AN ABSTRACT OF THE THESIS OF

Gökay Saldamlı for the degree of Doctor of Philosophy in

Electrical and Computer Engineering presented on May 23, 2005.

Title: Spectral Modular Arithmetic

Abstract approved: Redacted for Privacy Redacted for Privacy

In many areas of engineering and applied mathematics, spectral methods provide very powerful tools for solving and analyzing problems. For instance, large to extremely large sizes of numbers can efficiently be multiplied by using discrete Fourier transform and convolution property. Such computations are needed when computing π to millions of digits of precision, factoring and also big prime search projects.

When it comes to the utilization of spectral techniques for modular operations in public key cryptosystems two difficulties arise; the first one is the reduction needed after the multiplication step and the second is the cryptographic sizes which are much shorter than the optimal asymptotic crossovers of spectral methods.

In this dissertation, a new modular reduction technique is proposed. Moreover, modular multiplication is given based on this reduction. These methods work fully in the frequency domain with some exceptions such as the initial, final and partial transformations steps. Fortunately, the new technique addresses the reduction problem however, because of the extra complexity coming from the overhead of the forward and backward transformation computations, the second goal is not easily achieved when single operations such as modular multiplication or reduction are considered. On the

contrary, if operations that need several modular multiplications with respect to the same modulus are considered, this goal is more tractable.

An obvious example of such an operation is the modular exponentiation i.e., the computation of $c = m^e \bmod n$ where c, m, e, n are large integers. Therefore following the spectral modular multiplication operation a new modular exponentiation method is presented. Since forward and backward transformation calculations do not need to be performed for every multiplication carried during the exponentiation, the asymptotic crossover for modular exponentiation is decreased to cryptographic sizes. The method yields an efficient and highly parallel architecture for hardware implementations of public-key cryptosystems.

Spectral Modular Arithmetic

by

Gökay Saldamlı

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented May 23, 2005
Commencement June 2006

Doctor of Philosophy thesis of Gökay Saldamlı presented on May 23, 2005

APPROVED:

Redacted for Privacy

Co-Major Professor, representing Electrical and Computer Engineering

Redacted for Privacy

Co-Major Professor, representing Electrical and Computer Engineering

Redacted for Privacy

Associate Director of the School of Electrical Engineering and Computer Science

Redacted for Privacy

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Redacted for privacy

Gökay Saldamlı, Author

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

Page

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

Table Page

1. INTRODUCTION

1.1. Background and Motivation

In the near future, security will be an innate feature of virtually every piece of digital equipment. The performance metrics of space, time and power dissipation of cryptographic algorithms will always remain amongst the most critical aspects in the implementation of cryptographic services.

A major class of cryptographic algorithms comprises public-key schemes. These enable the verification of message integrity and authentication check, key distribution, and digital signature functions, among others. These schemes have become indispensable parts of present day secure communication. All public key cryptosystems require resource intensive arithmetic operations in certain mathematical structures such as finite fields, groups and rings. The most important operations in these structures, *modular multiplication, inversion, and exponentiation*, are at the center of very intensive research activities since they are usually the most resource-consuming operations in the many public key cryptosystems in use today.

In fact, Spectral techniques for integer multiplication have been known for over a quarter of a century. By using the spectral integer multiplication of Schönhage and Strassen [1], large to extremely large sizes of numbers can efficiently be multiplied. Such computations are needed when computing π to millions of digits of precision, factoring and also big prime search projects.

A naive way of utilizing the spectral techniques for modular multiplication is first computing the multiplication by using possibly Schönhage and Strassen [1] and then performing the reduction in the time domain. Such an approach is preferable if the input length is large enough to meet the asymptotic crossovers (somewhere in between 4 to 30 thousands bits) of Schönhage-Strassen – assuming the reduction has a constant cost –. But these figures are larger than the key sizes of most cryptosystems; thus, in practice the naive way is never used.

Moreover, a modular exponentiation realization with the naive method needs very expensive forward and backward transformation computations for every modular multiplication delivered. Thus the asymptotic crossover of modular exponentiation becomes the same as a single modular multiplication.

In this dissertation, new techniques of performing modular multiplication and exponentiation based on a new reduction operation are proposed (Chapter 3). These methods work fully in the frequency domain (spectrum) with some exceptions such as the initial, final and partial transformations steps. In other words, this work studies performing modular arithmetic operations with Fourier coefficients instead of the time sequences. Such an action is extremely effective for operations involving several modular multiplications because the convolution is readily available with component-wise multiplication when working with Fourier coefficients.

Obviously, the modular exponentiation has such a nature, it needs several modular multiplications with respect to the same modulus. Defining a reduction method working in the spectrum keeps the data in the spectrum and decreases the forward and backward transformation load which directly reduces the asymptotic crossover of the exponentiation to the cryptographic sizes.

1.2. Outline

In Chapter 2, after defining the convolutions and Discrete Fourier Transform (DFT) we give the basic properties of DFT (Section 2.1.3), the major bridge between time and frequency domain. Historically, the DFT is defined and found most of its applications over the complex numbers. Operating in the spectrum of the field of complex numbers causes some round-off errors unless an infinite precision for representing the complex numbers is used. In general, for simple computations these errors are controlled by increasing the precision used however, if one needs to perform massive computations in the spectrum —as in our case—, this is not practical.

Transforms analogous to DFT over real or complex numbers that admit no round-off errors are defined over some similar algebraic structures. A good candidate for such structures, proposed by Pollard [2], is the ring $\mathbb{Z}_q$ (i.e. the ring of integers with multiplication and addition modulo a positive integer q). Since the binary operations in these structures are modular multiplication and addition, the arithmetic is always exact. The DFT defined over $\mathbb{Z}_q$ is called the Number Theoretical Transforms (NTT). In Section 2.2 we give a formal definition of the NTT.

For computational purposes, rings for which q is of the form $2^v \pm 1$ are mostly desirable since the modular reduction operations for such q are simplified. The rings of the form $2^v - 1$ are called the *Mersenne rings*, while the rings of the form $2^v + 1$ are called the *Fermat rings*. Again in Section 2.2, we analyze the NTTs over these rings, which are also called the Mersenne number transform (MNT) and the Fermat number transform (FNT). Moreover, in Tables 2.1 and 2.2 we tabulated parameters of some nice MNT and FNT that are practically useful for our needs.

The Mersenne and Fermat rings are not the only suitable rings for efficient arithmetic. If p (not necessarily a prime) is a small divisor of q, the rings of the form

$Z_{q/p}$ are also quite useful. Since p divides q, the arithmetic modulo (q/p) is carried in the ring Z_q and by selecting Z_q as a Mersenne or Fermat ring, it is posssible to simplify the arithmetic. In Section 2.3, we present such rings which are also called *pseudo Mersenne* or *pseudo Fermat rings*. Additionally, in APPENDIX A, Tables 6.1 and 6.2 tabulate some suitable pseudo Mersenne and Fermat rings.

We note that most of the material of Chapter 2 is adapted from [2], [3], and [4]. Since we tailored the subject to our purposes, our presentation is simpler, in particular, we give a very clear explanation of the existence of pseudo transforms by Theorem 1.

Chapter 3 describes our main idea of spectral modular arithmetic. Primarily, we propose to do modular arithmetic in the frequency domain (spectrum) rather than in time domain. In general, if an operation is performed in the spectrum, the spectral coefficients do not reveal much about what has been done in time. That is why we construct the spectral modular algorithms as the image of some time algorithms that are easily be transformed to spectrum by using the properties of the DFT. By this we keep track of the activity of time sequences while working in the spectrum.

After covering some preliminaries in Sections 3.1 through 3.3, we define *spectral modular reduction* (SMR) in Section 3.4. In Section 3.5 we give a time simulation (Algorithm 4) that is translated into the complex spectrum in Section 3.6 and into a finite ring spectrum in Section 3.7 by using the linearity and shifting property of DFT. We present *Spectral Modular Product* (SMP) (Algorithm 8), which consists of convolution and spectral reduction, as the basic building block of both *spectral modular multiplication* (SMM) and *spectral modular exponentiation* (SME). Finally, we describe SMM and SME in Sections 3.8 and 3.9 respectively.

In Chapter 3 along with the presentation of algorithms, we find the minimal domains (i.e. smallest rings) in which our spectral algorithms work. Although working with a finite ring spectrum is free from round-off errors, if some parameters in time

simulations go out of predefined bounds the spectral algorithms produce erroneous outputs. These ill-favored effects are called overflows, which can be avoided by choosing larger q. By Theorem 12 we characterize the minimal q such that SME (with an SMP core) works without overflows. Lastly, we close the chapter with an illustrative example in Section 3.10.

In Chapter 4, we describe methodologies for selecting the parameters for SME in order to apply the algorithm to public key cryptography. Firstly, in Section 4.1 by using the bound (3.11), we demonstrate some convenient parameters for realizations in Table 4.2 for SME with SMP core. Secondly, in Section 4.2 with Algorithm 11 (i.e. modified spectral modular product (MSMP) algorithm), we modify SMP and improve the bound (3.11) by (4.3) at a cost of some extra memory. The improved parameter selection is tabulated in Table 4.3.

Finally and more importantly, in Section 4.3 we describe and discuss the use of the Chinese Remainder Theorem (CRT) for SME and SMM. CRT admits us to reach very large modulus sizes while at the same time work in small size rings. We tabulated possible implementation parameters in Tables 4.5 and 4.6 for both SMP and MSMP respectively.

In Chapter 5, we turn our attention to the architectural and performance analysis. The spectral algorithms yield efficient and highly parallel architectures for hardware implementations of public-key cryptosystems. In our analysis, we use some generic and specific architectures that lead us to have a general and a lower level unit-gate treatment. In Section 5.1 we briefly describe specified low-level architectures for Fermat and Mersenne ring arithmetic adapted from [5], [6], [7], [8], [9], and [10].

Since the initial and final transformation steps of SME algorithm are straightforward and computationally negligible, in Section 5.2 we describe detailed architectures along with performance analysis of individual steps of SMP and MSMP. Lastly, in

Section 5.3 we give the entire complexity by Tables 5.9 and 5.10 after combining the individual pieces described.

We conclude our work with some final comments in Chapter 6.

2. CONVOLUTION AND DFT

2.1. Convolution and DFT: the basic definitions and properties

We would like to begin with the basic notions of the cyclic convolution and discrete Fourier transform and their interrelationship.

2.1.1. Convolutions

A discrete-time signal $\{x_m\}$ or $x[m]$ is a sequence of real or complex numbers. By a sequence we mean a function that assigns to each natural number a unique real or complex number.

$$x : \mathbb{N} \to \mathbb{C} \text{ or } (\mathbb{R})$$

$$m \mapsto x_m$$

If the range of the discrete-time signal is a finite subset of real or complex numbers, the signal is called a digital signal, e.g. this is the case when representing numbers in a digital computer. Likewise, if the domain of a sequence is a finite subset of natural numbers, the signal is called a finite-length discrete-time signal. Typically, the finite domain is chosen to be the first d non-negative integers $\{m \in \mathbb{N} : 0 \leqslant m \leqslant d\}$ and the discrete-time signal is defined to be identically zero outside of this interval. Due to obvious practical reasons, we shall concentrate on finite-length discrete-time signals.

Definition 1 *Let* $\{x_m\} = (x_0, x_1, \ldots, x_{d-1})$ *and* $\{y_m\} = (y_0, y_1, \ldots, y_{d-1})$ *be two finite-length sequences. The* **cyclic convolution** *of* $\{x_m\}$ *and* $\{y_m\}$*, denoted by* $\{z_m\} = \{x_m\} * \{y_m\}$*, is a sequence of* d *elements (i.e. length* d*), defined by*

$$z_i := \sum_{k=0}^{d-1} x_{((i-k))} y_k, \qquad i = 0, 1, \ldots, d-1$$

where the double parenthesis denote modulo d.

An alternative and more compact way of representing cyclic convolutions is by making use of formal polynomials. In polynomial notation, calculating the terms of the cyclic convolution —also called the **convolution products**— is equivalent to computing the coefficients of the polynomial

$$z(t) = x(t)y(t) \bmod (t^d - 1) \tag{2.1}$$

where $x(t) = x_0 + x_1 t + \ldots + x_{d-1}t^{d-1}$ and $y(t) = y_0 + y_1 t + \ldots + y_{d-1}t^{d-1}$. If

$$\deg(x(t)) + \deg(y(t)) < d \tag{2.2}$$

one would not need a $t^d - 1$ reduction. In this case Equation (2.1) defines a linear convolution of two sequences which is the most important computation for our purposes. In general the linear convolution is introduced simply by $z(t) = x(t)y(t)$. In order to exhibit the connection between cyclic and linear convolution, we prefer to define the linear convolution as a cyclic convolution satisfying Equation (2.2). Indeed this connection describes a powerful method of linear convolution computation since cyclic convolutions can efficiently be computed by using the fast Fourier transform and the convolution property.

In many areas of engineering and applied mathematics, the effective computation of linear convolutions is essential. Integer multiplication can be given as an example; if the coefficients of $x(t) = x_0+x_1t+\ldots+x_{d-1}t^{d-1}$ and $y(t) = y_0+y_1t+\ldots+y_{d-1}t^{d-1}$ are the base 2^b representations of the integers x and y (i.e. $x = x(2^b)$ and $y = y(2^b)$), the computation of the linear convolution $z(t) = x(t)y(t)$ gives to the integer product $z = xy$ after the evaluation $z = z(2^b)$.

The direct computation of linear convolutions of two d-point sequences requires $O(d^2)$ operations. With an appropriate usage of the fast Fourier transform and the

convolution theorem it is possible to decrease this complexity drastically. We refer the reader to textbook presentations [3] and [4] for the computational aspects. In the next section, we briefly introduce the discrete Fourier transform and related notions.

2.1.2. Discrete Fourier Transform

Definition 2 *Let R be an arbitrary commutative ring. An element $\omega \neq 1$ of R is called a* **principal dth root of unity** *if $\omega^n = 1$ and*

$$\sum_{m=0}^{d-1} \omega^{mj} = 0 \quad \text{for } 1 \leqslant j < d \ .$$

*Moreover, the elements $\omega^0, \omega^1, \ldots, \omega^{d-1}$ are called the d***th roots of unity**.

For example; $e^{2\pi i/d}$, where $i = \sqrt{-1}$ is a principal dth root of unity in the ring of complex numbers.

Definition 3 *Let $x[m]$ be a length-d discrete-time signal of complex numbers. The* **Fourier transform** *of signal $x[m]$ is defined as a length $d' > 0$ signal $X[k]$ where*

$$X[k] := \sum_{m=0}^{d-1} x[m] e^{-2\pi i m/d'}, \qquad k = 0, 1, \ldots, d'-1; i = \sqrt{-1}$$

The signal $X[k]$ is called the **spectrum** *of the time signal $x[m]$, and the coefficients of $X[k]$ are called the* **spectrum coefficients**. *Moreover, we say that $X[k]$ and $x[m]$ are* **transform pair** *and denote this relation by*

$$X[k] \quad \xleftrightarrow{DFT} \quad x[m]$$

Observe that like as the time signal $x[m]$, the spectrum $X[k]$ is also a sequence of a finite length. If we restrict our attention to the special case in which both signals have the same length (i.e. $d = d'$) we obtain the d-point Discrete Fourier Transform (DFT).

Definition 4 *Let $x[m]$ be a length-d discrete-time signal of complex numbers. The d-point (or length-d)* **Discrete Fourier transform** *of signal $x[m]$ is defined by*

$$X[k] = DFT_d(x[m]) := \sum_{m=0}^{d-1} x[m]\omega_d^{mk}$$

where $k = 0, 1, \ldots, d-1$ and $\omega_d = e^{-2\pi i/d}$ (complex principal dth root of unity).

The inverse transform is shown to be

$$x[m] = IDFT_d(X[m]) := d^{-1} \cdot \sum_{m=0}^{d-1} X[k]\omega_d^{-mk}$$

Under some conditions, the Fourier transform preserves some properties of the time sequences. Linearity, convolution and time-frequency shifting are some of these invariants which are essential for a better understanding of the nature of the transform. Moreover, because of these properties the Fourier transform becomes a powerful tool for applied sciences.

2.1.3. Time-frequency dictionary

Most of the time, if some operation is performed in the frequency domain, the spectrum coefficients do not tell much about what has been done in the time domain. That is why we construct the spectral modular algorithms as the image of some time simulations that are easily translated to spectrum. By this we keep track of the activity of time sequences while working in the spectrum.

This translation of algorithms is done by using a dictionary of operations. We like to approach these operations from a computer arithmetic point of view.

Let $(\{x_m\}, \{X_k\})$ and $(\{y_m\}, \{Y_k\})$ be two transform pairs. If $\lambda \in \mathbb{C}$ then the DFT has the following properties.

(i) Linearity. From a computer architect's point of view, this property corresponds to additions and subtractions. Literally, addition and subtraction in the time domain correspond to addition and subtraction in the frequency domain,

$$\{x_m\} \pm \{y_m\} \quad \xleftrightarrow{DFT} \quad \{X_k\} \pm \{Y_k\}$$

$$\lambda \cdot \{x_m\} = \{\lambda x_m\} \quad \xleftrightarrow{DFT} \quad \lambda \cdot \{X_k\} = \{\lambda X_k\}$$

(ii) Convolution The main reason of our interest in the Fourier transform is the following convolution property. Let $\odot$ denote the component-wise multiplication of sequences then

$$\{x_m\} * \{y_m\} \quad \xleftrightarrow{DFT} \quad \{X_k\} \odot \{Y_k\}$$

$$\{x_m\} \odot \{y_m\} \quad \xleftrightarrow{DFT} \quad \{X_k\} * \{Y_k\}$$

Because of this property, the costly convolutions of the sequences $\{x_m\}$ and $\{y_m\}$ is calculated by transforming the sequences to $\{X_k\}$ and $\{Y_k\}$, doing a term-by-term multiplication and obtaining the result by an inverse transformation. On the other hand, convolutions in the frequency domain correspond to term-by-term multiplications in the time domain.

Notation 1 *Asumme that ω is a principal d-th root of unity; we let $\{\Gamma_k\}$ and $\{\Omega_k\}$ denote the negative and positive power sequence of ω as*

$$\{\Omega_k\} = (1, \omega^1, \omega^2, \ldots, \omega^{(d-1)})$$
$$\{\Gamma_k\} = (1, \omega^{-1}, \omega^{-2}, \ldots, \omega^{-(d-1)})$$

(iii) Time and frequency shifts. Time and frequency shifts correspond to circular shifts when working with finite-length signals. Let $\{x_m\} = (x_0, x_1, \ldots, x_{d-1})$

and $\{X_k\} = (X_0, X_1, \ldots, X_{d-1})$ be transform pair. The *one-term right circular shift* is defined as

$$\{x_m\} \circlearrowleft 1 := (x_{d-1}, x_0, \ldots, x_{d-2}) \quad \xleftrightarrow{DFT} \quad \{X_k\} \odot \{\Omega_k\}$$

The *one-term left circular shift* is similar, where multiplication of the coefficients with negative power sequence of the principal d-th root of unity gives the left circular shift:

$$\{x_m\} \circlearrowleft 1 := (x_1, x_2, \ldots, x_{d-1}, x_0) \quad \xleftrightarrow{DFT} \quad \{X_m\} \odot \{\Gamma_m\}$$

An arbitrary circular shift can be computed by applying the consecutive one-term shifts or by using a proper ω power sequence, for instance; an s $(0 \leqslant s \leqslant d-1)$ left shift is achieved by

$$\{x_m\} \circlearrowleft s := (x_s, x_{s+1}, \ldots, x_{d-1}, x_0, \ldots, x_{s-1}) \quad \xleftrightarrow{DFT} \quad \{X_k\} \odot \{\Gamma_k^s\}$$

where $\{\Gamma_k^s\} = (1, \omega^{-s}, \omega^{-st}, \ldots, \omega^{-s(d-1)})$.

(iv) Sum of sequence and first value. The sum of a sequence in the time domain is equal to the first value of its DFT; conversely the sum of the spectrum coefficients equals d^{-1} times the first value of the time sequence (Figure 2.1).

$$x_0 = d^{-1} \cdot \sum_{i=0}^{d-1} X_i \omega^{-i} \qquad \text{and} \qquad X_0 = \sum_{i=0}^{d-1} x_i \omega^i$$

(v) Left and right shifts. By using properties (iii) and (iv), it is possible to achieve the regular left and right shifts. We begin with a one-term left shift operation (this corresponds to division by the radix r if we assume that the terms of x are in radix r and x_0 is the least significant digit). Let $\{x_0\} = (x_0, x_0, \ldots, x_0)$ represent the constant sequence of length-d then

$$\{x_m\} \ll 1 = (x_1, x_2, \ldots, x_{d-1}, 0) \quad \xleftrightarrow{DFT} \quad (\{X_k\} - \{x_0\}) \odot \{\Gamma_k\}$$

The right shifts are similar, where one then uses the $\{\Omega_k\}$ sequence instead of $\{\Gamma_k\}$.

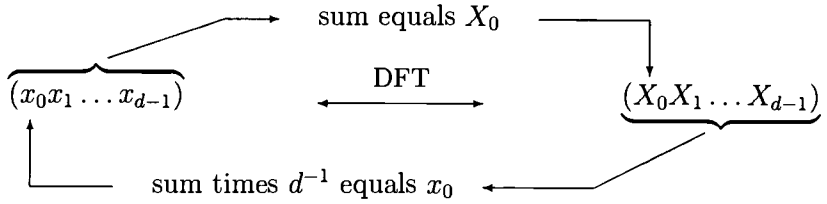

FIGURE 2.1. Sum of sequence and first value.

2.2. Number Theoretical Transforms

Operating in the spectrum of the field of complex numbers causes some round-off errors unless an infinite precision for representing the complex numbers is used. In general, for simple computations these errors are controlled by increasing the precision used; however, if one needs to perform massive computations in the spectrum — as in our case — this is not practical.

Transforms analogous to DFT over real or complex numbers that admit no round-off errors are defined over some similar algebraic structures. A good candidate for such structures, proposed by Pollard [2], is the ring $\mathbb{Z}_q$ (i.e. the ring of integers with multiplication and addition modulo a positive integer q). Since the binary operations in these structures are modular multiplication and addition, the arithmetic is always exact.

Additionally, from a computational point of view, these transforms exploit the special arithmetic of the underlying rings; hence, significant computational savings are possible. For instance; rings for which q is of the form $2^v \pm 1$ are mostly desirable since the modular reduction operations for such q are simplified. Furthermore, if the

principal root of unity is chosen as a power of 2, spectrum coefficients are computed only using additions and circular shifts.

We begin with formal definitions of transforms over finite rings:

Definition 5 *Let ω be a principal d-th root of unity in $\mathbb{Z}_q$ and $\{x_m\}$ be a length-d sequence of elements of $\mathbb{Z}_q$. Suppose that the sequence $\{X_k\}$ is defined as,*

$$X_i := \sum_{j=0}^{d-1} x_j \, \omega^{ij} \mod q, \qquad i = 0, 1, \ldots, d-1 \tag{2.3}$$

then the transform that sends $\{x_i\}$ to $\{X_i\}$ is called the **d-point (or length-d) Number Theoretical Transform (NTT)**. *Moreover, if q is a Mersenne number of the form $2^r - 1$, NTT is called the* **Mersenne Number Transform (MNT)** *whereas if q is a Fermat number of the form $2^r + 1$, transform is named as a* **Fermat Number Transform (FNT)**.

In a complex spectrum setting, for every $d > 0$ there exists a d-point DFT because a complex principal d-th root of unity and the inverse transform always exist; however, in a finite ring spectrum the existence of inverse transform and principal root of unity depend on some conditions. Therefore, a NTT exists if these conditions are satisfied.

Proposition 1 *The inverse transform*

$$x_i := d^{-1} \cdot \sum_{j=0}^{d-1} X_j \, \omega^{-ij} \mod q, \qquad i = 0, 1, \ldots, d-1 \tag{2.4}$$

exists if d is invertible, i.e. $d \cdot d' = 1$ mod q for some $d' \in \mathbb{Z}_q$

Proof: Clearly (2.4) gives the inverse transform. Since ω is a principal d root of unity, negative powers exist. Hence, the sum is well defined if the inverse of d exists in $\mathbb{Z}_q$. ■

Corollary 1 *If q is prime in (2.4) then the inverse transform exists.*

Proof: This is the case where $\mathbb{Z}_q$ is a field, which means that every non-zero element has a multiplicative inverse. ■

Proposition 2 *There exist a d-point NTT over the ring $\mathbb{Z}_q$ if and only if d divides $p-1$ for every prime p divides q. Moreover the greatest common divisor of the set $\{p-1 : p \text{ dividing } q\}$ gives the maximum NTT length that can be defined over $\mathbb{Z}_q$.*

Proof: see Theorem 6.1.1 in [3]. ■

Example 1 *Consider the Fermat ring $\mathbb{Z}_q$ with $q = 2^{32}+1$. Since*

$$2^{32}+1 = 641 \cdot 6700417$$

by Proposition 2 the maximum transform length equals $gcd(640, 6700416) = 128$ and NTTs with length powers of two up to 2^7 are possible.

Corollary 2 *If q is a prime then for every factor d of $q-1$, there exists a d-point NTT over $\mathbb{Z}_q$.*

One should choose the underlying ring and the principal root of unity with care. These two design criteria directly affect the transform lengths as well as the complexity of the computations. In order to have the maximum simplicity, the obvious selection for the principal root of unity is $\omega = \pm 2$ in $\mathbb{Z}_q$; multiplications with unity then correspond to circular shifts. As Proposition 2 characterizes the relation between the underlying rings and the transform lengths, the rings that admit longer transform lengths for $\omega = \pm 2$ should to be used; however, the complexity of the arithmetic has to be considered seriously.

Ruling out the trivial choice $q = 2^r$ with the maximum NTT length 2, the simplest choice is a Mersenne ring with $q = 2^r - 1$. To demonstrate the arithmetic in

a Mersenne ring suppose $x = x_0 + 2x_1 + 4x_2 + \ldots + 2^r x_r$, $x_i \in \{0,1\}$ and let's compute x^2 in $\mathbb{Z}_{2^r-1}$; for some $y_i \in \{0,1\}$, $i = 0,1,2,\ldots,2r$, one writes x^2 as

$$\begin{aligned} x^2 &= (x_0 + 2x_1 + 4x_2 + \ldots + 2^r x_r)^2 \\ &= y_0 + 2y_1 + 4y_2 + \ldots + 2^r y_r + 2^{r+1} y_{r+1} + 2^{r+2} y_{r+2} + \ldots + 2^{2r} y_{2r} \\ &= y_0 + 2y_1 + 4y_2 + \ldots + 2^0 y_r + 2^1 y_{r+1} + 2^2 y_{r+2} + \ldots + 2^r y_{2r} \ (\text{ since } 2^r = 1) \\ &= (y_0 + y_{r+1}) + 2(y_1 + y_{r+2}) + 4(y_2 + y_{r+3}) + \ldots + 2^r (y_r + y_{2r}) \end{aligned}$$

This corresponds to the well known one's complement arithmetic which is fairly simple. Furthermore, Mersenne rings admit reasonable transform lengths once the principal root of unity is chosen to be $\omega = \pm 2$.

Proposition 3 *If q is a Mersenne prime of the form $2^r - 1$ then there exist MNT of length r and $2r$ whose principal roots of unity are $\omega = 2$ and $\omega = -2$ respectively.*

Proof: Firstly, $2^r - 2 = 0 \bmod r$ implies that r divides $q - 1$; since $\omega = 2$ has clearly order r, we have the r-point MNT. Similarly, in case of $q - 1$ is even, $2r$ divides $q - 1$. Since $w = -2$ has order $2r$, we have the $2r$-point MNT. ■

ring $\mathbb{Z}_q$	prime factors	ω	MNT length	ω	MNT length
$2^{17} - 1$	131071	2	17	-2	34
$2^{19} - 1$	524287	2	19	-2	38
$2^{23} - 1$	$47 \cdot 178481$	2	23	-2	46
$2^{29} - 1$	$233 \cdot 1103 \cdot 2089$	2	29	-2	58
$2^{31} - 1$	2147483647	2	31	-2	62
$2^{37} - 1$	$223 \cdot 616318177$	2	37	-2	74
$2^{41} - 1$	$13367 \cdot 164511353$	2	41	-2	82

$2^{43}-1$	$431 \cdot 9719 \cdot 2099863$	2	43	-2	86
$2^{47}-1$	$2351 \cdot 4513 \cdot 13264529$	2	47	-2	94
$2^{53}-1$	$6361 \cdot 69431 \cdot 20394401$	2	53	-2	106
$2^{59}-1$	$179951 \cdot 3203431780337$	2	59	-2	118
$2^{61}-1$	2305843009213693951	2	61	-2	122
$2^{67}-1$	$193707721 \cdot 761838257287$	2	67	-2	134
$2^{71}-1$	$228479 \cdot 48544121 \cdot 212885833$	2	71	-2	142
$2^{73}-1$	$439 \cdot 2298041 \cdot 9361973132609$	2	73	-2	146
$2^{79}-1$	$2687 \cdot 202029703 \cdot 1113491139767$	2	79	-2	158
$2^{83}-1$	$167 \cdot 57912614113275649087721$	2	83	-2	166
$2^{89}-1$	618970019642690137449562111	2	89	-2	178
$2^{97}-1$	$11447 \cdot 13842607235828485645766393$	2	97	-2	196
$2^{101}-1$	$7432339208719 \cdot 341117531003194129$	2	101	-2	202
$2^{103}-1$	$2550183799 \cdot 3976656429941438590393$	2	103	-2	206
$2^{107}-1$	162259276829213363391578010288127	2	107	-2	214
$2^{109}-1$	$745988807 \cdot 870035986098720987332873$	2	109	-2	218
$2^{113}-1$	$3391 \cdot 23279 \cdot 65993 \cdot 1868569 \cdot 1066818132868207$	2	113	-2	226
$2^{127}-1$	170141183460469231731687303715884105727	2	127	-2	254

TABLE 2.1.: Parameters of MNT for $2^{16} < q < 2^{128}$

In Table 2.1 we show some nice MNT parameters for $2^{16} < q < 2^{128}$. The table can easily be extended for larger rings by using the Proposition 2 and 3. Observe especially that, the cases which q is not a prime are interesting. For instance; $q = 2^{71}-1$ is not a prime so do not satisfy the conditions of Proposition 3; however, the ring still admits r-point and $2r$-point MNTs by using Proposition 2.

Similarly constructions of FNT are possible for Fermat rings as the following proposition states. But first, we like to note that the arithmetic in Fermat rings is as simple as the one in Mersenne rings, see Section 5.1 for further details.

Proposition 4 *If q is a Fermat number of the form $2^{2^r}+1$ then there exist an FNT of length $2 \cdot 2^r$ and $4 \cdot 2^r$ with the principal roots of unity $\omega = 2$ and $\omega = \sqrt{2}$ respectively.*

Proof: See Section §8.3 of [4] ■

ring $\mathbb{Z}_q$	prime factors	ω	FNT length	ω	FNT length
$2^{16}+1$	65537	4	16	2	32
$2^{20}+1$	$17 \cdot 61681$	32	8	4100	16
$2^{24}+1$	$97 \cdot 257 \cdot 673$	8	16	$\sqrt{8}$	32
$2^{32}+1$	$641 \cdot 6700417$	4	32	2	64
$2^{40}+1$	$257 \cdot 4278255361$	32	16	$\sqrt{32}$	32
$2^{64}+1$	$274177 \cdot 67280421310721$	4	64	2	128
$2^{80}+1$	$414721 \cdot 44479210368001$	32	32	$\sqrt{32}$	64
$2^{96}+1$	$641 \cdot 6700417 \cdot 18446744069414584321$	8	64	$\sqrt{8}$	128
$2^{112}+1$	$449 \cdot 2689 \cdot 65537 \cdot 183076097 \cdot 358429848460993$	2^7	32	$\sqrt{128}$	64
$2^{128}+1$	$59649589127497217 \cdot 5704689200685129054721$	4	128	2	256

TABLE 2.2.: Parameters of FNT for $2^{16} < q < 2^{129}$.

