Spectral Modular Exponentiation

Gökay Saldamlı *
Samsung Electronics
Embedded Software Center
Yongin-city, Gyeonggi-do, S. Korea
g.saldamli@samsung.com

Çetin Kaya Koç
Oregon State University
School of EECS
Corvallis, OR 97331, USA
koc@eecs.oregonstate.edu

Abstract

We describe a new method to perform the modular exponentiation operation, i.e., the computation of $c = m^e \bmod n$*, where* c, m, e *and* n *are large integers. The new method uses the discrete Fourier transform over a finite ring, and relies on new techniques to perform multiplication and reduction operations. The method yields efficient and highly parallel architectures for hardware realizations of public-key cryptosystems requiring the modular exponentiation as the core computation, such as the RSA and Diffie-Hellman algorithms.*

1. Introduction

Most public key cryptosystems require resource-intensive arithmetic calculations in certain mathematical structures such as finite fields, groups and rings. The efficient realizations of the these operations including *modular multiplication, inversion, and exponentiation* are at the center of research activities in cryptographic engineering. In this study, new techniques of performing modular multiplication and exponentiation based on a new reduction operation are proposed (Section 2). These methods work completely in the frequency domain (spectrum) with some exceptions such as the initial, final and some partial transformations calculations.

Spectral techniques for integer multiplication have been known for over a quarter of a century. Using the spectral integer multiplication of Schönhage and Strassen [9], 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 using possibly Schönhage-Strassen and then performing the reduction in the time domain. This approach is preferable if the input length is large enough to meet the asymptotic crossover of Schönhage-Strassen, assuming the reduction has a constant cost. Additionally, if the naive method is used for operations involving consecutive multiplications, because of the costly forward and backward transformation computations, the asymptotic crossover of these operations would be similar to what a single modular multiplication has. Unfortunately these crossovers are larger than the key sizes of the most public-key cryptosystems; thus, in practice the naive way is hardly used.

We propose a modular multiplication that can be performed on the Fourier representations of integers. In such a representation, multiplications are readily available by the convolution property, and thus, operations involving several modular multiplications can be efficiently computed.

After giving some preliminary notation and a formal definition of Discrete Fourier Transform (DFT) over $\mathbb{Z}_q$ (i.e., the ring of integers with multiplication and addition modulo a positive integer q), we describe our main idea of spectral modular arithmetic including *spectral modular multiplication* (SMM) and *spectral modular exponentiation* (SME) in the next section.

In Section 3, we describe methodologies for selecting the parameters for SME in order to apply the algorithm to public key cryptography. In Section 4, we turn our attention to the architectural issues and performance analysis: we show that spectral algorithms yield efficient and highly parallel architectures for hardware implementations of public-key cryptosystems. We conclude our work with some final comments in the final section.

2 Spectral Modular Arithmetic

2.1 Discrete Fourier Transform

Spectral techniques are widely accepted and used in the field of digital signal processing, hence most existing nota-

*This work was performed while Gökay Saldamlı was a Ph.D. student at Oregon State University.

tion and concepts come from this theory. For many reasons, the signals and admissible operations on these signals of such a theory are quite different than of a theory of computer arithmetic. For instance, when we multiply integers using FFT (or convolutions), first we break the integers into words and then process on these partitions. Note that any small perturbation in these components would totally change represented integer. On the other hand, approximations on the signal components without changing the main characteristics of the original signal are allowable in digital signal processing.

Therefore, we believe we need to develop a more suitable notation that would permit us to have a better understanding of employing the spectral techniques for the computer arithmetic related problems. While doing this, we follow a polynomial representation instead of the standard sequence representation of digital signal processing. Such a presentation is necessary for our purposes and moreover, it states the different nature of number representing signals from a classical signal processing analysis. We start with a formal definition of DFT.

