O, P_2 \neq O$, and $P_1 \neq P_2$, the point $P_3 = (x_3, y_3) = P_1 + P_2$ (addition) is obtained as:

$$x_3 = \lambda^2 - x_2 - x_1; y_3 = -y_2 + \lambda(x_2 - x_3); \quad (22)$$

$$\lambda = \frac{y_1 - y_2}{x_1 - x_2}. \quad (23)$$

- For $P_1 \neq O$, the point $P_3 = (x_3, y_3) = P_1 + P_1 = 2P_1$ (doubling) is obtained as:

$$x_3 = \lambda^2 - 2x_1; y_3 = \lambda(x_1 - x_3) - y_1; \quad (24)$$

$$\lambda = \frac{3x_1^2 + a}{2y_1}. \quad (25)$$

- The point $P_3 = (x_3, y_3) = -P_1 = O - P_1$ (additive inverse) is obtained as:

$$x_3 = x_1; y_3 = -y_1. \quad (26)$$

- For $P_1 = O$, the point $P_3 = P_1 + P_2 = P_2$ and $P_3 = 2P_1 = O$.

ECC uses an operation designated EC point multiplication by a scalar, or simply EC point multiplication. For a scalar integer s and a point P_1 in $E(a, b, GF(p))$, the EC point multiplication P_3 is obtained as:

$$P_3 = sP_1 = \underbrace{P_1 + \dots + P_1}_{s \text{ times}}. \quad (27)$$

This operation grants the security of the EC cryptography given that knowing P_3 and P_1 it is not possible to compute s in polynomial time. Deriving s from P_3 and P_1 is known as the Elliptic Curve Discrete Logarithm Problem (ECDLP). One of the simplest and most used methods to compute the EC point multiplication is the double-and-add method, which is presented in Algorithm 4. With this algorithm, $n - 1$ EC point doublings and $h(s) - 1$ EC point additions are performed to compute the final result P_3 , where $h(s)$ stands for the Hamming weight of s .

B. Elliptic Curve Points with Projective Coordinates

Considering Definition 6, $n + h(s) - 2$ field inversions are required to compute P_3 with Algorithm 4, where $h(s)$ is the Hamming weight of s . Given the high computational demands of the field inversion in comparison with the other field operations, a different representation of the EC point can be adopted using more than two coordinates, namely three. These co-

ordinates are called *Projective Coordinates* while the coordinates originally introduced in Definition 5 are called *Affine Coordinates*.

Definition 7 (EC Projective Coordinates). The projective coordinates of an EC point $P_i \in E(a, b, \mathbb{F})$, where \mathbb{F} is a finite field, refer to the triple $(\widehat{x}_i, \widehat{y}_i, \widehat{z}_i)$ in $\mathbb{F} \times \mathbb{F} \times \mathbb{F}$ except for the triple $(0, 0, 0)$ in the surface of equivalence classes where $(\widehat{x}_1, \widehat{y}_1, \widehat{z}_1)$ is said to be equivalent to $(\widehat{x}_2, \widehat{y}_2, \widehat{z}_2)$ if there exists a $\lambda \in \mathbb{F}$ such that $\lambda \neq 0$ and $(\widehat{x}_1, \widehat{y}_1, \widehat{z}_1) = (\lambda^{\tau_x} \widehat{x}_2, \lambda^{\tau_y} \widehat{y}_2, \lambda \widehat{z}_2)$, where τ_x and τ_y are positive integers. In this surface, an EC point with affine coordinates (x_i, y_i) can be represented in projective coordinates by setting $\lambda \widehat{z}_i = 1 \Leftrightarrow \lambda = 1/\widehat{z}_i$. Therefore, the correspondence between affine and projective coordinates is given by $x_i = \widehat{x}_i/\widehat{z}_i^{\tau_x}$ and $y_i = \widehat{y}_i/\widehat{z}_i^{\tau_y}$.

The main types of projective coordinates used for ECs over $GF(p)$ are the Standard [35], Jacobi [36], and Chudnovsky [37]. The standard projective coordinates avoid inversions in the EC point multiplication and underlie an EC point multiplication algorithm whose execution time does not depend on s . This latter property is important to avoid leaking information on the secret scalar to attackers tracking the execution time [38]. The standard projective coordinates set $\tau_x = \tau_y = 1$ in Definition 7, which results in $x_i = \widehat{x}_i/\widehat{z}_i$ and $y_i = \widehat{y}_i/\widehat{z}_i$. Applying the projective coordinates to (21) results in the following equation to describe the EC:

$$\widehat{y}_i^2 \widehat{z}_i = \widehat{x}_i^3 + a \widehat{x}_i \widehat{z}_i^2 + b \widehat{z}_i^3, a, b \in GF(p). \quad (28)$$

The standard projective representation of the points is particularly interesting for EC point multiplication, namely the Montgomery EC point multiplication method, also known as Montgomery Ladder [39].

Definition 8 (Montgomery EC Point Multiplication). Consider the EC points $P_1 = (\widehat{x}_1, \widehat{y}_1, \widehat{z}_1)$, $P_2 = (\widehat{x}_2, \widehat{y}_2, \widehat{z}_2)$,

Algorithm 4: Double-and-add EC point multiplication.

Require: n -bit scalar $s = (s_{n-1} \dots s_0)$ and $P_1 \in E(a, b, GF(p))$;
Ensure: $P_3 = sP_1$;
 $P_3 = P_1$;
 /*Assume $s_{n-1} = 1$ */
for $i = n - 2$ **down to** 0 **do**
 $P_3 = 2P_3$;
 if $s_i = 1$ **then**
 $P_3 = P_3 + P_1$;
 end if
end for
return P_3 ;

Algorithm 5:
Montgomery ladder for computing the EC point multiplication.

Require: n -bit scalar $s = (s_{n-1} \dots s_0)$ and P_1 in $E(a, b, GF(q))$;
Ensure: $P_3 = sP_1$;
 $\widehat{x}_1 = x_1$; $\widehat{z}_1 = 1$; $\widehat{x}_2 = x_1^A + b$; $\widehat{z}_2 = x_1^2$.
for $i = n - 2$ **down to** 0 **do**
 if $s_i = 1$ **then**
 $(\widehat{x}_1, \widehat{z}_1) = \text{Madd}(\widehat{x}_1, \widehat{z}_1, \widehat{x}_2, \widehat{z}_2)$,
 $(\widehat{x}_2, \widehat{z}_2) = \text{Mdouble}(\widehat{x}_2, \widehat{z}_2)$;
 else
 $(\widehat{x}_2, \widehat{z}_2) = \text{Madd}(\widehat{x}_2, \widehat{z}_2, \widehat{x}_1, \widehat{z}_1)$,
 $(\widehat{x}_1, \widehat{z}_1) = \text{Mdouble}(\widehat{x}_1, \widehat{z}_1)$;
 end if
end for
return $P_3 = \text{Mxy}(\widehat{x}_1, \widehat{z}_1, \widehat{x}_2, \widehat{z}_2)$;

$P_{\text{add}} = (\widehat{x}_{\text{add}}, \widehat{y}_{\text{add}}, \widehat{z}_{\text{add}}) = P_1 + P_2$, and $P_{\text{sub}} = (x_{\text{sub}}, y_{\text{sub}}) = P_1 - P_2$. Then, the following holds:

$$\begin{aligned} \widehat{x}_{\text{add}} &= \begin{cases} -4b\widehat{z}_1\widehat{z}_2(\widehat{x}_1\widehat{z}_2 + \widehat{x}_2\widehat{z}_1) + (\widehat{x}_1\widehat{x}_2 - a\widehat{z}_1\widehat{z}_2)^2, & \text{if } P_1 \neq P_2, \\ (\widehat{x}_1^2 - a\widehat{z}_1^2)^2 - 8b\widehat{x}_1\widehat{z}_1^3, & \text{if } P_1 = P_2. \end{cases} \\ \widehat{z}_{\text{add}} &= \begin{cases} x_{\text{sub}}(\widehat{x}_1\widehat{z}_2 - \widehat{x}_2\widehat{z}_1)^2 & \text{if } P_1 \neq P_2, \\ 4\widehat{z}_1(\widehat{x}_1^3 + a\widehat{x}_1\widehat{z}_1^2 + b\widehat{z}_1^3) & \text{if } P_1 = P_2. \end{cases} \end{aligned} \quad (29)$$

which suggests that it is possible to compute the \hat{x} and \hat{z} coordinates of an addition knowing only the \hat{x} and \hat{z} coordinates of the input points and the x coordinate of the subtraction of the inputs, without requiring neither the y

Table 1.
Operations scheduling for each iteration of the loop in Algorithm 5 (assuming $s_i = 0$).

mult. 1	$A = \widehat{x}_1\widehat{z}_2$; $B = \widehat{x}_2\widehat{z}_1$; $C = \widehat{x}_1\widehat{x}_2$; $D = \widehat{z}_1\widehat{z}_2$; $E = \widehat{x}_1^2$; $F = \widehat{z}_1^2$; $H = b\widehat{z}_1$
reduce A, B, C, D, E, F, H	
add. 1	$\widehat{z}_2 = A - B$; $\widehat{x}_2 = A + B$; $C = C + 3D$; $A = E + 3F$
mult. 2	$D = D\widehat{x}_2$; $\widehat{x}_2 = C^2$; $\widehat{z}_2 = \widehat{z}_2^2$; $A = A^2$; $B = FH$; $E = E\widehat{x}_1$; $F = \widehat{x}_1F$
reduce $D, \widehat{x}_2, \widehat{z}_2, A, B, E, F$	
add. 2	$G = E + B - 3F$
mult. 3	$D = bD$; $\widehat{z}_2 = x_1\widehat{z}_2$; $B = \widehat{x}_1B$; $F = \widehat{z}_1G$
reduce D, \widehat{z}_2, B, F	
add. 3	$\widehat{x}_2 = \widehat{x}_2 - 4D$; $\widehat{x}_1 = A - 8B$; $\widehat{z}_1 = 4F$

nor the \hat{y} coordinates of any of the points. Note that the previous operations avoid the computation of multiplicative inverses.

Operations in (27) that depend on the affine coordinate x_{sub} are used to support the EC point multiplication in Algorithm 5, where Madd and Mdouble routines stand for the EC point addition and doubling, respectively. Algorithm 5 has the property that x_{sub} is invariant and corresponds to the affine x coordinate of the EC point multiplication input point, i.e., $x_{\text{sub}} = x_1$. Algorithm 5 has a final step Mxy that aims at computing the affine coordinates of $P_3 = (x_3, y_3)$. This can be obtained from the projective coordinates as [34]:

$$\begin{aligned} x_3 &= \widehat{x}_1 / \widehat{z}_1, \\ y_3 &= (x_1 + \widehat{x}_1 / \widehat{z}_1)(\widehat{x}_2 \widehat{z}_1 \widehat{z}_2)^{-1} \end{aligned} \quad (30)$$

$$[(\widehat{x}_1 + x_1 \widehat{z}_1)(\widehat{x}_2 + x_1 \widehat{x}_2) + (x_1^2 + y_1)(\widehat{z}_1 \widehat{z}_2)] + y_1. \quad (31)$$

If the application underpinned by the EC arithmetic does not need the y coordinate, the expression for y_3 does not have to be implemented at all.

C. Application of RNS to ECC

The strategy to apply RNS to EC point multiplication is similar to the way it was applied to RSA since both rely on $\mathbb{Z}_{p=N}$ arithmetic on the exponential base or EC point coordinates. Apart from the size of N and the number of bits of the key, the main difference is that RSA only requires multiplications, whereas ECC requires additions and subtractions as well.