FNT for various principal roots of unity and transform lengths are tabulated in Table 2.2 by using Proposition 4 and 2. Observe that, one can attain larger FNT lengths when $\omega = \sqrt{2}$ (or let's say when ω is not a power of 2); however, such a choice of the

principal root of unity brings some further complexity (to be specific; multiplications with roots of unity involve additions as well as cyclic shifts) in the computations. For instance: in $2^{20}+1$, $\omega_{16}=4100$ is not a power of 2, so multiplications with roots of unity become 20-bit by 20-bit multiplications. This is not tolerable for our purposes, therefore we recommend the use of the powers of two for roots of unity even if it is at the cost of shorter transform lengths.

In general, the transform lengths tabulated above are considered too short for most of the digital signal processing applications. On the other hand, these lengths seem reasonable for cryptographic applications and our purposes.

2.3. Pseudo Number Transforms

The Mersenne and Fermat rings are not the only suitable rings for efficient arithmetic. If p (not necessarily a prime) is a small divisor of q. The rings of the form $Z_{q/p}$ are also quite useful.

Definition 6 *Let q and p be positive integers and p divides q. The NTT defined over $\mathbb{Z}_{q/p}$ is called a* **pseudo number transform (PNT).**

In general, arithmetic in $\mathbb{Z}_{q/p}$ is difficult; however, since p is a factor of q, the arithmetic modulo (q/p) can be carried in the ring Z_q. By selecting Z_q as a Mersenne or Fermat ring one simplifies the overall arithmetic. The next theorem makes the importance of PNT more clear.

Theorem 1 *Let*

$$q = q_1 q_2 \dots q_n = p_1^{e_1} p_2^{e_2} \dots p_n^{e_n}, \quad \textit{where} \quad q_i = p_i^{e_i} \textit{ for } i = 1, 2, \dots, n$$

for distinct primes p_i and positive integers e_i and n. Let R be a proper subset of the set $\{q_1, q_2, \ldots, q_n\}$ and $R' = \{p_i - 1 : p_i^{e_i} \in R\}$. If $S = \{p_1 - 1, p_2 - 1, \ldots, p_n - 1\}$ then gcd(S) $\leqslant$ gcd(R') =: d' and a PNT of length-d' can be defined over $\mathbb{Z}_{q/p}$ for $p = \prod_{q_i \notin R} q_i$.

Proof: Firstly, $R \subsetneq S \Rightarrow \gcd(S) \leqslant \gcd(R')$. For the second part, let R be a proper subset of the set $\{q_1, q_2, \ldots, q_n\}$ such that $q/p = \prod_{q_i \in R} q_i$. By using the Proposition 2 there exists a NTT with length $d' = \gcd(\{p_i - 1 : p_i^{e_i} \in R\})$ over $\mathbb{Z}_{q/p}$. ■

Example 2 *In $\mathbb{Z}_{2^{15}-1}$, Proposition 2 states that the maximum transform length is gcd$(6, 30, 150) = 6$. This MNT length is very short if the size of the ring is considered. On the other hand, if a PNT is employed in the ring $\mathbb{Z}_{(2^{15}-1)/7}$, we get the transform lengths up to gcd$(30, 150) = 30$.*

At first glance, the arithmetic in the ring $\mathbb{Z}_{(2^{15}-1)/7}$ seems difficult; however, it is possible to perform the actual computation in the ring $\mathbb{Z}_{(2^{15}-1)}$ with a final reduction to modulo $(2^{15} - 1)/7$.

Remark 1 *Observe that PNT tailors the rings in a way that larger length transforms are possible. But while doing that the size of the ring shrinks. The most interesting PNTs are the ones which enlarge the lengths with minimal shrinkage. The effective size of the decreased ring has to be concerned when PNTs are used.*

In Appendix A, Tables 6.1 and 6.2 present parameters for some suitable pseudo Mersenne and Fermat rings. If the Tables 2.1 & 6.1 and 2.2 & 6.2 are combined, it is seen that for almost every v (recall that $q = 2^v \pm 1$) in between 16 and 128 there exist some set of parameters for a nice NTT. Therefore, PNTs enrich the possible design choices which equip us to meet the marginal needs of a particular applications.

2.4. DFT and Convolution Algorithms

The DFT of a sequence $\{x_m\} = (x_0, x_1, \ldots, x_{d-1})$ is defined as the sequence $\{X_k\} = (X_0, X_1, \ldots, X_{d-1})$ given by Equation (2.3), where ω is the principal d-th root of unity.

The sum can also be written as a matrix-vector product as

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ \vdots \\ X_{d-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{d-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(d-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{d-1} & \omega^{2(d-1)} & \cdots & \omega^{(d-1)(d-1)} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{d-1} \end{bmatrix} \pmod{q} .$$

We denote this matrix-vector product by $X = \mathcal{T}x$, where $\mathcal{T}$ is the $d \times d$ transformation matrix. The inverse DFT is defined as $x = \mathcal{T}^{-1}X$. It turns out that the inverse of $\mathcal{T}$ is obtained by replacing ω with ω^{-1} in the matrix, and by placing a multiplicative factor d^{-1} in front of the matrix. The inverse matrix is given as

$$\mathcal{T}^{-1} = d^{-1} \cdot \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \cdots & \omega^{-(d-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \cdots & \omega^{-2(d-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{-(d-1)} & \omega^{-2(d-1)} & \cdots & \omega^{-(d-1)(d-1)} \end{bmatrix} \pmod{q} .$$

The matrix-vector product definition of the DFT implies an algorithm to compute the DFT function, However, this requires d multiplications and $d-1$ additions to compute an entry of the output sequence $\{X_k\}$. Thus, the total number of multiplications is d^2, and the total number of additions is $d(d-1)$. This complexity is acceptable for our primary purpose centered around a modular exponentiation as discussed in Section 3.9 and specifically as illustrated in Figure 3.7. Although DFT calculations are

negligible for our purposes the *Fast Fourier Transform (FFT)* algorithm can be used which reduces the complexity from $O(d^2)$ to $O(d \log d)$. We refer the reader to [3] and [4] for excellent presentations of the subject.

2.5. Convolutions over the integer ring $\mathbb{Z}_q$

The computation of linear convolution is important in many fields of engineering. In practice, especially for long sequences, linear convolutions are computed by using the DFT in conjunction with the convolution property. If the DFT is specified in these computations, we say that **DFT respects the convolution.**

Earlier we defined the linear convolution as

$$z(t) = x(t)y(t) \quad \text{such that} \quad \deg(x(t)) + \deg(y(t)) < d \tag{2.5}$$

Observe that "mod $t^d - 1$" is dropped from the Equation (2.1) since the condition $\deg(x(t)) + \deg(y(t)) < d$ guarantees that the degree of $z(t)$ never exceeds d. In fact, by this condition the linear convolution is turned into a cyclic one which can be computed with a d-point DFT followed by a component-wise multiplication and an inverse DFT. On the other hand, if $\deg(x(t))+ \deg(y(t)) > d$, d-point DFT produces erroneous answers and DFT does not respect convolution of $x(t)$ and $y(t)$.

A similar situation occurs when using DFT over $\mathbb{Z}_q$ spectrum. In order to address this problem, Equation (2.5) has to be written as follows

$$z(t) = x(t)y(t)$$

such that $\deg(x(t)) + \deg(y(t)) < d$ and $0 \leqslant z_i < q$ for all $i = 0, 1, \ldots, d-1$.

Assume that $\deg(x(t)) + \deg(y(t)) < d$ is satisfied, by the above analysis the linear convolution computation by using DFT over $\mathbb{Z}_q$ and convolution property do not produce wrong answers; however, if any coefficient of $z(t)$ (the convolution products)

exceeds q, this procedure produces erroneous answers. The reason is simple, since DFT works over $\mathbb{Z}_q$, the inverse DFT computes $z(t)$ mod q, which is not equal to the actual $z(t)$. In the literature, such incidents are called **overflows**, in order to avoid overflows, input signals or coefficients of $x(t)$ and $y(t)$ should be bounded properly.

In Chapter 3, we define and analyze these incidents in a formal way. We close the section by an example which illustrates the above discussions.

Example 3 *Consider the integer ring $\mathbb{Z}_{31}$; the element 2 is a principal 5th root of unity and 5^{-1} exists since $gcd(31,5) = 1$, thus there exists a 5-point NTT over ring $\mathbb{Z}_{31}$. DFT can be achieved by a matrix multiplication where the transformation matrix $\mathcal{T}$ is given by*

$$\mathcal{T} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 2^2 & 2^3 & 2^4 \\ 1 & 2^2 & 2^4 & 2^6 & 2^8 \\ 1 & 2^3 & 2^6 & 2^9 & 2^{12} \\ 1 & 2^4 & 2^8 & 2^{12} & 2^{16} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 16 \\ 1 & 4 & 16 & 2 & 8 \\ 1 & 8 & 2 & 16 & 4 \\ 1 & 16 & 8 & 4 & 2 \end{bmatrix}$$

Now, consider the sequences $x = (1,2,3,0,0)$ and $y = (2,3,3,0,0)$, one finds that the condition $deg(x(t)) + deg(y(t)) = 2 + 2 < 5$ is satisfied if the polynomial representations of x and y are examined.

The transforms of x and y can be computed as

$$X = \mathcal{T}x^T \equiv (6,17,26,23,26) \ mod\ 31 \quad \text{and} \quad Y = \mathcal{T}y^T \equiv (8,20,0,1,12) \ mod\ 31$$

At this point, we perform point-wise multiplication (i.e. convolution property), and then take the inverse transform.

$$Z = X \odot Y \equiv (17,30,0,23,2) \ mod\ 31$$

$$z = \mathcal{T}^{-1}(17, 30, 0, 23, 2)^T \equiv (2, 7, 15, 15, 9) \ mod\ 31$$

which is indeed the convolution of the signals x *and* y.

If $x = (10, 5, 1, 0, 0)$ *is taken, by the same procedure one gets* $z = (20, 9, 20, 24, 9)$ *which is equivalent to* $z = (20, 40, 51, 24, 9)$ *modulo 31; however, this is a wrong answer caused by an overflow.*

3. SPECTRAL MODULAR ARITHMETIC

In this chapter we describe our main idea of spectral modular arithmetic. After covering some preliminaries in Sections 3.1 through 3.3, we define *spectral modular reduction* (SMR) in Section 3.4. In Section 3.5, we give a time simulation (Algorithm 4) that is translated into complex spectrum in Section 3.6 and into a finite ring spectrum in Section 3.7 by using the linearity and shifting property of DFT (see Section 2.1.3). We present *Spectral Modular Product* (SMP) (Algorithm 8), which consists of convolution and spectral reduction, as the basic building block of both *spectral modular multiplication* (SMM) and *spectral modular exponentiation* (SME). Finally, we describe our main contribution SMM and SME in Sections 3.8 and 3.9 respectively.

Furthermore, along with the presentation of algorithms, we investigate the minimal domains (i.e. smallest rings) in which our spectral algorithms work. In particular, Theorem 12 characterizes the size of a minimal ring that SME (with a SMP core) works without overflows.

3.1. Evaluation Polynomials

So far, we have had a presentation mostly adopted from the theory of digital signal processing. In such a theory, the signals are usually assumed to be statistically independent. However in the setting of modular or integer arithmetic, the signals are not really of such a nature. In fact, signals mostly correspond to representations of numbers in a certain radix. Therefore, we develop a more clear notation that permit us to explore this observation in a more convenient way.

Definition 7 *Let x and $b > 0$ be integers, if $x(t)$ is a polynomial in $\mathbb{Z}[t]$ such that $x(b) = x$ then we say $x(t)$ is an* **evaluation polynomial of** x. *For ease of notation we denote evaluation polynomials by the pair $(x, x(t))$.*

Remark 2 *The positive integer b is called the* **base** *or* **radix**. *In order not to have confusion with the frequent usage of word "radix" for another instance in FFT theory, we use the base for b.*

The set of all evaluation polynomials is a fairly big set if b is seen as a variable. For our purposes, we rather prefer to work on a subset.

Notation 2 *If the positive integer b is fixed, then the set of all evaluation polynomials is denoted by $\mathcal{B}$.*

Remark 3 *When b is fixed there exists a natural 1-1 correspondence between $\mathcal{B}$ and $\mathbb{Z}[t]$ which is given by $(x, x(t)) \mapsto x(t)$,*

Remark 4 *To be consistent with the previous notations, if*

$$x(t) = x_0 + x_1 t + \ldots + x_{d-1} t^{d-1} \in \mathbb{Z}[t]$$

is an evaluation polynomial of an integer x for a fixed positive integer b then the corresponding length d sequence $\{x_m\} = (x_0, x_1, \ldots, x_{d-1})$ is called the evaluation sequence.

In this context, we like to specify the evaluation polynomials (or sequences) of base (or radix) representation of numbers as follows:

Definition 8 *Let*

$$x(t) = x_0 + x_1 t + \ldots + x_{d-1} t^{d-1} \in \mathbb{Z}[t]$$

be an evaluation polynomial of an integer x for a fixed base b. If the coefficients of $x(t)$ satisfy $0 \leqslant x_i < b$ for all $i = 1, 2, \ldots, d-1$, $x(t)$ is called the **base evaluation polynomial** *or simply the* **base polynomial**.

Example 4 *A base $2^k, k > 0$ representation of an integer x ($(x_0 x_1 \dots x_d)$ with $0 \leqslant x_i < 2^k$ for $i = 1, 2, \dots, d-1$) has the base polynomial $x(t) = x_0 + x_1 t + x_2 t^2 \dots + x_{d-1} t^{d-1}$ where $y(t) = (x_0 + x_1 b) + 0 \cdot t + x_2 t^2 + \dots + x_{d-1} t^{d-1}$ is any one of its evaluation polynomials.*

As seen in Example 4; the evaluation polynomial (or sequence) of the integer x is not unique, indeed the same integer has infinitely many different evaluation polynomials. But note that the base polynomials (i.e. base representations) are unique.

Proposition 5 *Let $\mathcal{B}$ denote the set of all evaluation polynomials then $(\mathcal{B}, \oplus, \otimes)$ is a ring with the following operations;*

$$(x, x(t)) \oplus (y, y(t)) = (x + y, x(t) + y(t))$$

$$(x, x(t)) \otimes (y, y(t)) = (xy, x(t)y(t))$$

where $x(t), y(t) \in \mathbb{Z}[t]$ and $x, y \in \mathbb{Z}$.

Proof: With fixed b, it is easily seen that $(\mathcal{B}, \oplus)$ is surely an abelian group and $(\mathcal{B}, \otimes)$ is a closed set since the structures on the components of pair come from $\mathbb{Z}$ and $\mathbb{Z}[t]$. All we need to show is that the evaluation map is well defined on the components. This is trivial because $x+y = x(b)+y(b)$, and the distribution property comes naturally from this observation. Thus, $(\mathcal{B}, \oplus, \otimes)$ is a ring with identity $(1_\oplus, 1_\otimes) = (0, 1)$. ■

Proposition 6 *The map $\phi : \mathcal{B} \to \mathbb{Z}[t]$ sending $(x, x(t)) \mapsto x(t)$ is a ring isomorphism.*

Proof: Since $b > 0$ is fixed, the evaluation $x = x(b)$ is also fixed which implies that there exists a natural surjective map from $\mathcal{B}$ to $\mathbb{Z}[t]$ sending $(x, x(t)) \mapsto x(t)$ with a zero kernel. ■

Definition 9 *If $x(t)$ and $y(t)$ are evaluation polynomials for the same integer x, then we write $x(t) \sim y(t)$ and say $x(t)$ is related to $y(t)$.*

Proposition 7 *$x(t) \sim y(t)$ is an equivalence relation*

Proof: (i) $x(t) \sim x(t)$ since $x(b) = x(b)$

(ii) if $x(t) \sim y(t)$ then $y(t) \sim x(t)$ since $x(b) = y(b)$

(iii) if $x(t) \sim y(t)$ and $y(t) \sim z(t)$ then $x(t) \sim z(t)$ since $x(b) = y(b) = z(b)$ ■

Proposition 8 *Let $\mathcal{B}$ be the set of all evaluation polynomials then $\mathcal{B}/\sim$ is isomorphic to the ring of integers $\mathbb{Z}$.*

Proof: Let the base polynomials be the representative of the equivalence classes of the set $\mathcal{B}$ with respect to the relation $\sim$. Since base polynomials are unique for all integer $x \in \mathbb{Z}$. The map

$$\mathbb{Z} \to \mathcal{B}/\sim$$

$$x \mapsto [(x, x(b))]$$

gives the isomorphism.

■

Definition 10 *The set of all evaluation polynomials of degree less than a positive integer d is called a* **time frame** *and denoted by $\mathcal{B}^d$.*

Observe that, $\mathcal{B}^d \subset \mathcal{B}$; however, $(\mathcal{B}^d, \otimes)$ is not closed under the binary operation $\otimes$ thus $\mathcal{B}^d$ is not a subring of $\mathcal{B}$ but $(\mathcal{B}^d, \oplus)$ is still an abelian group.

Fortunately, the ring of the evaluation polynomials is isomorphic to the standard $\mathbb{Z}[t]$. One might like to state the same relation between $\mathcal{B}^d$ and $\mathbb{Z}[t]/(t^d - 1)$ which is not a good practice for many reasons, let us give an example.

Example 5 *Suppose* $x = 19$ *and* $y = 9$. *If the base* $b = 2$ *evaluation polynomials* $x(t) = 1 + t + 2t^3$ *and* $y(t) = 3 + t + t^2$ *of* x *and* y *are considered, the product*

$$x(t)y(t) = 3 + 4t + 2t^2 + 7t^3 + 2t^4 + 2t^5 \tag{3.1}$$

gives an evaluation polynomial for $19 \cdot 9 = 171$. *But since the product (3.1) does not belong to the set* $\mathcal{B}^4$, *the result becomes undefined or out of bounds. On the other hand, if we work in the ring* $\mathbb{Z}[t]/(t^4 - 1)$ *we are forced to reduce the product (3.1) to* $5 + 6t + 2t^2 + 7t^3$. *This clearly in* $\mathbb{Z}[t]/(t^4 - 1)$ *but evaluation of it at* $b = 2$ *gives* 81 *rather than* 171. *Since we suppress evaluations, such a reduction is not acceptable for our purposes, hence for the time we keep the result (3.1) as undefined rather than reducing a wrong evaluation polynomial, but later we are going to define the spectral modular reduction that respects the evaluations while reducing the degree.*

As computer arithmeticians, we started to build a terminology in time domain with a very simple observation that in some sense the sequences we deal with are not statistically independent. We describe that this dependence can be revealed by a pair; the polynomial and its evaluation. Actually with this notation, we put the emphasis on the evaluation as well as the representation. At first glance, adding the evaluation information to the representation seems redundant but such a convention leads us to a better understanding of spectral techniques for arithmetic operations. Moreover such a representation states the different nature of a number representing signals from a classical signal processing analysis.

Remark 5 *Note that the same theory can be built by taking sequences or vectors as building blocks.*

When the spectral sequences are considered our simple observation is not correct any more hence most of the terminology created so far seems not to make any sense for

spectral sequences. However, we like to use the polynomial representation for a unified notation. Therefore, when we write

$$x(t) \quad \overset{DFT}{\longleftrightarrow} \quad X(t)$$

we mean the coefficients of $x(t)$ and $X(t)$ are transform pair as sequences. We call $x(t)$ the **time polynomial** where $X(t)$ is called the **spectral polynomial of** $x(t)$. As before we reserve the capital letters for the spectrum variables.

Theorem 2 *If $\mathcal{F}$ denotes the set of all spectral polynomials over $\mathbb{C}$, then $\mathcal{F}$ forms a ring with component-wise addition and multiplication. $(\mathcal{F}, +, \odot)$ which is called the* **complex Fourier ring**.

Proof: Observe that, $(\mathcal{F}, +)$ is the same abelian group as the additive complex polynomials. $(\mathcal{F}, \odot)$ is closed since if $X(t), Y(t) \in \mathcal{F}$ then component-wise multiplication $X(t) \odot Y(t)$ is definitely in $\mathcal{F}$. The distribution property is inherited from complex numbers $\mathbb{C}$ on the coefficients. Note that a multiplicative identity do not exist. ■

Theorem 3 *If $\mathcal{F}^d$ is the set of spectral polynomials over $\mathbb{C}$ with degree less than a positive integer d then $\mathcal{F}^d$ is a subring of the Fourier ring $\mathcal{F}$ and it is isomorphic to $\mathbb{C}^d$ (as a direct sum of rings).*

Proof: All we need to show is $\mathcal{F}^d$ is closed under both binary operations. $(\mathcal{F}^d, +)$ is closed because addition of any two polynomial does not increase the degree of the result. The same reasoning is also correct for $(\mathcal{F}^d, \odot)$, thus $\mathcal{F}^d$ is a subring of $\mathcal{F}$. Moreover if $X(t) = X_0 + X_1 t + \ldots + X_d t^d$ then the map $X(t) \mapsto (X_0, X_1, \ldots, X_d) \in \mathbb{C}^d$ surely defines an isomorphism. ■

3.2. DFT and Convolution Revisited

In this section we establish an algebraic treatment for some concepts of Fourier transforms and convolutions.

3.2.1. DFT and Convolution over the Complex Spectrum

The DFT is an invertible set map from $\mathcal{B}^d$ to $\mathcal{F}^d$. Moreover the properties of DFT show that this map actually respects some structure as well. Definition 11 gives the standard expression of the DFT for a polynomial representation.

Definition 11 *The DFT is the invertible map*

$$DFT_d^\omega : \mathcal{B}^d \to \mathcal{F}^d, \quad \text{where } \omega \in \mathbb{C} \text{ is a principal } dth \text{ root of unity}$$

defined by

$$X_i = DFT_d^\omega(x(t)) := \sum_{j=0}^{d-1} x_j \omega^{ij}, \quad i = 0, 1, \ldots, d-1 \tag{3.2}$$

with the inverse

$$x_i = IDFT_d^\omega(X(t)) := \frac{1}{d} \sum_{j=0}^{d-1} X_j \omega^{-ij}, \quad i = 0, 1, \ldots, d-1 \tag{3.3}$$

Proposition 9 *The DFT map $DFT_d^\omega : (\mathcal{B}^d, \oplus) \to (\mathcal{F}^d, +)$ is a group homomorphism*

Proof: Let $(x, x(t))$ and $(y, y(t))$ be in $\mathcal{B}^d$, first of all the map $(x, x(t)) \mapsto x(t)$ is well defined since x is fixed by evaluation at b.

$$\begin{aligned} DFT_d^\omega((x, x(t)) \oplus (y, y(t))) &= DFT_d^\omega((x + y, x(t) + y(t))) \\ &= x(t) + y(t) \\ &= DFT_d^\omega((x, x(t))) + DFT_d^\omega((y, y(t))) \ . \end{aligned}$$

In fact, this corresponds to the linearity property (i.e. property (i) of Section 2.1.3). ■

When it comes to multiplicative group, $(\mathcal{B}^d, \otimes)$ is not closed; there exist elements $x(t), y(t) \in \mathcal{B}^d$ such that $x(t) \cdot y(t) \notin \mathcal{B}^d$, thus the DFT map is not even well defined. We conclude that DFT map is not a global group homomorphism on $(\mathcal{B}^d, \otimes)$ but locally there are still cases which the following equation is satisfied

$$\begin{aligned} DFT_d^\omega((x, x(t)) \otimes (y, y(t))) &= DFT_d^\omega((x, x(t)) \odot DFT_d^\omega(y, y(t)) \\ &= X(t) \odot Y(t) \end{aligned} \tag{3.4}$$

Indeed, Equation 3.4 is an another way of stating the convolution property. Fortunately, in the Section 2.5 we discussed whether a product is in the time frame $\mathcal{B}^d$ or not, and concluded that if the condition $\deg(x(t)) + \deg(y(t)) < d$ is satisfied, then DFT respects the convolution.

In fact, next proposition shows that on a convex or a more regular subset of $(\mathcal{B}^d, \otimes)$ the DFT map respects the convolution property.

Proposition 10 *A length d DFT map* $DFT_d^\omega : \mathcal{B}^d \to \mathcal{F}^d$ *respects the convolution property on the subset* $\mathcal{B}^s \subset \mathcal{B}^d$ *where* $s = \lceil d/2 \rceil$.

Proof: Let $x(t)$ and $y(t)$ be elements of the set $\mathcal{B}^{\lceil d/2 \rceil}$ then $\deg(x(t))$ and $\deg(y(t))$ has to be less than $\lceil d/2 \rceil$ which implies

$$\deg(x(t)y(t)) \leqslant \begin{cases} (\frac{d-1}{2}) + (\frac{d-1}{2}) = d - 1 & \text{if } d \text{ is odd} \\ (\frac{d}{2} - 1) + (\frac{d}{2} - 1) = d - 2 & \text{if } d \text{ is even} \end{cases}$$

which is less than d for either case. ■

3.2.2. DFT and Convolution over a Finite Ring Spectrum

In Sections 2.2 and 2.3 we have seen many suitable finite ring spectra for DFT. Moreover, in Section 2.5 we literally stated when DFT respects the convolution property in these finite structures. In this section, we formalize these findings in terms of our new notation.

We begin with adapting the formal definition of DFT (i.e. Definition 11) to a finite ring spectrum. We start with forming a suitable ring for the range followed by the domain.

Theorem 4 *If $\mathcal{F}_q^d$ is the set of all polynomials over $\mathbb{Z}_q$ of degree less than a positive integer d, then $\mathcal{F}_q^d$ forms a ring with standard polynomial addition and component-wise multiplication. The ring $\mathcal{F}_q^d$ is called the* **Fourier ring** *and it is isomorphic to $\mathbb{Z}_q^d$ (i.e. the direct sum of rings).*

Proof: All we need to show is that $\mathcal{F}_q^d$ is closed under both binary operations. $(\mathcal{F}_q^d, +)$ is closed because the degree of sum of two polynomials with degree less than d do not exceed $d-1$ and the coefficients stay in $\mathbb{Z}_q$. Indeed $(\mathcal{F}_q^d, +)$ is the same abelian group as additive group of the polynomial ring $\mathbb{Z}_q[t]$. On the other hand, the same reasoning is also correct for $(\mathcal{F}_q^d, \odot)$ since if $X(t), Y(t) \in \mathcal{F}_q^d$ then the degree of $X(t) \odot Y(t)$ do not exceed d so the result is in $\mathcal{F}_q^d$. The distribution property is inherited from $\mathbb{Z}_q$, the ring integers modulo q, on the coefficients. Note that a multiplicative identity do not exist.

Additionally, if $X(t) = X_0 + X_1 t + \ldots + X_d t^d \in \mathcal{F}_q^d$ then the map sending $X(t)$ to $(X_0, X_1, \ldots, X_d) \in \mathbb{Z}_q^d$ surely defines an isomorphism. That is to say $\mathcal{F}_q^d \approx \mathbb{Z}_q^d \approx \mathbb{Z}_q^d$.

■

Definition 12 *Let r and d be positive integers, we define a* **positive polynomial frame** *as*

$$\mathcal{H}_r^d = \{y(t) \in \mathcal{B} : \ deg(y(t)) < d \text{ and } 0 \leqslant y_i < r \text{ for all } i = 0, 1, \ldots, d-1\}$$

Remark 6 *Note that, the $\mathcal{H}_b^d$ is the set of base polynomials.*

We mentioned earlier that it is not a good practice to visualize $\mathcal{B}^d$ as the polynomial ring $\mathbb{Z}[t]/(t^d - 1)$. The same assertion can be made for $\mathcal{H}_r^d$ and $\mathbb{Z}_r[t]/(t^d - 1)$. We give an example to show this difference.

Example 6 *Let $x = 15$, $y = 12$ and base $b = 2$. Consider the evaluation polynomials of $x(t) = 11 + 2t$ and $y(t) = 6 + t + t^2$ of x and y. The product*

$$x(t)y(t) = 66 + 23t + 13t^2 + 2t^3 \tag{3.5}$$

sits in $\mathbb{Z}_{17}[t]/(t^4 - 1)$ without a degree reduction but since the underlying integer ring is $\mathbb{Z}_{17}$ we have to reduce the coefficients of the product. This gives the polynomial $15 + 6t + 13t^2 + 2t^3$ which is an evaluation polynomial for 95 rather than 180. On the contrary if $\mathcal{H}_{17}^4$ is considered, we would say that the result (3.5) is undefined or out of bounds. Actually, this ill-favored situation corresponds to overflows discussed in Section 2.5.

Example 6 shows that the positive polynomial frame $\mathcal{H}_r^d$ does not form a group with either usual addition or convolution. But as in the time frame case, we can speak of some subsets of $\mathcal{H}_r^d$ such that DFT respects these two operations. Moreover, when we define the spectral modular reduction, a reduction that respects evaluations, we are going to speak of closed frames with respect to spectral modular addition and multiplication and later we embed these structured frames into Fourier rings to apply the DFT theory. Let us first give a formal definition of DFT in the finite ring setting.

Definition 13 *The DFT map is the invertible map*

$$DFT_d^\omega : \mathcal{H}_q^d \rightarrow \mathcal{F}_q^d, \quad \text{where } \omega \in \mathbb{Z}_q \text{ is a principal } dth \text{ root of unity}$$

defined by

$$X_i = DFT_d^\omega(x(t)) := \sum_{j=0}^{d-1} x_j \omega^{ij} \bmod q, \quad i = 0, 1, \ldots, d-1 \tag{3.6}$$

with the inverse

$$x_i = IDFT_d^\omega(X(t)) := \frac{1}{d} \sum_{j=0}^{d-1} X_j \omega^{-ij} \bmod q, \quad i = 0, 1, \ldots, d-1 \tag{3.7}$$

Next we formally state when a DFT respects a convolution.

Definition 14 *Suppose that* $(x(t), x)$ *and* $(y(t), y)$ *are evaluation polynomials in the frame* $\mathcal{H}_q^s$ *with* $s = \lceil d/2 \rceil$. *We say a* **DFT map** DFT_d^ω **respects the convolution** *if the the product* $(z(t), z) = (x(t), x) \otimes (y(t), y)$ *sits in* $\mathcal{H}_q^d$. *Moreover, we say an* **overflow occurs** *for those cases in which this condition is not satisfied.*

Lemma 1 *Suppose that* $(x(t), x)$ *and* $(y(t), y)$ *are base* b *polynomials in the frame* $\mathcal{H}_b^s$ *with* $s = \lceil d/2 \rceil$. *The product* $(z(t), z) = (x(t), x) \otimes (y(t), y)$ *sits in* $\mathcal{H}_q^d$ *where* $q > sb^2$.