Definition 1 *Let ω be a primitive d-th root of unity in $\mathbb{Z}_q$ and, let $x(t)$ and $X(t)$ be polynomials of degree $d-1$ having entries in $\mathbb{Z}_q$. The DFT map over $\mathbb{Z}_q$ is an invertible set map sending $x(t)$ to $X(t)$ given by the following equation;*

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

with the inverse

$$x_i = IDFT_d^{\omega}(X(t)) := d^{-1} \cdot \sum_{j=0}^{d-1} X_j \omega^{-ij} \bmod q,$$

for $i, j = 0, 1, \ldots, d-1$. We say $x(t)$ and $X(t)$ are transform pairs, $x(t)$ is called a **time polynomial** *and sometimes $X(t)$ is named as the* **spectrum** *of $x(t)$.*

In the literature, DFT over a finite ring spectrum (1) is also known as the *Number Theoretical Transform (NTT)*. Moreover, if q has some special form such as a Mersenne or a Fermat number, the transform named after this form; i.e., *Mersenne Number Transform (MNT)* or *Fermat Number Transform (FNT)*.

Note that, unlike the DFT over the complex numbers, the existence of DFT over finite rings is not trivial. In fact, Pollard [7] mentions that the existence of primitive root d-th of unity and the inverse of d do not guarantee the existence of a DFT over a ring. He adds that a DFT exists in ring R if and only if each quotient field R/M (where M is maximal ideal) possesses a primitive root of unity. If $R = \mathbb{Z}_q$ is taken, one gets the following corollary;

Corollary 1 *There exists a d-point DFT over the ring $\mathbb{Z}_q$ that supports the circular convolution if and only if d divides $p-1$ for every prime p factor of q.*

Proof. See Chapter 6 of Blahut [2] or Chapter 8 of Nausbaumer [6]. ■

2.2 Spectral Modular Reduction (SMR)

In order to compute the modular reduction, a Montgomery type [5] method can be employed. Instead of attacking to compute "$x \bmod n$" directly, it is possible to derive the reduction after performing a related computation

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

where $\gcd(n, r) = 1$. At first glance, this seems computationally pointless because of inversion involved but the selection of r changes this first impression drastically. In Algorithm 1, we present such a reduction methodology after presenting a related notation. We gently refer the reader to [8] for a proof.

Notation 1 *Let $a \in \mathbb{Z}$ be a constant number, a degree d polynomial with all of its coefficients equal to a (i.e., $a(t) = a + at + at^2 + \ldots + at^d$) is denoted by $a(t)$.*

Algorithm 1 *Spectral Reduction Algorithm*
Suppose that there exists a d-point DFT map for some principal root of unity ω in $\mathbb{Z}_q$. Let $x(t)$ & $X(t)$ and $n(t)$ & $N(t)$ be transform pairs with $deg(n(t)) = e \leq d = deg(x(t))$ and $n(b)$ is a multiple of the modulus n where $n_0 = 1$.
Input: *$X(t)$ and $N(t)$; spectral polynomials*
Output: *$Y(t) = DFT(y(t))$ where $y \equiv x2^{-db} \bmod n$ and $y = y(b)$,*

1: $Y(t) := X(t)$
2: $\alpha := 0$
3: **for** $i = 0$ **to** $d-1$
4: $\quad y_0 := d^{-1} \cdot (Y_0 + Y_1 + \ldots + Y_{d-1}) \bmod q$
5: $\quad \beta := -(y_0 + \alpha) \bmod b$
6: $\quad \alpha := (y_0 + \alpha + \beta)$ **div** b
7: $\quad Y(t) := Y(t) + \beta \cdot N(t) \bmod q$
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: $Y(t) := Y(t) + A(t)$,
12: **return** $Y(t)$

where $\odot$ stands for component-wise multiplication; $A(t)$ is the DFT pair of the base polynomial of $\alpha(t)$ and $\Gamma(t) = 1 + \omega^{-1}t + \omega^{-2}t^2 + \ldots + \omega^{-(d-1)}t^{(d-1)}$.

2.3 Spectral Modular Multiplication

Convolution and the SMR algorithm can easily be combined to harvest a spectral modular multiplication algorithm in a finite ring spectrum. In order to have a clear presentation we divide our presentation into 3 sub-procedures as seen in Figure 1. Note that the initial and final stages consist of some data arrangements where the *Spectral Modular Product* (SMP) procedure 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 again. SMP procedure and SMM are given as follows:

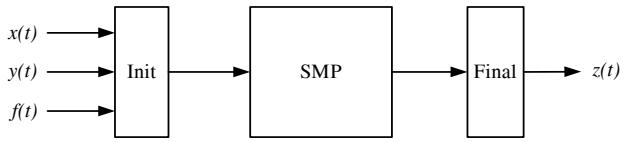

Figure 1. Spectral Modular Multiplication.