A scheduling for the operations in the loop of Algorithm 5 can be found in Table 1. Note that this schedule groups operations into addition/multiplication and reduction blocks. The former can be mapped directly to operations in the RNS channel whereas the latter requires employing the base extension technique, which was discussed in Section III-B and led to Algorithm 3.

The maximum dynamic range has to be identified for implementing Algorithm 5 according to Table 1. Assuming the maximum output of a reduction block in Table 1 is ψ , we can determine the maximum input of a reduction block to be \widehat{x}_1 (block add. 3 of the loop), bounded by 9ψ . From Section III-B we know that $\psi = (h_1 + 2)N$. Also, we need to make sure that the dynamic range of \mathfrak{B}_1 respects: $M_1 > (9\psi)^2 / N = (9(h_1 + 2))^2 N$.

To compute the reduction, the moduli and dynamic range M_2 for \mathfrak{B}_2 have to be defined so that $\hat{\alpha}_2$ is correctly computed using the Kawamura et al. method [33]. Taking into account the results in Section III-B we know that $M_2 > M_1 > (9(h_1 + 2))^2 N$. Moreover, we know that $MU < (h_1 + 2)N < (1 - \beta)M_2$ which allows us to set an upper bound for β :

$$(h_1 + 2)N < (1 - \beta)(9(h_1 + 2))^2 N \Leftrightarrow \beta < 1 - \frac{1}{81(h_1 + 2)}. \quad (32)$$

We know that the following also holds:

$$\beta = \Delta = \frac{h_2}{2^k m_{\min}} (2^k (m_{\min} + 2^{k-t} - 1) - m_{\min}^2). \quad (33)$$

where m_{\min} is the minimum value among all the moduli in \mathfrak{B}_2 . This can be used along with (32) to derive a minimum value for t . Finally, $\hat{\alpha}_2$ can be computed as:

$$\hat{\alpha}_2 = \left\lfloor \sum_{e=1}^{h_2} \frac{\text{truncH}_t(q_{e,2})}{2^{k-t}} + \Phi \right\rfloor 2^t, \quad (34)$$

where $\Phi = \lfloor 2^t \beta \rfloor$. All divisions are by a power of two, which makes the computation of $\hat{\alpha}_2$ efficient.

A different approach for the application of RNS to ECC relies on specific properties of the prime that underlies the EC. In [40], it is suggested to use primes of the form $q = M_1^2 - 2$. An element $X \in \mathbb{F}_q$ with RNS representation $(x_{1,1}, \dots, x_{1,h_1}, x_{2,1}, \dots, x_{2,h_2}) \in \mathfrak{B}_1 \times \mathfrak{B}_2$ is mapped into $K_X, R_X \in \mathfrak{B}_1 \times \mathfrak{B}_2$ such that $X = K_X M_1 + R_X \bmod q$. For $X, Y \in \mathbb{F}_q$,

$$\begin{aligned} XY &= (K_X R_Y + K_Y R_X) M_1 + 2K_X K_Y + R_X R_Y \\ &= VM_1 + U \bmod q \end{aligned} \quad (35)$$

with $U, V < 3M_1^2$ if $K_X, K_Y, R_X, R_Y < M_1$, where the property $M_1^2 = 2 \bmod q$ was used. Since these values must be representable in $\mathfrak{B}_1 \times \mathfrak{B}_2$, one needs to set $3M_1^2 < M_1 M_2$, which represents a considerable reduction in the required RNS dynamic range, when compared with the previous approach. However the value of $U + VM_1$ is too large for $\mathfrak{B}_1 \times \mathfrak{B}_2$, thus U and V need also to be split into (K_U, R_U) and (K_V, R_V) , to produce $K_Z < 5M_1$ and $R_Z < 6M_1$ such that:

$$\begin{aligned} XY &= U + VM_1 \\ &= K_U M_1 + R_U + K_V M_1^2 + R_V M_1 \\ &= (K_U + R_V) M_1 + R_U + 2K_V \\ &= K_Z M_1 + R_Z \bmod q \end{aligned} \quad (36)$$

The repeated application of this multiplication algorithm leads to an increasing bound on the magnitude of its outputs. The solution proposed in [40] corresponds to setting a larger value of M_2 that can handle a few multiplications, and afterwards applying a “compression” method that bounds the magnitude of the values in a pair (K, R) [40].

V. RNS Lattice-Based Cryptography

A lattice is a repeated arrangement of points, which is used to support certain cryptographic systems, as formally expressed in Definition 9.

Definition 9 (Lattice). Let $r_1, \dots, r_n \in \mathbb{R}^m$ be a set of linearly independent vectors. The lattice \mathcal{L} generated by r_1, \dots, r_n is the set of linear combinations of r_1, \dots, r_n with coefficients a_i in \mathbb{Z} :

$$\mathcal{L} = \left\{ \sum_{i=1}^n a_i r_i : a_i \in \mathbb{Z} \right\}$$

A basis for \mathcal{L} is any set of independent vectors that generates \mathcal{L} . If the basis is represented as a matrix R , having the basis vectors as rows, the lattice generated by $R \in \mathbb{R}^{n \times m}$ 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 multiplication of the vector x by the matrix R . The dimension of \mathcal{L} is the number of vectors in a basis for \mathcal{L} . Herein, vectors r_1, \dots, r_n are restricted to \mathbb{Z}^m , for computational convenience, and thus we will consider herein integral lattices. Further, we will only consider full-rank lattices, wherein $m = n$.

If U is a unimodular matrix (i.e. with integer coefficients and determinant ± 1), UR generates the same lattice as R [41]. In fact, any lattice admits an infinite number of bases as long as $n \geq 2$ [41]. Figure 7 graphically represents a lattice with $n = 2$, as well as the fundamental parallelepiped of a specific basis, established according to Definition 10.

Definition 10 (Fundamental Parallelepiped). Let \mathcal{L} be a lattice of dimension n and let r_1, r_2, \dots, r_n be a basis for \mathcal{L} . The fundamental parallelepiped for \mathcal{L} corresponding to this basis is the set:

$$\mathcal{F}(r_1, \dots, r_n) = \left\{ \sum_{i=1}^n x_i r_i : 0 \leq x_i < 1 \right\}$$

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 v modulo a basis R corresponds to finding the unique

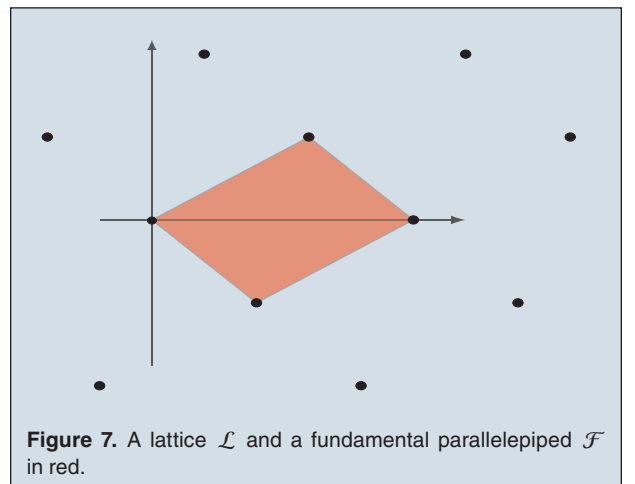


Figure 7. A lattice \mathcal{L} and a fundamental parallelepiped \mathcal{F} in red.

point $w \in \mathcal{F}(R)$ such that $v \equiv_{\mathcal{L}} w$, where $\mathcal{F}(R)$ is the fundamental parallelepiped of the rows of R .

The lattice problem which we will focus on, the Closest Vector Problem (CVP), is based on distance minimization. Herein, we measure distance through the L^2 norm: for $v \in \mathbb{Z}^n$, $\|v\|_2 = \sqrt{v_0^2 + \dots + v_{n-1}^2}$. The distance between two points $x, y \in \mathbb{Z}^n$ is defined as $d(x, y) = \|x - y\|_2$, and the distance between a vector $x \in \mathbb{Z}^n$ and a lattice \mathcal{L} is defined as $d(x, \mathcal{L}) = \min_{y \in \mathcal{L}} \{d(x, y)\}$. The CVP can be formulated as in Definition 11.

Definition 11 (Closest Vector Problem). *Given a basis R of a n -rank lattice $\mathcal{L} \subseteq \mathbb{Z}^n$ and a target vector $x \in \mathbb{Z}^n$, find a lattice vector $y \in \mathcal{L}$ such that $d(x, y) = d(x, \mathcal{L})$.*

A. Lattice-Based Cryptography

Some classical approaches to lattice-based cryptography are supported on the complexity of the computation of the closest vector of a lattice \mathcal{L} to a vector $c \in \mathbb{Z}^n$. The public-key of a user corresponds to a lattice basis with which solving the CVP is hard. Messages are ciphered by producing a point in \mathbb{Z}^n that can be interpreted as a perturbation to a point in the lattice. If one knows a nearly orthogonal basis of the lattice, Babai's Round-Off procedure [42] is a natural way to compute the closest vector to the cryptogram. Given a vector $c \in \mathbb{Z}^n$ and a basis $r_1, \dots, r_n \in \mathbb{Z}^n$ of nearly orthogonal vectors, this procedure comprises computing $a_1, \dots, a_n \in \mathbb{Q}$ such that $c = \sum_{i=1}^n a_i r_i$, and outputting $w = \sum_{i=1}^n \lfloor a_i \rfloor r_i$ ($\lfloor a_i \rfloor$ denotes the rounding of a_i to the nearest integer). A cryptogram can be deciphered by computing w , and outputting the difference between the cryptogram and w , which corresponds to the perturbation to the lattice.

Algorithm 6: Encryption algorithm of Rose's cryptosystem.

Require: $p \in \mathbb{Z}^n, B1 \in \mathbb{Z}^n$
Ensure: $c \in \mathbb{Z}$
1: $c = p[1]$
2: **for** $i = n; i \geq 2; i--$ **do**
3: $c = c - p[i] \times B1[i]$
4: **end for**
5: $c = c \bmod B1[1]$
6: **return** c

Algorithm 7: Decryption algorithm of Rose's cryptosystem.

Require: $c \in \mathbb{Z}, R_1^{-1} \in \mathbb{Q}^n$, the first row of $R^{-1}, R \in \mathbb{Z}^{n \times n}$
Ensure: $p \in \mathbb{Z}^n$
1: $p = (c, 0, \dots, 0) - \lfloor c R_1^{-1} \rfloor R$
2: **return** p

Let us present a lattice-based encryption scheme using Rose's approach [43]. The private basis is produced as a rotated nearly-orthogonal basis such that Babai's Round-Off procedure may be used to compute the closest vector. The public basis has a Hermite Normal Form (HNF). The HNF is a lower triangular basis of $\mathcal{L}(R)$, $H \in \mathbb{Z}^{n \times n}$, such that:

$$\forall 1 \leq i, j \leq n, \begin{cases} = 0 & \text{if } i < j \\ \geq 1 & \text{if } i = j \\ < H_{j,j} & \text{if } i > j \end{cases} \quad (37)$$

This normal form is unique, and can be efficiently computed from any basis of \mathcal{L} . Furthermore, the vectors of the basis are rather skewed, which makes it difficult to solve the CVP with H . With current algorithms it is also hard to convert it to a nearly orthogonal basis of the same lattice [44]. 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. More formally, H is an OHNF basis of \mathcal{L} if and only if H is an HNF basis and $\forall 2 \leq i \leq n, H_{i,i} = 1$. As an example, most lattices with a prime determinant have an OHNF basis.