Proof: Let $x(t) = x_0 + x_1 t + \ldots + x_{d-1} t^{d-1}$ and $y(t) = y_0 + y_1 t + \ldots + y_{d-1} t^{d-1}$ be polynomials such that $\deg(x(t)) + \deg(y(t)) < d$ and $0 \leqslant x_i, y_i < b$ for some $b > 0$. Without loss of generality, assume $\deg(x(t)) \geqslant \deg(y(t))$. If $z(t) = x(t)y(t)$ then the coefficients of $z(t)$ can be written as follows

$$z_k = \sum_{k=i+j} x_i y_j, \quad k = 0, 1, 2, \ldots, d-1 \ .$$

For any k, z_k can be found by adding at most $\deg(y(t)) + 1$ nonzero terms, but since $\deg(y(t)) + 1 \leqslant \lceil d/2 \rceil$, letting $s = \lceil d/2 \rceil$ gives

$$z_k \leqslant (\deg(y(t)) + 1) \cdot b \cdot b \leqslant s \cdot b^2$$

thus choosing $q > s \cdot b^2$ gives the result. ■

The following result gives the condition when the d-point DFT map respects the convolution of two elements of a positive frame $\mathcal{H}_q^d$

Theorem 5 *If $DFT_d^\omega : \mathcal{H}_q^d \rightarrow \mathcal{F}_q^d$ is a length d DFT map then DFT_d^ω respects the convolution as well as the addition on the subset $\mathcal{H}_b^s \subset \mathcal{H}_q^d$ where $s = \lceil d/2 \rceil$ and $b^2 s < q$ for an integer $b > 0$.*

Proof: Let $x(t)$ and $y(t)$ be polynomials in $\mathcal{H}_b^s$ for some $b > 0$. By using Lemma 1, the product $z(t) = x(t)y(t)$ is in $\mathcal{H}_q^d$ where $q > s \cdot b^2$. Therefore, the DFT map DFT_d^ω respects the convolution, and the convolution can be computed by using the combination of DFT and convolution property. ■

3.2.3. Time Simulations and Spectral Algorithms

In the previous two sections we stated the conditions when the convolution and linearity property hold for either complex or finite ring spectra.

At this point, we state that our primary interest with spectral techniques is the ability of replacing convolutions by component-wise multiplications. An algorithm involving many convolutions benefits most from such a transformation. For instance; the encryption algorithm RSA [11] over some integer ring has such a nature, it involves significant number of multiplications. But since these multiplications are modular, one has to deal with reductions. In general, reductions are performed as a combination or a sequence of multiplications and additions/subtractions, hence by using the properties of DFT it is possible to perform reduction in the spectrum. But this still needs an analysis of conditions when a DFT map respects such sequences of operations. We start by an example in order to illustrate our methodology.

Example 7 *Consider an algorithm in time domain doing the following;*

Algorithm 1 .

input: $x(t), y(t) \in \mathbb{Z}[t]$ *polynomials of degree* d

output: $z(t) := x(t)(5y(t) + x(t)) - 3x(t)$

1: $z(t) := x(t) + 5y(t)$

2: $z(t) := x(t) \cdot z(t)$

3: $z(t) := z(t) - 3x(t)$

4: **return** $z(t)$

As long as all the intermediate results and the output in the above algorithm belong to the domain of DFT map, DFT respects the algorithm and we have a dual algorithm in the spectrum which is presented as follows.

Algorithm 2 .

input: $X(t), Y(t) \in \mathbb{Z}[t]$ *polynomials of degree* d

output: $Z(t) := X(t) \odot (5Y(t) + X(t)) - 3X(t)$

1: $Z(t) := X(t) + 5Y(t)$

2: $Z(t) := X(t) \odot Z(t)$

3: $Z(t) := Z(t) - 3X(t)$

4: **return** $Z(t)$

Observe that when the inputs of Algorithm 1 and 2 agree, a parallel run produces the agreeing intermediate results as well as the actual output. By agreeing we mean that there exists a DFT relation between the data in two domain at all times.

Algorithm 1 is called the **time simulation** of spectral algorithm. In fact, Example 7 tells a lot about our methodology. In the following sections, we start with describing some time simulations of modular reduction algorithm. The challenge is to

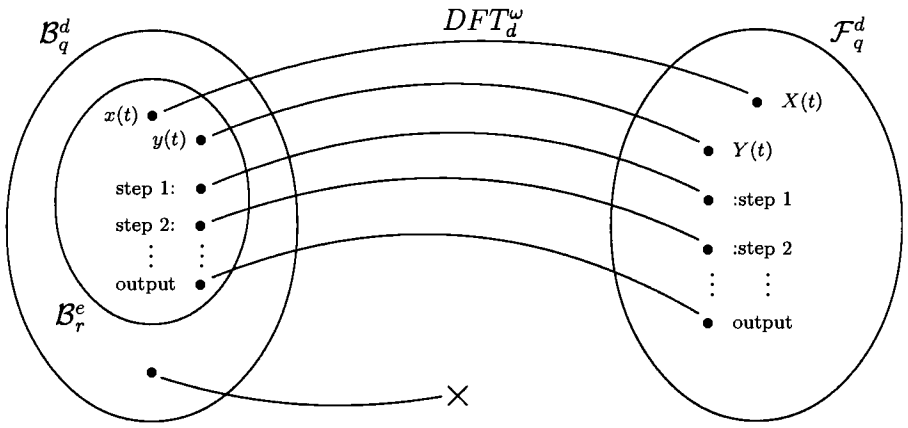

FIGURE 3.1. A translation of a time simulation into the spectrum, two algorithms agree as long as the intermediate results and the output belong to the domain $\mathcal{B}_r^e$ for some positive integers e and r.

come up with some nice algorithm that has a simple translation. Then we translate the time simulations to complex and finite ring spectrum. The majority of our work is dedicated to find the minimal domains (i.e. smallest rings) in which our spectral algorithms work.

3.3. Modular Reduction

Before introducing the notion of spectral reduction, we need to make a few points clear about the modular arithmetic over the ring of integers;

In calculations of integers involving division it often happens that we are interested in remainder, but not the quotient. Those numbers having the same remainder when divided by a fixed number n are called congruent, to be more formal:

Definition 15 *Let $n > 0$ be a fixed integer. We say x* **is congruent to** *y* **modulo** *n and write*

$$y \equiv x \bmod n \quad \textit{if } n \textit{ divides } (y - x). \tag{3.8}$$

From the division algorithm we know that for each $x \in \mathbb{Z}$ there is an equation

$$x = nq + r, \quad \text{for some } q \in \mathbb{Z} \text{ and } 0 \leqslant r \leqslant n$$

This means that each $x \in \mathbb{Z}$ can be assigned to one of the element of the set $\{0, 1, 2, \ldots, n-1\}$. This set is called the **least residues** mod n and it is clear that no two of the elements are congruent to each other mod n. We define the modular reduction as follows.

Definition 16 *Let $n > 0$ be a fixed integer. We say y is the modular reduction of x modulo n and write*

$$y = x \bmod n \quad \textit{if } y \textit{ is a least residue } \bmod n$$

Remark 7 *The expressions "$y = x \bmod n$" and "$y \equiv x \bmod n$" have different meanings. Observe that the first one with "=" states that y is in the range $[0, n-1]$.*

The equivalence on $\mathbb{Z}$ defined by the relation (3.8) partitions $\mathbb{Z}$ into n blocks, called the residue classes of $\mathbb{Z}$ mod n. Indeed $\mathbb{Z}_n := \mathbb{Z}/n\mathbb{Z}$ is the set of these residue classes. If we denote the residue class mod n containing y by $\bar{y}$, then the $\mathbb{Z}_n$ can be seen as the ring having the following n elements $\bar{0}, \bar{1}, \ldots, \overline{(n-1)}$. For instance, when $n = 2$, the residue classes are the set of even and odd numbers.

The ring $\mathbb{Z}_n$ is mostly represented by the least residues mod n. But using a different representation set is equally valid as long as the set contains a unique representative of the disjoint residue classes of $\mathbb{Z}_n$. Such representations require a modified modular reduction.

Definition 17 *Let $n > 0$ be a fixed integer, and let the set $\mathcal{R} = \{y_1, y_2, \ldots, y_{n-1}\}$ be a representation set of the residue classes of $\mathbb{Z} \bmod n$ such that no two of elements are congruent to each other $\bmod n$. The modular reduction of $x \in \mathbb{Z}$ with respect to modulus n will be defined by*

$$y = x \bmod_{\mathcal{R}} n \quad \textit{where } y \in \mathcal{R} \textit{ is congruent to } x,$$

We use "$\bmod_{\mathcal{R}}$" notation in order to emphasize the representation set $\mathcal{R}$.

Example 8 *For $n = 3$, the three disjoint residue classes are $\{\ldots, -8, -5, -2, 1, 4, 7, 10, \ldots\}$, $\{\ldots, -7, -4, -1, 2, 5, 8, 11, \ldots\}$ and $\{\ldots, -9, -6, -3, 0, 3, 6, 9, \ldots\}$. The standard representation set is $\{0, 1, 2\}$. If the representation $\mathcal{R} = \{-3, 1, 5\}$ is used $21 = -3 \bmod_{\mathcal{R}} 3$ where with the standard representation we have $21 = 0 \bmod 3$.*

We, now define a subset of $\mathcal{B}$ that corresponds to the least magnitude residue representation

$$\mathcal{R} = \{\lceil -n/2 \rceil, \ldots, -1, 0, 1, 2, \ldots, \lceil n/2 \rceil\} .$$

Definition 18 *Let r and d be positive integers, the following subset of $\mathcal{B}$*

$$\mathcal{B}_r^d = \{y(t) \in \mathcal{B} : \ deg(y(t)) < d \textit{ and } \lceil -r/2 \rceil \leqslant y_i \leqslant \lceil r/2 \rceil \textit{ for all } i = 0, 1, \ldots, d-1\}$$

will be called a **polynomial frame with radius r and degree d.**

Remark 8 *Note that, $\mathcal{H}_r^d$ and $\mathcal{B}_r^d$ are related since the set $\lceil -r/2 \rceil \leqslant y_i \leqslant \lceil r/2 \rceil$ is a shifted version of the set $0 \leqslant y_i < r$.*

While performing computations such as modular exponentiation, in order to have some computational advantage sometimes exact modular reduction calculations can be postponed for the intermediate values [12]. As long as these values belong

to the correct residue classes such modifications do not tend to misleading modular reductions. Now, we stretch the definition of the modular reduction for ease of our construction.

Definition 19 *Let $\varepsilon > 0$ be an integer, we call the set*

$$F(\varepsilon) = \{y \in \mathbb{Z} : |y| < \varepsilon\}$$

the **integer frame of radius** ε.

Definition 20 *Let $x, n > 0$ and $\varepsilon \geqslant n$ be integers. Then the elements of the set*

$$\{y : y \in F(\varepsilon) \text{ and } y \equiv x \bmod n\}$$

are called the **almost modular reductions of** x **with respect to the modulus** n.

Example 9 *Let $\varepsilon = 6$ then $F(\varepsilon) = (-6, 6)$ and the set of almost modular reductions of $x = 1$ with respect to modulus $n = 3$ is $\{-5, -2, 1, 4\}$.*

The choice of the radius ε completely depends on the nature or needs of the problem. Most of the time the reductions followed by a squaring or a multiplication therefore, as ε gets larger the operand sizes of the follow up operations increase. Obviously $\varepsilon = n$ is the optimal choice in that sense. But as we pointed out earlier, we are after some approximations of the optimal solution for some obvious reasons. In other words, we are looking for some small ε such that after finding an element of almost modular reduction set, deducing the exact modular reduction has to be simple. Indeed, that is why it is appropriate to use the adjective "almost" to describe the elements of this set.

3.4. Spectral Modular Reduction

In this section we give a formal definition for the spectral modular reduction and build up the necessary terminology for a better understanding of the algorithms in the spectrum. We return to our main objects: the set of evaluation polynomials, $\mathcal{B}$, and its subsets.

Proposition 11 *The evaluations of the polynomials in $\mathcal{B}_r^d$ form an integer frame $F(\varepsilon)$ in $\mathbb{Z}$ where $\varepsilon = (\lceil r/2 \rceil - 1) + (\lceil r/2 \rceil - 1)b + (\lceil r/2 \rceil - 1)b^2 + \ldots + (\lceil r/2 \rceil - 1)b^{d-1}$.*

Proof: It is easily seen that the polynomial $x(t) = (\lceil r/2 \rceil - 1) + (\lceil r/2 \rceil - 1)t + (\lceil r/2 \rceil - 1)t^2 + \ldots + (\lceil r/2 \rceil - 1)t^{d-1} \in \mathcal{B}_r^d$ attains the maximum evaluation value at base b which is the integer $(\lceil r/2 \rceil - 1) + (\lceil r/2 \rceil - 1)b + (\lceil r/2 \rceil - 1)b^2 + \ldots + (\lceil r/2 \rceil - 1)b^{d-1}$ (see Figure 3.2). Likewise $-x(t)$ takes the minimum value. Setting $\varepsilon = (\lceil r/2 \rceil - 1) + (\lceil r/2 \rceil - 1)b + (\lceil r/2 \rceil - 1)b^2 + \ldots + (\lceil r/2 \rceil - 1)b^{d-1}$ gives the result. ■

Definition 21 *Let $n(t)$ be a base b polynomial of n with degree $d-1$. The elements of the set*

$$\mathcal{A} = \{(y, y(t)) : y \equiv x \bmod n \text{ and } y(t) \in \mathcal{B}_r^d \text{ for some } r \geqslant b\}$$

are called **almost spectral reductions of the evaluation polynomial** $(x, x(t))$ **with respect to** $(n, n(t))$.

Lemma 2 *Let $\mathcal{A}$ be the set of all almost spectral modular reductions of $(x, x(t))$ with respect to $(n, n(t))$. If $y(t)$ is the base polynomial for $y = x \bmod n$ then $(y, y(t)) \in \mathcal{A}$.*

Proof: If $y(t) = y_0 + y_1 t + \ldots + y_{d-1}t^{d-1}$ is the base polynomial for $y = x \bmod n$ then $|y_i| < b$ for all $i = 0, 1, \ldots, d-1$. Since $r \geqslant d$, $y(t) \in \mathcal{B}_b^d \subset \mathcal{B}_r^d \Rightarrow (y, y(t)) \in \mathcal{A}$ ■

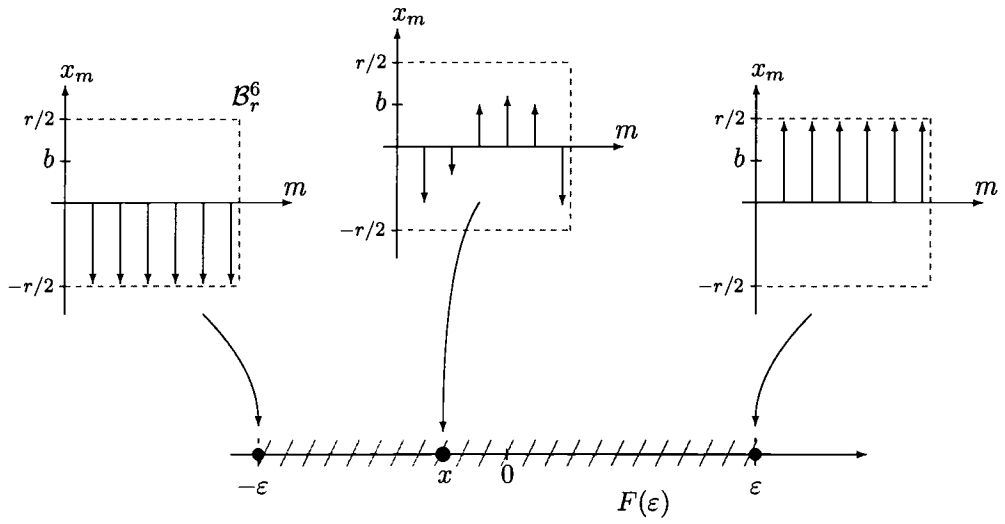

FIGURE 3.2. Evaluation map sends the polynomial frame $\mathcal{B}_r^d$ to an integer frame $F(\varepsilon)$.

Definition 22 *We call the base polynomial* $y(t)$ *of* $y = x \bmod n$ *as the* **spectral (modular) reduction** *of* $(x, x(t))$ *with respect to* $(n, n(t))$ *and we simply write*

$$y(t) = x(t) \ mods \ n(t).$$

Moreover the expression

$$y(t) \equiv x(t) \ mods \ n(t)$$

mean n *divides the evaluation of* $(x(t) - y(t))$ *at base* b.

The spectral reduction can be viewed as a projection of the usual modular operation in $\mathbb{Z}$ to the set of (evaluation) polynomials. Clearly, it is defined over the polynomials but it is different from the standard modular reduction in $\mathbb{Z}[t]$. To indicate this difference, in place of "mod" we choose to use "mods" where "s" stands for spectral.

Similar objects can be defined for spectral polynomials; however, we note that unlike time polynomials, evaluation of spectral polynomials do not have any special meaning that serves our needs. To be specific for a spectral polynomial $X(t)$, $X(b)$ does not have a special meaning, where $x(b)$ mostly represents an input integer. Therefore, our derivation for spectral polynomials is a canonical continuation of the notation that is developed for time polynomials.

Definition 23 *Let $x(t)$ be a base polynomial for $b > 0$ of an integer x. We call the spectral polynomial $X(t)$, the transform pair of $x(t)$, the* **spectral base polynomial**.

Definition 24 *Let $y(t)$ be an almost spectral reduction of $x(t)$ with respect to $n(t)$ in some frame $\mathcal{B}_r^d$. The spectral polynomial $Y(t)$, transform pair of $y(t)$, is called the* **almost spectral reduction of $X(t)$ with respect to $N(t)$** *where $(X(t), x(t))$ and $(N(t), n(t))$ are transform pairs.*

Definition 25 *We call the base polynomial $Y(t)$ in the spectrum the* **spectral modular reduction of $X(t)$ with respect to $N(t)$** *and we write*

$$Y(t) = X(t) \ mods \ N(t).$$

Moreover the expression

$$Y(t) \equiv X(t) \ mods \ N(t)$$

means n divides the evaluation of $IDFT((X(t) - Y(t))$ at base b.

In Figure 3.3, we present a visual description of the above definitions for signals of length 6. We assume the line on the bottom is the $\mathbb{Z}$ line, the region $[-\varepsilon, \varepsilon]$ represents the frame $F(\varepsilon)$ and the shaded region shows $\mathbb{Z}_n$. The set of solid points gives an almost modular reduction set of x with respect to n, hence any solid point y satisfies

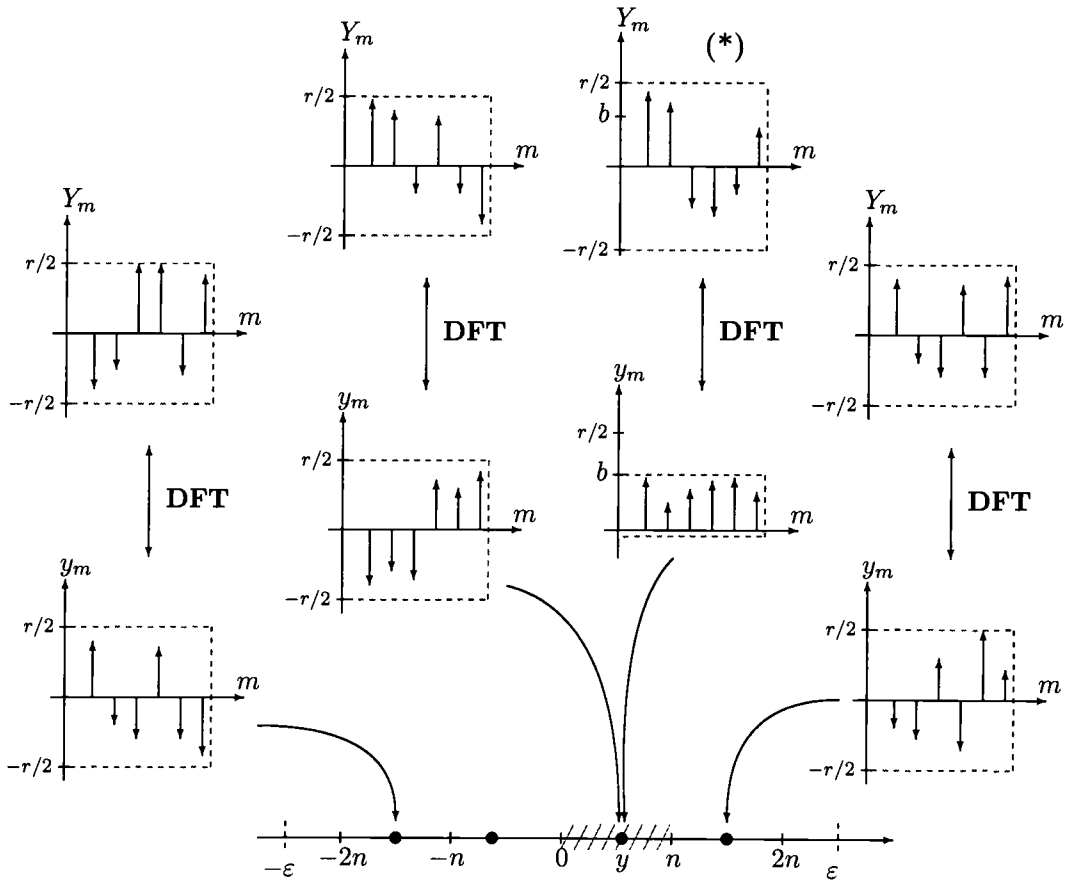

FIGURE 3.3. A description of definitions (21) through (25). (*) represents the spectral modular reduction.

$y \equiv x \bmod n$. The four time signals represent the evaluation polynomials for these solid points. Observe that all of the polynomials are elements of the polynomial frame $\mathcal{B}_r^d$ and they all belong to the almost spectral reduction set. The above four spectral signals are transform pairs of the time signals and specifically the signal labeled with (*) represents the spectral modular reduction of an $X(t)$ with respect to some $N(t)$ of degree 6.

3.5. Time Simulation of Spectral Modular Reduction

The spectral reduction can easily be achieved by deducing the base polynomial of the usual modular reduction ($y = x \bmod n$). But with such an approach one needs to perform classical modular reduction routines, which do not have simple spectral meanings. Our next step is to give a description of an algorithm that computes an almost spectral reduction of an evaluation polynomial. But, before presenting the algorithm we state the relation of the set $\mathcal{A}$ with time frame $F(\varepsilon)$.

Lemma 3 *Let $\mathcal{A}$ be the set defined in Definition 21. Then $\mathcal{A}/\sim$ gives an almost modular reduction set in the time frame $F(\varepsilon)$ for some $\varepsilon > 0$.*

Proof: Let $n > 0$ and $d = \deg(n(t))$. The set $\mathcal{A}$ is defined to be the set of all evaluation polynomials of $y \equiv x \bmod n$ in the frame $\mathcal{B}_r^d$, $(\mathcal{A}/\sim) \subset F(\varepsilon)$ by Lemma 2 and since all evaluations satisfy $y \equiv x \bmod n$, $\mathcal{A}$ is an almost modular reduction set of of n. ■

Algorithm 3 *Time Simulation of Spectral Reduction Algorithm*
Suppose that n and b are positive numbers with $gcd(b, n) = 1$, $n(t)$ is the base evaluation polynomial of n with degree $d-1$ and $x(t)$ is an evaluation polynomial of x with degree $e \geqslant d$.
Input: *$x(t)$ and $n(t)$.*
Output: *$y(t)$, an almost spectral modular reduction of $x(t)$ with respect to $n(t)$.*

1: *Compute $\overline{n} = \rho n$ such that $\overline{n}_d = 1$ and $|\overline{n}_i| < b/2$*
2: *Compute $\underline{n} = \delta n$ such that $\underline{n}_0 = 1$ and $|\underline{n}_i| < b/2$*
3: $y(t) = x(t) \cdot t^d$
4: **for** $i = 0$ **to** e *(degree reduction)*

5: $y(t) := y(t) - y_{e-i} \cdot \overline{n}(t) \cdot t^{e-i}$

6: **end for**

7: $\alpha := 0$

8: **for** $i = 0$ **to** $d - 1$ *(radius reduction)*

9: $\beta := (y_0 + \alpha)$ **rem** b

10: $\alpha := y_0$ **div** b

11: $y(t) := y(t) - \beta \cdot \underline{n}(t)$

12: $y(t) := (y(t) - y_0)/t$

13: **end for**

14: $y(t) := y(t) + \alpha(t)$, *where* $\alpha(t)$ *is the base polynomial for* α.

15: **return** $y(t)$

In Figure 3.4 we illustrate the steps of the Algorithm 3. In Step 1 and 2; we compute the multiples of the modulus n that is used in reduction steps. The computation of $\overline{n} = \rho n$ can be performed as follows:

- multiply n with a power of two such that $b > \overline{n}_d \geqslant b/2$,
- deduce the base polynomial $\overline{n}(t)$ of $\overline{n}$ by breaking $\overline{n}$ into words of $\lceil \log(b) \rceil$ bits.
- encode the base polynomial $\overline{n}(t)$ by the following procedure:

  Procedure 1

  1: **for** $i = d - 1$ **to** 0

  2: if $\overline{n}_i > b/2$ then

  3: $\overline{n}_i := \overline{n}_i - b$

  4: $\overline{n}_{i+1} := \overline{n}_{i+1} + 1$

  5: **return** $y(t)$

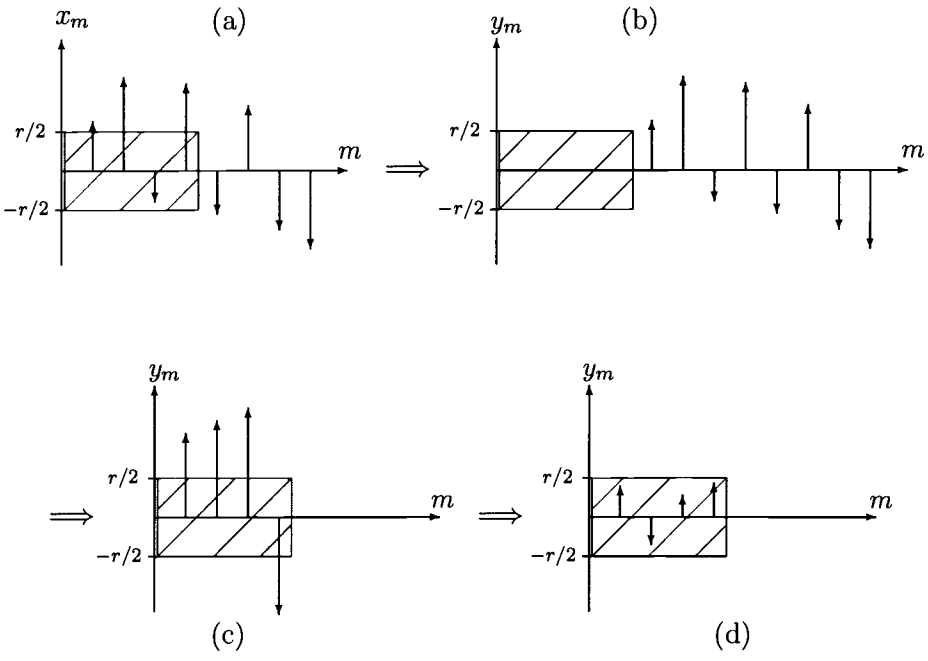

FIGURE 3.4. The time simulation of spectral reduction algorithm; **(a)** A time sequence of length 8. **(b)** Step 3; the extension of $x(t)$ **(c)** Step 4–6; degree reduction, **(d)** Step 8–13; radius reduction, fits the signal into the $\mathcal{B}_r^4$ frame for some $r > 0$.

which gives the desired polynomial with $|\overline{n}_i| < b/2$ for $i = 0, 1, \ldots, d-1$.

The computation of $\underline{n}(t)$ is similar. Since $\gcd(b, n) = 1$, $\delta = n_0^{-1} \bmod b$ exists and can be found by using the extended Euclidean algorithm. By employing a similar encoding to the base polynomial of δn, the desired polynomial $\underline{n}(t)$ is computed with $\underline{n}_0 = 1$ and $|\underline{n}_i| < b/2$, $i = 1, 2, \ldots, d-1$.

At Step 3; we extend the input polynomial (this corresponds to a d-word shift with computer arithmetic terms). The reason for this extension becomes clear once we discuss the radius reduction steps. The first **for** loop implements a classical polynomial degree reduction of $x(t) \cdot t^d$ with respect to $\overline{n}(t)$.

Steps 7–15 play the most important role in the algorithm. Observe that radius reduction is very similar to a Montgomery reduction for binary polynomials [13]. Thus if $y(t)$ is the input to this core, the core computes the reduction $y(t) \cdot t^{-d}$ but since our input was $x(t) \cdot t^d$, the output of the whole algorithm becomes $x(t) \cdot t^d \cdot t^{-d}$. Moreover the carry-save methodology brings the magnitudes of the coefficient of the signals to fit into a polynomial frame. The boundaries of radius and degree of this frame are traced in the following theorem;

Theorem 6 *Algorithm 3 computes an almost spectral reduction $y(t) \equiv x(t)$ mods $n(t)$ such that the output signal $y(t)$ fits into the polynomial frame $\mathcal{B}_r^d$ where $r = b^2d + b$.*

Proof: First of all the algorithm computes an evaluation polynomial $y(t)$ such that $y(t) \equiv x(t)$ mods $n(t)$. This is seen as follows; at only Steps 5, 11 and 12 the value of $y(t)$ is accumulated either by adding a multiple of $n(t)$ or a word shift. Since we work in the ring $\mathcal{B}$ adding or subtracting a multiple of $n(t)$ do not change the residue class that $y(t)$ belongs to. On the other hand, the word shift in Step 12 is performed after eliminating the least significant b-bits of y_0 (i.e. by adding a proper multiple of $n(t)$ to $y(t)$ at Step 11) and passing the necessary carry information to the successive digit. Such a trivial operation does not change the residue class that $y(t)$ belongs to. Therefore $y(t)$ satisfies $y(t) \equiv x(t)$ mods $n(t)$.

When it comes to figure out which frame $y(t)$ belongs to, let $x(t) = x_0 + x_1 t + \ldots + x_{e-1}t^{e-1}$ be an evaluation polynomial with $|x_i| < u$ for all $i = 0, 1, \ldots, e-1$ and let $\underline{n}(t) = 1 + \underline{n}_1 t + \ldots + \underline{n}_d t^d$ and $\overline{n}(t) = \overline{n}_0 + \overline{n}_1 t + \ldots + 1 \cdot t^d$ be evaluation polynomials of some multiples of $n(t)$ (i.e. with $|\underline{n}_i|, |\overline{n}_i| < b/2$ for all $i = 0, 1, \ldots, d$). Observe that, the degree reduction part of the algorithm implements a classical polynomial reduction of $x(t) \cdot t^d$ with respect to $\overline{n}(t)$ and after the first **for** loop, the degree of the polynomial

$y(t)$ drops from $d+e$ to $d-1$, on the contrary the magnitude of the coefficients can possibly be large.