Algorithm 2 *Spectral Modular Product*
Suppose that there exist a d-point DFT map for some principal root of unity ω in $\mathbb{Z}_q$, and $X(t), Y(t)$ and $N(t)$ are transform pairs of $x(t), y(t)$ and $n(t)$ respectively. We assume that $x(b) = x$, $y(b) = y$ and $n(b)$ is a multiple of modulus n with $n_0 = 1$ for some integers $x, y < n$ and $b > 0$.
Input: *$X(t), Y(t)$ and $N(t)$; spectral polynomials*
Output: *$Z(t) = DFT(z(t))$ where $z \equiv xy2^{-db} \bmod n$ and $z(b) = z$,*

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

Algorithm 3 *(Spectral Modular Multiplication)*
Suppose that there exist a d-point DFT map and $n(b)$ is a multiple of modulus n where $n_0 = 1$, $deg(n(t)) = s - 1$, $s = \lceil d/2 \rceil$, $gcd(b, n) = 1$ and $b > 0$.
Input: *A modulus $n > 0$ and $x, y < n$*
Output: *$z \equiv xy \bmod n$.*

1: *Compute* $\lambda(t)$, $\lambda = b^d \bmod n$.
2: *Compute* $x^d(t) := x(t) \cdot t^d$ *for* $x \cdot \lambda \bmod n$.
3: $X^d(t) :=$ ***DFT***$(x^d(t))$
4: $Y(t) :=$ ***DFT***$(y(t))$
5: $N(t) :=$ ***DFT***$(n(t))$
6: $Z(t) :=$ ***SMP***$(X^d(t), Y(t), N(t))$
7: $z(t) :=$ ***IDFT***$(Z(t))$
8: **return** $z(b)$

Since we operate in a finite ring spectrum, one has to deal with overflows that might occur during the computations. In fact, Algorithm 3 gives a correct result if the intermediate values stay bounded. The following theorem states the condition when overflows do not occur. The reader is referred to [8] for a proof.

Theorem 1 *Algorithm 3 computes $z \equiv xy \bmod n$, if the parameters b, q and s satisfies $2sb^2 < q$.*

2.4 Spectral Modular Exponentiation

In general, a single classical modular multiplication is faster than a single SMM; however, spectral methods are very effective when several modular multiplications with respect to the same modulus are needed. 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. Such a setup needs a consecutive use of SMM; also means a consecutive use of DFT and IDFT operations (obviously redundant computations as seen Figure 2). Therefore, if the data is kept in the Fourier domain at all times, the backward and forward transforms are bypassed. Consequently, this approach decreases the asymptotic crossovers of the spectral methods to cryptographic sizes while computing the modular exponentiation.

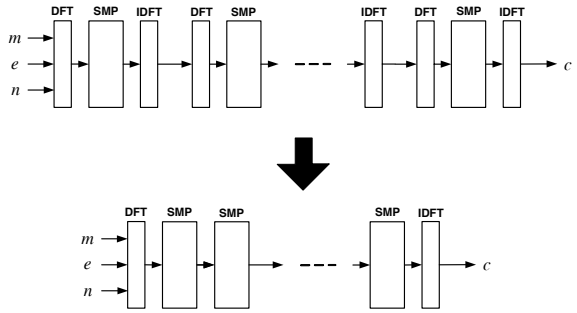

Figure 2. Spectral Modular Exponentiation.

There are many methods for carrying general exponentiation. Mostly efficiency comes from two resources; one is to decrease the time to multiply; the other is to reduce the number of multiplications. In practice one does both. Notice that, until now our objective was reducing the modular

multiplication which is categorized in the first category. For the rest of this study we keep this goal and simply consider using the binary method (see [4]) 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 bit value, a subsequent multiplication is performed. We describe the spectral modular exponentiation algorithm by using a left-to-right binary method below.