The private/public key generation consists of finding a rotated nearly-orthogonal private basis, such that its HNF is optimal. Due to its OHNF, the public-key can be represented as a single column, denoted as $B1$. 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 Algorithm 6. Due to the public basis structure, the cryptogram corresponds to a vector, where all the entries but the first are zero. Therefore, it suffices a scalar c to store its value. The decryption procedure is represented in Algorithm 7. It corresponds to the computation of the difference between $(c, 0, \dots, 0)$ and the closest vector to $(c, 0, \dots, 0)$ in the lattice. The computation of the CVP is supported on Babai's Round-Off procedure. If

$$p \in \left[-\left\lfloor \frac{\sqrt{n}}{2} \right\rfloor + 1, \left\lfloor \frac{\sqrt{n}}{2} \right\rfloor - 1 \right]^n \quad (38)$$

then Babai's Round-Off produces the correct closest vector [43].

These procedures are illustrated in Figure 8, for $n = 2$, where lattice points are represented as black dots. To generate the private-key, we first produce a random rotation matrix $T = \begin{pmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} \end{pmatrix}$, a random matrix $M = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \in \{0, 1\}^{2 \times 2}$, and output $R = \lfloor 4T \rfloor + M = \begin{pmatrix} 4 & 2 \\ -2 & 3 \end{pmatrix}$. The HNF basis of this lattice is of an OHNF, and corresponds to $H = \begin{pmatrix} 16 & 0 \\ 10 & 1 \end{pmatrix}$. In Figure 8(a), the plain-text $p = (0, 1)$ is reduced modulo the OHNF basis H , producing the value $c = (6, 0)$. It should be noted that p was

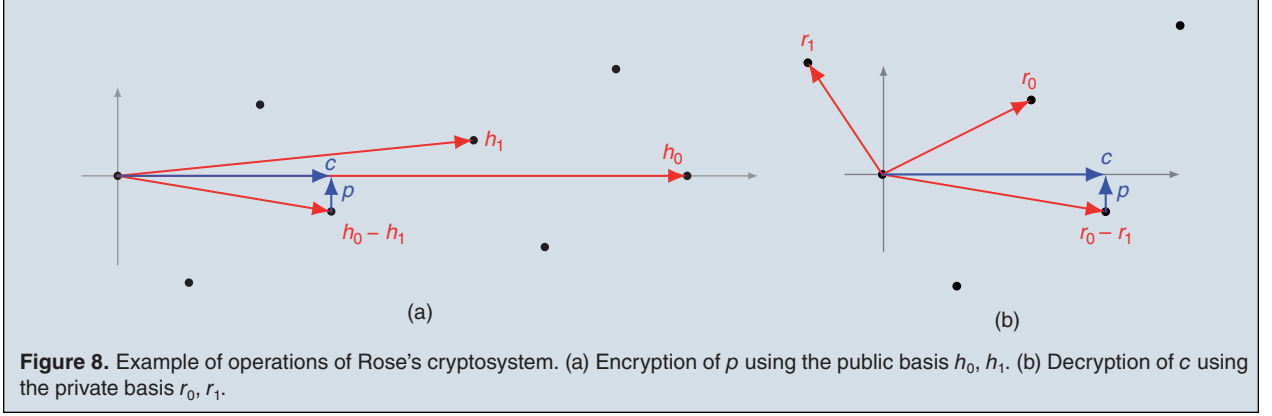


Figure 8. Example of operations of Rose's cryptosystem. (a) Encryption of p using the public basis h_0, h_1 . (b) Decryption of c using the private basis r_0, r_1 .

chosen for illustrative purposes. While it does not satisfy (38), and hence it is not guaranteed that the corresponding cryptogram will decrypt correctly, we shall see that decryption is possible in this particular case. The public basis vectors are very large and skewed, when compared with the private basis, making an attacker incapable of decyphering the resulting ciphertext. When decrypting, the private basis R is used to find the closest vector of the lattice, $(6, -1)$, to c . The value of p is then computed as $c - (6, -1) = (0, 1)$.

B. Application of RNS to LBCs

For exploiting RNS features to accelerate Babai's Round-Off algorithm, the computation described in Algorithm 7 has to be converted to integer arithmetic, as described in [19]. Hence $R' = \det(R)R^{-1}$ ($R \in \mathbb{Z}^{n \times n}$) is used instead of R^{-1} . Moreover, considering $v_1 = (1, \dots, 1)$, $\lfloor cR_1^{-1} \rfloor$ is rewritten as:

$$\lfloor cR_1^{-1} \rfloor = \left\lfloor \frac{cR'_1}{\det(R)} \right\rfloor = \left\lfloor \frac{cR'_1}{\det(R)} + \frac{1}{2}v_1 \right\rfloor = \frac{2cR'_1 + \det(R)v_1 - [2cR'_1 + \det(R)v_1 \bmod (2\det(R))]}{2\det(R)} \quad (39)$$

Furthermore, since p is constrained by (38), if $\beta > 2\lceil \sqrt{n}/2 \rceil - 1$ then the computation of Algorithm 7 may be performed modulo β . The value β can be selected to be $m_{2,1}$, which is the value of the first element of the second RNS basis [19] in Montgomery multiplication.

A description of the resulting decryption scheme can be found in Algorithm 8. The value of v'_1 , which corresponds to $\det(R)$, is used to compute $a = 2cR'_1 + (v'_1, \dots, v'_1) \bmod m_{2,1}$. Afterwards, the value of $a' = |2cR'_1 + (v'_1, \dots, v'_1)|_{D_R} \bmod m_{2,1}$, where $D_R = 2 \times \det(R)$, is computed using RNS. In order to compute this value, $R'_1 = 2R'_1M_1m_3 \bmod D_R$ and $v'_1 = \det(R)M_1m_3 \bmod D_R$ are precomputed. The value of M_1 regards the first RNS basis, and m_3 is an extra modulo that is introduced

to simplify the reduction operation. The extra M_1m_3 term will be eliminated during the RNS reduction.

The *ReduceModDR* function is presented in Algorithm 9 and is based on the reduction algorithm presented in Section III-B. The main and secondary RNS bases $\mathfrak{B}_1 = \{m_{1,1}, \dots, m_{h_1,1}\}$ and $\mathfrak{B}_2 = \{m_{1,2}, \dots, m_{h_2,2}\}$ have dynamic ranges M_1 and M_2 , respectively. Firstly, q_1 is computed such that $cR'_1[i] + v'_1 + q_1D_R$ is divisible by M_1 . Since $cR'_1[i] + v'_1 < \det(R)D_R + D_R$, M_1 should satisfy $M_1 > \det(R) + 1$. Afterwards, q_1 is extended to the second basis \mathfrak{B}_2 , by evaluating (8) for each of the moduli of \mathfrak{B}_2 . It should be noted that the first extension uses a value of α_1 in Algorithm 3 that is set to zero, and therefore there will be an extension error less than $(h_1 - 1)M_1$ [45]. As such, $a'_2 = ((cR'_1[i] + v'_1) + D_R \times q_2) \times M_1^{-1}$ is bounded by $a'_2 < (h_1 + 1)D_R$. Hence, a'_2 should be reduced a second time, by using an extra modulus m_3 . If $m_3 > (h_1 + 1)(\forall 1 \leq i \leq h_2, m_3 < m_{i,2})$, $a'_2 = (a'_2 - D_R \times \hat{q}) \times m_3^{-1}$ is computed, and $-D_R < a'_2 < D_R$, where $\hat{q} = D_R^{-1}a'_3 \bmod m_3$.

Lastly, if $a'_2 < 0$, an addition by D_R must be performed 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 $D_R < \lfloor m_{h_2,2}/2 \rfloor (M_2/m_{h_2,2}) < M_2 - D_R$, checking if $a'_2 < 0$ is equivalent to testing if $\hat{a}'_{h_2,2} \geq \lfloor m_{h_2,2}/2 \rfloor$, where $\hat{a}'_{h_2,2}$ denotes the h_2^{th} MRS digit of a'_2 in (9). In Algorithm 9, the RNS digits are overwritten with the MRS digits, and

Algorithm 8: Improved decryption algorithm of rose's cryptosystem.

Require: $c \in \mathbb{Z}, R'_1 \bmod m_{1,2} \in \mathbb{Z}^n, R'_1 \in \mathbb{Z}^n, R \bmod m_{1,2} \in \mathbb{Z}^{n \times n}, v'_1 \in \mathbb{Z}, v''_1 \in \mathbb{Z}$
Ensure: $p \in \mathbb{Z}^n$
1: $a = 2cR'_1 + (v'_1, \dots, v'_1) \bmod m_{1,2}$
2: $a' = \text{ReduceModDR}(c, R'_1, v'_1)$
3: $b = \frac{a - a'}{2\det(R)} \bmod m_{1,2}$
4: $p = (c, 0, \dots, 0) - bR \bmod m_{1,2}$
5: **return** p

Algorithm 9:
RNS modular reduction (ReduceModDR).

Require: $c \in \mathbb{Z}, R_1' \in \mathbb{Z}^n, v_1' \in \mathbb{Z}$, RNS constants

Ensure: $a' \in \mathbb{Z}^n$

```

1: for  $i = 1; i \leq n; i++$  do
2:   In ch.  $e, 1$ :  $q_{e,1} = \lfloor -D_R^{-1} (cR_1'[i] + v_1') \rfloor_{m_{e,1}}$ 
3:   In ch.  $e, 2$ :  $q_{e,2} = \left\lfloor \sum_{j=1}^{h_1} q_{j,1} \lfloor M_{j,1}^{-1} \rfloor_{m_{e,1}} \times M_{j,1} \right\rfloor_{m_{e,2}}$ 
4:   In ch.  $e, 2$ :
        $a'_{e,2} = \lfloor (cR_1'[i] + v_1' + D_R \times q_{e,2}) \times \lfloor M_1^{-1} \rfloor_{m_{e,2}} \rfloor_{m_{e,2}}$ 
5:   In ch. 3:  $q_3 = \left\lfloor \sum_{j=1}^{h_1} q_{j,1} \lfloor M_{j,1}^{-1} \rfloor_{m_{e,1}} \times M_{j,1} \right\rfloor_{m_3}$ 
6:   In ch. 3:  $a'_3 = \lfloor (cR_1'[i] + v_1' + D_R \times q_3) \times \lfloor M_1^{-1} \rfloor_{m_3} \rfloor_{m_3}$ 
7:   In ch. 3:  $\hat{q} = \lfloor (D_R^{-1}) a'_3 \rfloor_{m_3}$ 
8:   In ch.  $e, 2$ :  $a'_{e,2} = \lfloor (a'_{e,2} - D_R \times \hat{q}) \times \lfloor m_3^{-1} \rfloor_{m_{e,2}} \rfloor_{m_{e,2}}$ 
9:   for  $j = 1; j \leq h_2; j++$  do
10:    In ch.  $e, 2$  for  $e \in \{j+1, \dots, h_2\}$ :
        $a'_{e,2} = \lfloor (a'_{e,2} - a'_{j,2}) \lfloor m_{j,2}^{-1} \rfloor_{m_{e,2}} \rfloor_{m_{e,2}}$ 
11:   end for
12:   if  $a'_{h_2,2} < \lfloor \frac{m_{h_2,2}}{2} \rfloor$  then
13:      $a'[i] = a'_{i,2}$ 
14:   else
15:      $a'[i] = \lfloor a'_{i,2} + D_R \rfloor_{m_{i,2}}$ 
16:   end if
17: end for
18: return  $a'$ 

```

computed in the innermost loop. Afterwards, D_R is added to the result modulo $m_{1,2}$ if $a'_2 < 0$.

Subsequent to the conditional subtraction, the returned value from Algorithm 9 corresponds to $a' = a \bmod D_R \bmod m_{1,2}$. The plain-text p is then evaluated as $p = (c, 0, \dots, 0) - \frac{a-a'}{2\det(R)} R \bmod m_{1,2}$ (line 4 of Algorithm 8).