No matter how big these coefficients are, in the radius reduction loop, we decrease the radius by an add-shift methodology which is described as follows. At every run of the loop we set $y_0 = 0$ (i.e. by adding a proper multiple of $\underline{n}(t)$ to $y(t)$ at Step 11 and passing the necessary carry information to the successive digit) and then divide by t (which corresponds to $\lceil \log_2(b) \rceil$-bit shifts). At every next run of the loop, y_d satisfies $|y_d| < \beta \cdot |\underline{n}_d| < b^2/2$ because $y(t)$ drops degree from d to $d-1$ whenever divided by t in the radius reduction. Since $e > d$ after d steps of reduction y_0 has the maximum accumulation which decides the following bound

$$|y_i| < \frac{b^2 d}{2} \quad \text{for all } i = 0, 1, \ldots, d-1$$

The only detail left is the last carry (i.e. α), α can significantly change the bound for y_0 if it is large and is directly added to y_0 (i.e. $y_0 := y_0 + \alpha > b^2 d$). But instead of adding it straightaway if an encoded base polynomial of α is added to the $y(t)$, this bound is iterated to $|y_i| < (b^2 d + b)/2$. Thus, $y(t)$ fits into $\mathcal{B}_r^d$ with $r = b^2 d + b$. Note that, here we assume $\deg(\alpha(t)) < d$ and $|\alpha_i| < b/2$. ■

Algorithm 3 presents a direct reduction method, instead it is possible to use some indirect reduction methodology, such as make use of the Montgomery's trick [14]. Montgomery trick is a method which allows efficient implementation of modular arithmetic operations without explicitly carrying out the classical modular reductions. Indeed, it replaces the modular reduction by a multiplication and some trivial shifts. The trick is instead of attacking to compute the x mod n directly, it proposes to derive it after performing a related computation

$$x \cdot \tau^{-1} \bmod n$$

for $\tau > n$ and $\gcd(n, \tau) = 1$. At first glance this seems computationally pointless because of inversion involved but The selection of τ changes this first impression drastically. After giving some related notation, with Algorithm 4 we employ such a methodology.

Notation 3 *The polynomial product $x(t) \cdot t^e$ is denoted by $x^e(t)$, so in this context*

$$x^{-e}(t) = x(t) \cdot t^{-e}$$

Algorithm 4 *Time Simulation of Spectral Reduction Algorithm (Montgomery type)*
Suppose that n and b are positive numbers with $gcd(b, n) = 1$, $n(t)$ is the base evaluation polynomial of n with degree $d-1$ and $x(t)$ is an evaluation polynomial of x with degree $(e-1) \geqslant (d-1)$.

Input: *$x(t)$ and $n(t)$.*

Output: *$y(t)$, an almost spectral reduction of $x^{-e}(t)$ with respect to $n(t)$.*

1: *Compute $\underline{n} = \delta n$ such that $\underline{n}_0 = 1$ and $|\underline{n}_i| < b/2$*

2: $y(t) := x(t)$

3: $\alpha := 0$

4: **for** $i = 0$ **to** $e - 1$

5: $\beta := (y_0 + \alpha)$ **rem** b

6: $\alpha := y_0$ **div** b

7: $y(t) := y(t) - \beta \cdot \underline{n}(t)$

8: $y(t) := (y(t) - y_0)/t$

9: **end for**

10: $y(t) := y(t) + \alpha(t)$, *where $\alpha(t)$ is the base polynomial for α.*

11: **return** $y(t)$

Algorithm 4 reduces the degree while reducing the radius since the core of the Algorithm 4 and the radius reduction part of the Algorithm 3 are identical. An other salient feature of Algorithm 4 is that it also bounds the coefficients of the intermediate values quite desirably which literally described in Section 3.2.3. We present these results in the next theorem.

Theorem 7 *Algorithm 4 computes an almost spectral reduction* $y(t) \equiv x^{-e}(t)$ *mods* $n(t)$ *such that the output signal* $y(t)$ *has the form*

$$y(t) = y_0 + y_1 t + y_2 t^2 + \ldots + y_{d-1} t^{d-1}, \quad \textit{where } |y_i| < \frac{(d-i)b^2 + b}{2}$$

that fits into the polynomial frame $\mathcal{B}_r^d$ *where* $r = b^2 d + b$. *Moreover, if* $|x_i| < u/2$ *for all* $i = 1, 2, \ldots, e-1$ *then the coefficients of the intermediate values satisfy* $|y_i| < (u + b^2 d)/2$ *for* $i = 0, 1, \ldots, d-1$.

Proof: The algorithm computes the almost spectral reduction of $x^{-e}(t) = x(t) \cdot t^{-e}$; since the loop runs e times that implies we divide by t exactly e times. if $|x_i| < u$ for all $i = 0, 1, \ldots, e-1$, at every step of the loop, y_{d-i} would satisfy $|y_{d-i}| < \beta \cdot |\underline{n}_d(i+1)| + u/2 < (b^2(i+1) + u)/2$ where $i = 0, 1, \ldots, d-1$. A bound for $|y_{d-i}|$ is attained when $i = d-1$ which is $|y_i| < (b^2 d + u)/2$ for $i = 0, 1, \ldots, d-1$.

In particular, if the last $d-1$ steps of reduction is considered $y(t)$ takes the form

$$y(t) = y_0 + y_1 t + y_2 t^2 + \ldots + y_{d-1} t^{d-1}, \quad \text{where } |y_i| < \frac{(d-i)b^2 + b}{2}$$

for $i = 0, 1, \ldots, d-1$. ■

Note that both Algorithm 3 and 4 describe reduction routines of an arbitrary evaluation polynomial (i.e. of degree larger than or equal to $d-1$) which do not really address our targeted problems. To be specific, in general modular exponentiation is

performed by a multiply-reduce methodology, so we practically deal with reducing polynomials into single sizes frames from approximately doubled ones. In order to cooperate with transform parameter related discussions, we give a corollary to Theorem 7 by fixing the degree of $x(t)$ to $d-1$ and the degree of the $n(t)$ to $\lceil \frac{d}{2} \rceil - 1$.

Corollary 3 *Let $n(t)$ be a base b polynomial with degree $s-1$ such that $s = \lceil d/2 \rceil$ and let $x(t) \in \mathcal{B}_r^d$ where $r = sb^2$. Algorithm 4 computes an almost spectral reduction, $y(t) \equiv x(t) \cdot t^{-d}$ mods $n(t)$ in the polynomial frame $\mathcal{B}_{r'}^s$ where $r' = b^2 s + b$. Moreover, coefficients of all the intermediate values do not exceed $2b^2 s$.*

Proof: Let $n(t)$ be a base b polynomial with degree $s-1$ such that $s = \lceil d/2 \rceil$ and let $x(t) \in \mathcal{B}_r^d$ where $r = b^2 s$ then the coefficients of $x(t)$ satisfy $|x_i| < r/2 = b^2 s/2$ for all $i = 0, 1, \ldots, d-1$ (note that we take $x(t)$ with the maximum degree $d-1$ in order to find the upper bounds). First of all taking $e - 1 = d - 1$, the algorithm drops the $\deg(x(t))$ to $\deg(n(t) = s-1$ and computes the almost spectral reduction of $x_{-d}(t) = x(t) \cdot t^{-d}$ in the frame $\mathcal{B}_{r'}^s$. The radius $r' = b^2 s + b$ can be deduced by applying the Theorem 7, moreover since $|x_i| < b^2 s/2$ the intermediate values are bounded by

$$|y_i| < \frac{b^2 s}{2} + \frac{r}{2} = \frac{b^2 s + b^2 s}{2} = b^2 s$$

■

3.6. Spectral Modular Reduction in a Complex Spectrum

Now, it is time to translate the spectral modular reduction algorithm into the spectrum. Observe that the extension step in Algorithm 3 forces us to use larger transform lengths. For convenience we only transform Algorithm 4 which do not require any extensions, but with a little bit more work similar steps can be employed for Algorithm 3 as well. Firstly, we translate the time simulation (i.e. Algorithm 4 into a

complex spectrum followed by a translation to a finite ring spectrum. The translation is performed line by line by using the dictionary, established earlier in Section 2.1.3. We start by adapting some notation, in particular Notation 1, to our polynomial based presentation;

Notation 4 *Let ω be a principal d-th root of unity, we denote the sequences $\{\Omega_k\}$ and $\{\Gamma_k\}$ by*

$$\Omega(t) = 1 + \omega^1 t + \omega^2 t^2 + \ldots + \omega^{(d-1)} t^{d-1} \quad \textit{and}$$
$$\Gamma(t) = 1 + \omega^{-1} t + \omega^{-2} t^2 + \ldots + \omega^{-(d-1)} t^{d-1}$$

in polynomial notation respectively.

Notation 5 *Additionally, recall that if $a \in \mathbb{Z}$ is a constant then $\{a\} = (a, a, \ldots, a)$ represents the constant sequence of length d. In polynomial notation we denote constant sequences by*

$$a(t) = a + at + at^2 + \ldots + at^d$$

Algorithm 5 *Spectral Reduction Algorithm (in a complex spectrum)*
Suppose that there exist a DFT of length e over $\mathbb{C}$ and

$$x(t) \xleftrightarrow{DFT} X(t) \qquad \underline{n}(t) \xleftrightarrow{DFT} \underline{N}(t)$$

where $x(t)$ is an evaluation polynomial of x with degree $e-1$ and $\underline{n}(t)$ is a degree $d \leqslant e$ evaluation polynomial of a multiple of modulus n such that $\underline{n}_0 = 1$ (we assume $gcd(b, n) = 1$ therefore such a multiple always exists).
Input: *$X(t)$ and $\underline{N}(t)$, spectral polynomials*
Output: *$Y(t) \equiv X^{-e}(t)$ mods $N(t)$,*

1: $Y(t) := X(t)$

2: $\alpha := 0$

3: **for** $i = 0$ **to** $e - 1$

4: $\quad y_0 := e^{-1} \cdot (Y_0 + Y_1 + \ldots + Y_e)$

5: $\quad \beta := (y_0 + \alpha)$ **rem** b

6: $\quad \alpha := (y_0 + \alpha)$ **div** b

7: $\quad Y(t) := Y(t) - \beta \cdot \underline{N}(t)$

8: $\quad Y(t) := Y(t) - (y_0 - \beta)(t)$

9: $\quad Y(t) := Y(t) \odot \Gamma(t)$

10: **end for**

11: $Y(t) := Y(t) + A(t)$, *where* $A(t)$ *is the DFT pair of the base polynomial of* α.

12: **return** $Y(t)$

In Figure 3.5 we illustrate the relation between time and spectrum algorithms. Notice that input and output of spectral polynomials do not show any correspondence, once the time simulation considered the action of the spectral reduction algorithm recognized.

Our next step is proving that Algorithm 4 and 5 agree; in other words there exists a DFT relation between the intermediate and output data in two domains at all times. In a complex spectrum it is simpler to show this relation since the transform domain is the time frame $\mathcal{B}^d$ which admits no radius overflows. Now let's formalize what we have said.

Theorem 8 *Algorithm 5 computes the almost spectral reduction,* $Y(t) \equiv X^{-e}(t)$ *mods* $N(t)$ *such that the inverse of the output signal* $Y(t)$ *gives* $y(t) \equiv x^{-e}(t)$ *mods* $n(t)$ *(i.e. the output of the Algorithm 4).*

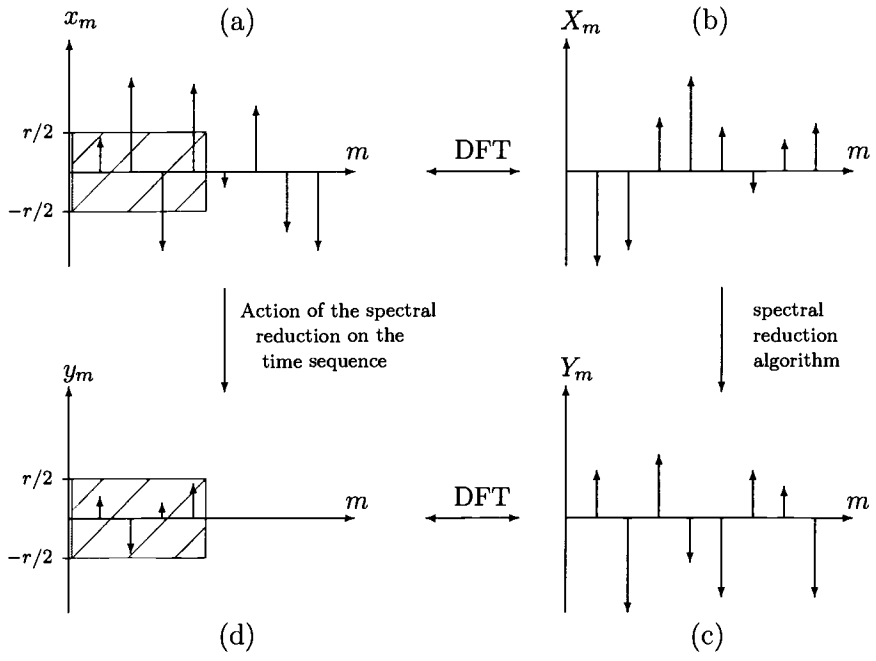

FIGURE 3.5. The action of the spectral reduction algorithm on the time sequence **(a)** A time sequence of length 8, possibly a result of a convolution. **(b)** DFT of $\{x_m\}$ **(c)** the signal after the spectral reduction algorithm applied **(d)** the inverse DFT of spectral reduction which also shows the desired time simulation of spectral reduction.

Proof: We need to show that Algorithm 4 and 5 generate agreeing intermediate and output values.

Let $(x(t), X(t))$ and $(\underline{n}(t), \underline{N}(t))$ are transform pairs. In Step 4 $y_0 := e^{-1} \cdot (Y_0 + Y_1 + \ldots + Y_e)$ computes the first coefficient of the time polynomial $y(t)$ (by using the property (iii) of Section 2.1.3). Note that in Algorithm 4, y_0 comes freely. Once y_0 is computed in Step 5 and 6 the parameters β which is the actual value of the constant term and the next carry α are generated.

In Step 7, β multiple of $\underline{N}(t)$ is subtracted from $Y(t)$, this updates $Y(t)$ such that the new $y_0 = 0 \bmod b$. By linearity this is equivalent to Step 6 of Algorithm 4.

Since carry α is saved to be added to the consecutive digit in the next run of the loop, a division by t can be performed. Before this shift we need to eliminate the contribution of y_0 to the spectral polynomial $Y(t)$ completely. But as Step 7 changed y_0 to $y_0 - \beta$, the computation of $(Y(t) - (y_0 - \beta)(t))$ in Step 8 sets zeroth time term of $Y(t)$ to zero (observe that $(y_0 - \beta) \in \mathbb{Z}$ is a constant so $(y_0 - \beta)(t)$ is a constant polynomial, see Notation 5. If this is followed by the component-wise multiplication with $\Gamma(t)$ polynomial, Step 8 and 9 implement a circular shift (property (iv) of Section 2.1.3).

Therefore Algorithm 4 and 5 generate transform pairs. As Algorithm 4 computes $y(t) \equiv x^{-e}(t)$ mods $n(t)$, Algorithm 5 computes $Y(t) \equiv X^{-e}(t)$ mods $N(t)$. ■

Algorithm 5 presents a reduction in the complex spectrum. If an infinite precision for representing the complex numbers is used this algorithm works; however, because of the obvious reasons we are limited to finite precisions. Hence every operation on the spectral polynomials comes with some additional round off errors. For single convolutions, these errors are controlled by adjusting the precision. But when it comes to work mostly in the spectrum one needs very large precisions. For the time being we stop working in the complex spectrum and leave these analysis for a future research.

As we discuss earlier, exact calculations are possible if one switches to work in a finite ring spectrum. Next section raises the methods for an almost spectral reduction calculation in these spectra in which the round off errors are not issues any longer but overflows naturally stay in our center of attention.

3.7. Spectral Modular Reduction in a Finite Ring Spectrum

While exposing the convolutions over finite rings we urge to control the two parameters namely transform lengths and overflows. We noted that if the radius or the degree of the intermediate values go out of predefined bounds the spectral algorithms produce erroneous outputs. Unlike convolution the spectral reduction does not extend the degree (excluding Algorithm 3) thus, transform lengths do not have any ill-favored effect on the contrary radius overflows can still be problematical.

We have already demonstrated some examples of overflows for convolutions. A careless translation of the Algorithm 4 to the spectrum causes overflows of some different kind namely generated by the subtraction step (i.e. Step 7) of the Algorithm 4. Let us give an example;

Example 10 *Let $b = 4$, $x = 777$ and $n = 55$. If the evaluation polynomial $x(t) = -3 + 11t - 2t^2 - t^4 + t^5$ of x and the base b polynomial $n(t) = 3 + t + 3t^2$ of n are considered, Theorem 8 states that the Algorithm 4 outputs an almost spectral modular reduction in the frame $\mathcal{B}_r^d$ where the radius $r = b^2 d$. Since Algorithm 4 works totally in time, choosing the correct frame does not cause any overflow. But when a DFT that respects the convolution is considered, the situation may become tricky because of a carelessly chosen spectrum.*

If we fix $d = 6$, we have $r = 4^2 \cdot 6 = 116$ and for $q = 127 > r$ the range of the DFT map becomes the Fourier ring $\mathcal{F}_{127}^6$. Recall that since $\mathcal{F}_q^d \approx \mathbb{Z}_q^d$, the operation on spectral coefficients are performed in the integer ring $\mathbb{Z}_q$ (i.e. the usual "mod q"). This implies working with

$$x(t) = 124 + 11t + 125t^2 + 126t^4 + t^5 \pmod{127}$$

which is an evaluation polynomial of 35448 *but not for* 777. *Therefore any method fails to work from that moment since the input evaluation polynomial* $x(t)$ *has changed during the initial embedding to* $\mathcal{F}_{127}^2$

We suggest two methods to overcome such overflow propagations. The first one advances a use of least magnitude representation for $\mathbb{Z}_q$ where the second one proposes to use the positive polynomial frame $\mathcal{H}_q^d$ (signals are positive at all times, see Definition 12) plus some related changes in the time simulation.

3.7.1. The Use of a Least Magnitude Representation for $\mathbb{Z}_q$

Algorithm 4 can be embedded to the finite ring spectrum $\mathcal{F}_q^d \approx \mathbb{Z}_q^d$ where $\mathbb{Z}_q$ is represented by a least magnitude representation.

Algorithm 6 *Spectral Reduction Algorithm (in a finite ring spectrum)*
Suppose that there exist a DFT map $DFT_d^\omega : \mathcal{B}_q^e \rightarrow \mathcal{F}_q^e$ *and*

$$x(t) \xleftrightarrow{DFT} X(t) \qquad \underline{n}(t) \xleftrightarrow{DFT} \underline{N}(t)$$

where $(x(t), x) \in \mathcal{B}_r^e$ *for* $r < q - b^2 d$ *and* $(\underline{n}(t), \underline{n}) \in \mathcal{B}_b^{d+1}$ *such that* $deg(n(t)) = d \leqslant e$ *and* $\underline{n}$ *is a multiple of modulus* n *with* $\underline{n}_0 = 1$ *(we assume* $gcd(b, n) = 1$*).*
Input: $X(t)$ *and* $\underline{N}(t)$, *spectral polynomials*
Output: $Y(t) \equiv X^{-e}(t)$ *mods* $N(t)$,

1: $Y(t) := X(t)$
2: $\alpha := 0$
3: **for** $i = 0$ **to** $e - 1$
4: $\quad y_0 := e^{-1} \cdot (Y_0 + Y_1 + \ldots + Y_e) \bmod_{\mathcal{R}} q$
5: $\quad \beta := (y_0 + \alpha)$ **rem** b

6: $\alpha := (y_0 + \alpha)$ **div** b

7: $Y(t) := S(t) - \beta \cdot \underline{N}(t) \bmod_{\mathcal{R}} q$

8: $Y(t) := Y(t) - (y_0 - \beta)(t) \bmod_{\mathcal{R}} q$

9: $Y(t) := Y(t) \odot \Gamma(t) \bmod_{\mathcal{R}} q$

10: **end for**

11: $Y(t) := Y(t) + A(t)$, *where* $A(t)$ *is the DFT pair of the base polynomial of* α.

12: **return** $Y(t)$

Algorithm 6 is a spectral equivalent procedure of the Algorithm 4. Theorem 8 gives the boundaries of the time simulation while carrying the spectral steps. If the parameter q is chosen large enough that any coefficient of time signal fits completely into the frame $\mathcal{B}_q^e$, no overflows occur and Algorithm 6 and 4 produce the transform pairs. Let us state this formally;

Theorem 9 *Algorithm 6 computes the almost spectral reduction,* $Y(t) \equiv X^{-e}(t)$ *mods* $N(t)$ *such that the inverse of the output signal* $Y(t)$ *gives* $y(t) \equiv x^{-e}(t)$ *mods* $n(t)$ *(i.e. the output of the Algorithm 4).*

Proof: Notice the Algorithm 5 and 6 are similar except the modular arithmetic carried in Steps 4, 7 and 8. Hence the analysis we did in Theorem 8 is valid. We need to prove that no overflows occurs. First of all by using the least magnitude residue representation for $\mathbb{Z}_q$ the overflows of kind described in Example 10 do not occur. When it comes to the ones which occur because of magnitude violations, the Theorem 7 is applied. We assume $\deg(x(t)) = e$ for $x(t) \in \mathcal{B}_r^e$ which implies that $|x_i| < r/2 = (q - b^2 d)/2$ for $i = 0, 1, \ldots, e-1$ and since $\underline{n}$ is a multiple of modulus n with $\underline{n}_0 = 1$. We conclude by Theorem 7 that the intermediate values and the output $y(t)$ of the time simulation are bounded by

$$|y_i| < \frac{r + b^2 d}{2} = \frac{q - b^2 d + b^2 d}{2} = \frac{q}{2}$$

which states that Algorithm 4 and 6 generate the transform pair $y(t)$ and $Y(t)$. As Algorithm 4 computes $y(t) \equiv x^{-e}(t)$ mods $n(t)$, Algorithm 6 performs $Y(t) \equiv X^{-e}(t)$ mods $N(t)$. ■

3.7.2. The Use of Positive Frames

An other method of by-passing the overflows of type illustrated in Example 10 is to use the positive polynomial frame $\mathcal{H}_q^d$. This needs some some minor modifications in the time simulation (i.e. Algorithm 4) without touching the $\mathbb{Z}_q$ representation. These overflows are caused mainly subtraction of Step 7 which is;

7: $\quad y(t) := y(t) - \beta \cdot \underline{n}(t) \bmod q$

once any coefficient of $y(t)$ becomes negative, standard representation of $\mathbb{Z}_q$ force us to bring these negative coefficients to $[0, q-1]$ range. Notice that, the role of Step 7 is setting the least significant $\lceil \log(b) \rceil$-bits of y_0 to zero by subtracting $\beta \cdot \underline{n}(t)$ from $y(t)$. But the same engagement can be done by adding a $-\beta \bmod b$ multiple of $\underline{n}(t)$ to $y(t)$, with a slight change in carry to the next digit. This corresponds to the following modification of the Steps 5–7 of the time simulation Algorithm 4;

5': $\quad \beta := -(y_0 + \alpha) \bmod b$

6': $\quad \alpha := (y_0 + \alpha + \beta)$ **div** b

7': $\quad y(t) := y(t) + \beta \cdot \underline{n}(t) \bmod q$

If we translate the time simulation with the above modification we get the following spectral algorithm. Notice that unlike Algorithm 6 here the coefficients of

the spectral polynomials are positive at all times and the usual "mod" operation is used.

Algorithm 7 *Spectral Reduction Algorithm (in a finite ring spectrum)*
Suppose that there exist a DFT map $DFT_d^{\omega}: \mathcal{H}_q^e \rightarrow \mathcal{F}_q^e$ *and*

$$x(t) \xleftrightarrow{DFT} X(t) \qquad \underline{n}(t) \xleftrightarrow{DFT} \underline{N}(t)$$

where $(x(t), x) \in \mathcal{H}_r^e$ *for* $r < q - b^2 d$ *and* $(\underline{n}(t), \underline{n}) \in \mathcal{H}_b^{d+1}$ *such that* $deg(n(t)) = d \leqslant e$ *and* $\underline{n}$ *is a multiple of modulus* n *with* $\underline{n}_0 = 1$ *(we assume* $gcd(b, n) = 1$*).*

Input: $X(t)$ *and* $\underline{N}(t)$*, spectral polynomials*

Output: $Y(t) \equiv X^{-e}(t)$ *mods* $N(t)$*,*

1: $Y(t) := X(t)$

2: $\alpha := 0$

3: **for** $i = 0$ **to** $e - 1$

4: $\quad y_0 := e^{-1} \cdot (Y_0 + Y_1 + \ldots + Y_d) \bmod q$

5: $\quad \beta := -(y_0 + \alpha) \bmod b$

6: $\quad \alpha := (y_0 + \alpha + \beta)$ **div** b

7: $\quad Y(t) := Y(t) + \beta \cdot \underline{N}(t) \bmod q$

8: $\quad Y(t) := Y(t) - (y_0 + \beta)(t) \bmod q$

9: $\quad Y(t) := Y(t) \odot \Gamma(t) \bmod q$

10: **end for**

11: $Y(t) := Y(t) + A(t)$*, where* $A(t)$ *is the DFT pair of the base polynomial of* α*.*

12: **return** $Y(t)$

Theorem 10 *Algorithm 7 computes the almost spectral reduction,* $Y(t) \equiv X^{-e}(t)$ *mods* $N(t)$ *such that the inverse of the output signal* $Y(t)$ *gives* $y(t) \equiv x^{-e}(t)$ *mods* $n(t)$ *(i.e. the output of the Algorithm 4).*

Proof: By using the above analysis and the similarity of Algorithm 7 to 5, we conlcude that 7 computes an almost spectral reduction; however, we still need to find the optimal domain for the algorithm since we have changed the domain to $\mathcal{H}_r^e$. We assume $\deg(x(t)) = e$ for $x(t) \in \mathcal{H}_r^e$ which implies that $0 \leqslant x_i < r = (q - b^2 d)$ for $i = 0, 1, \ldots, e-1$. Since $\underline{n}$ is a multiple of modulus n with $\underline{n}_0 = 1$, we conclude by Theorem 7 that the intermediate values and the output $y(t)$ of the time simulation bounded by

$$0 \leqslant y_i < r + b^2 d = q - b^2 d + b^2 d = q$$

Therefore, no overflows occur, Algorithm 4 and 7 generate the transform pair $y(t)$ and $Y(t)$. As Algorithm 4 computes $y(t) \equiv x^{-e}(t)$ mods $n(t)$, Algorithm 7 performs $Y(t) \equiv X^{-e}(t)$ mods $N(t)$. ■

With Algorithm 7 we have completed our primary discussion on spectral modular reduction. We leave the improvement related comments to Chapter 4. Notice that our presentation so far targets the reduction of an arbitrary evaluation polynomial of degree e with respect to a base polynomial of degree $d < e$. In the next section we change this routine and target to reduce an evaluation polynomial which is a result of a convolution. By this, we introduce the spectral modular multiplication.

3.8. Spectral Modular Multiplication (SMM)

Convolutions and the spectral reduction algorithms can easily be combined to harvest a spectral modular multiplication algorithm in a finite ring spectrum. For convenience we present SMM with the reduction Algorithm 7 but Algorithm 6 can also be used instead. Note that we consider the SMM for only finite ring spectrum.

In order to have a clear presentation we divide the spectral multiplication algorithm into 3 sub-procedures as seen in Figure 3.6. Note that the initial and final

stages consist of some data arrangements where the procedure, *spectral modular product* (SMP) consists of the actual multiplication and reduction steps (i.e. convolution and spectral reduction). Later, while presenting the spectral exponentiation algorithm, SMP is going to be the basic building block. SMP procedure and SMM are given as follows:

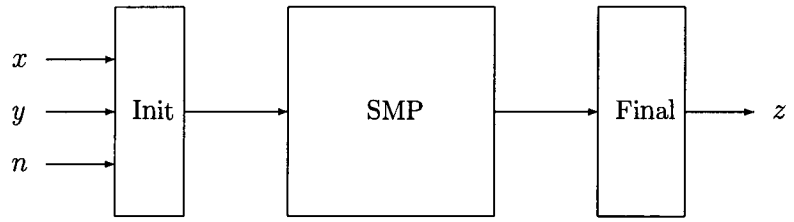

FIGURE 3.6. Spectral modular multiplication.

Algorithm 8 *Spectral Modular Product*

Suppose that there exist a DFT map $DFT_d^{\omega} : \mathcal{H}_q^d \rightarrow \mathcal{F}_q^d$, *and* $X(t), Y(t)$ *and* $\underline{N}(t)$ *be transform pairs of* $x(t), y(t)$ *and* $\underline{n}(t)$ *respectively where* $(x(t), x)$ *and* $(y(t), y)$ *are evaluation polynomials in the frame* $\mathcal{H}_r^s$ *with* $r > 0$ *and* $s = \lceil d/2 \rceil$, *and* $(\underline{n}(t), \underline{n}) \in \mathcal{H}_b^{s+1}$ *such that* $deg(\underline{n}(t)) \leqslant s$ *and* $\underline{n}$ *is a multiple of modulus* n *with* $\underline{n}_0 = 1$ *(we assume* $gcd(b, n) = 1$*).*

Input: $X(t), Y(t)$ *and* $\underline{N}(t)$*; spectral polynomials*

Output: $Z(t) \equiv (X(t) \odot Y^{-d}(t))$ *mods* $N(t)$,

procedure *SMP*$(X(t), Y(t))$

1: $Z(t) := X(t) \odot Y(t)$

2: $\alpha := 0$

3: **for** $i = 0$ **to** $d - 1$

4: $z_0 := d^{-1} \cdot (Z_0 + Z_1 + \ldots + Z_d) \bmod q$

5: $\beta := -(z_0 + \alpha) \ mod \ b$

6: $\alpha := (z_0 + \alpha + \beta)/b$

7: $Z(t) := Z(t) + \beta \cdot \underline{N}(t) \bmod q$

8: $Z(t) := Z(t) - (z_0 + \beta)(t) \bmod q$

9: $Z(t) := Z(t) \odot \Gamma(t) \bmod q$

10: **end for**

11: $Y(t) := Y(t) + \alpha(t)$

12: **return** $Z(t)$

Algorithm 9 *(Spectral Modular Multiplication)*

Suppose that there exist a DFT map $DFT_d^\omega : \mathcal{H}_q^d \rightarrow \mathcal{F}_q^d$. *Let* $n(t)$ *be a base* b *polynomial for* n *where* $deg(n(t)) = s - 1$, $s = \lceil d/2 \rceil$ *and* $gcd(b, n) = 1$.

Input: *A modulus* $n > 0$ *and* $x, y < n$

Output: *an almost modular reduction* $z \equiv xy \bmod n$

1: Compute $\underline{n} = \delta \cdot n$ *such that the base polynomial* $\underline{n}(t)$ *has degree* d *and* $\underline{n}_0 = 1$

2: $\underline{N}(t) := DFT_d^\omega(\underline{n}(t))$

3: Compute the base polynomial $\lambda(t)$ *for* $\lambda = b^d \bmod n$.

4: Compute the base polynomial $x^d(t) = x(t)t^d$ *for* $x \cdot \lambda \bmod n$.