Algorithm 4 *(Spectral Modular Exponentiation)*
Suppose that there exist a d-point DFT map and $n(b)$ is a multiple of modulus n where $n_0 = 1$, deg$(n(t)) = s - 1$, $s = \lceil d/2 \rceil$, gcd$(b, n) = 1$ and $b > 0$.
Input: *A modulus $n > 0$ and $m, e < n$*
Output: *$c \equiv m^e \bmod n$*

1: *Compute $\lambda(t)$ such that $\lambda = b^{2d} \bmod n$.*
2: $N(t) :=$ *DFT*$(n(t))$
3: $\Lambda(t) :=$ *DFT*$(\lambda(t))$
4: $M(t) :=$ *DFT*$(m(t))$
5: $M'(t) :=$ *SMP*$(M(t), \Lambda(t), N(t))$
6: $C(t) :=$ *SMP*$(1(t), \Lambda(t), N(t))$
7: **for** $i = j - 2$ **downto** *0*
8: $\quad C(t) :=$ *SMP*$(C(t), C(t), N(t))$
9: $\quad$ **if** $e_i = 1$ **then** $C(t) :=$ *SMP*$(C(t), M'(t), N(t))$
10: $C(t) :=$ *SMP*$(C(t), 1(t))$
11: $c(t) :=$ *IDFT*$(C(t))$
12: **return** $c(b)$

Once again, we need to guarantee that overflows do not occur, in other words the coefficients of intermediate or final results should not be winding over q.

Theorem 2 *Algorithm 4 computes an almost modular reduction, $c \equiv m^e \bmod n$ if the parameters b, q and s satisfies the following inequality*

$$(b^2 + b)^2 B(s) + b^2 s < q \qquad (2)$$

where

$$\begin{aligned} B(s) &= \frac{-2s^3}{3} - s^2 + 2s^2 r_1 - \frac{s}{3} + \frac{r_1}{3} + 2sr_1 \\ r_1 &= -2 + \frac{1}{3}\sqrt{3 + 18s^2 + 18s}\,. \end{aligned}$$

3 Applications and Further Improvements

Modular exponentiation is one of the most important arithmetic operation in modern cryptography. For example, the RSA algorithm requires exponentiation in Z_n for some positive integer n, whereas Diffie-Hellman key agreement and the ElGamal scheme use exponentiation in some large prime fields (see [3]). In this section, we describe the methodologies of selecting the parameters for SME in order to use the method in public-key cryptography. In particular the Inequality (2) presents a solid basis for the relation between the parameters b, q and s. This bound can be improved in many ways which we also investigate throughout this section.

3.1 Parameter Selection for RSA

In this section we tabulate some example parameters for modular exponentiations using the SME method. Once the underlying ring, the DFT length and the principal root of unity are selected, the maximum modulus size used in the SME method is computed by finding the base $b = 2^u$. The relation between these parameters is computed after determining the maximum b satisfying the Inequality (2).

Bits k	Ring $\mathbb{Z}_q$	DFT d	Root ω	Wordsize u	Words s
518	$2^{73} - 1$	73	2	14	37
704	$2^{64} + 1$	128	2	11	64
1,185	$2^{79} - 1$	158	-2	15	79
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

Table 1. SMP Parameter selection for SME.

In Table 1, some sample rings with DFT parameters are given. We give an example to show how we get these figures. We first select a ring, for instance, lets take $q = 2^{79} - 1$. This comes with the principal root of unity $\omega = -2$, the length $d = 158$ and $s = \lceil d/2 \rceil = 79$. Plugging these values into the Inequality (2) gives

$$138754.3b^4 + 277508.7b^3 + 138833.3b^2 < 2^{79} - 1$$

and then by inspection $b = 2^{15} \Rightarrow u = 15$ is found. Therefore, we may perform an exponentiation of maximum operand size equals to $k = s \cdot u = 79 \cdot 15 = 1185$ using SME with the specified parameters.