There are other approaches in the literature that apply RNS to LBCs. [20] implements part of the CVP procedure in the RNS domain, namely by computing $x = \left\lfloor \sum_{i=1}^{h_1} M_{i,1} \lfloor cR_1' \rfloor_{m_{i,1}} \right\rfloor_{M_1}$, with $r_i = R_1^{-1} \det(R) \bmod m_{i,1}$, $r_i \in \mathbb{Z}^n$, and finalizing with $w = \left\lfloor \frac{x}{\det(R)} \right\rfloor R$. In [46], the decryption procedure is changed so that the value of $\lfloor \gamma cR^{-1} \rfloor$ is computed, instead of using (39) to compute $\lfloor cR^{-1} \rfloor$. This enables one to detect when there has been an extension error when performing basis extension in Algorithm 9, by checking whether the result is divisible by γ or not, instead of performing a conversion to the MRS. Therefore, the second RNS basis is comprised of $\mathfrak{B}_2 = \{m_{1,2}, m_{2,2}\}$, where $m_{2,2} = \gamma$. After extending the result of $q_{e,1} = \lfloor -D_R^{-1} (\gamma cR_1'[i] + v_1') \rfloor_{m_{e,1}}$, with $R_1' = 2R_1 M_1 \bmod D_R$ and $v_1' = \det(R) M_1 \bmod D_R$, to the second basis, and com-

puting $a'_{e,2} = \lfloor (\gamma cR_1'[i] + v_1' + D_R \times q_{e,2}) \times \lfloor M_1^{-1} \rfloor_{m_{e,2}} \rfloor_{m_{e,2}}$, one can take the difference between the results in channels $m_{1,2}$ and $m_{2,2}$ to get the offset-free result. The result of (37) would then be divided by γ before multiplying it by R .

VI. RNS Based Public-Key Cryptosystems

This section addresses the design of efficient parallel algorithms for RSA, ECC and LBC, targeting platforms that support data-level parallelism. RNS is used to expose parallelism and transform algorithms targeting processors with SIMD extensions, GPUs and dedicated architectures and circuits.

A. Circuits and Architectures for Implementing Cryptosystems

General purpose processors, for example the Intel Core i7 processors, with four cores supported on the Nehalem microarchitecture, as well as other superscalar processors for embedded systems [47] have been used. The extensions of these microarchitectures to efficiently support Single-Instruction-Multiple-Data (SIMD) processing of vectors of integer or floating point numbers have been exploited (NEON for the ARM processors accommodates 128 bits while the Intel AVX2 technology supports 256 bits).

Beside general purpose multiprocessors, programmable and hardware accelerators have been adopted in order to improve performance and energy efficiency. GPUs are parallel programmable accelerators based on a massive number of simple high-pipeline multi-cores that execute multiple threads on a Single Instruction Multiple Threading (SIMT) approach to exploit data parallelism. In the past few years, the numbers of cores in a GPU has significantly increased, up to thousands of cores, usually organized around dozens of stream multiprocessors. We are using NVIDIA GPUs to obtain the results replicated in this paper, ranging from one of the first unified Tesla architectures, the GeForce GTX 285 GPU, that has 240 unified shaders, to one of the most recent GPUs, the K40c GPU, created for server purposes, supported on the Kepler architecture with 2880 cores. Furthermore, Field Programmable Gate Arrays (FPGA) have been employed to implement the RNS-based cryptographic circuits presented in this work, namely the Virtex-4, Virtex-5 and Kintex-7 FPGAs, all from Xilinx [48].

B. RSA and ECC Implementation Strategy

As previously stated, the strategy to obtain an RNS-based implementation of the RSA modular exponentiation and EC point multiplication is similar in the sense that both rely on $\mathbb{Z}_{p=N}$ arithmetic on the exponential base or EC point coordinates. The details of the computation

required for a RSA modular exponentiation are presented in Algorithm 1 while for the EC point multiplication are presented in Algorithm 5 and complemented with (29). Whereas RSA only requires modular multiplications, EC requires additions/subtractions as well. This makes the latter slightly more elaborate to implement and therefore it will be our focus in this Section.

The first step to design an efficient RNS implementation is to determine an RNS channel width and moduli that suit the target technology. In order to better illustrate the design strategy, we adopt a concrete target in this section consisting of NVIDIA GPUs and a underlying CUDA implementation. Also, we will implement an RNS channel using one CUDA thread.

The channel width has to be such that enables the efficient execution of channel arithmetic, i.e. additions and subtractions modulo the channel assigned modulus. A possible implementation of the channel reduction is presented in Algorithm 10. Knowing the target machine uses a 32-bit wide physical register, a suitable value for the RNS channel width would be $k = 16$. This ensures z_L and z_H fit in a register and Algorithm 10 is executed without arithmetic overflow. As discussed in [17], the least number of iterations in the *while* loop in Algorithm 10 is obtained if the modulus is closest to 2^k . Therefore, the moduli for the different channels i should be of the form $2^k - c_i$, where c_i is a small constant that ensures that all moduli are pairwise coprime. Further, the operation $\min(z, z - r_{m_i})$ is performed using unsigned arithmetic; and the *while* loop therein is unrolled, taking into consideration the maximum value to be reduced and the minimum value in the RNS bases, so that divergences between the multiple threads in the GPU are avoided.

A second step is to obtain a valid schedule of the algorithm one wants to accelerate with RNS. We have discussed a possible scheduling in Section IV-C. Assuming we have h_1 moduli in \mathcal{B}_1 , each supported by a CUDA thread, the computation of modular reductions is as depicted in Figure 9. If we select a 224-bit finite field to

Algorithm 10: GPU modular reduction.

Require: $z \in \mathbb{Z}, m_i \in \mathbb{Z}$

Ensure: $z \in \mathbb{Z}$

```

1: while  $z \geq 2^k$  do
2:    $z_L = z \& (2^k - 1)$ 
3:    $z_H = z \gg k$ 
4:    $z = z_L + (2^k - m_i)z_H$ 
5: end while
6: return  $z = \min(z, z - m_i)$ 

```

underly the EC arithmetic (N has 224 bits) and looking for pairwise coprime moduli $2^k - c_i$ for ascending values of c_i , we can conclude that $h_1 = 15$, based on the bounds determined in Section IV-C. Moreover, for this particular example ($k = 16$ and $h_1 = 15$) we get $t = 5$ for the computation of (34).

Other target-specific considerations can be made to improve the performance of the implementation. Examples are how the data is organized in memory and how constants are accessed. For a CUDA implementation, it is important to ensure that the same value encoded in the different channels is stored in consecutive memory positions so that the accesses by the threads that implement a channel are coalesced. For constants used in the implementation (e.g. the λ constants in Section III-B) it is important to store them in proper memory locations (CUDA constant memory) or in some cases to compute them and store them in scratchpad memories (CUDA shared memory) close to the processing cores to avoid expensive memory transactions. In the work presented in [17] it is suggested that constants λ_{2ej} and λ_{6ej} benefit from being computed at runtime and stored in CUDA shared memory whereas the other constants should rather be looked up from CUDA constant memory.

Note that, notwithstanding the strategy presented herein is mostly targeted at a GPU device, it can be

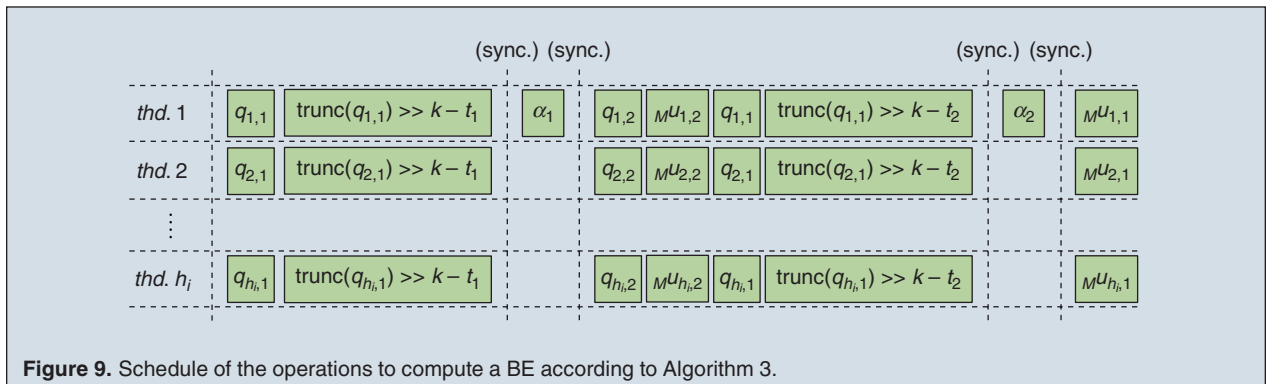


Figure 9. Schedule of the operations to compute a BE according to Algorithm 3.

Parallelism in different platforms can be exploited to speed up the execution of RNS LBCs by modifying the ReduceModDR function.

easily extended to other devices like threaded CPU or dedicated/reconfigurable hardware implementations. Different devices offer different degrees of freedom (e.g. number of channels, width of each channel) and finding the sweet spot for the different parameters requires iterating on the different steps stated above.

C. LBCs Implementation Strategy

In this section, it is explained how parallelism can be exploited to speed up the execution of RNS LBCs. In a first scenario, we consider an heterogeneous platform with a CPU and a GPU. One may implement Babai's Round-Off algorithm by offloading the execution of the *ReduceModDR* function to the GPU, preceded by the required data transfers. During the GPU execution of line 2 of Algorithm 8, the CPU executes line 1 simultaneously. After synchronizing the CPU and the GPU operations, which assures that the *ReduceModDR* results were fully transferred to the CPU, the computation of lines 3 and 4 takes place on the CPU.

The computation of $a = 2cR'_0 + (v'_1, \dots, v'_1) \bmod m_{1,2}$ is split among the threads of the CPU. Each core computes a subset of the result. Concerning the execution of the *ReduceModDR* function in the GPU, some RNS constants may be collapsed: it suffices to store $-D_R^{-1} |M_{i,1}^{-1}|_{m_{i,1}} \bmod m_{i,1}$, and $|M_1^{-1}|_{m_3} |D_R^{-1}|_{m_3} \bmod m_3$, instead of $-D_R^{-1} \bmod m_{i,1}$, $M_{i,1}^{-1} \bmod m_{i,1}$, $M_1^{-1} \bmod m_3$ and $D_R^{-1} \bmod m_3$. This reduces the amount of memory required by the algorithm.

The *ReduceModDR* function presented in Algorithm 9 was implemented as an OpenCL kernel. Each work-group was associated with a single dimension, and each work-item with a modulus from \mathfrak{B}_1 and another from \mathfrak{B}_2 . The resulting kernel only requires 3 barriers: after lines 2, 7 and 10 of Algorithm 9. Moreover, lines 5 up to 7, and 12 up to 16 are executed on a single thread.

After the reduction result is transferred to the CPU, $b = ((a - a')/2 \det(R)) \bmod m_{1,2}$ is co-jointly computed by multiple threads in the multiple available CPU cores. Then the vector-matrix multiplication $bR \bmod m_{1,2}$ takes place: each core multiplies a set of entries of b by the corresponding lines of R , and the partial results of each core are added to produce the multiplication result afterward. Finally, the value of $p = (c, 0, \dots, 0) - bR \bmod m_{1,2}$ is determined, and each thread computes a subset of the final result.

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 previously executed on the CPU are accomplished in a similar way. Additionally, the *ReduceModDR* function, which still makes use of the RNS, is enhanced with multi-threading, with each thread computing part of the loop iterations in line 1 of Algorithm 9.