5: $X^d(t) := DFT_d^\omega(x^d(t))$

6: $Y(t) := DFT_d^\omega(y(t))$

7: $Z(t) := SMP(X^d(t), Y(t))$

8: $z(t) := IDFT_d^\omega(Z(t))$

9: **return** $z(b)$

Remark 9 *The Algorithm 9 outputs an almost modular reduction of* $xy \bmod n$. *If the reduction is desired in the range* $[0, n-1]$ *a final short reduction is needed.*

Lemma 4 *SMP*$(X^d(t), Y(t)) \equiv X(t) \odot Y(t)$ *mods* $N(t)$

Proof: Since SMP$(X^d(t), Y(t))$ computes the spectral coefficients of almost modular reduction, $Z(t) \equiv (X^d(t) \odot Y^{-d}(t))$ mods $N(t)$ hence taking the inverse transform gives

$$x^d(t)y^{-d}(t) = x(t) \cdot t^d y(t) \cdot t^{-d} = x(t) \cdot y(t) \text{ mods } n(t) \ .$$

■

Lemma 5 *The Procedure 8 computes the* $Z(t) \equiv (X(t) \odot Y^{-d}(t))$ *mods* $N(t)$ *if the parameters* b, q *and* s *satisfies the following inequality*

$$2sb^2 < q \tag{3.9}$$

Proof: The convolution and reduction are put together in SMP procedure. In the previous sections, we described the action of convolution and how the steps of reduction works. Here, we concentrate on driving the Inequality (3.9). Assume that the conditions of SMP are satisfied, we investigate the time simulation of the algorithm in order to trace the overflows. By using Theorem 7 we conclude that after convolution at Step 1 the time polynomial $z(t)$ doubles its degree to $2s-2$ and at the same time its coefficients increase upto sb^2 since $x(t)$ and $y(t)$ are base b polynomials (in other words $z(t) \in \mathcal{H}_r^d$ where $r = sb^2$). When it comes to analyze the reduction steps: applying Corollary 3 assures us that the output $z^{-d}(t) \in \mathcal{H}_{r'}^d$ where $r' = b^2 s + b$ and coefficients of all the intermediate values do not exceed $2b^2 s$. Therefore, if q chosen as $\max(2b^2 s, b^2 s + b) = 2b^2 s < q$, no overflows could be generated and SMP computes the

desired result. Note here that the carry added in Step 12 is no longer large because of working with a convolution output hence we take it as a constant rather than breaking it into words. ■

Theorem 11 *Algorithm 9 computes an almost modular reduction,* $z \equiv xy \bmod n$, *if the parameters* b, q *and* s *satisfies* $2sb^2 < q$.

Proof: If the steps of the algorithm are examined, it is seen that in initialization, before computing the Fourier coefficients, we compute $xb^d \bmod n$ (i.e. $x^d(t) \bmod n(t)$). Hence by Lemma 4 step computes the product $x(t)y(t)$ mods $n(t)$ unless overflows occur. Fortunately the initialization and finalization parts do not have anything to do with the coefficient bounds, hence it sufficies to find the minimal domain for the core SMP which is stated in Lemma 5 as $2sb^2 < q$. ■

The most important feature of the spectrum algorithms is the use of convolution theorem for multiplication operation. By some small arrangements, it is possible to transform the Steps 3–5 of Algorithm 9 into the into the spectrum. Such an organization decreases the area by excluding the usual modular multiplication steps in the initialization. the following fragments show such an arrangement:

3a: Compute the base polynomial $\lambda'(t)$ for $\lambda' = b^{2d} \bmod n$.
3b: $\Lambda'(t) := \mathrm{DFT}_d^{\omega}(\lambda'(t))$
4: $X(t) := \mathrm{DFT}_d^{\omega}(x(t))$
5: $\overline{X}(t) := \mathrm{SMP}(X(t), \Lambda'(t))$

Observe that, the output of Step 5 of the above modification gives a spectral evaluation $\overline{X}(t) = X^d(t)$ with a transform pair $x^d(t) \in \mathcal{H}_r^s$ with $r = 2b^2 s$ as described in Lemma 5 but if we consider the actual SMP (i.e. Algorithm 9), at Step 5 $X^d(t)$ is computed as the DFT of a base polynomial $x^d(t) \in \mathcal{H}_b^s$. Therefore, $2sb^2 < q$ bound does

not work for the above modification. We are going to come across with a consequent use of SMP procedure when we deal with modular exponentiation so we leave this analysis to the next section.

3.9. Spectral Modular Exponentiation

In general using a classical modular multiplication is superior for performing a single modular multiplication; however, SMM is very effective when several modular multiplications with respect to the same modulus are needed. In such a setup, if we keep the data in the Fourier domain at all times, the backward and forward transforms are by-passed as seen in Figure 3.7. An example is the case when one needs to compute a modular exponentiation, i.e., the computation of $m^e \bmod n$, where m, e and n are positive integers.

In fact, our motivation is centered around studying efficient algorithms and architectures for modular exponentiation. Now it is time to harvest what we have developed in the previous sections.

There are many methods for carrying general exponentiation. Most of the time the efficiency comes from two resources one is to decrease the time to multiply two operands; the other is to reduce the number of multiplications used to compute desired result. Generally, one do both. Notice that, until now our objective was reducing the modular multiplication which is categorized in the first group. For the rest of this study we keep this goal and simply put the objective of reducing the number of multiplications used out of scope, moreover, we consider using the binary method (see [14]) for the rest of our presentation.

The binary method scans the bits of the exponent either from left to right or from right to left. A squaring is performed at each step, and depending on the scanned

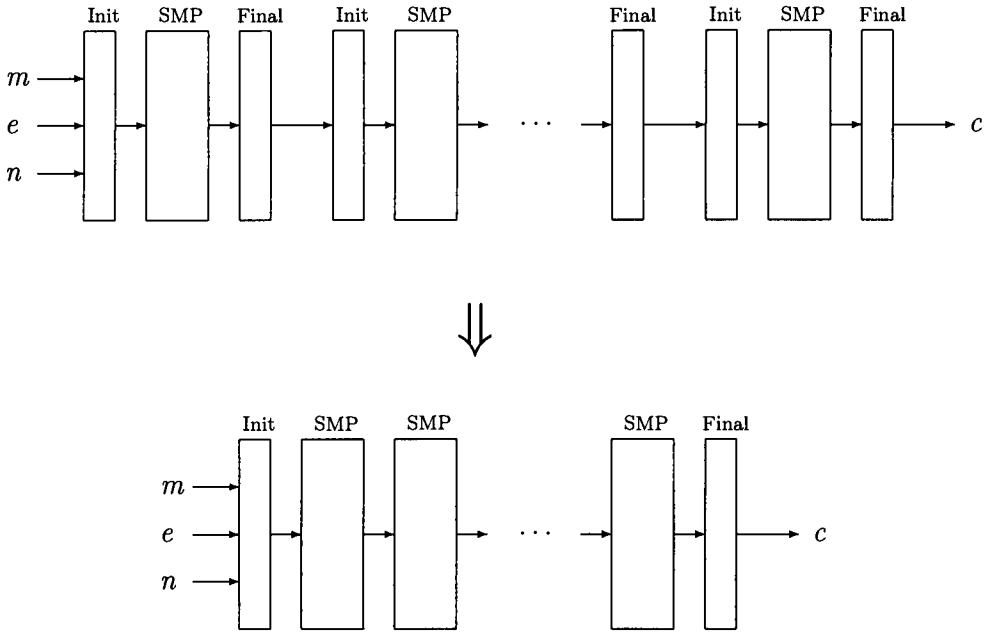

FIGURE 3.7. Spectral modular exponentiation.

bit value, a subsequent multiplication is performed. We describe the spectral modular exponentiation algorithm by using a left-to-right binary method below.

Algorithm 10 *(Spectral Modular Exponentiation)*
Suppose that there exist a DFT map $DFT_d^{\omega} : \mathcal{H}_q^d \rightarrow \mathcal{F}_q^d$. *Let* $n(t)$ *be a base* b *polynomial for* n *where* $deg(n(t)) = s - 1$, $s = \lceil d/2 \rceil$ *and* $gcd(b, n) = 1$.
Input: *A modulus* $n > 0$ *and* $m, e < n$
Output: *an almost modular reduction,* $c \equiv m^e \bmod n$

1: Compute $\underline{n} = \delta \cdot n$ *such that the base polynomial* $\underline{n}(t)$ *has degree* d *and* $\underline{n}_0 = 1$

2: $\underline{N}(t) := DFT_d^{\omega}(\underline{n}(t))$

3: *Compute the base polynomial* $\lambda'(t)$ *for* $\lambda' = b^{2d} \bmod n$.

4: $\Lambda'(t) := DFT_d^{\omega}(\lambda'(t))$

5: $M(t) := DFT_d^{\omega}(m(t))$

6: $\overline{M}(t) := SMP(M(t), \Lambda'(t))$

7: $\overline{C}(t) := SMP(1(t), \Lambda'(t))$

8: **for** $i = j - 2$ **downto** *0*

9: $\quad \overline{C}(t) := SMP(\overline{C}(t), \overline{C}(t))$

10: $\quad$ **if** $e_i = 1$ **then** $\overline{C}(t) := SMP(\overline{C}(t), \overline{M}(t))$

11: $C(t) := SMP(\overline{C}(t), 1(t))$

12: $c(t) := IDFT_d^{\omega}(C(t))$

13: **return** $c(b)$

Remark 10 *Since the SME algorithm computes an almost modular reduction of* $c \equiv m^e \bmod n$, *a final reduction is needed if the output is desired in the range* $[0, n-1]$.

Observe that SME is presented in such a way that even the initialization steps carried in the spectrum but if doing the initialization in time domain brings some advantage the Steps 3–7 can be arranged as follows

3: Compute the base polynomial $\lambda(t)$ for $\lambda = b^d \bmod n$.

4: Compute the base polynomial $\overline{m}(t)$ for $\overline{m} = m \cdot \lambda \bmod n$.

5: $\overline{M}(t) := \mathrm{DFT}_d^{\omega}(\overline{m}(t))$

6: $\overline{C}(t) := \mathrm{DFT}_d^{\omega}(\lambda(t))$

We close the section by presenting the minimal domain bounds for SME algorithm, we start with a lemma that states the how big the coefficients of a special

polynomial get after a convolution, then by using this result we state how q has to be chosen in order not to have some overflow effects.

Lemma 6 *Let $s > 0$ and $x(t)$ be the special polynomial, $x(t) = 1+2t+3t^3+\ldots+st^{s-1}$, then the coefficients of $z(t) = (x(t))^2$ are bounded by*

$$B(s) = \frac{-2s^3}{3} - s^2 + 2s^2 r_1 - \frac{s}{3} + \frac{r_1}{3} + 2sr_1 \tag{3.10}$$

where $r_1 = -2 + \sqrt{\frac{3+18s^2+18s}{9}}$.

Proof: Let $x(t) = 1+2t+3t^3+\ldots+st^{s-1}$ be a polynomial of degree $s-1$ observe that if the convolution $z(t) = (x(t))^2$ is computed the coefficients of $z(t)$ satisfies the following recurence:

$$\begin{aligned}
z_0 &= 1 & z_s &= (s-1)(s+1) + z_{s-2} \\
z_1 &= 2^2 & z_{s+1} &= 2(s-2)(s+1) + z_{s-3} \\
z_2 &= 3^2 + z_0 & z_{s+2} &= 3(s-3)(s+1) + z_{s-4} \\
z_3 &= 4^2 + z_1 & &\vdots \\
&\vdots & z_{s+i-1} &= i(s-i)(s+1) + z_{s-i-1} \\
z_{s-2} &= (s-1)^2 + z_{s-4} & &\vdots \\
z_{s-1} &= s^2 + z_{s-3} & z_{2s-3} &= (s-2)2(s+1) + z_1 \\
& & z_{2s-2} &= (s-1)(s+1) + z_0
\end{aligned}$$

If these coefficients examined carefully one realizes that maximum magnitude, which also decides the bound we are interested, has to be located somewhere in between coefficients $s-1$ and $2s-2$ since the coefficients upto the $(s-2)$th are monotonously increasing and $z_s > z_{s-2}$ for $s > 1$.

Our first observation is z_i is a telescoping sequence for $r < s$ hence

$$z_{r+1} + z_r = 1^2 + 2^2 + \ldots + (r+2)^2 = \sum_{i=1}^{r+2} i^2$$

this sum can be found by

$$z_{r+1} + z_r = (\frac{(B + (r+2) + 1)^3 - B^3}{3}$$

where B^i stands for the ith Bernolli number (i.e. $B^0 = B_0 = 1, B_1 = -1/2, B_2 = 1/6$ and $B_3 = 1/30$, plugging these values to the equation gives

$$\begin{aligned} z_{r+1} + z_r &= \frac{1}{6}(2(r+2)^3 + 3(r+2)^2 + (r+2)) \\ &= \frac{1}{6}(2r^3 + 15r^2 + 37r + 30) \end{aligned}$$

Here if $r+1$ is even,z_{r+1} and z_r can be found as

$$\begin{aligned} z_r = z_1 + z_3 + \ldots + z_r &= 4(\frac{(B + \frac{r+1}{2} + 1)^3 - B^3}{3}) = \frac{1}{6}(r^3 + 6r^2 + 11r + 6) \\ z_{r+1} = \frac{1}{6}(2r^3 + 15r^2 + 37r + 30) - \frac{1}{6}(r^3 + 6r^2 + 11r + 6) &= \frac{1}{6}(r^3 + 9r^2 + 26r + 24) \end{aligned}$$

and in case of $r+1$ is odd, one would get

$$\begin{aligned} z_{r+1} = z_1 + z_3 + \ldots + z_{r+1} &= 4(\frac{(B + \frac{r+2}{2} + 1)^3 - B^3}{3}) = \frac{1}{6}(r^3 + 9r^2 + 26r + 24) \\ z_r = \frac{1}{6}(2r^3 + 15r^2 + 37r + 30) - \frac{1}{6}(r^3 + 9r^2 + 26r + 24) &= \frac{1}{6}(r^3 + 6r^2 + 11r + 6) \end{aligned}$$

Therefore, in either case z_r is written as $\frac{1}{6}(r^3+6r^2+11r+6)$. A general term of the recurrence is found by plugging z_r into the system after replacing the index $s+i-1$ by r, if explicitly written;

$$z_r = \begin{cases} \frac{1}{6}(r^3 + 6r^2 + 11r + 6) & \text{if } r < s \\ -(s-(r+1))(2s-(r+1))(s+1) + z_{2s-r-2} & \\ \quad = -\frac{2s^3}{3} + s^2 r + \frac{5s}{3} - \frac{r^3}{6} - \frac{11r}{6} + s^2 + sr - r^2 - 1) & \text{if } s \leqslant r < 2s-1 \end{cases}$$

Since the explicit function for z_r is found, we simply use the usual arguments of finding the maximum value of a function. In other words finding the roots of the derivative $\frac{\partial z_r}{\partial r}$ and plugging the root into the equation of z_r. In this case the root

$r_1 = -2 + \sqrt{\frac{3+18s^2+18s}{9}}$ gives the local maximum in the range $s \leqslant r < 2s - 1$ and if r_1. if it is plugged into z_r we would get the bound $B(s)$ as a function of s

$$B(s) = \frac{-2s^3}{3} - s^2 + 2s^2 r_1 - \frac{s}{3} + \frac{r_1}{3} + 2sr_1,$$

which gives the desired result. ■

Theorem 12 *Algorithm 9 computes an almost modular reduction,* $c \equiv m^e \bmod n$ *if the parameters* b, q *and* s *satisfies the following inequality*

$$(b^2 + b)^2 B(s) + b^2 s < q \tag{3.11}$$

where $B(s)$ *is given by Equation (3.10).*

Proof: First of all since SME implements the binary exponentiation method in the spectrum and by Lemma 4, SMP computes the DFT of $x(t)t^d$ with the finalization Step 11, residue form of spectral polynomials normalized hence the algorithm works as long as a nice domain is chosen for all intermediate values and output causing no overflows. By using Theorem 7 we realize that when the inputs of SMP are spectral base b polynomials the output of SMP algorithm is a spectral polynomial with an inverse image that fits into the frame $\mathcal{H}_r^s$ where $r = b^2 s + b$. But if a consecutive SMP is fed with this input, the output of the second SMP has larger time coefficients. For instance in Steps 9, 10 and 11 the input $\overline{C}(t)$ is a spectral polynomial with time coefficients larger than b. If these steps examined further we understand that maximum magnitudes are attained from the computation of SMP$(\overline{C}(t), \overline{C}(t))$ in Step 9 since for both Steps 10 and 11 $\overline{M}(t)$ and $1(t)$ are spectral base b polynomials.

Thus we investigate how big the coefficients of the time polynomial get after in Step 9. Indeed this gives us the bounds for our domain of DFT. In order to have a better analysis, we need to say more about the distribution of the coefficients of the

time polynomial after applying an SMP. In Theorem 7 we showed that after a reduction $c(t) \in \mathcal{H}_r^s$ has the form

$$c(t) = c_0 + c_1 t + c_2 t^2 + \ldots + c_{s-1} t^{s-1}, \quad \text{where } c_i < (s-i)b^2 + b$$

for $i = 0, 1, \ldots, s-1$. Since $c_i < (s-i)b^2 + b < (s-i)(b^2+b)$ instead we can write

$$c(t) < (b^2+b)y(t), \quad \text{where } y(t) = s + (s-1)t + (s-2)t^2 + \ldots + 1t^{s-1},$$

If $(c(t))^2$ is computed as seen in Step 9, it is seen that

$$(c(t))^2 < (b^2+b)^2 (y(t))^2$$

But by using Lemma 6 we guarantee that the coefficients of $(y(t))^2$ are bounded by $B(s)$ [1] which implies that coefficients of $(c(t))^2$ is bounded by $(b^2 s + b)^2 B(s)$. And finally since the reduction steps increase this bound slightly which is given by Theorem 7 as $(b^2+b)^2 B(s) + b^2 s$ for intermediate values. Therefore if q is chosen larger than this final bound no overflows is generated and SME works over the ring $\mathbb{Z}_q^d$. ■

Fortunately, Inequality (3.11) gives the relation between the parameters b, s and q in a consecutive use of SMP algorithm. In practice, these parameters are generated in two different ways: the first one is picking s and b and then try to find a ring $\mathbb{Z}_q$ that admits a DFT of size d. The second one is picking a ring with q elements, decide on s and then find out the base b.

We discuss the parameter selection related issues in the next chapter, but before we like to demonstrate an illustrative example for better understanding of the presented subject.

---

[1] $c(t)$ of Lemma 6 is the mirror image of $y(t)$

3.10. Illustrative Example

In this section, we give an example exponentiation computation using the SME method with the input values as $m = 2718$, $e = 53$, and $n = 3141$. We will describe the steps of the SME method performing this modular exponentiation operation, giving the temporary results and the final result $c = m^e \pmod{n}$.

We select the length of a single word as $b = 2^3$ and $s = 4$, by using the Inequality (3.11) we conclude that $q > 131845.0 > 2^{17}$. Thus we need to search for a Fermat or Mersenne ring with $q \geqslant 2^{18} - 1$ that admit a DFT with length $d = 7$ or $d = 8$ for ω equals to a power of two. It turns out that the ring $\mathbb{Z}_{2^{20}+1}$ satisfies these conditions with $\omega = 32$. With these selections we compute $d^{-1} \bmod q$ and $\Gamma(t)$ as

$$d^{-1} = 8^{-1} \pmod{2^{20} + 1} = 917505 \ .$$

and

$$\begin{aligned} \Gamma(t) &= 1 + w^{-1}t + w^{-2}t^2 + w^{-3}t^3 + w^{-4}t^4 + w^{-5}t^5 + w^{-6}t^6 + w^{-7}t^7 \\ &= 1 + 1015809t + 1047553t^2 + 1048545t^3 + 1048576t^4 + \\ &\quad 32768t^5 + 1024t^6 + 32t^7 \ . \end{aligned}$$

We start with writing m and n in polynomial representation

$$\begin{aligned} n(t) &= 5 + 0 \cdot t + 1t^2 + 6t^3 \ , \\ m(t) &= 6 + 3t + 2t^2 + 5t^3 \ . \end{aligned}$$

note that $\deg n(t) = s - 1 = 3$ and $\gcd(n, b) = 1$.

The steps of the SME method computing this modular exponentiation are described below.

1. Given $n = 3141$, we have $n_0 = 5$. Finding the inverse of n_0 modulo b gives δ which is mostly achieved by Extended Euclidean Algorithm:

$$\delta = n_0^{-1} \bmod b = 5 \bmod 8 \ .$$

Thus, $\underline{n} = \delta n = 5 \cdot 3141 = 15705$ which equal to

$$\underline{n}(t) = 1 + 3t + 5t^2 + 6t^3 + 3t^4 \ .$$

in polynomial representation. Recall that $\deg(\underline{n}(t)) = s = 4$ and $\underline{n}_0 = 1$

2. The computation of $\underline{n}(t) = \mathrm{DFT}[\underline{n}(t)]$ is accomplished using the DFT function. In Section 2.4 , we simply gave the DFT computation as a matrix multiplication but some FFT can be employed. We obtain the result of the DFT as

$$\underline{N}(t) = 18 + 201822t + 1045504t^2 + 93374t^3 + 856991t^5 + 3071t^6 + 944959t^7$$

Recall that we work in the finite ring $\mathbb{Z}_q$ with $q = 2^{20} + 1 = 1048577$ represented by the last residue set, thus, the coefficients of the polynomial $\underline{N}(t)$ are in the range $[0, 2^{20})$.

3. after computing $\lambda' = 2^{2d} \bmod n = 8^{16} \bmod 3141 = 415$, the polynomial representation of λ' is found

$$\lambda'(t) = 7 + 3t + 6t^2 \ .$$

Furthermore, we obtain the spectral coefficients of $\Lambda'(t)$ using the DFT as

$$\Lambda'(t) = 16 + 6247t + 3073t^2 + 92167t^3 + 10t^4 + 6055t^5 + 1045506t^6 + 944136t^7$$

4. Given $m(t)$, we obtain its spectral representation $M(t)$ using the DFT as

$$M(t) = 16 + 165990t + 1046533t^2 + 96422t^3 + 886695t^5 + 2052t^6 + 948071t^7$$

5. The SMP algorithm is used to compute $\overline{M}(t) = \text{SMP}[M(t), \Lambda'(t)]$ with inputs

$$M(t) = 16 + 165990t + 1046533t^2 + 96422t^3 + 886695t^5 + 2052t^6 + 948071t^7$$

$$\Lambda'(t) = 16 + 6247t + 3073t^2 + 92167t^3 + 10t^4 + 6055t^5 + 1045506t^6 + 944136t^7$$

We then use the SMP to find the resulting polynomial $\overline{M}(t)$ given the inputs $M(t)$ and $\Lambda'(t)$. First we execute Step 1 in the SMP method, and obtain the initial value of $Z(t)$ using the rule $Z_i = M_i \cdot \Lambda'_i \bmod q$ for $i = 0, 1, \ldots, 7$ as

$$Z(t) = 256 + 945454t + 10250t^2 + 236399t^3 + 223985t^5 + 1038347t^6 + 691376t^7$$

In Step 2 of the SMP method, We assign the initial value of $\alpha = 0$, and start the loop for $i = 0, 1, \ldots, 7$. We illustrate the computation of the instance of the loop for $i = 0$ in Table 3.1. The **for** loop needs to execute for the remaining values of i as $i = 1, 2, \ldots, 7$ in order to compute the resulting product $\overline{M}(t)$ given by

$$\overline{M}(t) = 135 + 324054t + 36891t^2 + 398677t^3 + 27t^4 +$$
$$779927t^5 + 1011740t^6 + 594712t^7 \ .$$

Step	Operation and Result
4:	$z_0 = d^{-1} \cdot (Z_0 + Z_1 + Z_2 + Z_3 + Z_4 + Z_5 + Z_6 + Z_7) \pmod q$ $z_0 = 917505 \cdot (256 + 945454 + 10250 + 236399 + 223985 +$ $1038347 + 691376) \pmod{1048567} = 42$
5:	$\beta = -(z_0 + \alpha) \pmod b = -(42 + 0) \pmod{16} = 6$
6:	$\alpha = (z_0 + 0 + \beta)/b = (42 + 6)/16 = 3$
7:	$Z_i = Z_i + \beta \cdot \overline{N}_i \pmod q$ $Z(t) = 364 + 59232t + 1040389t^2 + 796643t^3 +$ $123046t^5 + 8196t^6 + 69668t^7$

8:	$Z_i = Z_i - (z_0 + \beta) = Z_i - 48 \pmod q$ $Z(t) = 316 + 59184t + 1040341t^2 + 796595t^3 + 1048529t^4 +$ $122998t^5 + 8148t^6 + 69620t^7$
9:	$Z_i = Z_i \cdot \Gamma_i \pmod q$ $Z(t) = 316 + 526138t + 45048t^2 + 723385t^3 + 48t^4 +$ $717053t^5 + 1003513t^6 + 130686t^7$

TABLE 3.1.: The SMP **for** loop instance $i = 0$.

6. In this step, the SMP method is used to compute $\overline{C}(t) = \text{SMP}[1(t), \Lambda'(t)]$ with inputs

$$1(t) = 1 + t + t^2 + t^3 + t^4 + t^5 + t^6 + t^7 ,$$

$$\Lambda'(t) = 16 + 6247t + 3073t^2 + 92167t^3 + 10t^4 + 6055t^5 + 1045506t^6 + 944136t^7$$

We will not give the details of this multiplication since it is similar to the previous one. The result is obtained as

$$\overline{C}(t) = 106 + 13591t + 39979t^2 + 217142t^3 + 28t^4 + 11095t^5 + 1008684t^6 + 806969t^7 .$$

7. **Exponentiation Loop:** The loop starts with the values of $\overline{M}(t)$ and $\overline{C}(t)$ computed above as

$$\overline{M}(t) = 135 + 324054t + 36891t^2 + 398677t^3 + 27t^4 + 779927t^5 + 1011740t^6 + 594712t^7 ,$$

$$\overline{C}(t) = 106 + 13591t + 39979t^2 + 217142t^3 + 28t^4 +$$
$$11095t^5 + 1008684t^6 + 806969t^7 \ .$$

Given the exponent value $e = (53)_{10} = (110101)_2$, the exponentiation algorithm performs squarings and multiplications using the SMP method. Since $j = 6$, the value of i starts from $i = 5$ and moves down to zero, and computes the new value of $\overline{C}(t)$ using the binary method of exponentiation as described. The steps of the exponentiation and intermediate values of $\overline{C}(t)$ are tabulated in Table 3.2. The final value is computed as

$$\overline{C}(t) = 174 + 327348t + 43062t^2 + 592243t^3 + 54t^4 +$$
$$782837t^5 + 1005623t^6 + 395062t^7 \ .$$

i	e_i	Operation	$\overline{C}(t)$
		Start	$106 + 13591t + 39979t^2 + 217142t^3 +$ $28t^4 + 11095t^5 + 1008684t^6 + 806969t^7$
5		$\overline{C}(t)$ =SMP$[\overline{C}(t), \overline{C}(t)]$	$127 + 13519t + 36931t^2 + 118862t^3 +$ $55t^4 + 11215t^5 + 1011780t^6 + 905297t^7$
	1	$\overline{C}(t)$ =SMP$[\overline{C}(t), \overline{M}(t)]$	$135 + 324054t + 36891t^2 + 398677t^3 +$ $27t^4 + 779927t^5 + 1011740t^6 + 594712t^7$
4		$\overline{C}(t)$ =SMP$[\overline{C}(t), \overline{C}(t)]$	$127 + 118020t + 35890t^2 + 178339t^3 +$ $45t^4 + 967557t^5 + 1012787t^6 + 833510t^7$
	1	$\overline{C}(t)$ =SMP$[\overline{C}(t), \overline{M}(t)]$	$175 + 434919t + 42016t^2 + 648646t^3 +$ $45t^4 + 693672t^5 + 1006625t^6 + 320201t^7$
3		$\overline{C}(t)$ =SMP$[\overline{C}(t), \overline{C}(t)]$	$119 + 526344t + 16391t^2 + 982536t^3 +$ $27t^4 + 589897t^5 + 1032200t^6 + 1047114t^7$
	0		$119 + 526344t + 16391t^2 + 982536t^3 +$ $27t^4 + 589897t^5 + 1032200t^6 + 1047114t^7$

2		$\overline{C}(t)$ =SMP$[\overline{C}(t), \overline{C}(t)]$	$202 + 128046t + 60499t^2 + 955597t^3 +$ $72t^4 + 976047t^5 + 988244t^6 + 37904t^7$
	1	$\overline{C}(t)$ =SMP$[\overline{C}(t), \overline{M}(t)]$	$192 + 628407t + 33843t^2 + 586390t^3 +$ $54t^4 + 494072t^5 + 1014836t^6 + 388633t^7$
1		$\overline{C}(t)$ =SMP$[\overline{C}(t), \overline{C}(t)]$	$265 + 755301t + 60454t^2 + 460547t^3 +$ $63t^4 + 422502t^5 + 988199t^6 + 459208t^7$
	0		$265 + 755301t + 60454t^2 + 460547t^3 +$ $63t^4 + 422502t^5 + 988199t^6 + 459208t^7$
0		$\overline{C}(t)$ =SMP$[\overline{C}(t), \overline{C}(t)]$	$296 + 546702t + 74843t^2 + 734828t^3 +$ $90t^4 + 606607t^5 + 973916t^6 + 209585t^7$
	1	$\overline{C}(t)$ =SMP$[\overline{C}(t), \overline{M}(t)]$	$174 + 327348t + 43062t^2 + 592243t^3 +$ $54t^4 + 782837t^5 + 1005623t^6 + 395062t^7$

TABLE 3.2.: The steps of the exponentiation loop.

8. After the exponentiation loop is completed, we have the final value $\overline{C}(t)$. In this step, we have an SMP execution followed by and inverse DFT calculation.

9. We obtain $C(t)$ using $C(t) = \text{SMP}[\overline{C}(t), 1(t)]$ using the inputs

$$\overline{C}(t) = 174 + 327348t + 43062t^2 + 592243t^3 + 54t^4 + 782837t^5 + 1005623t^6 + 395062t^7 .$$

$$1(t) = 1 + t + t^2 + t^3 + t^4 + t^5 + t^6 + t^7 .$$

This computation finds $C(t)$ as

$$C(t) = 169 + 438168t + 48142t^2 + 842167t^3 + 27t^4 + 696537t^5 + 1000463t^6 + 120506t^7 ,$$

10. We obtain $c(t)$ using the inverse DFT function $c(t) = \text{IDFT}[C(t)]$, which gives

$$c(t) = 56 + 59t + 42t^{+}12t^{2} \ .$$

Thus, we obtain the final value as $c(b) = 9360 \equiv 3078 \bmod 3141$, which is equal to

$$3078 = 22718^{53} \bmod 3141$$

as required.

4. APPLICATIONS AND FURTHER IMPROVEMENTS

Modular exponentiation is one of the most important arithmetic operation for modern cryptography. For example, the RSA requires exponentiation in $\mathbb{Z}_n$ for some positive integer n, whereas Diffie-Hellman key agreement and the ElGamal scheme use exponentiation in some large prime fields (see [15]).

The sizes of the modulus which also determines the key size are the main security measure for those systems. In this chapter, we describe the methodology for selecting the parameters for *Spectral Modular Exponentiation* (SME) in order to use the method in public-key cryptography. We like to mention that the description of any cryptosystems is not with in the scope of this work, however we prefer to present our SME parameters that match the popular RSA key sizes which also provides perspective for similar cryptosystems.

Symbol	Meaning	Relationship
c, m, e, n	Output and input integers	$c = m^e \pmod{n}$
k	Number of bits in c, m, n	usually $k \geqslant 512$
u	Length of a single word	k is a multiple of u
s	Number of words in c, m, n	$k = su$
b	Base representation	$b = 2^u$
q	# of elements of $\mathbb{Z}_q$	$q = 2^v \pm 1$
v	The word size of q	$q = 2^v \pm 1$
d	Length of the DFT	$s = \lceil d/2 \rceil$
ω	Principal dth root of unity in $\mathbb{Z}_q$	$\omega^d = 1 \pmod{q}$

TABLE 4.1. The symbols used in the SME method.

We gave the hints of parameter selection in the previous chapter, in particular the Inequality (3.11) presents a solid basis for the relation between the parameters b, q and s. This bound can be improved in many ways which we investigate through out this chapter. In order to have a friendly presentation we give a look up table (Table 4.1) for the symbols used with in this chapter.