3.2 Modified Spectral Modular Product

If the SMP (i.e., Algorithm 2) is considered, the boundary analysis depends heavily on $\beta \cdot N(t)$ multiplication of Step 7. It is possible to replace this multiplication by a multi-operand addition at a cost of some pre-computation and extra memory. This replacement gives a reasonable amount of radius shrinkage. Additionally, replacing the multiplication by an array adder 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 $n(t) = n_1(t)$). We can now write $\beta \cdot N(t)$ as

$$\beta \cdot N(t) = \sum_{i=1}^{u} \beta_i \cdot N_i(t) \ , \tag{3}$$

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

Plugging the Equation (3) into the Algorithm 2 gives us the modified Spectral modular product algorithm;

Algorithm 5 *Modified Spectral Modular Product (MSMP) Suppose that there exist a d-point DFT map for some principal root of unity ω in $\mathbb{Z}_q$, and $X(t)$ and $Y(t)$ are transform pairs of $x(t)$ and $y(t)$ respectively where $x(b) = x$ and $y(b) = y$. Let $\mathcal{N} = \{N_1(t), N_2(t), \ldots, N_u(t)\}$ be the set of special polynomials as described above;*
Input: *$X(t), Y(t)$ and a basis set $\mathcal{N}$*
Output: *$Z(t) = DFT(z(t))$ where $z \equiv xy2^{-db} \bmod n$ and $z(b) = z$,*

1: $Z(t) := X(t) \odot Y(t)$
2: $\alpha := 0$
3: **for** $i = 0$ **to** $d - 1$
4: $\quad z_0 := d^{-1} \cdot (Z_0 + Z_1 + \ldots + Z_{d-1}) \bmod q$
5: $\quad \beta := -(z_0 + \alpha)$ *mod* b
6: $\quad \alpha := (z_0 + \alpha + \beta)/b$
7: $\quad Z(t) := Z(t) + \sum_{i=1}^{u} \beta_i \cdot N_i(t) \bmod q$
8: $\quad Z(t) := Z(t) - (z_0 + \beta)(t) \bmod q$
9: $\quad Z(t) := Z(t) \odot \Gamma(t) \bmod q$
10: **end for**
11: $Z(t) := Z(t) + A(t)$
12: **return** $Z(t)$

Observe that the basis set $\mathcal{N}$ needs to be pre-computed and stored. This requirement 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 at the beginning of Algorithm 3 or 4. 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 (3), 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

$$(b \log(b) + b)^2 B(s) + b \log(b) s < q. \tag{4}$$

This new bound is a good improvement for a reasonable space cost and gives us the improved parameters which are tabulated in Table 2 below.

Bits k	Ring $\mathbb{Z}_q$	DFT d	Root ω	Wordsize u	Words s
540	$2^{59} - 1$	59	2	18	30
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
4,251	$2^{109} - 1$	218	-2	39	109

Table 2. MSMP parameter selection for SME.

4 Architectures and Performance Analysis

In this section, we give architectures and performance analysis of spectral modular algorithms. Spectral methods brings parallelism in computations by dividing the larger problems into smaller pieces. As a result, it is possible to employ some low level parallelism on the resulting partitions. In general, large size problems suffer from this kind of parallelism. For instance, a parallel realization of a multiplier has quadratic area complexity and infeasible to realize for large input sizes.

Spectral modular algorithms are dominated by multi-operand additions and multiplications over some special rings. We briefly review the architectures described in [11] and [12], in particular multiplication and multi-operand addition for Fermat and Mersenne rings. In Section 4.2, beside a generic analysis, we append a combined analysis of high and low level parallelism.

There are other efficient ways of carrying low level operations, for instance, check [10] and [1] for table-lookup methods. Further, in [1], discussions on the use of Booth's recoding can be found.

4.1 Fermat/Mersenne Ring Arithmetic

An integer ring having $q = 2^v \pm 1$ elements is the most suitable one for the SME computation since the modular arithmetic operations for such q are simplified. Moreover, if the principal root of unity is chosen as a power of 2, spectral coefficients are computed only using additions and circular shifts. The rings with $q = 2^v - 1$ are called the *Mersenne rings*, while the rings having $q = 2^v + 1$ are called the *Fermat rings*.

Observe that carrying arithmetic in Mersenne rings is equivalent to doing one's complement operations. Although the arithmetic in Fermat rings is more complicated than one's complement arithmetic, with certain encoding techniques this difference can be minimized [8].