The OpenMP `#pragma omp for` directive is used to split the multi-dimensional computation of $a = 2cR'_0 + (v'_1, \dots, v'_1) \bmod m_{1,2}$, $b = ((a - a')/2 \det(R)) \bmod m_{1,2}$ and $p = (c, 0, \dots, 0) - bR \bmod m_{1,2}$ among the threads. The vector-matrix multiplication bM requires not only the use of `#pragma omp for` but also of `#pragma omp critical` for the modular sum of the thread partial results. For executing the *ReduceModDR* function on the CPU, a `#pragma omp for` directive was applied to line 1 of Algorithm 9. It should be noted that, since the targeted CPUs featured data-paths of 64-bits, the moduli bit-width was changed to 32-bits.

SIMD parallelism can also be used to enhance the execution on the CPU. In particular the *ReduceModDR* is modified to exploit SIMD extensions. First, it is possible to process multiple channels at a time for the steps in lines 2, 4, 8 and 10 of Algorithm 9. Second, it is possible to accelerate all the summations by splitting their computation over multiple summations and perform those in parallel.

An implementation of RNS-based LBC for FPGAs was described in [46]. It proposes the use of an array of Rox-units. These units compute not only modular multiplication but also modular addition, and are connected sequentially to perform calculus of the form $\sum_{i=1}^k a_i b_i \bmod m_j$. This architecture is exploited to compute fast RNS basis extensions and modular vector-matrix multiplications. Furthermore, a second Montgomery domain level is used for representing integers in the computation channels, each bound to a modulus of the RNS basis.

VII. Experimental Results and Computer Assisted Tools

This section provides experimental results for state-of-the-art implementations of RNS-based cryptosystems, and discuss the possibility of systematizing the design of RNS-based cryptosystems by using computer-assisted techniques and tools.

A. EC experimental results

The EC point multiplication design strategy based on RNS and described in Section VI-B has been materialized into CUDA and OpenCL implementations and thoroughly tested in systems whose specifications are in Table 2. The results are summarized in Figure 10 for a 224-bit underlying finite field. The minimum latency is 29.2 ms for when a single multiplication per block is being computed. The maximum throughput is 9827 multiplications/s registered when using 30 CUDA blocks and 20 EC point multiplications per block. Up to 600 EC point multiplications can be computed by a single device in 61.3 ms.

Results were obtained for both GPU and CPU for the same RNS-based implementation in OpenCL and different underlying finite fields. Additionally, we implemented for the CPU the same version of Algorithm 5 using the GNU Multiprecision Library (GMP) and OpenMP to parallelize.

These results are summarized in Figure 11. The GPU minimum latency obtained for the OpenCL implementation is the same as in CUDA (29.2 ms) whereas the throughput drops to 7474 multiplications/s. This drop is attributed to overheads introduced by the OpenCL toolchain. Our aim is not to compare different technologies as the ones selected herein for the GPU and CPU device. Instead our interest is to evaluate how the implementation scales in different devices. We observe that for smaller field sizes, the CPU latency is up to 2.5 smaller than the GPU's, but for the larger field sizes, the results converge and the CPU latency is only 1.05 smaller. This suggests that the computation time does not increase with the field size for the GPU as fast as for the CPU. This relates with penalties with the thread switching/synchronization which for the GPU is smaller. This

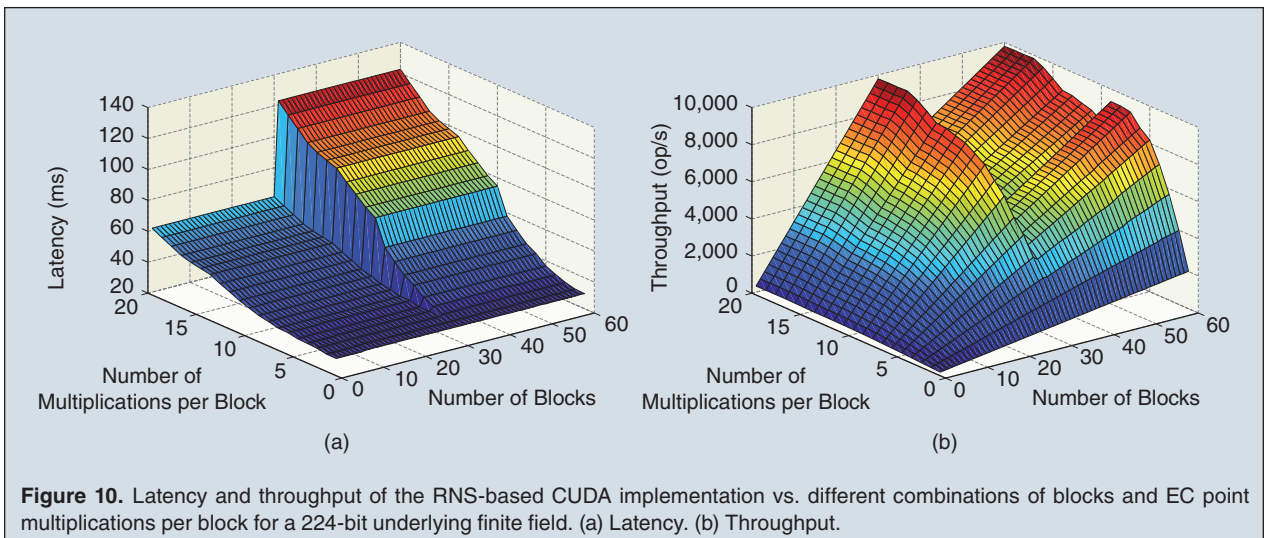
Table 2.
Experimental setup summary for the CUDA and OpenCL-based implementations.

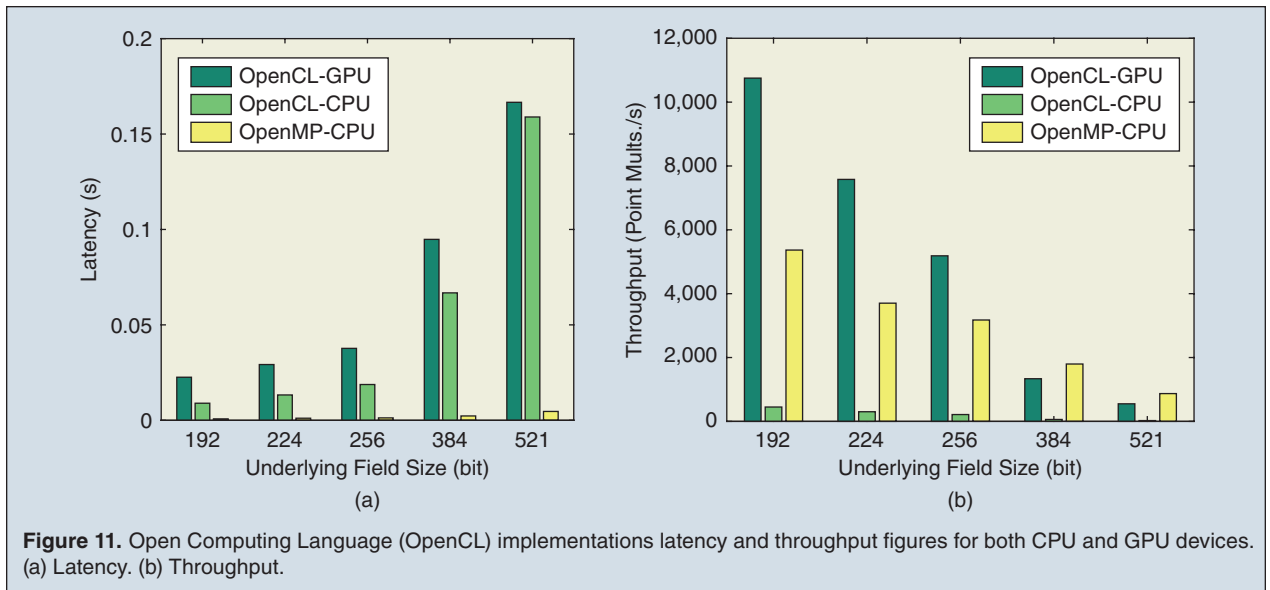
Implementation	Characteristics
GPU (CUDA)	NVIDIA GeForce GTX 285 30 multiprocessors (1476 MHz) 1 GB video memory (1242 MHz) NVIDIA CUDA version 3.1 NVIDIA CUDA-OpenCL version 3.1
Host	Intel Core 2 Quad Q9550 (2.83 GHz) 2 × 2GB Memory (DDR2 1066) PCI Express 16x
CPU (OpenCL)	AMD Ph. II X4 945 4-Core (3.0 GHz) OpenSUSE 11.0 Operating System 2 × 2GB Memory (DDR3 1333) 64-bit datapath AMDAPP OpenCL support GCC 4.5.1 OpenMP implementation

also justifies why the OpenMP implementation provides much lower latency - it does not requires thread switching/synchronization to compute an instance of EC point multiplication.

Throughput-wise, we observe that the GPU implementation provides up to 2 times more throughput for the smaller field sizes and up to 1.5 times less throughput for the larger field sizes when comparing with the OpenMP implementation. This result suggests the existence of a tipping point for which the RNS parallelization does not pay off the increase of the modular reduction's complexity.

Although the second method to apply RNS to ECC proposed in [40], and described in Section IV-C, looks promising, no experimental results are presented therein for the EC point multiplication. Nevertheless, one could expect a reduction in latency in the order of 23-42% [40].





B. LBCs Experimental Results

The first method described in Section VI-C for LBC decryption was implemented and thoroughly verified. It should be noted that the described encryption procedure is very optimized and its execution time is negligible when compared with decryption. Also, the sequential method of Algorithm 7, which does not make use of RNS, was implemented using the NTL 6.2.1 library [49] for comparison purposes. They were tested on a system with an i7 4770 K with 32GB of RAM and 4 cores, operating at 3.5 GHz, a Tesla K40c with 12GB of main memory and 2888 Shader Processing Units (SPUs), operating at 0.7 GHz, and a GeForce GTX 780 Ti with 3GB and 2880 SPUs, operating at 0.9 GHz. 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 Table 3. 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.

There is a direct link between the GPU performance and their memory bandwidth, with the GTX 780 Ti outperforming the K40c. This results from the low arithmetic intensity of the kernels. Moreover, the K40c is outperformed by the i7 platform. 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 Algorithm 8 in parallel on the CPU, this step has a small arithmetic complexity. The other concerns

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 significantly reduced. This hampers the negative impact of the reduced number of cores in the CPU.

As n grows, there is not only an increase in the number of dimensions, but the RNS bases also grow. This is due to the fact that the determinant of R also grows with n . When using RNS, one has to select a dynamic range that accommodates the maximum value to be represented, and the computational load is directly related to that value. In contrast, for the sequential implementation, since a multi-precision library is used, this range is modified as needed, which means that computational complexity does not grow as rapidly. As such, speed-up tends to decrease as the considered dimensions grow.

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 [41], a cryptosystem similar to the previously considered was implemented using the NTL 5.5.2 library on an Intel Core 2 Duo platform, running at 2.1 GHz, with 4 Gb RAM. It should be noted that the key generation step is slightly different, and thus the plain-text space is limited to $\{-1, 0, 1\}^n$. There, decryption took place as described in Algorithm 7, except that cR'_1 was computed using RNS, by computing each entry of the result for all moduli of the RNS base and reconstructing the values with the CRT. The implementation only uses a single core and performs decryption in 294.42 and 1323 millions of

The design of a cryptographic algorithm using RNS offers several degrees of freedom and there are tradeoffs that are hard or time consuming to evaluate.

clock cycles for $n = 500$ and $n = 800$, respectively, in an Intel Core Duo. Even though different platforms were used, the reported numbers of clock cycles are in the same order of magnitude to those of Table 3 for the sequential method. As such, one may conclude that not only is most beneficial to employ RNS for the whole Baibai Round-Off procedure, but also that LBCs are greatly enhanced with data parallelism.