4.1. Parameter Selection for RSA

The well known RSA algorithm mostly operate with key sizes ranges from 512 to 16,384, addressing different levels of security needs. In this section we tabulate some example parameters for modular expònentiations using the SME method, starting from 513 bits up to 4,260 bits. Our figures are centered around those sizes that are powers of 2 which are popular for architectural reasons.

In Tables 2.1 and 2.2, we give suitable Mersenne and Fermat rings for the SME algorithm, additionally in Tables 6.1 and 6.2 in the Appendix A we tabulate the pseudo Mersenne and Fermat rings which are also suitable to use. These tables exhibit principal roots of unity and the DFT lengths for each ring. Whenever possible, we select the principal root of unity as $w = 2$ or $w = -2$ since multiplication with such numbers are accomplished by shifting.

Once the underlying ring and the DFT length and the principal root of unity are selected, the maximum modulus size in the SME method is computed by finding the base $b = 2^u$. The relation between these parameters is computed by finding the maximum b satisfying the Inequality (3.11). In Table 4.2, we give some example rings and their parameters. As an example, we first select a ring from this table such as $q = 2^{73} + 1$. This comes with the principal root of unity $w = 2$, the length $d = 73$ and $s = \lceil d/2 \rceil = 37$. Plugging these values into the Inequality (3.11) gives

$$14564.0b^4 + 29128.0b^3 + 14601.0b^2 < 2^{73} - 1$$

by inspection $b = 2^{14} \Rightarrow u = 14$ is found. Therefore, we compute the maximum bit length of the exponentiation as $k = s \cdot u = 37 \cdot 14 = 518$ as given in Table 4.2.

Bits k	Ring $\mathbb{Z}_q$	DFT d	Root w	Wordsize u	Words s
513	$(2^{57} - 1)/7$	114	-2	9	57
518	$2^{73} - 1$	73	2	14	37
518	$(2^{73} + 1)/3$	73	4	14	37
704	$2^{64} + 1$	128	2	11	64
1,185	$2^{79} - 1$	158	-2	15	79
1,728	$2^{128} + 1$	128	4	27	64
2,060	$(2^{103} + 1)/3$	206	2	20	103
2,163	$2^{103} - 1$	206	-2	21	103
3,456	$(2^{128} + 1)$	256	2	27	128
4,260	$(2^{142} + 1)/5$	284	2	30	142

TABLE 4.2. Parameter selection for SME with SMP.

4.2. Modified Spectral Modular Product

If the SMP (i.e. Algorithm 8) is considered, one realizes that our bound analysis depends heavily on $\beta \cdot \underline{N}(t)$ multiplication of Step 7. It is possible to replace this multiplication by a multi-operand addition at a cost of some pre-computations and extra memory. This approach is very beneficial since this replacement gives a reason-

able amount of radius shrinkage. Additionally, replacing the multiplication by an array addition improves the computational complexity.

Let $b = 2^u$ and $n_i(t)$ be the polynomial representation of an integer multiple of n such that the zeroth coefficient of $n_i(t)$ satisfies $(n_i)_0 = 2^{i-1}$ for $i = 1, 2, \ldots, u$ (note that $\underline{n}(t) = n_1(t)$). We can now write $\beta \cdot \underline{N}(t)$ as

$$\beta \cdot \underline{N}(t) = \sum_{i=1}^{u} \beta_i \cdot N_i(t) \ , \tag{4.1}$$

where β_i is a binary digit of β and $N_i(t) = \mathrm{DFT}_d^{\omega}(n_i(t))$ for $i = 1, 2, \ldots, u$. Note that $\beta < b$ and $\beta_i = 0$ for $i \geqslant u$.

Plugging the Equation (4.1) into the Algorithm 8 gives us the modified Spectral modular product algorithm;

Algorithm 11 *Modified Spectral Modular Product Algorithm (MSMP)*
Assume that there exist a NTT of length d over $\mathbb{Z}_q$, and $(X(t), x(t))$ and $(Y(t), y(t))$ are transform pairs where $x(t)$ and $y(t)$ are evaluation polynomials for x and y in the frame $\mathcal{B}_r^{\lfloor d/2 \rfloor}$ for some r. Let the basis set $\mathcal{N} = \{N_1(t), N_2(t), \ldots, N_u(t)\}$ is consist of spectral polynomials as described above.
Input: *$X(t), Y(t)$ and a basis set $\mathcal{N}\{N_1(t), N_2(t), \ldots, N_u(t)\}$*
Output: *$Z(t) \equiv X(t) \odot Y^{-d}(t)$ mods $N(t)$,*

1: $Z(t) := X(t) \odot Y(t)$
2: $\alpha := 0$
3: **for** $i = 0$ **to** $d - 1$
4: $\quad z_0 := d^{-1} \cdot (Z_0 + Z_1 + \ldots + Z_d) \bmod q$
5: $\quad \beta := -(z_0 + \alpha) \ mod \ b$
6: $\quad \alpha := (z_0 + \alpha + \beta)/b$

7: $Z(t) := Z(t) + \sum_{i=1}^{u} \beta_i \cdot N_i(t) \bmod q$

8: $Z(t) := Z(t) - (z_0 + \beta)(t) \bmod q$

9: $Z(t) := Z(t) \odot \Gamma(t) \bmod q$

10: **end for**

11: $Y(t) := Y(t) + \alpha(t)$

12: **return** $Z(t)$

Observe that the basis set $\mathcal{N} = \{N_1(t), N_2(t), \ldots, N_u(t)\}$ needs to be pre-computed and stored. Therefore memory requirement is obviously

$$\text{memory}(\mathcal{N}) = uvd \ . \tag{4.2}$$

This can be seen as a handicap for certain platforms; however, in general the MSMP delivers smaller realizations. While inserting MSMP into either SME or SMM the pre-computation is done during Step 1 of Algorithm 9 or 10. After computing $n_1(t)$ the rest of the basis set is computed by multiplying n_1 by powers of 2. We then apply the DFT function to corresponding polynomials in order to get $N_i(t) = \text{DFT}[n_i(t)]$ for $i = 1, 2, \ldots, u$.

With the adjustment (4.1), the time coefficients of $\sum_{i=1}^{u} \beta_i \cdot N_i(t)$ become less than $b\log(b)$ which gives us a better inequality

$$(b\log(b) + b)^2 B(s) + b\log(b)s < q \tag{4.3}$$

which is a pretty good improvement for a reasonable space cost. This bound gives us the improved parameters which are tabulated in Table 4.3 below.

Additionally, whenever the memory requirement of the above method is excessive, a hybrid stragedy that delights the usage of combination of multiplication and addition array is also possible. This is summarized with the following equation

Bits k	Ring $\mathbb{Z}_q$	DFT d	Root w	Wordsize u	Words s
540	$2^{59}-1$	59	2	18	30
564	$2^{47}-1$	94	-2	12	47
589	$2^{61}-1$	61	2	19	31
640	$2^{64}+1$	64	4	20	32
1,098	$2^{61}-1$	122	-2	18	61
1,080	$2^{79}-1$	79	2	27	40
1,216	$2^{64}+1$	128	2	19	64
2,054	$2^{79}-1$	158	2	26	79
2,160	$2^{107}-1$	107	2	40	54
3,200	$2^{128}+1$	128	4	50	64
4,251	$2^{109}-1$	218	-2	39	109
6,144	$2^{128}+1$	256	2	48	128

TABLE 4.3. Improved parameter selection for SME with MSMP.

$$\beta \cdot N(t) = \beta' \cdot N_1(t) + \sum_{i=u'}^{u} \beta_i \cdot N_i(t), \tag{4.4}$$

where $\beta' = \beta \bmod b'$ and β_i stands for binary digits of β/b' for some $0 \leqslant b' \leqslant b$. We leave the analysis of the hybrid method for future research and continue with some further observations.

If Tables 4.3 and 4.2 are examined, one realizes that the maximum modulus size is given by $k = su$. Therefore by scaling s and u one enjoys the same modulus sizes for different parameters. Recall that the loop in the SMP or MSMP algorithm runs

d times so a decrease in d is desirable for some designs even if it is at a cost of using larger rings (means larger b and u). In Table 4.4, we demonstrate parameters of this nature.

Bits k	Ring $\mathbb{Z}_q$	DFT d	Root w	Wordsize u	Words s
528	$(2^{115}-1)/31$	23	32	44	12
544	$(2^{93}+1)/9$	31	64	34	16
592	$2^{96}+1$	32	64	37	16
1,024	$(2^{155}+1)/33$	31	1024	64	16
1,122	$(2^{129}+1)/9$	43	64	51	22
2,150	$(2^{129}+1)/9$	86	64	50	43

TABLE 4.4. Parameter selection having smaller d for SME with MSMP.

4.3. The use of Chinese Remainder Theorem (CRT)

In many situations it is desirable to break a congruence mod n into a system of small congruences mod factors of n. Once computations are performed in the small factor rings, by using CRT, the resultant system of congruences is replaced by a single congruence under certain conditions.

Theorem 13 Chinese Remainder Theorem. *For $i \geqslant 2$, let $p_1, p_2, \ldots, p_l$ be non-zero integers which are pairwise relatively prime: $gcd(p_i, p_j) = 1$ for $i \neq j$. Then, for any integers $a_1, a_2, \ldots, a_l$, the system of congruences*

$$x \equiv a_1 \bmod p_1,\ x \equiv a_2 \bmod p_2, \ldots,\ x \equiv a_l \bmod p_l,$$

has a solution, and this solution is uniquely determined modulo $p_1 p_2 \cdots p_l$.

A formal proof of CRT can be found in many text books including [16], we rather interested in computation (lift) of final congruence. Although there are more efficient ways of lifting, throughout this chapter, we consider the single-radix conversion (SRC) method. Going back to the general system, SRC algorithm computes x using the following summation for given $a_1, a_2, \ldots, a_l$ and $p_1, p_2, \ldots, p_l$.

$$x = \sum_{i=1}^{l} a_i c_i n_i' \pmod{n}, \quad \text{where } n_i' = p_1 p_2 \cdots p_{i-1} p_{i+1} \cdots p_l = \frac{n}{p_i} \tag{4.5}$$

and c_i is the multiplicative inverse of n_i' mod p_i.

When the algorithms of Chapter 3 are considered CRT can be used in two different ways. The first one is for degree where the second one is for radius. We start with the first one.

4.3.1. CRT for Degree

When n is composite number such as the RSA modulus n, the CRT is very beneficial if the prime factorization of n is known (e.g. RSA decryption). This use of CRT was first proposed by Quisquater and Couvreur [17], and it is easily be adopted to SME.

Suppose that $n = p_1 p_2$ is a typically RSA modulus with two large distinct prime factors p_1 and p_2. The computation m^e mod n can be put into the system of two small congruences as follows:

$$m_1 := c^e \bmod p_1$$

$$m_2 := c^e \bmod p_2$$

However, applying Fermat's theorem to the exponents, we only need to compute

$$m_1 := c^{e_1} \bmod p_1$$

$$m_2 := c^{e_2} \bmod p_2$$

where $e_1 := e \bmod (p_1 - 1)$ and $e_2 := e \bmod (p_2 - 1)$.

Observe that the computation of m_1 and m_2 is performed by using the two separate SME with inputs m_1, e_1 and m_2, e_2 respectively and when they are computed, proceeding with the SRC algorithm, we achieve m by using the sum (4.5)

$$m = m_1 c_1 p_2 + m_2 c_2 p_1 \bmod n \tag{4.6}$$

where $c_1 = p_2^{-1} \bmod p_1$ and $c_2 = p_1^{-1} \bmod p_2$.

4.3.2. CRT for Radius

CRT is proposed to use for integer multiplication by J. M. Pollard [2] and independently by A. Schönhage and V. Strassen [1] who further recommend to use FFT over Fermat rings $\mathbb{Z}_q$ with $q = 2^{2^r} + 1$ and $d = 2^r$ for some $r > 0$ (see Proposition 4 for the existence of such transforms), they proved the famous time bound $O(n \log n \log\log n)$ for integer multiplication, with some care their ideas can be applied to spectral modular algorithms defined in Chapter 3.

Let us go back to Example 3 of Chapter 2, we mention that if $x = (10, 5, 1, 0, 0)$ is chosen we get

$$z = x \odot y = (10, 5, 1, 0, 0) \odot (2, 3, 3, 0, 0) = (20, 9, 20, 24, 9) \bmod 31$$

If the same computation is done modulo 2

$$z = x \odot y = (10, 5, 1, 0, 0) \odot (2, 3, 3, 0, 0) = (0, 1, 0, 0, 1) \bmod 2$$

is computed and this suffices to recover the real coefficients $z = (20, 40, 51, 24, 9)$ from the congruences $z = (20, 9, 20, 24, 9) \bmod 31$ and $z = (0, 0, 1, 0, 1) \bmod 2$ by using CRT on the coefficients. For instance, the second term can be found by solving the congruences

$$z_2 \equiv 9 \bmod 31, \text{ and } z_2 \equiv 0 \bmod 2 \text{ which gives } z_2 \equiv 40 \bmod 62$$

by using the Equation (4.6).

Schönhage-Strassen multiplication picks some special rings and applies the CRT to recover the actual coefficients of a convolution. As a second example we present their algorithm by using our new notation.

Suppose that $(x(t), x)$ and $(y(t), y)$ are base b polynomials in the frame $\mathcal{H}_b^s$ with $d = 2^{r+1}$, $b = 2^{2^{r-1}}$ and $s = \lceil d/2 \rceil = 2^r$. Lemma 1 states that $(z(t), z) = (x(t), x) \otimes (y(t), y) \in \mathcal{H}_d^q$ is well defined if $q > sb^2 = 2^r(2^{2^{r-1}})^2 = 2^r 2^{2^r}$. If $q = 2^r(2^{2^r} + 1) > sb^2$ is picked, CRT is applicable to compute the coefficients of $z(t) = x(t)y(t) \bmod q$.

In other words, if the following two polynomials are calculated

$$z_1(t) = x_1(t)y_1(t) \bmod 2^r, \tag{4.7}$$

$$z_2(t) = x_2(t)y_2(t) \bmod (2^{2^r} + 1) \tag{4.8}$$

where $x_1(t) = x(t) \bmod 2^r$, $y_1(t) = y(t) \bmod 2^r$, $x_2(t) = x(t) \bmod (2^{2^r} + 1)$ and $y_2(t) = y(t) \bmod (2^{2^r} + 1)$, the coefficients of $z(t) = x(t)y(t) \bmod 2^r(2^{2^r} + 1)$ are deduce by using CRT since $\gcd(2^r, 2^{2^r} + 1) = 1$.

In general the computation of Equation (4.7) is done by using the usual polynomial multiplication routines because 2^r is small on the other hand the computation of the Equation (4.8) is mostly performed by using FFT and convolution theorem in the Fermat ring with $q' = 2^{2^r} + 1$. Recall that by Proposition 4 there exists a length $d = 2 \cdot 2^r = 2^{r+1}$ DFT map with principal root of unity $\omega = 2$. Although we know that overflows occur (since $b^2 > q'/s$), exact coefficients of the convolution are recovered by using the congruence (4.7).

When it comes to the spectral modular operations the methodology changes slightly. One has to recover the actual least significant time digit at every step of the reduction process.

Suppose that $(x(t), x)$ and $(y(t), y)$ are evaluation polynomials in the frame $\mathcal{H}_q^s$ with $q > 0$ and $s = \lceil d/2 \rceil$. Let $p_1, p_2, \ldots, p_l$ be positive, pairwise relatively prime numbers such that $p_1 p_2 \cdots p_l > q$. Moreover, assume that for all $j = 1, 2, \ldots, l$ there exists a length d_j DFT over $\mathbb{Z}_{p_j}$ such that $d_j \geqslant d$. It is possible to break the computation of the congruence

$$z(t) = x(t)y(t) \text{ mods } n(t)$$

into the congruences in small rings $\mathbb{Z}_{p_j}$. Let for $j = 1, 2, \ldots, l$

$$x_j(t) = x_{j0} + x_{j1}t + \ldots + x_{j(d_j-1)}t^{d_j-1} \equiv x(t) \bmod p_j \ ,$$
$$y_j(t) = y_{j0} + y_{j1}t + \ldots + y_{j(d_j-1)}t^{d_j-1} \equiv y(t) \bmod p_j \ ,$$
$$n_j(t) = n_{j0} + n_{j1}t + \ldots + n_{j(d_j-1)}t^{d_j-1} \equiv n(t) \bmod p_j$$

be the transform pairs of $X_j(t), Y_j(t)$, and $N_j(t)$ respectively. If $\mathbf{X}(t) = (X_1(t), X_2(t), \ldots, X_{l-1}(t))$ and $\mathbf{Y}(t) = (Y_1(t), Y_2(t), \ldots, Y_{l-1}(t))$ stand for vectors of spectral polynomials, the following procedure computes $\mathbf{Z}(t) = (Z_1(t), Z_2(t), \ldots, Z_{l-1}(t))$ where

$$z_j(t) = x_j(t)y_j(t) \text{ mods } n_j(t), \text{ for } j = 1, 2, \ldots, l$$

Algorithm 12 *SMP by using CRT*

Input: $\mathbf{X}(t)$ *and* $\mathbf{Y}(t)$ *vector spectral polynomials*

Output: $\mathbf{Z}(t)$,

1: $\mathbf{Z}(t) := \mathbf{X}(t) \odot \mathbf{Y}(t)$

2: $\alpha := (0, 0, \ldots, 0)$, *vector zero of dimension* l

3: **for** $i = 0$ **to** $d - 1$

4: **for** $j = 1$ **to** l

5: $z_{j0} := d_j^{-1} \cdot (Z_{j0} + Z_{j1} + \ldots + Z_{jd}) \bmod p_j$

6: **end for**

7: $z_0 := \sum_{j=1}^{l} z_{j0} c_j p'_j \pmod{q}$

8: **for** $j = 1$ **to** l

9: $\beta_j := (-(z_0 + \alpha_j) \bmod b) \bmod p_j$

10: $\alpha_j := (z_{0j} + \alpha_j + \beta_j)/b \bmod p_j$

11: $Z_j(t) := Z_j(t) + \beta_j \cdot N_j(t) \bmod p_j$

12: $Z_j(t) := Z_j(t) - (z_{0j} + \beta_j)(t) \bmod p_j$

13: $Z_j(t) := Z_j(t) \odot \Gamma_j(t) \bmod p_j$

14: **end for**

15: **end for**

16: $\mathbf{Z}(t) := \mathbf{Z}(t) + \alpha(t)$

17: **return** $\mathbf{Z}(t)$

Once $\mathbf{Z}(t) = (Z_1(t), Z_2(t), \ldots, Z_{l-1}(t))$ is computed, after applying the inverse DFT, one recovers $z(t)$ from small congruences or in case of an exponentiation keep using the above core consequently as we described in Section 3.9. We give an example to demonstrate the use of CRT in SME setting.

Example 11 *In Section 3.10 we demonstrated the computation of* $2718^{53} \bmod 3141$ *by using the 8 point DFT over the Fermat ring* $\mathbb{Z}_{2^{20}+1}$. *Now, let's change the parameter* b *from* 2^3 *to* 2^7 *and increase the inputs to* $m = 27182818$ *and* $n = 31415927$ *and try to compute* $c = 27182818^{53} \bmod 31415927$ *this time. It is obvious that the spectral exponentiation algorithm does not work in* $\mathbb{Z}_{2^{20}+1}$ *because of the overflows. But if we use the ring* $\mathbb{Z}_q$ *with* $q = (2^{20}+1)(2^{16}+1)$, *we are safe because*

$$25.4b^4 + 50.8b^3 + 29.4b^2 < q$$

for $b = 2^7$.

If Algorithm 12 is used as a core for an exponentiation algorithm one can compute the spectral modular reduction in $\mathbb{Z}_{2^{20}+1}$ *and* $\mathbb{Z}_{2^{19}-1}$ *and then combine these results by using CRT to obtain the final value as* $c(t) = 13125 + 7039t + 2831t^2 + 520t^3$. *Evaluating* $c(t)$ *at* $b = 2^7$ *gives* $c(2^7) = 1137816261 \equiv 6842889 \bmod 31415927$, *which is same as*

$$6842889 = 27182818^{53} \bmod 31415927$$

as required.

In Tables 4.5, 4.6, and 4.7, we tabulate some nice rings that CRT can be applied. As before the popular RSA sizes are targeted.

Bits k	Ring $\mathbb{Z}_q$		DFT d	Root w	Words s	Wordsize u
518	q_1	$2^{37}-1$	74	-2	37	14
	q_2	$(2^{37}+1)/3$	74	2	37	
1,071	q_1	$(2^{51}-1)/7$	102	-2	51	21
	q_2	$2^{53}-1$	106	-2	53	
2,130	q_1	$(2^{71}-1)$	141	-2	71	30
	q_2	$(2^{71}+1)/3$	141	2	71	
4,171	q_1	$(2^{97}-1)$	194	-2	97	43
	q_2	$(2^{97}+1)/3$	194	2	97	

TABLE 4.5. Parameter selection by using CRT for SME with SMP.

Bits k	Ring $\mathbb{Z}_q$		DFT d	Root w	Words s	Wordsize u
512	q_1	$2^{16}+1$	32	2	16	32
	q_2	$2^{17}-1$	34	-2	17	
	q_3	$(2^{17}+1)/3$	34	2	17	
	q_4	$2^{19}-1$	38	-2	19	
	q_5	$(2^{20}+1)$	40	2	20	
551	q_1	$2^{29}-1$	58	-2	29	19
	q_2	$2^{31}-1$	62	-2	31	
560	q_1	$2^{31}-1$	32	2	16	35
	q_2	$(2^{31}+1)/3$	32	-2	16	
	q_3	$2^{32}+1$	32	4	16	
1,054	q_1	$2^{31}-1$	62	-2	31	34
	q_2	$(2^{31}+1)/3$	62	2	31	
	q_3	$2^{32}+1$	64	2	32	
1,148	q_1	$2^{41}-1$	82	-2	41	28
	q_2	$(2^{41}+1)/3$	82	2	41	
2,015	q_1	$2^{31}-1$	62	-2	31	65
	q_2	$(2^{31}+1)/3$	62	2	31	
	q_3	$2^{32}+1$	64	2	32	
	q_4	$2^{64}+1$	64	2	32	
2,052	q_1	$2^{71}-1$	71	2	36	57
	q_2	$(2^{71}+1)/3$	71	-2	36	
2,067	q_1	$2^{53}-1$	106	-2	53	39
	q_2	$(2^{53}+1)/3$	106	2	53	

TABLE 4.6. Parameter selection by using CRT for SME with MSMP.

Bits k	Ring $\mathbb{Z}_q$		DFT d	Root w	Words s	Wordsize u
4,118	q_1	$(2^{58}+1)/5$	116	2	58	71
	q_2	$2^{59}-1$	118	-2	59	
	q_3	$(2^{59}+1)/3$	118	2	59	
4,161	q_1	$2^{73}-1$	146	-2	73	57
	q_2	$(2^{73}+1)/3$	146	2	73	
8,216	q_1	$2^{79}-1$	158	-2	79	104
	q_2	$(2^{79}+1)/5$	158	2	79	
	q_3	$2^{83}-1$	166	-2	83	
8,357	q_1	$2^{61}-1$	122	-2	61	137
	q_2	$(2^{61}+1)/3$	122	2	61	
	q_3	$(2^{62}+1)/5$	124	2	62	
	q_4	$2^{62}+4$	128	2	64	
	q_5	$(2^{64}-1)/31$	130	-2	65	
8,484	q_1	$2^{101}-1$	202	-2	101	84
	q_2	$(2^{101}+1)/3$	202	2	101	
16,428	q_1	$(2^{111}-1)/7$	222	-2	111	148
	q_2	$(2^{111}+1)/9$	222	2	111	
	q_3	$2^{113}-1$	226	-2	113	
16,827	q_1	$2^{71}-1$	142	-2	71	237
	q_2	$2^{73}-1$	146	-2	73	
	q_3	$(2^{74}+1)/5$	148	4	74	
	q_4	$(2^{75}-1)/217$	150	-2	75	
	q_5	$(2^{76}+1)/17$	152	4	76	
	q_6	$2^{79}-1$	158	-2	79	
	q_7	$(2^{79}+1)/3$	158	2	79	

TABLE 4.7. Parameter selection by using CRT for SME with MSMP.

5. ARCHITECTURES AND PERFORMANCE ANALYSIS

In this chapter, we give architectures and performance analysis of spectral modular algorithms. While doing this we use some generic and specific architectures that lead us to have a general and a lower level unit-gate analysis. We immediately mention that because of a word level parallelism potential, spectral modular arithmetic is notably suitable for hardware realizations. By word level parallelism we mean the capability of chopping the large numbers into small pieces, then carrying out a sequence of independent calculations on these components and at the same is able to get a result of the actual problem. To have this kind of parallelism is itself a big achievement for computations, furthermore having word level parallelism make lower level parallelism feasible. In other words, if the sizes of the pieces are small enough bit level parallelism is beneficially emerged for the treatment of the basic operations (i.e. addition, multiplication, etc.) on small pieces. In general, applying the spectral techniques have this action on the problems.

Favorably, word level parallelism in spectral modular arithmetic algorithms is expressible with any kind of Application Specific Integrated Circuit (ASIC) including full-custom, semi-custom (cell-based, gate-array-based) and programmable (PLD, FPGA). When it comes to the bit level parallelism i.e. addition, multiplication and division, ASIC technologies naturally differ (see Table 5.1). For instance; in FPGA technology a CPA design is superior to a CSA design for multi-operand addition, even though CSAs are known to be fastest parallel adders.

Our stragedy is to use some generic notation that permits us to exhibit the word level parallelism in spectral arithmetic algorithms. We speak more about generic notation in Sections 5.2 and 5.3 . At the same time, we delightfully attempt to embed the

ASIC Category	bit level parallelism $(+/-, \cdot, /)$	Word level parallelism spectral
full-custom	good	good
semi-custom	fair	good
programmable	poor	good

TABLE 5.1. ASIC Categories and Parallelism.

bit level parallelism into the spectral parallelism that melts graceful designs especially suitable for full-custom and semi-custom ASIC[1].

It is true that spectral modular algorithms are dominated by multi-operand additions and multiplications over some special rings. These kind of operations are efficiently realized by networks based on a Wallace tree structure (see [5] and [6]). But when working with fairly large rings the delay of irregular interconnections in these networks become expensive. Although a more regular compressor networks are well suited for overcoming these issues, we follow the optimal Wallace tree network and neglect these delays. we briefly review the architectures described in [5] and [6], in particular multiplication and multi-operand addition for Fermat and Mersenne rings. In Section 5.2 and 5.3 beside the generic analysis, we append this combined analysis of word and bit level parallelism.

Remark 11 *There are other efficient ways of carrying lower level operations especially multiplication and multi-operand addition. Check [19] and [20] for table-lookup*

[1]see [18] for a FPGA discussion

methods. Further in [20] approaches based on regular integer multiplication followed by modular reduction and treatments of Booth's recoding can be found.

5.1. Arithmetic for Fermat and Mersenne Rings

We mentioned earlier that carrying arithmetic in Mersenne rings is equivalent to doing one's complement operations. Arithmetic in Fermat rings are slightly complicated than one's complement arithmetic, when certain encoding techniques are performed.

5.1.1. Designated Adders and Wallace Tree Structures

There are vast variety of adder architectures for different applications and purposes. Carry save adders (CSA) seem to be the most useful adder for our applications. It is simply a parallel assemble of full-adders without any horizontal connection with a functionality of reducing an input of three operands to an output of two operands. The main draw back is if a normal representation of the output is needed, a final carry propagate adder (CPA) has to be employed in order to add the output of the CSA.

We do not want to go into the design details of CPAs (see [7] and for a textbook treatment check [8]). Considering the sizes of targeted rings we indicate that we favor the parallel-prefix adders in our analysis, to be more specific, we use the optimal Sklansky parallel-prefix adders [9]. For a regular integer addition these adders perform with a $2\log n + 3$ delay and need $(3/2)n\log n + 4n$ area. When it comes to addition in Fermat or Mersenne rings, these figures slightly change which we review in the next section.

When multi-operand additions are examined, Wallace tree structure is known for optimal delay time. The optimality is obtained at an expense of irregular inter-

connection which causes the largest number of wiring tracks that directly increases the wiring complexity. In Figure 5.1, a 27-operand Wallace tree is shown where CSA indicates a carry-save adder having three multi-bit inputs and two multi-bit outputs.

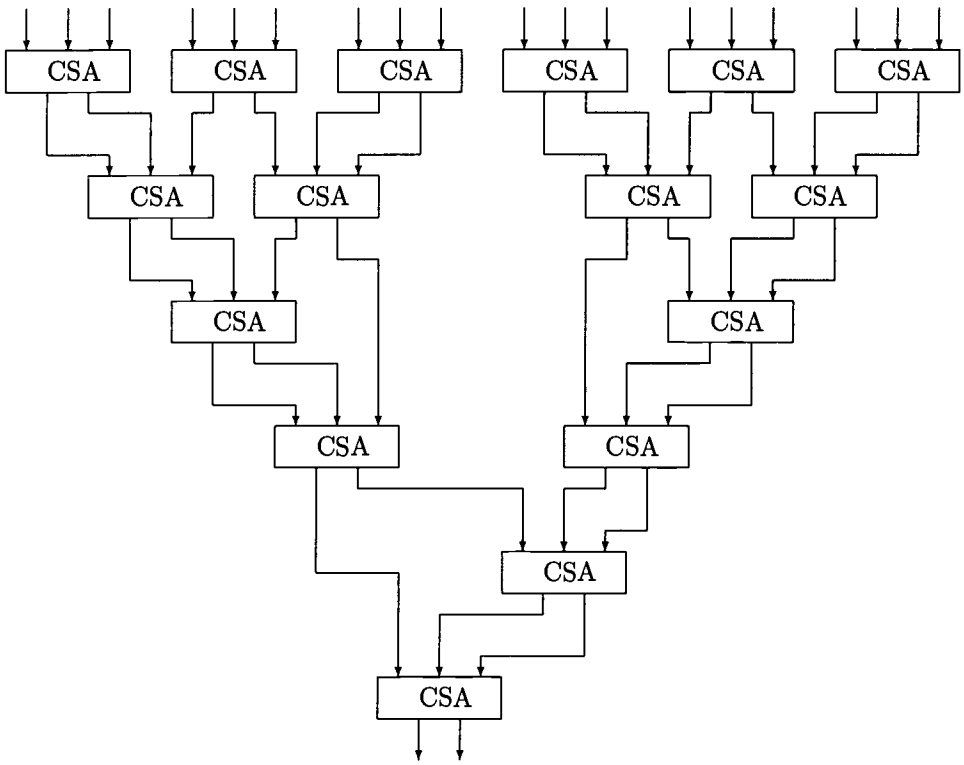

FIGURE 5.1. 27 operand Wallace tree.

Since we are dealing with trees, there are two important parameters that gauge both the time and area complexity of the resulting networks; namely height and number of nodes. Let us declare the relation between number of operands and height as well as the number of nodes of a Wallace tree structure.

Let $\theta(x)$ denotes the height function of the Wallace tree. $\theta(x)$ is a step function incrementing at the following sequence (given recursively);

$$\sigma_{i+1} = \lfloor \frac{3\sigma_i}{2} \rfloor \text{ where } \sigma_0 = 2$$

Obviously the height of a 3 input CSA is 1, additionally the following table shows the height of a carry-save Wallace tree for operands up to 141;

x	3	4	5–6	7–9	10–13	14–19	20–28	29–42	43–63	64–94	95–141	...
$\theta(x)$	1	2	3	4	5	6	7	8	9	10	11	...