An implementation of an RNS LBC for FPGAs can be found in [46]. The technique of computing $\lfloor \gamma c R^{-1} \rfloor$ instead of $\lfloor c R^{-1} \rfloor$ described in Section VI-C for detecting extensions errors was used. It should be noted that the public-key was not restricted to an OHNF. Therefore, c was not a scalar value but a vector, and this value should be multiplied by R^{-1} instead of R_1^{-1} as described in Algorithm 7. However, the designed architecture did not fit the target FPGAs (Virtex-5 and Kintex-7) for lattices with dimensions greater than 128. Therefore it is hard to compare the experimental results with the previous implementations.

C. Computer Assisted Tools

The design of a cryptographic algorithm using RNS offers several degrees of freedom. Different combinations of channel width and number of channels can be used for similar dynamic ranges, and notwithstanding the designer can offer a good guess regarding these parameters given his knowledge of the technology, there are tradeoffs that are hard to evaluate. That would require iterating the design which can be time consuming and error prone given that changing a single parameter would require all the others to be calculated and evaluated. The Computing with the RNS Framework (CRNS) was proposed in [50] to tackle this problem. It departs

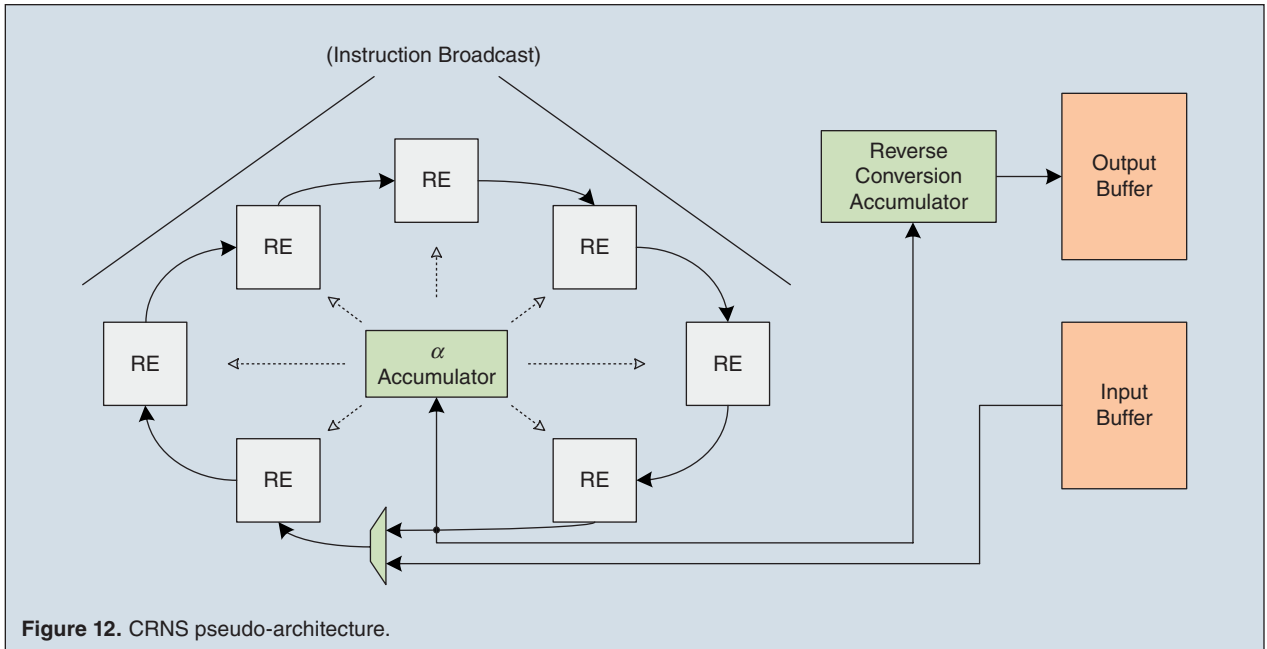
from the observation that calculating all RNS-based constants can be made automatic and save the designer the trouble of having to master complicated field-arithmetic details. It consists of a tool that allows the designer to describe the target field-arithmetic algorithm using a C-like programming language and obtain within a few seconds a fully functional RNS-based implementation. The tool also takes as input the channel width and by adjusting that parameter the designer can easily obtain an implementation for different channel widths that he can evaluate and eventually deploy.

CRNS uses the LLVM compiler infrastructure and the GNU multi-precision library to process the input code and compute all the implementation parameters. It also proposes the pseudo-architecture in Figure 12 that can model different target devices such as GPUs or FPGAs. This pseudo-architecture has the shape of a ring and mimics the data dependencies required to compute a base extension and each of its components can be implemented by components of the target device. E.g. for a GPU, a Ring Element (RE) consists of a thread, α accumulator is the thread in charge of collecting the contributions to the base extension offsets, and the input/output buffer are the GPU global memory. If the device is an FPGA, RE would be an arithmetic unit (DSP), α accumulator a real hardware accumulator, and the input/output buffers would be memory connected to the device I/O pads.

On top of the design automation, CRNS can also provide tuned implementation by working at the PTX ISA level for an NVIDIA GPU and HDL level for an FPGA. That allows the tool to control lower level details like carry-out/in bits that are not exposed by high-Level languages like CUDA but important to implement efficient modular

Table 3.
Decryption performance for the i7 4770K, K40c, and GTX 780 Ti platforms.

Execution Time [$\times 10^6$ 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 w/AVX2 (i7 4770K)	8.668 (11.2)	29.05 (9.8)	74.78 (8.3)	148.5 (8.2)



arithmetic. The EC point multiplication and RSA modular exponentiation were implemented using the CRNS high-level language and results were obtained for a GTX 580 GPU and a Virtex 4 FPGA, which can be found in [15], [50].

VIII. RNS for Improving Resistance to Attacks

Cryptanalysis is the field of study where systems are analyzed in the perspective of exploiting their vulnerabilities. It is used to breach cryptographic security systems and gain access to the contents of encrypted information. Cryptanalysis covers Side-Channel Attacks (SCAs), which exploit weaknesses in the implementation of the cryptosystems. The analysis of the power consumption of cryptographic hardware devices and systems seems to be the most common and also easiest way to perform those attacks.

The Simple Power Analysis (SPA) side-channel attack directly analyses variations in power consumption measurements collected over time as different cryptographic operations are performed. As an example, consider that the squaring operation of Algorithm 1 has been implemented in a way different from multiplication, so that it was more efficient. In this case, the power trace of squaring would differ from the one of multiplication. Thus, analyzing the power trace of the exponentiation algorithm will enable one to retrieve the used exponent, which corresponds to the private-key when deciphering or signing data with RSA.

Differential Power Analysis (DPA), a more advanced attack, uses statistical analysis by correlating predictions with the actual power measurements from a

cryptosystem [51]. A simple example follows. If one has a multitude of power traces for the RSA decryption of $r = c_i$ using Algorithm 1 and wants to obtain the Most Significant Bit (MSB) of the private-key (corresponding to the exponent b), one can compute the Hamming weight (i.e. the number of nonzero bits) of r after the first iteration for both $b_{k-2} = 0$ and $b_{k-2} = 1$, for all cryptograms c_i . Assuming that the power consumption of storing r in memory depends linearly on the Hamming weight, it is possible to compute the value of b_{k-2} , by determining which predicted Hamming weight best matches the power traces. A large number of power traces is necessary to mitigate the power consumption noise of other parts of the circuit. This process can be repeated for the bit b_{k-3} , and then for b_{k-4} and so forth, until all private-key bits are recovered.

Several attacks against RSA have been reported and countermeasures discussed [52]. There are DPA attacks that need a lot of traces, and some of them have been thwarted by exponent blinding whereas others by message blinding [53]. With exponent blinding, each time a message is decrypted or signed, a random multiple r of $\phi(N) = (p-1)(q-1)$ is added to the private-key d so that the result of exponentiating $c^{d+r\phi(N)} \bmod N$, is still the same as exponentiating $c^d \bmod N$, but correlation attacks between multiple power traces are prevented. With message blinding, the value to be exponentiated c is first multiplied by r^e , where r is a random value and e is the public-key, such that $m = (cr^e)^d r^{-1} \bmod N$ is obtained, and the Hamming weight of intermediary results cannot be guessed. More recently, several SCA attacks have been reported also for ECC. A survey on known

**RNS can also be used to strengthen the security of private-key cryptosystems.
In particular, it may be used to prevent fault attacks against the
Advanced Encryption Standard.**

SCAs for ECC can be found in [54]. With ECC, the resistance to DPA is efficiently tackled by randomizing the projective coordinates, the EC base point or the scalar, which has given rise to various countermeasures that thwart known attacks [55].

An internal correlation has been proposed to counteract message blinding with a reduced number of traces [56]. The general idea is to use as reference the power consumption at smartly chosen time instances as predictions for the DPA attack. For instance, one could choose as reference the power trace of loading the value of $a \bmod N$ in Algorithm 1 to detect squarings instead of modeling power usage with the Hamming weight. The Big Mac attack is also a DPA attack against sliding or fixed window algorithms [57]. In particular, it reduces the amount of power traces necessary to conduct a DPA attack, by analyzing the power consumption associated with each digit individually during the multiplication algorithm. In some cases, the attack may only use one trace, and so thwarts exponentiation blinding.

The properties of RNS, namely the potential parallelism, can also be exploited to leverage the randomization of processed data in order to address SCAs. In particular, Leak Resistant Arithmetic (LRA) was proposed to provide protection at the arithmetic level supported on RNS [58]. An implementation of modular exponentiation based on the LRA has been proposed in [59]. Two randomization approaches have been proposed on these works: *i*) random choice of the initial bases: randomization of the input data is provided by randomly choosing the elements of \mathfrak{B}_1 and \mathfrak{B}_2 moduli sets from a predefined pool of $2h$ moduli; and *ii*) random permutation of the moduli during processing.

With the first approach, a random draw of \mathfrak{B}_1 and \mathfrak{B}_2 , with h moduli each, is seen as a permutation γ over the predefined set $\mathfrak{B} = \mathfrak{B}_{1,\gamma} \cup \mathfrak{B}_{2,\gamma}$ of size $2h$ [58]. As pointed out in Section III, to obtain the Montgomery representation of any input X , its value is multiplied by $M_{1,\gamma}$ modulo N , where $M_{1,\gamma}$ represents the range of $\mathfrak{B}_{1,\gamma}$. In practice, this value is obtained by multiplying X by $M_{1,\gamma}^2 \bmod N$, followed by the Montgomery reduction that divides the result by $M_{1,\gamma}$. However, since $M_{1,\gamma}$ is the product of h randomly chosen moduli, $M_{1,\gamma}^2 \bmod N$ is not known beforehand. On one hand, the huge number of possibilities for $M_{1,\gamma}$ makes the precomputation of the products of all the subsets of h elements of \mathfrak{B} unpractical. On the other hand, the on-the-fly evaluation of

$M_{1,\gamma}^2 \bmod N$ would be very expensive. Therefore it was proposed to multiply X by M , where M is the dynamic range of \mathfrak{B} , and afterwards apply the Montgomery reduction with the roles of \mathfrak{B}_1 and \mathfrak{B}_2 reversed, producing (40):

$${}_M X = \frac{X(M \bmod N) + QN}{M_2} \equiv XM_1 \bmod N \quad (40)$$

where $Q \equiv -X(M \bmod N)N^{-1} \bmod M_2$, and the required amount of precomputed constants is small. The result is obtained in RNS for the two bases, and the processing is continued with the two bases $\mathfrak{B}_{1,\gamma}$ and $\mathfrak{B}_{2,\gamma}$ playing their usual role.