Observe that once the wiring delays are neglected, the height function $\theta(x)$ depends only on the single variable x which stands for the number of operands. In other words delay does not depend on the size of operands. Although such a convention does not reflects the reality[2] it makes the analysis easier at a common ground. Therefore in such a model the delay of a Wallace tree of x operands becomes same either the operands are integers or elements of Fermat or Mersenne rings. On the other hand area is directly proportional with the number of nodes in a tree which in this case corresponds to number of CSAs. With the following theorem we give the relation of number of nodes and operands in a Wallace tree:

Theorem 14 *Let x be the number of operands and $\nu(x)$ denotes the number of nodes in the Wallace tree. Then*

$$\nu(x) = x - 2$$

Proof: let $\sigma_i \leqslant x < \sigma_{i+1}$ then

$$\nu(x) = (x - \sigma_i) + \nu(\sigma_i) \tag{5.1}$$

---

[2]especially as the transistors get smaller and smaller the wire delay dominates the gate delay in the integrated circuits

Observe that $\nu(\sigma_i) = (\sigma_i - \sigma_{i-1}) + (\sigma_{i-1} - \sigma_{i-2}) + \ldots + (\sigma_1 - \sigma_0) + \nu(\sigma_0)$. But $\sigma_0 = 2$ and $\nu(\sigma_0) = 0$ so the Equation (5.1) can be written as

$$\nu(x) = (x - \sigma_i) + (\sigma_i - 2) = x - 2.$$

■

Indeed Theorem 14 describes a methodology to attain the minimal number of CSAs in a Wallace tree design which is; first reduce the number of operands to the smaller nearest element of the $\{\sigma_n\}$ sequence then follow the optimality of the sequence.

Unlike the height function, area depends on the sizes of the operands. Obviously, as we keep adding numbers the size of result gets larger and the demand for more gates increases but in Fermat or Mersenne rings the operand sizes can easily be kept constant by an operate-reduce argument. Such an arrangement delights an easy computation of the actual number of full adders needed for a Wallace tree network for Fermat and Mersenne ring which we is addressed in Section 5.1.3 after briefly reviewing the nature of addition in Fermat and Mersenne rings in the next section.

5.1.2. Addition

In this section we characterize the methodology of addition in Fermat and Mersenne rings. We begin with two equations that can easily be traced

Lemma 7 *Let* $r > 0$ *be an integer and* $0 \leqslant x \leqslant (2^v + 1)^2$ *and* $0 \leqslant y \leqslant (2^v - 1)^2$

$$x \bmod (2^v + 1) = (x \bmod 2^v - x \ div \ 2^v) \bmod (2^v + 1) \tag{5.2}$$

$$y \bmod (2^v - 1) = (y \bmod 2^v + y \ div \ 2^v) \bmod (2^v - 1) \tag{5.3}$$

Proof: Since $2^v = \pm 1$ the modular reduction corresponds to adding/subtracting the most significant r bits to/from the least significant r bits. This procedure needs

a correction in case of this addition/subtraction exceeds the modulus which is carried by a final modular reduction. ■

Mersenne ring addition (Modulo $2^v - 1$) Let $x, y \in \mathbb{Z}_{2^v-1}$, by using the Lemma 7 the addition in the Mersenne ring can be formulated as follows

$$x + y \bmod (2^v - 1) = \begin{cases} x + y + 1 \bmod 2^v & \text{if } x + y \geqslant 2^v \\ x + y & \text{otherwise} \end{cases}$$

Therefore this is the regular addition unless there exists an overflow which is normally handled by employing an end-around carry adder.

Fermat ring addition (Modulo $2^v + 1$) The binary representation of all the elements of the ring $\mathbb{Z}_{2^v+1}$ need $r + 1$ bits. The additional bit is required only for the number 2^v. In order to alter this redundancy, a modified binary system — diminished-1 — can be employed. where the number x is represented by $x' = x - 1$ and the value 0 is not used or handled separately [10].

Let $x, y \in \mathbb{Z}_{2^v+1}$ with diminished-1 representation x' and y' respectively. Observe that

$$(x + y)' = (x + y) - 1 = (x' + 1) + (y' + 1) - 1 = x' + y' + 1$$

Hence by using Lemma 7, addition in the Fermat ring can be formulated as

$$x' + y' + 1 \bmod (2^v + 1) = \begin{cases} x' + y' \bmod 2^v & \text{if } x' + y' + 1 \geqslant 2^v \\ x' + y' + 1 & \text{otherwise} \end{cases}$$

which leads us the following important identity

$$x' + y' + 1 \bmod 2^v + 1 = x' + y' + \overline{c}_{out} \bmod 2^v \tag{5.4}$$

Note that Equation (5.4) is also valid for normal representations of numbers. Therefore it is possible to do arithmetic by using normal representations (see [6]).

Remark 12 *For those algorithms heavily use a normal number representation of numbers, a diminished-1 representation suffer from the conversion from and to the normal number representation. In our case we generally do not care about a presentation of a number unless it is the final result.*

Likewise Mersenne arithmetic, if the overflows of the regular addition are fed to the adder after an inversion, a Fermat arithmetic is achieved. Observe that the above methodology function well for CPAs and CSAs, moreover these adders are named as modular CPAs and CSAs after above arrangements.

Notation 6 *Through out this document, for a gate-unit analysis we assume each two input monotonic gate (e.g., AND, NAND) counts as one gate (area or delay), an XOR as two gates (area or delay), and a full adder has an area of seven gates and a delay of four gates.*

If Sklansky parallel-prefix adders are considered for a CPA implementation, the area and time complexity of CPA becomes same for both Fermat and Mersenne rings which is very similar to the standard integer propagate addition (see Table 5.2)

ring	area	delay
Mersenne or Fermat (M/F)	$\frac{3}{2}n \log n + 7n$	$2 \log n + 5$

TABLE 5.2. The cost of CPA (Sklansky parallel-prefix adder).

5.1.3. Multi-operand Addition with Modular Carry-save Adders

As we mention earlier, multi-operand binary addition are optimally implemented by a Wallace tree network. In Section 5.1.1, we derived the relations between

number of operands and the two decisive parameters; height and number of nodes in case of regular integer additions. When it comes to Fermat or Mersenne rings, because of the round around carry feature the complexity of multi-operand addition changes slightly. We continue our analysis with a corollary to Theorem 14;

Corollary 4 *Let $x_1, x_2, \ldots, x_m$ be k-bit numbers. The number of full adders in order to realize a Wallace tree network for Fermat or Mersenne rings is given by*

$$\# \; FA = k(m-2)$$

Proof: Since we are working in finite rings, after every addition the result can be reduced. This reduction is performed by complementing the carry outs and feeding them to the LSB position of the adders in the successive stage. Therefore the size k of the operands of the CSAs remain constant at all times but this means that at each level the number of full adders needed is only k. By using Theorem 14 the number of CSA is given by $\nu(x) = x - 2$ thus $\#$ FA $= k(m-2)$. ■

Remark 13 *Observe that because of the increase in output size one needs more resources for a multi-operand integer addition. On the other hand, for Mersenne and Fermat arithmetic because of round around carry feature the size of output and intermediate sums remain same which fortunately brings a smaller and more importantly a more regular circuit layout.*

In Table 5.3 area and time analysis of an k-bit m operand addition are presented, it can be seen that the delay of Mersenne arithmetic is same as regular integer operations with a less area constrained. When Fermat arithmetic is considered, because of the correction logic one needs slightly larger area and an additional multiplexer delay.

Mersenne	area	delay
Wallace tree	$7k(m-2)$	$4\theta(m)$
CPA	$\frac{3}{2}k\log k + 7k$	$2\log m + 5$
total	$7km + \frac{3}{2}k\log k - 7k$	$4\theta(m) + 2\log m + 5$

Fermat	area	delay
Wallace tree	$8k(m-1)$	$4\theta(m+1)$
corrections	$12k$	3
CPA	$\frac{3}{2}k\log k + 7k$	$2\log m + 5$
total	$8km + \frac{3}{2}k\log k + 11k$	$4\theta(m+1) + 2\log m + 8$

TABLE 5.3. The cost of a multi-operand addition in Fermat and Mersenne aritmetic.

5.1.4. Multiplication

A multiplier consists of three stages: Partial Product Generator (PPG), partial product accumulator (PPA), and carry propagate adder (CPA). The PPG stage generates the partial products from the multiplicand and multiplier. For our discussions PPA corresponds to a Wallace tree network; a multi-operand addition for all the generated partial products that produces the sum in a carry-save form. If a normal form is needed the carry-save form is converted to a regular representation by using a CPA. Since we have already introduced the second and third stages in Sections 5.1.3 and 5.1.2, we comment briefly on PPG.

A naive way of carrying partial products is simply deducing the logical AND of the multiplicand with the multiplier. More advanced methods employ various encoding

techniques aiming to reduce the number of partial products. Among these methods Booth's encoding is the most effective method for many applications. For our immediate purposes an analysis of these encoding techniques are not examined but they are readily available and surely improve the delay function. We simply leave this analysis for further research and choose to use the naive way for our further discussions. With these remarks, in Table 5.4 the delay and area complexity for a $k \times k$ multiplier is given.

Mersenne	area	delay
PPG	k^2	1
Wallace tree	$7k(k-2)$	$4\theta(k)$
CPA	$\frac{3}{2}k\log k + 7k$	$2\log k + 5$
total	$8k^2 + \frac{3}{2}k\log k - 7k$	$4\theta(k) + 2\log k + 6$

Fermat	area	delay
PPG	k^2	1
Wallace tree	$8k(k-1)$	$4\theta(k+1)$
corrections	$12k$	3
CPA	$\frac{3}{2}k\log k + 7k$	$2\log k + 5$
total	$9k^2 + \frac{3}{2}k\log k + 11k$	$4\theta(k+1) + 2\log k + 9$

TABLE 5.4. The cost of $k \times k$ multiplication in Fermat and Mersenne arithmetic.

5.2. Software and Hardware Architectures for Spectral Modular Arithmetic

In the light of previous section, it is possible to give a unit-gate analysis of our algorithms with the specified low level arithmetic architectures. In addition to this analysis we also give a generic evaluation in order to have a platform independent treatment. The following setup is our basis for this evaluation;

- $A(k,l)$; addition of k numbers of l bits modulo q
- $M(k,l)$; $k \times l$ bit multiplication modulo q
- T_f; the delay of f function, i.e. $T_{A(2,l)}$ stands for delay of adding two l-bit numbers modulo q.
- A_f; the area for f function, i.e. $A_{A(2,l)}$ represents the area needed for adding two l-bit numbers modulo q.

Specifically, we describe architectures for SME which could immediately be reduced to outline designs for the SMP or MSMP core since initialization and finalization stages are cost negligible. Recall that SMP and MSMP differ only in calculation of $\beta \cdot \underline{N}(t)$. The former one performs a direct multiplication where the second one achieve this by a successive addition of some pre-computed values. At first glance the second approach seem as an acceleration attempt of modular multiplication by using look-up tables. But we have seen in the previous section that when the modulus is special i.e. $2^v \pm 1$ the use of look up table is not promising. In reality the second approach is more about increasing the size of b which immediately increases the bit size of the realizable modular arithmetic for a fixed transform size d (or decreases the transform size for a fixed modulus).

We make the following conventions in order to come up with the simplest design as possible. We assume

- $\mathbb{Z}_q$, where $q = 2^v \pm 1$ i.e. Mersenne or Fermat ring[3]
- there exist a d point DFT with $\omega = 2^l$ for some $l > 0$
- base b is a power of two i.e. $b = 2^u$,
- the parameters b, q and $s = \lceil d/2 \rceil$ satisfy Inequalities (3.11) and (4.3) for SMP and MSPM respectively (as a core for SME).

In Figure 5.2 we exhibit a higher level architecture for SMP. Before zooming into the boxes and discussing the possible design practices, we like to say a few words about the data flow.

In this architecture, the outputs of the convolution (i.e. Step 1) feed the Z MUXes. For the initial case, each MUX Z chooses the input from Step 1, and then the reduction loop starts. The loop runs d times: at every run, the outputs of the processing units are passed to the interpolation and also fed back to the unit itself. The processing engine waits until the parameters $z_0 + \beta$ and β are generated from the parameter generation logic. After d runs of the loop, the processing units from 0 to $d-1$ outputs the coefficient of the resultant spectral polynomial $Z(t)$. It is important to realize that in this architecture:

- All processing units work in parallel.
- The cyclic shifts are different for each unit in other words the units are not completely identical.
- We do not need a cyclic shifter in unit $d-1$
- The same $z_0 + \beta$ and β are passed to all of the processing units.

---

[3]this could easily be extended to pseudo Mersenne or Fermat rings

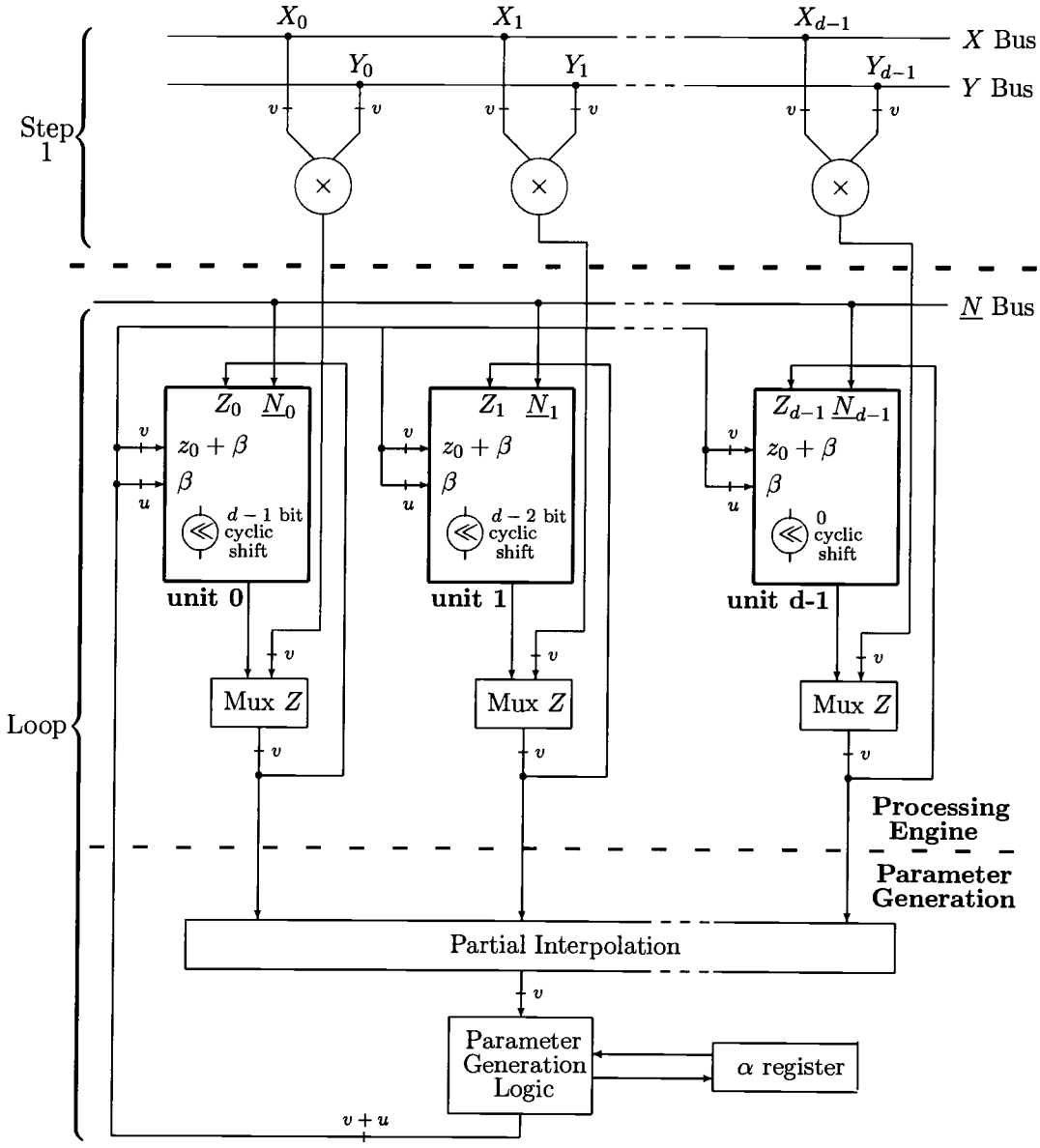

FIGURE 5.2. The hardware architecture of the SMP algorithm.

In Step 1, the convolution property is employed. Given the spectral polynomials $X(t)$ and $Y(t)$, we compute $Z(t)$ such that $Z_i = X_i Y_i \pmod{q}$ for $i = 0, 1, \ldots, d-1$ as seen in Figure 5.3 These multiplications can be realized by employing $v \times v$ modulo multipliers with a generic delay $T_{M(v,v)}$ and area $T_{A(v,v)}$.

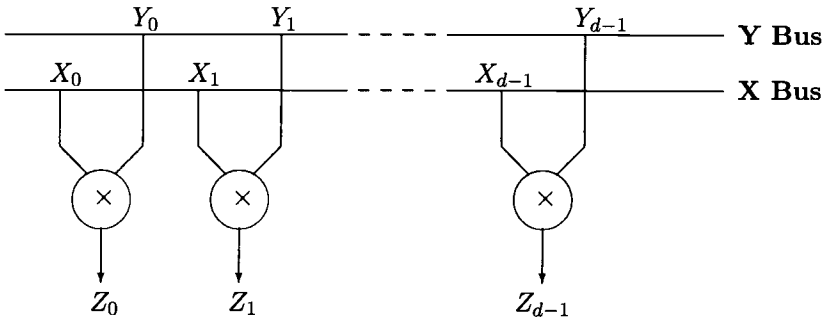

FIGURE 5.3. The archictecture for Step 1 of SMP and MSMP.

If the specific architectures described in Section 5.1 are applied to Step 1 a twofold (word+bit) level parallelism is achieved. We note that, we try to keep the intermediate data in carry-save form at all times. Later, we see that during the reduction loop there is only one step that data need to be partially in normal form. That is why the CPA after the component-wise multiplication is discarded in order to keep the output of the convolution in a carry-save format.

In case of a Mersenne arithmetic, the critical path of the $v \times v$ multiplication without a CPA needs; a simplified 2-to-1 multiplexer for partial product generation followed by a CSA Wallace tree network. When same calculation is considered for Fermat rings we need some additional hardware and time for some corrections. Thus by using Table 5.4, the cost of Step 1 is summarized in Table 5.5.

step	Algorithm: SMP and MSMP		
1	ring	area	delay
generic sequential	F/M	$A_{M(v,v)}$	$d \cdot T_{M(v,v)}$
generic parallel	F/M	$d \cdot A_{M(v,v)}$	$T_{M(v,v)}$
specific parallel	$2^r - 1$	$8v^2 - 14v$	$4\theta(v) + 1$
	$2^r + 1$	$9v^2 + 3v$	$4\theta(v+1) + 4$

TABLE 5.5. The cost of Step 1.

Now it is time to consider the reduction steps; for simplicity we divide the loop with i into two parts;

- **Parameter Generation:** This corresponds to Steps 4, 5 and 6. Here, we compute the parameters z_0, α, and β, and feed them to the main processing units.
- **Processing Engine:** This corresponds to Steps 7, 8, and 9, in which we add a multiple of the modulus to the partial sum and then divide it by the base.

In **Parameter Generation**, Step 4 corresponds to a partial interpolation in which a d-input multi-operand addition followed by a multiplication by d^{-1}. This computes the zeroth coefficient of the time polynomial. Figure 5.4 shows the architecture for these computations.

Observe that d^{-1} is a constant v-bit number so the multiplication is indeed another multi-operand addition. With some recoding techniques, it is possible to deliver this multiplication by a $v/2$ operand addition, and since this addition has to wait for the output of d operand addition, in total, Step 2 can be realized by $d + v/2$ operand addition. On a sequential system we compute that much of additions as

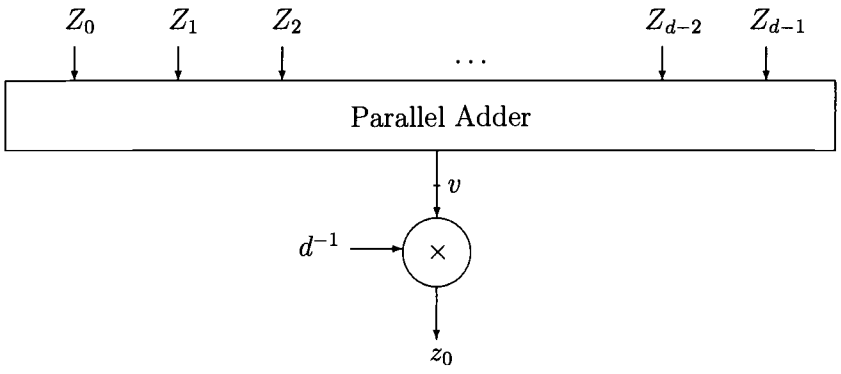

FIGURE 5.4. Partial interpolation.

$$T_{A(d,v)} + T_{A(\frac{v}{2},v)} = (d-1)T_{A(2,v)} + (\frac{v}{2}-1)T_{A(2,v)} = (d+\frac{v}{2}-2)T_{A(2,v)} \ ,$$

and the area needed by the this system is simply $T_{A(2,v)}$. On the contrary, the complexity of a parallel system can be modeled by a complete binary tree network where the nodes of the tree associated with two operand adders, $A(2,v)$. Next lemma traces the delay and area expansion.

Proposition 12 *Let CBT be a complete binary tree network. If k is the number of operands to be added then the tree has height $h = \log k$ and $\frac{3k}{2} - \log(2k)$ nodes.*

Proof: Let CBT be a complete binary tree network with k operands. Being a binary tree CBT obviously has height $\log k$. Additionally the number of nodes is given as follows

$$2^{h-1} - 1 + k - h = 2^{\log k - 1} + k - \log k - 1 = \frac{3k}{2} - \log(2k) \ .$$

■

By using Prosition 12, one realizes a parallel system with delay;

$$T_{A(d,v)} + T_{A(\frac{v}{2},v)} = \log d T_{A(2,v)} + \log(\frac{v}{2}) T_{A(2,v)} = (\log d + \log v - 1) T_{A(2,v)}$$

at an area cost

$$\begin{aligned} A_{A(d,v)} + A_{A(\frac{v}{2},v)} &= (\frac{3d}{2} - \log(2d)) A_{A(2,v)} + (\frac{3v}{4} - \log(\frac{2v}{2})) A_{A(2,v)} \\ &= (\frac{6d+3v}{4} - \log(2vd)) A_{A(2,v)} \end{aligned}$$

If the specifications of Section 5.1 are chosen, we have a simpler analysis of a twofold (word+bit) parallelism. By using the figures in Table 5.3, we tabulate the cost of $d + v/2$ operand addition without the final CPA in Table 5.6.

In a special Fermat ring with some other desirable condition it is possible to replace d^{-1} multiplication by circular shifts.

Proposition 13 *Whenever d and ω are both a power of 2, multiplication by d^{-1} is replaced by a circular shift.*

Proof: Let $\omega = 2^l$, since we have $\omega^d = 2^{ld} = 1 \bmod q$ and d^{-1} is written as

$$d^{-1} = 2^{ld - \log d} \bmod q$$

the multiplication by d^{-1} modulo q can be achieved by a $r = ld - \log d$ bit circular shift. ■

Once d^{-1} multiplication changed by circular shifts, the complexity shrinks for both time and area. In Table 5.6, We summarize what have been said about Step 4.

Steps 5 and 6 as seen in the Figure 5.5.a are called the **Parameter Generation Logic** (PGL) which computes the parameters $z_0 + \beta$ and β. The adders seen in Figure 5.5.a are the usual v-bit adders and they do not need modular reductions since $\alpha, (z_0 + \alpha), (z_0 + \alpha + \beta) < q$.

If the Steps 5 through 8 of SMP are examined carefully, it is seen that the second adder can be discarded by making the following modifications:

step 4	Algorithm: SMP and MSMP		
	ring	area	delay
generic sequential	F/M	$A_{A(2,v)}$	$(d+\frac{v}{2}-2)T_{A(2,v)}$
	F/M $d=2^r$	$A_{A(2,v)}$	$(d-1)T_{A(2,v)}$
generic parallel	F/M	$(\frac{6d+3v}{4}-\log(2vd))A_{A(2,v)}$	$(\log d+\log v-1)T_{A(2,v)}$
	F/M $d=2^r$	$(\frac{3d}{2}-\log(2d))A_{A(2,v)}$	$\log d \cdot T_{A(2,v)}$
specific parallel	M	$7vd+\frac{7}{2}v^2-28v$	$4\theta(d)+4\theta(v/2)$
	F	$8vd+4v^2+4v$	$4\theta(d+1)+4\theta(v/2+1)+6$
	F/M $d=2^r$	$8vd+2v$	$4\theta(d+1)+3$

TABLE 5.6. The cost of Step 4.

5: $\beta := -(z_0+\alpha) \bmod b$

6: $\alpha := (z_0+\alpha+\beta)/b$

7: $Z(t) := Z(t)+\beta \cdot N(t) \bmod q \quad \Rightarrow Z(t) := Z(t)+\beta \cdot (N(t)-1(t)) \bmod q$

8: $Z(t) := Z(t)-(z_0+\beta)(t) \bmod q \quad \Rightarrow Z(t) := Z(t)-z_0(t) \bmod q$

Therefore pre-computing and storing $N(t)-1(t) =: DFT(n(t)-1)$ instead of $N(t)$ eliminate the second adder as seen in Figure 5.5.b A similar analysis is achieved for MSMP by pre-computing and storing

$$\{N_1(t)-1(t), N_2(t)-1(t), \ldots, N_u(t)-1(t)\}$$

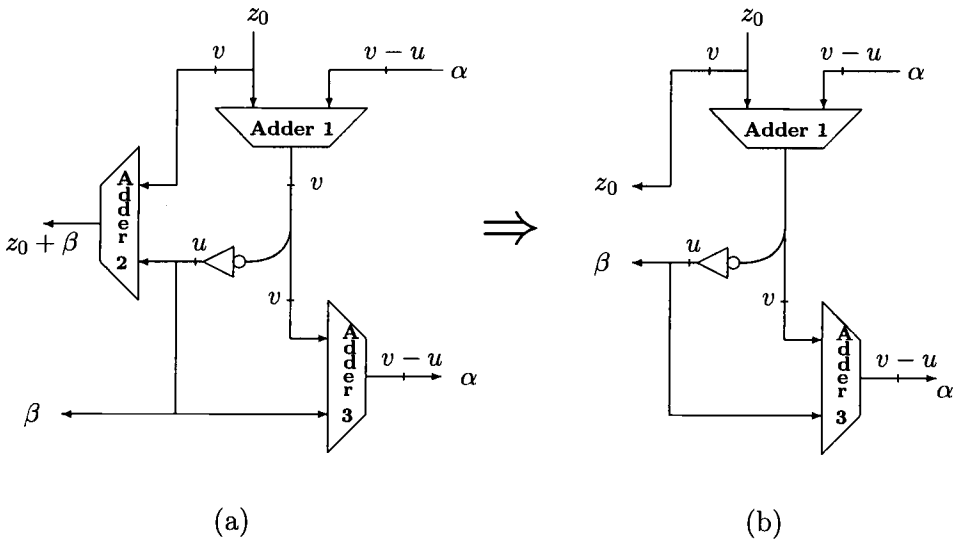

FIGURE 5.5. Parameter Generation Logic.

instead of $\{N_1(t), N_2(t), \ldots, N_u(t)\}$. With these adjustments the cost of PGL becomes equal to the delay of the first adder which is a single two operand addition $A(2, v)$ when a generic notation considered.

For a specific analysis; first of all the output of the partial interpolation is in carry save form. With the first adder the carry (initially 0) is added to the output of the partial interpolation. Since β is the multiplicand for the multiplication in Step 7 we need β in a normal form rather than a carry-save format. Therefore, here we have to engage with a u-bit CPA on the critical path. On the contrary the addition of most significant $v - u$ bits are going to be the carry for the next run which is not in the critical path.

Once the second adder has discarded the delay of PGL becomes equal to a u-bit CPA delay which is $2 \log u + 5$. This is same in both Fermat and Mersenne arithmetic

since u is small enough that addition do not produce any round around carry, hence this adder is realized as an regular integer adder. In Table 5.7 we tabulated the cost for Steps 5 and 6 this analysis along with generic one.

step	Algorithm: SMP and MSMP		
5 - 6	ring	area	delay
generic sequential	F/M	$A_{A(2,v)}$	$T_{A(2,v)}$
generic parallel	F/M	$A_{A(2,u)}$	$T_{A(2,u))}$
specific parallel	F/M	$\frac{3}{2}u\log u + 7u$	$2\log u + 5$

TABLE 5.7. The cost of Step 5–6 after modification.

The **Processing Engine** is the most resource-consuming stage and it corresponds to Steps 7, 8, and 9. In Figure 5.6 the architecture of a single processing unit for SMP is seen if multiplier is changed with a multi-operand addition, a schematic for MSMP can be achieved. The processing engine consists of d such units.

Note that both adders in Figure 5.6 are modulo q adders. The shift operation at the bottom of the figure corresponds to Step 9 of the SMM core. As we pick ω as a power of 2, the multiplications with the coefficients of $\Gamma(t)$ correspond to the constant $d-1-i$ bit circular shifts for $i = 1, 2, \ldots, d-1$.

For a generic evaluation the delay and area are readily available which is the cost of a $u \times v$ multiplier plus 2 additions for SMP and $u+2$ additions in total for MSMP. If the binary tree is used for modeling the multi-operand additions one gets the figures in Table 5.8.

On the other hand if the specific architectures of Section 5.1 are considered, Step 8 can be buried into the Wallace tree of the multiplication or multi-operand addition of Step 7. Therefore, we simply need $u+1$ operand Wallace tree network which is valid

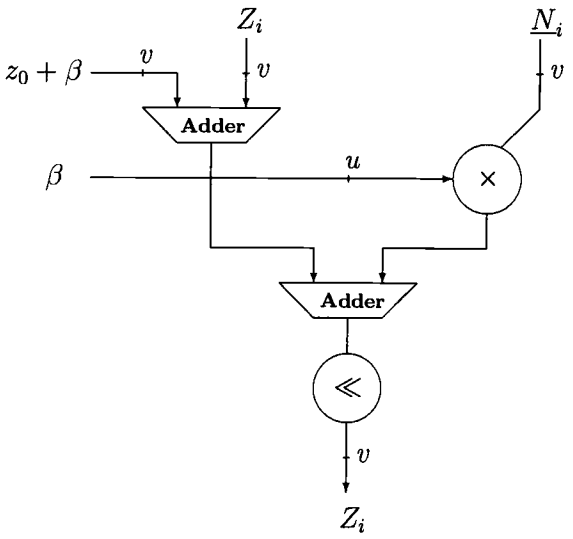

FIGURE 5.6. A Processing unit.

for both SMP and MSMP. The accurate costs are separately tabulated for SMP and MSMP in Table 5.8.

We close the section with some further remarks on possible architectural improvements that can be employed. We believe that these remarks bring some speed-up and optimized hardware for the realizations but we leave the analysis of these remarks for a future research.