When randomly permuting the moduli during processing, corresponding to the aforementioned second approach, the Montgomery representation of any RNS number changes from $XM_1 \bmod N$ to $XM'_1 \bmod N$. To perform this change, it has been suggested to multiply $XM_1 \bmod N$ by $M'_1 \bmod N$ (using the previously described approach of multiplying by $M \bmod N$ and then use Montgomery reduction with the roles of \mathfrak{B}'_1 and \mathfrak{B}'_2 reversed), and then using a Montgomery reduction with the old basis to divide the result by M_1 .

The previously referred works have directly applied RNS to make asymmetric-key encryption systems based on the RSA more resistant to SCA, namely DPA. In particular, the initial permutation of the moduli works as a means of message blinding. Furthermore, the shuffling of the moduli during the exponentiation makes the Big Mac attack harder to perform, requiring more power traces. Since more than one trace is now required, the Big Mac attack is no longer effective due to exponent blinding. Additionally, the moduli shuffling invalidates the internal correlation attack. Thus, the combination of exponent blinding with the two RNS moduli permutation techniques prevents most DPA attacks.

The main weakness of the ECC regarding SPA comes from the difference in complexity between the addition and the doubling on elliptic curves. To overcome this difference and mitigate the problem, one can use representations of the curve for which the two operations are obtained with the same formulae [60], [61] or by using the Montgomery ladder algorithm for the scalar multiplication [62]. RNS can also be combined with LRA [63] or other approaches to randomize the moduli sets [64] on elliptic curves over $GF(p)$. With these approaches, one gets the security benefits of LRA, while benefiting from the efficiency of both RNS and ECC.

We close this section by noting that even though in this paper we have focused on public-key cryptography, the RNS can also be used to strengthen the security of private-key cryptosystems. In particular, it may be used to prevent fault attacks against the Advanced Encryption Standard (AES) [65].

IX. Further Developments

This paper was focused on accelerating the encryption and decryption procedures of existing cryptosystems by exploiting the RNS. Nevertheless, its use is more extensive and has been used to accelerate other cryptographic operations. It has also been used as the foundation of certain cryptographic constructions.

Bilinear Pairings (BPs) have found extensive use in cryptography. They support Identity-Based Encryption (IBE) [66], wherein public-keys are derived from identities; Attribute-Based Encryption (ABE) [67], wherein ciphertexts and secret-keys are associated with a set of attributes, and a user is only capable of deciphering a ciphertext if the attributes of his secret-key match those of the ciphertext; and short signatures [68]. Implementations of pairings supported on RNS can be found in [69], [70].

Recently, RNS has also been used to develop Fully Homomorphic Encryption (FHE) systems [71]–[73]. FHE corresponds to a cryptographic system where cryptograms can be added and multiplied using specific operations. These operations produce new cryptograms encrypting, respectively, the addition and the multiplication of the underlying plaintexts. With these two operations it is possible to describe any function as a circuit, enabling the processing of data in encrypted format.

X. Conclusion

This paper shows how Residue Number Systems (RNSs) can support the design of faster, more efficient and resilient circuits and systems for asymmetrical encryption. RNS allows numbers to be represented by a set of residue digits, which can be processed in separate channels.

Modular reduction is a key operation in public-key cryptography, therefore it is of the utmost importance to map the Montgomery reduction algorithm efficiently into the RNS domain. While RSA modular exponentiation is implemented by repeatedly applying modular multiplication, Elliptic Curve Cryptography (ECC) requires a more complex set of arithmetic operations over finite fields. It was shown in this paper that, in order to implement RNS-based ECC, these operations can be properly scheduled resulting in an efficient computation in each RNS channel under a given Montgomery domain. Since current public-key cryptographic algorithms, such as RSA and ECC, will not be able to provide

security if quantum computers become viable, the scientific community has been investigating alternatives. Research work carried out recently led to the conclusion that RNS also benefits post-quantum Lattice-based Cryptography (LBC).

Using RNS in cryptography was also approached in this paper from the perspective of improving security. Recent research work has shown that some features of RNS, mainly the representation through independent residue digits, whose processing order can be randomized, can be leveraged in order to mitigate side-channel attacks.

Experimental results presented in the literature, and reproduced in this paper, have shown that RNS enables the design of more efficient cryptographic systems and reinforces the prevention of side-channel attacks, improving their security. Another important conclusion from the most recent body of work is that the usage of RNS is relevant not only to design Application Specific Integrated Circuits (ASICs) but also to exploit data parallelism featured on general purpose multi-cores, Single Instruction Multiple Data (SIMD) devices and Graphical Processing Units (GPU). Last but not least, to expand the use of RNS in cryptography, tools and platforms that assist the design, test, and evaluation of cryptosystems are required. The Residue Number System Framework (CRNS) was described in this paper, but more are expected to appear in the future.

Acknowledgment

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013 and by the Ph.D. grant with reference SFRH/BD/103791/2014.



Leonel Sousa received a Ph.D. degree in Electrical and Computer Engineering from the Instituto Superior Técnico (IST), Universidade de Lisboa (UL), Lisbon, Portugal, in 1996, where he is currently Full Professor. He is also a Senior Researcher with the R&D Instituto de Engenharia de Sistemas e Computadores (INESC-ID). His research interests include VLSI architectures, parallel computing, computer arithmetic, and cryptography. He has contributed to more than 200 papers in journals and international conferences, for which he got several awards, including: DASIP'13 Best Paper Award, SAMOS'11 'Stamatis Vassiliadis' Best Paper Award, DASIP'10 Best Poster Award, and the Honorable Mention Award UTL/Santander Totta for the quality of the publications in 2009. He has contributed to the organization of several international conferences, namely as program chair and as

general and topic chair, and has given keynotes in some of them. He has edited four special issues of international journals, and he is currently Associate Editor of the IEEE Transactions on Multimedia, IEEE Transactions on Circuits and Systems for Video Technology, IEEE Access, IET Electronics Letters and Springer JRTIP, and Editor-in-Chief of the Eurasip JES. Co-editor of the book *Circuits and Systems for Security and Privacy*, CRC. He is Fellow of the IET, Distinguished Scientist of ACM and Senior Member of IEEE.



Samuel Antão earned his PhD in Electrical and Computer Engineering from the Instituto Superior Técnico (IST), Universidade de Lisboa (UL), Lisbon, Portugal, in 2013. He is currently with IBM Research where he designs next-generation compiler technologies for high-performance computing. His research interests include compiler optimization, ranging from parallelization to instruction scheduling, programming models and runtime libraries. He also has a keen interest in hardware description languages and high-level synthesis approaches targeting the design automation for several applications, including cryptography.



Paulo Martins received the MSc degree in Electrical and Computer Engineering from the Instituto Superior Técnico (IST), Universidade de Lisboa (UL), Lisbon, Portugal, in 2014. He is a Junior Researcher with the R&D Instituto de Engenharia de Sistemas e Computadores (INESC-ID). His research interests include computer architectures, parallel computing, computer arithmetic, and cryptography. He is also a PhD student of IST and a student member of both IEEE and HiPEAC.

References

- [1] H. L. Garner, "The residue number system," in *Proc. Western Joint Computer Conf., Ser. IRE-AIEE-ACM '59 (Western)*, 1959, pp. 146–153.
- [2] C. H. Chang, A. S. Molahosseini, A. A. E. Zarandi, and T. F. Tay, "Residue number system: A new paradigm to datapath optimization for low-power and high-performance digital signal processing applications," *IEEE Circuits Syst. Mag.*, 2016.
- [3] R. Lee, "Subword parallelism with MAX-2," *IEEE Micro.*, vol. 16, pp. 51–59, 1996.
- [4] F. Sheikh and L. Sousa, Eds., *Circuits and Systems for Security and Privacy, Devices, Circuits, and Systems Series*. Boca Raton, FL: CRC, 2015.
- [5] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.
- [6] N. Kobitz, "Elliptic curve cryptosystems," *Math. Comput.*, vol. 48, no. 177, pp. 203–209, 1985.
- [7] V. S. Miller, "Use of elliptic curves in cryptography," in *Advances in Cryptology—CRYPTO Proceedings Series. Lecture Notes in Computer Science*. New York: Springer, 1985, pp. 417–426.
- [8] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," in *Proc. 37th Annu. ACM Symp. Theory of Computing*, 2005, pp. 84–93.
- [9] O. Goldreich, S. Goldwasser, and S. Halevi, "Public-key cryptosystems from lattice reduction problems," in *Proc. 17th Annu. Int. Cryptology Conf. Advances in Cryptology*, 1997, pp. 112–131.
- [10] P. Nguyen, "Cryptanalysis of the Goldreich-Goldwasser-Halevi cryptosystem from crypto 97," in *Proc. CRYPTO, vol. 1666 LNCS*, 1999, pp. 288–304.
- [11] M. Yoshino and N. Kunihiro, "Improving GGH cryptosystem for large error vector," in *Proc. Int. Symp. Information Theory and Its Applications*, 2012, pp. 416–420.
- [12] C. F. de Barros and L. M. Schechter, "GGH may not be dead after all," in *Anais do XXXV Congresso Nacional de Matemática Aplicada e Computacional*, 2014.
- [13] GNU. (2012). The GNU multiple precision arithmetic library [Online]. Available: <http://gmplib.org/>
- [14] M. C. Ring. (2012). MAPM: A portable arbitrary precision math library in C [Online]. Available: <http://gmplib.org/>
- [15] S. Antao and L. Sousa, "An RNS-based architecture targeting hardware accelerators for modular arithmetic," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing*, 2013, pp. 2572–2576.
- [16] P. Martins and L. Sousa, "On the evaluation of multi-core systems with SIMD engines for public-key cryptography," in *Proc. Int. Symp. Computer Architecture and High Performance Computing Workshop*, Paris, 2014, pp. 48–53.
- [17] S. Antao, J. C. Bajard, and L. Sousa, "RNS-based elliptic curve point multiplication for massive parallel architectures," *Comput. J.*, vol. 55, no. 5, pp. 1–19, 2011.
- [18] M. Rose, T. Plantard, and W. Susilo, "Improving BDD cryptosystems in general lattices," in *Information Security Practice and Experience Series. Lecture Notes in Computer Science*, vol. 6672. New York: Springer, 2011, pp. 152–167.
- [19] J. C. Bajard, J. Eynard, N. Merkiche, and T. Plantard, "Babai round-off CVP method in RNS: Application to lattice based cryptographic protocols," in *Proc. 14th Int. Symp. Integrated Circuits*, 2014.
- [20] T. Plantard, M. Rose, and W. Susilo, "Improvement of lattice-based cryptography using CRT," in *Proc. QuantumComm'09*, 2009, pp. 275–282.
- [21] J. C. Bajard, J. Eynard, N. Merkiche, and T. Plantard, "RNS arithmetic approach in lattice-based cryptography: Accelerating the rounding-off procedure," in *Proc. IEEE 22nd Symp. Computer Arithmetic*, 2015, pp. 113–120.
- [22] P. Martins, L. Sousa, J. Eynard, and J. Bajard, "Programmable RNS lattice-based parallel cryptographic decryption," in *Proc. 26th IEEE Int. Conf. Application-Specific Systems, Architectures and Processors*, 2015, pp. 149–153.
- [23] J. A. Ambrose, H. Pettenghi, and L. Sousa, "DARNS: A randomized multi-modulo RNS architecture for double-and-add in ECC to prevent power analysis side channel attacks," in *Proc. 18th Asia and South Pacific Design Automation Conf.*, 2013, pp. 620–625.
- [24] G. Perin, L. Imbert, L. Torres, and P. Maurine, "Practical analysis of RSA countermeasures against side-channel electromagnetic attacks," in *Smart Card Research and Advanced Applications Series. Lecture Notes in Computer Science*, vol. 8419, A. Francillon and P. Rohatgi, Eds. New York: Springer, 2014, pp. 200–215.
- [25] D. Schinianakis and T. Stouraitis. "Residue number systems in cryptography: Design, challenges, robustness," in *Secure System Design and Trustable Computing*. Cham, Switzerland: Springer, 2016. pp. 115–161.
- [26] F. Oggier. (2014). Lecture notes in algebraic methods [Online]. Available: <http://www1.spms.ntu.edu.sg/~frederique/Teaching.html>
- [27] G. A. Jones and J. M. Jones, *Elementary Number Theory, Springer Undergraduate Mathematics Series*. New York: Springer, 1998.
- [28] X. Wang, G. Xu, M. Wang, and X. Meng, *Mathematical Foundations of Public Key Cryptography*. Boca Raton, FL: CRC Press, 2015.
- [29] *Digital Signature Standard*, NIST (DSS)-FIPS PUB 186-3, 2009, pp. 1–119.
- [30] P. L. Montgomery, "Modular multiplication without trial division," *Math. Comput.*, vol. 44, no. 170, pp. 519–521, 1985.
- [31] J. C. Bajard, L. S. Didier, and P. Kornerup, "Modular multiplication and base extensions in residue number systems," in *Proc. IEEE Symp. Computer Arithmetic*, Vail, CO, 2001, pp. 59–65.
- [32] A. P. Shenoy and R. Kumaresan, "Fast base extension using a redundant modulus in RNS," *IEEE Trans. Comput.*, vol. 38, no. 2, pp. 292–297, 1989.

- [33] S. Kawamura, M. Koike, F. Sano, and A. Shimbo, "Cox-Rower architecture for fast parallel montgomery multiplication," in *Lecture Notes in Computer Science: Advances in Cryptology-EUROCRYPT 2000*, B. Preneel, Ed. Heidelberg, Germany: Springer, 2000, pp. 523–538.
- [34] F. Rodríguez-Henríquez, A. D. Pérez, N. A. Saqib, and C. K. Koç, *Cryptographic Algorithms on Reconfigurable Hardware*. New York, NY: Springer Science+Business Media, 2006.
- [35] E. Brier and M. Joye, "Weierstrass elliptic curves and side-channel attacks," in *Lecture Notes in Computer Science: Public Key Cryptography*, D. Naccache and P. Paillier, Eds. Heidelberg, Germany: Springer, 2002, pp. 183–194.
- [36] P. Longa and C. Gebotys, "Analysis of efficient techniques for fast elliptic curve cryptography on x86-64 based processors," *IACR Cryptol ePrint Arch*, vol. 335, pp. 1–34, 2010.
- [37] D. Chudnovsky and G. Chudnovsky, "Sequences of numbers generated by addition in formal groups and new primality and factorization tests," *Adv. Appl. Math.*, vol. 7, no. 4, pp. 385–434, Dec. 1986.
- [38] S. Antao, R. Chaves, and L. Sousa, "Compact and flexible microcoded elliptic curve processor for reconfigurable devices," in *Proc. IEEE Symp. Field Programmable Custom Computing Machines*, Napa, CA, 2009, pp. 193–200.
- [39] P. L. Montgomery, "Speeding the Pollard and elliptic curve methods of factorization," *Math. Comput.*, vol. 48, no. 177, pp. 243–243, 1987.
- [40] K. Bigou and A. Tisserand, "Single base modular multiplication for efficient hardware rns implementations of ecc," in *Cryptographic Hardware and Embedded Systems-CHES 2015*. Heidelberg, Germany: Springer, 2015, pp. 123–140.
- [41] M. Rose, "Lattice-based cryptography: A practical implementation," M.S. thesis, School of Computer Science and Software Engineering, Univ Wollongong, Australia, 2011.
- [42] L. Babai, "On Lovász' lattice reduction and the nearest lattice point problem (shortened version)," in *Proc. 2nd Symp. Theoretical Aspects of Computer Science, Series STACS '85*, London, UK, 1985, pp. 13–20.
- [43] M. Rose, T. Plantard, and W. Susilo, "Improving BDD cryptosystems in general lattices," in *ISPEC, Series Lecture Notes in Computer Science*, vol. 6672. F. Bao and J. Weng, Eds. New York: Springer, 2011, pp. 152–167.
- [44] T. Plantard, W. Susilo, and K. T. Win, "A digital signature scheme based on CVP," in *Public Key Cryptography-PKC 2008*. Heidelberg, Germany: Springer, 2008, pp. 288–307.
- [45] J. C. Bajard and L. Imbert, "A full RNS implementation of RSA," *IEEE Trans. Comput.*, vol. 53, no. 6, pp. 769–774, June 2004.
- [46] J. Bajard, J. Eynard, N. Merkiche, and T. Plantard, "RNS arithmetic approach in lattice-based cryptography: Accelerating the rounding-off core procedure," in *22nd IEEE Symp. Computer Arithmetic*, Lyon, France, 2015, pp. 113–120.
- [47] ARM Architecture Group. "ARMv8 instruction set overview," ARM Limited, Tech. Rep., 2011.
- [48] Xilinx. (2016). FPGA leadership across multiple process nodes [Online]. Available: <http://www.xilinx.com/products/silicon-devices/fpga.html>
- [49] V. Shoup. (2014). NTL 6.6.1: A library for doing number theory [Online]. Available: www.shoup.net/ntl
- [50] S. Antao and L. Sousa, "The CRNS framework and its application to programmable and reconfigurable cryptography," *ACM Trans. Architect. Code Optim.*, vol. 9, no. 4, pp. 33:1–33:25, 2013.
- [51] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology-CRYPTO 99, Series Lecture Notes in Computer Science*, vol. 1666. New York: Springer, 1999, pp. 388–397.
- [52] F. Koeune and F. X. Standaert, "A tutorial on physical security and side-channel attacks," in *Foundations of Security Analysis and Design III: FOSAD 2004/2005 Tutorial Lectures*. Heidelberg, Germany: Springer, 2005, pp. 78–108.
- [53] T. Messerges, E. Dabbish, and R. Sloan, "Power analysis attacks of modular exponentiation in smartcards," in *Cryptographic Hardware and Embedded Systems, Series Lecture Notes in Computer Science*, vol. 1717. Heidelberg, Germany: Springer, 1999, pp. 144–157.
- [54] J. Fan, X. Guo, E. De Mulder, P. Schaumont, B. Preneel, and I. Verbauwhede, "State-of-the-art of secure ECC implementations: A survey on known side-channel attacks and countermeasures," in *Proc. IEEE Int. Symp. Hardware-Oriented Security and Trust*, 2010, pp. 76–87.
- [55] J. S. Coron, "Resistance against differential power analysis for elliptic curve cryptosystems," in *Cryptographic Hardware and Embedded Systems, Series Lecture Notes in Computer Science*, vol. 1717. Heidelberg, Germany: Springer, 1999, pp. 292–302.
- [56] M. F. Witteman, J. G. J. Woudenberg, and F. Menarini, "Defeating RSA multiply-always and message blinding countermeasures," *Topics in Cryptology-CT-RSA 2011*. Heidelberg, Germany: Springer, 2011, pp. 77–88.
- [57] C. Walter, "Sliding windows succumbs to big mac attack," in *Cryptographic Hardware and Embedded Systems-CHES 2001, Series Lecture Notes in Computer Science*, vol. 2162. Heidelberg, Germany: Springer, 2001, pp. 286–299.
- [58] J. C. Bajard, L. Imbert, P. Y. Liardet, and Y. Teglia, "Leak resistant arithmetic," in *Cryptographic Hardware and Embedded Systems-CHES 2004, Series Lecture Notes in Computer Science*, vol. 3156. Heidelberg, Germany: Springer, 2004, pp. 62–75.
- [59] D. Mesquita, B. Badrignan, L. Torres, G. Sassattell, M. Robert, and J. Bajard, F. Moraes, "A leak resistant architecture against side channel attacks," in *Proc. Int. Conf. Field Programmable Logic and Applications*, 2006, pp. 1–4.
- [60] M. Joye and J. J. Quisquater, "Hessian elliptic curves and side-channel attacks," in *Cryptographic Hardware and Embedded Systems, Series Lecture Notes in Computer Science*, vol. 2162. Heidelberg, Germany: Springer, 2001, pp. 402–410.
- [61] E. Brier and M. Joye, "Weierstrass elliptic curves and side-channel attacks," in *Public Key Cryptography, Series Lecture Notes in Computer Science*, vol. 2274. Heidelberg, Germany: Springer, 2002, pp. 335–345.
- [62] P. L. Montgomery, "Speeding the pollard and elliptic curve methods of factorization," *Math. Comput.*, vol. 48, no. 177, pp. 243–264, 1987.
- [63] J. Bajard, S. Duquesne, and M. D. Ercegovac, "Combining leak-resistant arithmetic for elliptic curves defined over F_p and RNS representation," *IACR Cryptology ePrint Archive*, 2010.
- [64] J. Ambrose, H. Pettenghi, and L. Sousa, "DARNS: a randomized multi-modulo RNS architecture for double-and-add in ECC to prevent power analysis side channel attacks," in *Proc. 18th Asia and South Pacific Design Automation Conf.*, 2013, pp. 620–625.
- [65] J. Chu and M. Benaissa, "Error detecting AES using polynomial residue number systems," *Microprocessors Microsyst.*, vol. 37, no. 2, pp. 228–234, 2013. Digital System Safety and Security. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0141933112000853>
- [66] D. Boneh and M. K. Franklin, "Identity-based encryption from the weil pairing," in *Proc. 21st Annu. Int. Cryptology Conf. Advances in Cryptology*, 2001, pp. 213–229.
- [67] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proc. 2007 IEEE Symp. Security and Privacy*, Washington, DC, 2007, series SP '07, pp. 321–334.
- [68] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *Proc. 7th Int. Conf. Theory and Application of Cryptology and Information Security: Advances in Cryptology*, 2001, Series ASIACRYPT '01, pp. 514–532.
- [69] R. C. C. Cheung, S. Duquesne, J. Fan, N. Guillermine, I. Verbauwhede, and G. X. Yao, "FPGA implementation of pairings using residue number system and lazy reduction," in *Proc. 13th Int. Workshop Cryptographic Hardware and Embedded Systems*, Nara, Japan, 2011, pp. 421–441.
- [70] S. Duquesne and N. Guillermine. (2011). A FPGA pairing implementation using the residue number system. Cryptology ePrint Archive, Report 2011/176 [Online]. Available: <http://eprint.iacr.org/>
- [71] J. Kim, M. S. Lee, A. Yun, and J. H. Cheon. (2013). CRT-based fully homomorphic encryption over the integers. Cryptology ePrint Archive, Report 2013/057 [Online]. Available: <http://eprint.iacr.org/>
- [72] C. Gentry, S. Halevi, and N. P. Smart. (2012). Homomorphic evaluation of the aes circuit. Cryptology ePrint Archive, Report 2012/099 [Online]. Available: <http://eprint.iacr.org/>
- [73] J. C. Bajard, J. Eynard, A. Hasan, and V. Zucca. (2016). A full rns variant of fv like somewhat homomorphic encryption schemes. Cryptology ePrint Archive, Report 2016/510 [Online]. Available: <http://eprint.iacr.org/2016/510>