Remark 14 *A pipelining is possible for β computation (Step 3) and $\beta \cdot (N(t) - 1(t))$ computation. The Wallace tree do not need to wait the whole computation of the β, instead it can go with the ready bits of β from the CPA. By this the delay for CPA can be embedded into the Wallace tree delay of Step 7.*

Remark 15 *Speaking about the pipelining; the overall architecture is extremely suitable for pipelining. For several modular exponentiation calculations — with or without the same modulus — the throughoutput can be improved excessively.*

step		Algorithm SMP	
7 – 8	ring	area	delay
generic sequential	F/M	$(A_{M(u,v)} + 2A_{A(2,v)})$	$d(T_{M(u,v)} + 2T_{A(2,v)})$
generic parallel	F/M	$d(A_{M(u,v)} + 2A_{A(2,v)})$	$T_{M(u,v)} + 2T_{A(2,v)}$
specific	M	$8uv$	$4\theta(u+1)+5$
parallel	F	$9uv + 20v$	$4\theta(u+2)+8$

step		Algorithm MSMP	
7 – 8	ring	area	delay
generic sequential	F/M	$A_{A(u,v)} + 2A_{A(2,v)}$	$d(u+2)T_{A(2,v)}$
generic parallel	F/M	$d(\frac{3u}{2} - \log(2u+2) + \frac{5}{2})A_{A(2,v)}$	$(1 + \log(u+1))T_{A(2,v)}$
specific	M	$7vu$	$4\theta(u+1)+4$
parallel	F	$9uv + 20v$	$4\theta(u+2)+7$

TABLE 5.8. The cost of Step 7–8 for SMP and MSMP.

Remark 16 *The use of encoding techniques for multiplication (e.g. Booth recoding) can be employed to shrink the size and delay of the Wallace tree.*

Remark 17 *Lastly, we like mention the area can be squeezed by using the convolution step hardware for multiplications or multi-operand additions in the loop.*

5.3. Performance Analysis; Adding everything up

In Tables 5.9 and 5.10 we give the entire complexity of SMP and MSMP by combining the individual pieces described in the previous section.

SMP	ring	area	delay
generic sequential	F/M	$A_{M(v,v)} + 4A_{A(2,v)}$ $+A_{M(u,v)}$	$d \cdot T_{M(v,v)} + d(d \cdot T_{M(u,v)}$ $+(3d + \frac{v}{2} - 1)T_{A(2,v)})$
	F/M $d = 2^r$	$A_{M(v,v)} + 4A_{A(2,v)}$ $+A_{M(u,v)}$	$d \cdot T_{M(v,v)} + d(d \cdot T_{M(u,v)}$ $+3d \cdot T_{A(2,v)})$
generic parallel	F/M	$d(A_{M(v,v)} + A_{M(u,v)})+$ $(\frac{14d+3v}{4} - \log(vd))A_{A(2,v)}$	$T_{M(v,v)} + d(T_{M(u,v)}$ $+\log(4dv)T_{A(2,v)})$
	F/M $d = 2^r$	$d(A_{M(v,v)} + A_{M(u,v)})$ $+(\frac{7v}{2} - \log d)A_{A(2,v)}$	$T_{M(v,v)} + d(T_{M(u,v)}$ $+\log(8d)T_{A(2,v)})$
specific parallel	M	$\frac{23}{2}v^2 + 7vd - 42v$ $+8uv + \frac{3}{2}u\log u + 7u$	$4\theta(v) + 1 + 2d(\lg(u) + 5)+$ $4d(\theta(d) + \theta(v/2) + \theta(u+1))$
	F	$13v^2 + 8vd + 27v$ $+9uv + \frac{3}{2}u\log u + 7u$	$4\theta(v+1) + 4 + 2d(\lg(u) + 19/2)+$ $4d(\theta(d+1) + \theta(v/2+1) + \theta(u+2))$
	F/M $d = 2^r$	$9v^2 + 8vd + 25v$ $+9uv + \frac{3}{2}u\log u + 7u$	$4\theta(v+1) + 4 + 2d(\lg(u) + 8)$ $+4d(\theta(d+1) + \theta(u+2))$

TABLE 5.9. The performance analysis of SMP algorithm.

Remark 18 *Observe that remark 14 readily implies a* $2d(\log u + 3)$ *reduction in delay from the specific parallel implementation.*

Remark 19 *When MSMP algorithm is considered, we mentioned that the basis set* $\{N_1(t), N_2(t), \ldots, N_u(t)\}$ *has to be stored. Therefore one has to consider to allocate a space given by Equation (4.2) in realizations.*

Eventually, we are going to give an example for computing the complexities. We pick our samples from Tables 4.2 as follows:

MSMP	ring	area	delay
generic sequential	F/M	$A_{M(v,v)} + 4A_{A(2,v)}$ $+A_{M(u,v)}$	$d \cdot T_{M(v,v)} + d(du$ $+(3d + \frac{v}{2} - 1))T_{A(2,v)}$
	F/M $d = 2^r$	$A_{M(v,v)} + 4A_{A(2,v)}$ $+A_{M(u,v)}$	$d \cdot T_{M(v,v)} + d(du$ $+3d \cdot T_{A(2,v)})$
generic parallel	F/M	$dA_{M(v,v)} + (\frac{16d+3v+6ud}{4} -$ $\log(vd) - d\log(2u+2))A_{A(2,v)}$	$T_{M(v,v)} + d(\log(u+1)$ $+\log(2dv)T_{A(2,v)})$
	F/M $d = 2^r$	$d(A_{M(v,v)} + A_{M(u,v)})$ $+(\frac{7v}{2} - \log d)A_{A(2,v)}$	$T_{M(v,v)} + d(\log(u+1)$ $+\log(4d)T_{A(2,v)})$
specific parallel	M	$\frac{23}{2}v^2 + 7vd - 42v$ $+8uv + \frac{3}{2}u\log u + 7u$	$4\theta(v) + 1 + 2d(\lg(u) + 9/2)$ $+4d(\theta(d) + \theta(\frac{v}{2}) + \theta(u+1))$
	F	$13v^2 + 8vd + 27v$ $+9uv + \frac{3}{2}u\log u + 7u$	$4\theta(v+1) + 4 + 2d(\lg(u) + 9) +$ $4d(\theta(d+1) + \theta(\frac{v}{2}+1) + \theta(u+2))$
	F/M $d = 2^r$	$9v^2 + 8vd + 25v$ $+9uv + \frac{3}{2}u\log u + 7u$	$4\theta(v+1) + 4 + 2d(\lg(u) + 15/2)$ $+4d(\theta(d+1) + \theta(u+2))$

TABLE 5.10. The performance analysis of MSMP algorithm.

Example 12 *Let's compute the complexity of SMP with a modulus* 704*-bits by using the DFT in the ring* $\mathbb{Z}_q$ *with* $q = 2^{64} + 1$, $\omega = 2$ *and* $u = 11$ *as seen in Table 4.2. We start with the area and delay of a generic sequential setting;*

$$\begin{aligned} T_{SMP} &= d \cdot T_{M(v,v)} + d(d \cdot T_{M(u,v)} + 3d \cdot T_{A(2,v)}) \\ &= 128 \cdot T_{M(64,64)} + 128^2 \cdot T_{M(11,64)} + 3 \cdot 128^2 \cdot T_{A(2,64)} \\ A_{SMP} &= A_{M(v,v)} + A_{M(u,v)} + 4 \cdot A_{A(2,v)} \\ &= A_{M(64,64)} + A_{M(11,64)} + 4 \cdot A_{A(2,64)} \end{aligned}$$

when generic parallel setting is considered we get;

$$\begin{aligned} T_{SMP} &= T_{M(v,v)} + d(T_{M(u,v)} + \log(8d) \cdot T_{A(2,v)}) \\ &= T_{M(64,64)} + 128T_{M(11,64)} + 1280 \cdot T_{A(2,64)} \\ A_{SMP} &= d(A_{M(v,v)} + A_{M(u,v)}) + (\frac{7v}{2} - \log d)A_{A(2,v)} \\ &= 128A_{M(64,64)} + 128A_{M(11,64)} + 217A_{A(2,64)} \end{aligned}$$

With these figures we conclude that the the parallel algorithm is almost optimal in on sequential algorithm.

Lastly, we consider the complexity of the specific implementation described in Section 5.1 which is given for by a unit gate delay and area (i.e. two input monotonic gate such as AND or NAND).

$$\begin{aligned} T_{SMP} &= 4\theta(v+1) + 4 + 2d(\lg(u) + 8) + 4d(\theta(d+1) + \theta(u+2)) \\ &= 4\theta(64+1) + 4 + 2 \cdot 128(\log(11) + 8) + 4 \cdot 128(\theta(128+1) + \theta(11+2)) \\ &= 4 \cdot 10 + 4 + 256(4+8) + 512(11+5) = 11308 \\ A_{SMP} &= 9v^2 + 8vd + 25v + 9uv + \frac{3}{2}u\log u + 7u = 177871 \end{aligned}$$

In other words, a realization of SMP with the above parameters allocate 177871 *gates and have* 11308 *gate delay for an output. If this SMP is used in a SME, with the same area complexity one approximately has a* $11308K$ *gate delay.*

Example 12 gives the performance analysis of for an instance SMP over a Fermat ring. One similarly calculates the complexities of SMP and MSMP over Mersenne or pseudo rings. Moreover, one can combine these in order to find the complexity of using CRT.

6. CONCLUSION

In this dissertation new techniques of performing modular multiplication and exponentiation are proposed. Especially, modular exponentiation is one of the most important arithmetic operation for methods of modern cryptography, such as the RSA and Diffie-Hellman algorithms. Proposed methods use the Discrete Fourier Transform over finite rings, and relies on new techniques to perform the modular reduction operation.

Our initial motivation was obtaining modular arithmetic algorithms fully working in the spectrum in order to benefit the convolution property at a maximum extend. For carrying modular arithmetic one has obviously need to deal with the concept of modular reduction. After defining spectral reduction and related concepts we introduce a spectral reduction algorithm by using the linearity and shifting property of DFT, and than spectral modular multiplication (SMM) and spectral modular exponentiation (SME) came quite naturally.

Most of the time, if some operation is performed in spectrum, the spectral coefficients do not tell much about what has been done in the time. That is why we construct the spectral modular algorithms as the image of some time simulations that are easily be transformed to spectrum. By this we were able to keep track of the activity of time sequences while working in the spectrum.

When it comes to the practicability of the proposed methods, there were many directions to go because of the richness of the spectral theory. First we experiment to apply the methods in a complex spectrum but because of massive computations in the spectrum we understand that the round off errors are hard to control. Therefore we eager to employ finite ring spectrums for not admitting the round off errors in the

computations. Additionally, from a computational point of view, calculations in some special rings such as Fermat and Mersenne can exploit the special arithmetic.

On the contrary, we realized that while running the spectral algorithms in finite ring spectrums, if some parameters in time simulations go out of predefined bounds the spectral algorithms produce erroneous outputs. These ill-favored effects are called overflows and can be easily controlled by choosing larger domains. In Chapter 3 along with the presentation of algorithms, we were dedicated to find the optimal domains for our spectral modular algorithms.

Spectral Modular Product (SMP) which consists of convolution and spectral reduction is presented as the basic building block of both SMM and SME. Indeed SMP decides the optimal domain of SMM and SME, hence for the rest of the dissertation we basically concentrated in analysis and improvements of SMP algorithm.

By using the Theorem 7 we proved the relation between the effective transform length s, wordsize u and ring size q as $2sb^2 < q$ for SMM and

$$(b^2 + b)^2 B(s) + b^2 s < q$$

for SME. Additionally, we demonstrated some convenient parameters for realizations in Table 4.2 for SME. After that we modify SMP in order to improve the optimal domain with a space trade off. We gave the relation

$$(b \log b + b)^2 B(s) + sb \log b < q$$

at a cost of some pre-computations and storage space when the Modified Spectral Modular Product (MSMP) is used. Since with MSMP the effective word size almost doubles therefore we managed to double the operational maximum modulus size for SME.

More importantly we described the use of Chinese Remainder Theorem (CRT) for SME and SMM which permits us to reach maximum modulus sizes by using smaller

processing units. Indeed CRT admits us to reach very large modulus sizes at the same time work in small size rings. We tabulated possible implementation parameters in Tables 4.5 and 4.6 for both SMP and MSMP respectively.

Because of working in the spectrum there exists a vast amount of parallelism potential in computations. Therefore we had the chance of yielding efficient and highly parallel architectures for especially hardware implementations. In Chapter 5 we describe detailed architectures and unit-gate analysis of SMP and MSPM algorithms. While doing this we use some generic and also specific architectures that leads us to have a general and a lower level unit-gate performance analysis.

BIBLIOGRAPHY

[1] A. Schönhage and V. Strassen, "Schnelle multiplikation grafer zahlen," *Computing*, vol. 7, pp. 281–292, 1971.

[2] J. M. Pollard, "The fast Fourier transform in a finite field," *Mathematics of Computation*, vol. 25, pp. 365–374, 1971.

[3] R. E. Blahut, *Fast Algorithms for digital signal processing*, Addison-Wesley publishing Company, 1985.

[4] H. J. Naussbaumer, *Fast Fourier transform and convolution algorithms*, Springer, Berlin, Germany, 1982.

[5] G. A. Jullien Z. Wang and W. C. Miller, "An efficient tree architecture for modulo $2^n + 1$ multiplication," *J. VLSI Signal Processing Systems*, vol. 14, no. 3, pp. 241–248, Dec. 1996.

[6] R. Zimmermann, "Efficient VLSI implementation of modulo $(2^n \pm 1)$ addition and multiplication," in *Proceedings of the 14th IEEE Symposium on Computer Architecture*, 1999, pp. 158–167.

[7] R. Zimmermann, *Binary Adder Architectures for Cell Based VLSI and their Synthesis*, Ph.D. thesis, Swiss Federal Institute of Technology (ETH) Zurich, 1998.

[8] I. Koren, *Computer Arithmetic Algorithms*, Springer, Berlin, Germany, 1993.

[9] J. Sklansky, "Conditional sum addition logic," *IRE Trans. Electron. Comput.*, vol. EC-9, no. 6, pp. 226–231, June 1960.

[10] L.M. Leibowitz, "A simplified binary arithmetic for the Fermat number transform," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 24, pp. 356–359, 1976.

[11] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[12] T. Yanık, E. Savaş, and Ç. K.Koç, "Incomplete reduction in modular arithmetic," *IEE Proceedings - Computers and Digital Techniques*, vol. 149, no. 2, pp. 46–52, Mar. 2002.

[13] Ç. K. Koç and T. Acar, "Montgomery multiplication in $GF(2^k)$," *Designs, Codes and Cryptography*, vol. 14, no. 1, pp. 57–69, Apr. 1998.

[14] Ç. K. Koç, "High-Speed RSA Implementation," Tech. Rep. TR 201, RSA Laboratories, 73 pages, November 1994.

[15] N. Koblitz, *A Course in Number Theory and Cryptography*, Springer, Berlin, Germany, Second edition, 1994.

[16] I. N. Herstein, *Topics in Algebra*, Wiley, Second edition, 1975.

[17] J.-J. Quisquater and C. Couvreur, "Fast decipherment algorithm for RSA public-key cryptosystem," *Electronics Letters*, vol. 18, no. 21, pp. 905–907, Oct. 1982.

[18] J.-L. Beuchat, "A family of modulo (2^n+1) multipliers," Tech. Rep. 5316, Institut National de Recherche en Informatique et en Automatique (INRA), Sept. 2004.

[19] A. Skavantzos and P.B.Rao, "New multipliers modulo 2^n-1," *IEEE Transactions on Computers*, vol. 41, no. 8, pp. 957–961, Aug. 1992.

[20] H. Bonnenberg A.V. Curiger and H. Kaeslin, "Regular VLSI architectures for multiplication modulo (2^n+1)," *IEEE J. Solid State Circuits*, vol. 26, no. 7, pp. 990–994, July 1991.

APPENDICES

APPENDIX A. Pseuodo Mersenne and Fermat Transform Tables

Ring	Prime Factors	Modulus	ω	d	ω	d
$2^{25}-1$	$31 \cdot 601 \cdot 1801$	$(2^{25}-1)/31$	2	25	-2	50
$2^{26}-1$	$3 \cdot 2731 \cdot 8191$	$(2^{26}-1)/3$	2	26	-2	52
$2^{27}-1$	$7 \cdot 73 \cdot 262657$	$(2^{27}-1)/511$	2	27	-2	54
$2^{34}-1$	$3 \cdot 43691 \cdot 131071$	$(2^{34}-1)/3$	2	34	-2	68
$2^{35}-1$	$31 \cdot 71 \cdot 127 \cdot 122921$	$(2^{35}-1)/3937$	2	35	-2	70
$2^{38}-1$	$3 \cdot 174763 \cdot 524287$	$(2^{38}-1)/3$	2	38	-2	76
$2^{39}-1$	$7 \cdot 79 \cdot 8191 \cdot 121369$	$(2^{39}-1)/7$	2	39	-2	78
$2^{46}-1$	$3 \cdot 47 \cdot 178481 \cdot 2796203$	$(2^{46}-1)/3$	2	46	4	23
$2^{49}-1$	$127 \cdot 4432676798593$	$(2^{49}-1)/127$	2	49	-2	98
$2^{51}-1$	$7 \cdot 103 \cdot 2143 \cdot 11119 \cdot 131071$	$(2^{51}-1)/7$	2	51	-2	102
$2^{57}-1$	$7 \cdot 32377 \cdot 524287 \cdot 1212847$	$(2^{57}-1)/7$	2	57	-2	114
$2^{58}-1$	$3 \cdot 59 \cdot 233 \cdot 1103 \cdot 2089 \cdot 3033169$	$(2^{58}-1)/3$	2	58	4	29
$2^{62}-1$	$3 \cdot 715827883 \cdot 2147483647$	$(2^{62}-1)/3$	2	62	-2	124
$2^{64}-1$	$3 \cdot 5 \cdot 17 \cdot 257 \cdot 641 \cdot 65537 \cdot 6700417$	$(2^{64}-1)/255$	2	64	-2	128
$2^{65}-1$	$31 \cdot 8191 \cdot 145295143558111$	$(2^{65}-1)/31$	2	65	-2	130
$2^{74}-1$	$3 \cdot 223 \cdot 1777 \cdot 25781083 \cdot 616318177$	$(2^{74}-1)/3$	2	74	-2	148
$2^{75}-1$	$7 \cdot 31 \cdot 151 \cdot 601 \cdot 1801 \cdot 100801 \cdot 10567201$	$(2^{75}-1)/217$	2	75	-2	150
$2^{78}-1$	$3^2 \cdot 7 \cdot 79 \cdot 2731 \cdot 8191 \cdot 121369 \cdot 22366891$	$(2^{78}-1)/63$	2	78	4	39
$2^{82}-1$	$3 \cdot 83 \cdot 13367 \cdot 164511353 \cdot 8831418697$	$(2^{82}-1)/3$	2	82	4	41
$2^{85}-1$	$31 \cdot 131071 \cdot 9520972806333758431$	$(2^{85}-1)/31$	2	85	-2	170
$2^{86}-1$	$3 \cdot 431 \cdot 9719 \cdot 2099863 \cdot 2932031007403$	$(2^{86}-1)/3$	2	86	4	43
$2^{91}-1$	$127 \cdot 911 \cdot 8191 \cdot 112901153 \cdot 23140471537$	$(2^{91}-1)/127$	2	91	-2	182

$2^{93}-1$	$7 \cdot 2147483647 \cdot 658812288653553079$	$(2^{93}-1)/7$	2	93	-2	186
$2^{94}-1$	$3 \cdot 283 \cdot 2351 \cdot 4513 \cdot$ $13264529 \cdot 165768537521$	$(2^{94}-1)/3$	2	94	4	47
$2^{106}-1$	$3 \cdot 107 \cdot 6361 \cdot 69431 \cdot 20394401$ $\cdot 28059810762433$	$(2^{106}-1)/3$	2	106	4	53
$2^{111}-1$	$7 \cdot 223 \cdot 321679 \cdot 26295457 \cdot 616318177$ $\cdot 319020217$	$(2^{111}-1)/7$	2	111	-2	222
$2^{114}-1$	$3^2 \cdot 7 \cdot 571 \cdot 32377 \cdot 174763 \cdot$ $524287 \cdot 1212847 \cdot 160465489$	$(2^{114}-1)/63$	2	114	4	57
$2^{115}-1$	$31 \cdot 47 \cdot 14951 \cdot 178481 \cdot 4036961$ $\cdot 2646507710984041$	$(2^{115}-1)/1457$	2	115	-2	230
$2^{118}-1$	$3 \cdot 2833 \cdot 37171 \cdot 179951 \cdot$ $1824726041 \cdot 3203431780337$	$(2^{118}-1)/3$	2	118	4	59
$2^{121}-1$	$23 \cdot 89 \cdot 727 \cdot$ $1786393878363164227858270210279$	$(2^{121}-1)/2047$	2	121	-2	242
$2^{122}-1$	$3 \cdot 768614336404564651 \cdot$ 2305843009213693951	$(2^{122}-1)/3$	2	122	-2	244
$2^{128}-1$	$3 \cdot 5 \cdot 17 \cdot 257 \cdot 641 \cdot 65537 \cdot$ $274177 \cdot 6700417 \cdot 67280421310721$	$(2^{128}-1)/255$	2	128	4	64

TABLE 6.1.: Suitable pseudo Mersenne rings with ω and d values.

Ring	Prime Factors	Modulus	w	d	w	d
$2^{17}+1$	$3 \cdot 43691$	$(2^{17}+1)/3$	$-2,4$	17	2	34
$2^{19}+1$	$3 \cdot 174763$	$(2^{19}+1)/3$	$-2,4$	19	2	38
$2^{20}+1$	$17 \cdot 61681$	$(2^{20}+1)/17$	4	20	2	40
$2^{21}+1$	$3^2 \cdot 43 \cdot 5419$	$(2^{21}+1)/9$	$-2,4$	21	2	42
$2^{22}+1$	$5 \cdot 397 \cdot 2113$	$(2^{22}+1)/5$	4	22	2	44
$2^{23}+1$	$3 \cdot 2796203$	$(2^{23}+1)/3$	$-2,4$	23	2	46
$2^{28}+1$	$17 \cdot 15790321$	$(2^{28}+1)/17$	4	28	2	56
$2^{29}+1$	$3 \cdot 59 \cdot 3033169$	$(2^{29}+1)/3$	$-2,4$	29	2	58
$2^{31}+1$	$3 \cdot 715827883$	$(2^{31}+1)/3$	$-2,4$	31	2	62
$2^{34}+1$	$5 \cdot 137 \cdot 953 \cdot 26317$	$(2^{34}+1)/5$	4	34	2	68
$2^{37}+1$	$3 \cdot 25781083 \cdot 1777$	$(2^{37}+1)/3$	$-2,4$	37	2	74
$2^{38}+1$	$5 \cdot 229 \cdot 457 \cdot 525313$	$(2^{38}+1)/5$	4	38	2	76
$2^{39}+1$	$3^2 \cdot 22366891 \cdot 2731$	$(2^{39}+1)/9$	$-2,4$	39	2	78
$2^{40}+1$	$257 \cdot 4278255361$	$(2^{40}+1)/257$	4	40	2	80
$2^{41}+1$	$3 \cdot 83 \cdot 8831418697$	$(2^{41}+1)/3$	$-2,4$	41	2	82
$2^{43}+1$	$3 \cdot 2932031007403$	$(2^{43}+1)/3$	$-2,4$	43	2	86
$2^{44}+1$	$17 \cdot 353 \cdot 2931542417$	$(2^{44}+1)/17$	4	44	2	88
$2^{46}+1$	$5 \cdot 277 \cdot 1013 \cdot 1657 \cdot 30269$	$(2^{46}+1)/5$	4	46	2	92
$2^{47}+1$	$3 \cdot 283 \cdot 165768537521$	$(2^{47}+1)/3$	$-2,4$	47	2	94
$2^{52}+1$	$17 \cdot 308761441 \cdot 858001$	$(2^{52}+1)/17$	4	52	2	102
$2^{53}+1$	$3 \cdot 107 \cdot 28059810762433$	$(2^{53}+1)/3$	$-2,4$	53	2	106
$2^{56}+1$	$257 \cdot 54410972897 \cdot 5153$	$(2^{56}+1)/257$	4	56	2	112
$2^{57}+1$	$3^2 \cdot 571 \cdot 160465489 \cdot 174763$	$(2^{57}+1)/9$	$-2,4$	57	2	114
$2^{58}+1$	$5 \cdot 107367629 \cdot 536903681$	$(2^{58}+1)/5$	4	58	2	116

$2^{59}+1$	$3 \cdot 1824726041 \cdot 37171 \cdot 2833$	$(2^{59}+1)/3$	$-2,4$	59	2	118
$2^{60}+1$	$17 \cdot 241 \cdot 4562284561 \cdot 61681$	$(2^{60}+1)/17$	4	60	2	120
$2^{61}+1$	$3 \cdot 768614336404564651$	$(2^{61}+1)/3$	$-2,4$	61	2	122
$2^{62}+1$	$5 \cdot 384773 \cdot 49477 \cdot 8681 \cdot 5581$	$(2^{62}+1)/5$	4	62	2	124
$2^{65}+1$	$3 \cdot 11 \cdot 131 \cdot 409891 \cdot 7623851 \cdot 2731$	$(2^{65}+1)/3$	$-2,4$	65	2	130
$2^{66}+1$	$5 \cdot 13 \cdot 397 \cdot 4327489 \cdot 312709 \cdot 2113$	$(2^{66}+1)/5$	4	66	2	132
$2^{67}+1$	$3 \cdot 6713103182899 \cdot 7327657$	$(2^{67}+1)/3$	$-2,4$	67	2	134
$2^{68}+1$	$17^2 \cdot 2879347902817 \cdot 354689$	$(2^{68}+1)/17^2$	4	68	2	136
$2^{71}+1$	$3 \cdot 56409643 \cdot 13952598148481$	$(2^{71}+1)/3$	$-2,4$	71	2	142
$2^{73}+1$	$3 \cdot 1795918038741070627 \cdot 1753$	$(2^{73}+1)/3$	$-2,4$	73	2	146
$2^{74}+1$	$5 \cdot 149 \cdot 593 \cdot 184481113 \cdot 231769777$	$(2^{74}+1)/5$	4	74	2	148
$2^{76}+1$	$17 \cdot 1217 \cdot 24517014940753 \cdot 148961$	$(2^{76}+1)/17$	4	76	2	152
$2^{79}+1$	$3 \cdot 20148763660243819578 4363$	$(2^{79}+1)/3$	$-2,4$	79	2	158
$2^{82}+1$	$5 \cdot 181549 \cdot 12112549 \cdot 43249589 \cdot 10169$	$(2^{82}+1)/5$	4	82	2	164
$2^{83}+1$	$3 \cdot 499 \cdot 1163 \cdot 13455809771 \cdot 155377 \cdot 2657$	$(2^{83}+1)/3$	$-2,4$	83	2	166
$2^{85}+1$	$3 \cdot 11 \cdot 26831423036065352611 \cdot 43691$	$(2^{85}+1)/33$	$-2,4$	85	2	170
$2^{86}+1$	$5 \cdot 173 \cdot 1759217765581 \cdot 500177 \cdot 101653$	$(2^{86}+1)/5$	4	86	2	172
$2^{87}+1$	$3^2 \cdot 59 \cdot 96076791871613611 \cdot 3033169$	$(2^{87}+1)/531$	$-2,4$	87	2	174
$2^{88}+1$	$257 \cdot 43872038849 \cdot 119782433 \cdot 229153$	$(2^{88}+1)/257$	4	88	2	176
$2^{89}+1$	$3 \cdot 179 \cdot 18584774046020617 \cdot 62020897$	$(2^{89}+1)/3$	$-2,4$	89	2	178
$2^{91}+1$	$3 \cdot 43 \cdot 25829691707 \cdot 1210483 \cdot 2731 \cdot 224771$	$(2^{91}+1)/129$	$-2,4$	91	2	182
$2^{92}+1$	$17 \cdot 291280009243618888211558641$	$(2^{92}+1)/17$	4	92	2	184
$2^{93}+1$	$3^2 \cdot 529510939 \cdot 2903110321 \cdot 715827883$	$(2^{93}+1)/9$	$-2,4$	93	2	186
$2^{94}+1$	$5 \cdot 7484047069 \cdot 140737471578113 \cdot 3761$	$(2^{94}+1)/5$	4	94	2	188
$2^{96}+1$	$641 \cdot 18446744069414584321 \cdot 6700417$	$(2^{96}+1)/641$	4	96	2	192

$2^{97}+1$	$3 \cdot 971 \cdot 1553 \cdot 1100876018364883721 \cdot 31817$	$(2^{97}+1)/3$	$-2,4$	97	2	194
$2^{101}+1$	$3 \cdot 845100400152152934331135470251$	$(2^{101}+1)/3$	$-2,4$	101	2	202
$2^{103}+1$	$3 \cdot 8142767081771726171 \cdot 415141630193$	$(2^{103}+1)/3$	$-2,4$	103	2	206
$2^{104}+1$	$257 \cdot 7891988172627109114376362 3681$	$(2^{104}+1)/257$	4	104	2	208
$2^{106}+1$	$5 \cdot 15358129 \cdot 586477649 \cdot 1801439824104653$	$(2^{106}+1)/5$	4	106	2	212
$2^{107}+1$	$3 \cdot 643 \cdot 84115747449047881488635567801$	$(2^{107}+1)/3$	$-2,4$	107	2	214
$2^{109}+1$	$3 \cdot 2077756847362348863128179 \cdot 104124649$	$(2^{109}+1)/3$	$-2,4$	109	2	218
$2^{111}+1$	$3^2 \cdot 1777 \cdot 3331 \cdot 17539$ $\cdot 25781083 \cdot 107775231312019$	$(2^{111}+1)/9$	$-2,4$	111	2	222
$2^{113}+1$	$3 \cdot 227 \cdot 48817$ $\cdot 636190001 \cdot 491003369344660409$	$(2^{113}+1)/3$	$-2,4$	113	2	226
$2^{114}+1$	$5 \cdot 13 \cdot 229 \cdot 457 \cdot 131101$ $\cdot 160969 \cdot 525313 \cdot 275415303169$	$(2^{114}+1)/65$	$-2,4$	114	2	228
$2^{116}+1$	$17 \cdot 59393$ $\cdot 822801951671441198323905681 77$	$(2^{116}+1)/17$	4	116	2	232
$2^{118}+1$	$5 \cdot 1181 \cdot 3541 \cdot 157649$ $\cdot 174877 \cdot 5521693 \cdot 104399276341$	$(2^{118}+1)/5$	$-2,4$	118	2	236
$2^{120}+1$	$97 \cdot 257 \cdot 673 \cdot 394783681$ $\cdot 46908728641 \cdot 4278255361$	$(2^{120}+1)/257$	32	48	$\sqrt{32}$	96
$2^{121}+1$	$3 \cdot 683 \cdot 117371$ $\cdot 1105418458279780045573606 1107$	$(2^{121}+1)/4098$	$-2,4$	121	2	242
$2^{122}+1$	$5 \cdot 733 \cdot 1709 \cdot 3456749$ $\cdot 8831418697 \cdot 13194317913029593$	$(2^{122}+1)/5$	4	122	2	244
$2^{123}+1$	$3^2 \cdot 83 \cdot 739 \cdot 165313$ $\cdot 8831418697 \cdot 13194317913029593$	$(2^{123}+1)/747$	$-2,4$	123	2	146

$2^{124}+1$	$17 \cdot 290657 \cdot 3770202641$ $\cdot 1141629180401976895873$	$(2^{124}+1)/17$	4	124	2	248
$2^{127}+1$	$3 \cdot 5671372782015641057722910123862803524 3$	$(2^{127}+1)/3$	$-2,4$	127	2	254

TABLE 6.2.: Suitable pseudo Fermat rings with ω and d values.