Obviously addition and multiplication are the basic operations of our interest. Recall that Partial Product Generator (PPG), partial product accumulator (PPA), and carry propa-

gate adder (CPA) are the main stages of a multiplier. For our discussions PPA corresponds to a Wallace tree network having a height function, $\theta(x)$ for operands number between 14 and 95 as follows;

x	14-19	20-28	29-42	43-63	64-94
$\theta(x)$	6	7	8	9	10

In Table 3, we tabulate the delay and area functions of basic operations for Fermat and Mersenne rings.

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

Fermat	area	delay
PPG	km	1
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	$9km + \frac{3}{2}k\log k + 11k$	$4\theta(m+1) + 2\log m + 9$

Table 3. A $k \times m$ multiplication cost for Fermat and Mersenne rings.

Remark that we tend to keep the data in a carry-save form to avoid the horizontal carry propagation delay. If a normal form is needed the carry-save form is converted to a regular representation by using a CPA.

4.2 Software and Hardware Architectures for Spectral Modular Arithmetic

In this section we 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.

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 $\beta \cdot N(t)$. The first 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. 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. For simplicity, a complete analysis for MSMP is given in this text. A similar analysis for SMP can be found in [8].

In Figure 3 we exhibit a higher level architecture for MSMP. Before zooming into the boxes and discussing the possible design practices, we like to say a few words about the data flow.

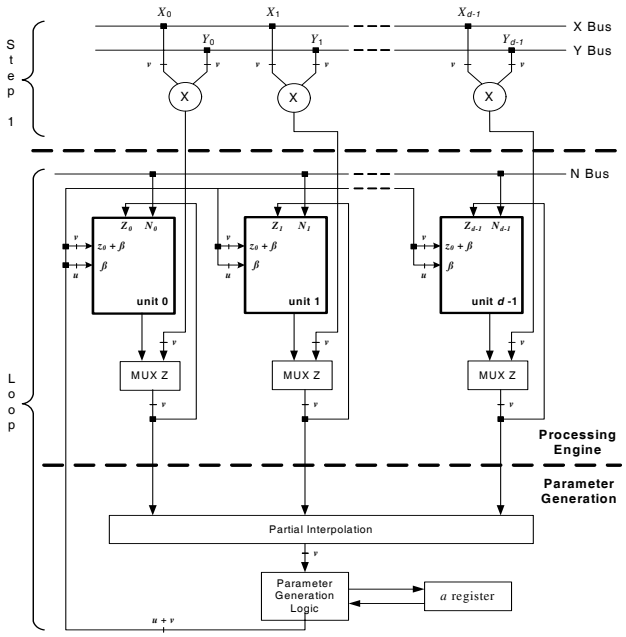

Figure 3. MSMP architecture.

In this architecture, the outputs of the convolution (i.e., Step 1) feed the Z MUXes. For the initial case, each Z MUX 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. Notice that all processing units work in parallel. 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 $d-1$ returns the coefficient of the resultant spectral polynomial $Z(t)$.

Going back to MSMP 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 $i = 0, 1, \ldots, d-1$. Note that these multiplications can be realized by employing $v \times v$ modulo multipliers. In case of a Mersenne arithmetic, these multipliers without a CPA has $4\theta(v) + 1$ delay with a $8v^2 - 14v$ area need. When same calculation is considered for Fermat rings we need some additional hardware and time for some corrections which is given by $9v^2 + 3v$ area and $4\theta(v+1) + 4$ delay.

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.

Observe that d^{-1} is a constant v-bit number so we need another multi-operand addition. With some recoding techniques, it is possible to deliver this multiplication by a $v/2$ operand addition. Therefore, Step 2 can be realized by a $d+v/2$-operand addition in total having the complexity tabulated in Table 4.

In a special Fermat ring with some other desirable conditions, it is possible to replace d^{-1} multiplication by a circular shift.

Proposition 1 *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 a circular shift, the complexity shrinks for both time and area. In Table 4, We summarize what have been said about Step 4.

ring	area	delay
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 4. The cost of Step 4.

Steps 5 and 6 are called the **Parameter Generation Logic** (PGL) in which we compute the parameters $z_0 + \beta$ and β (see Figure 4.a). Note that, the adders in Figure 4.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, the second adder can be discarded by making the following modifications:

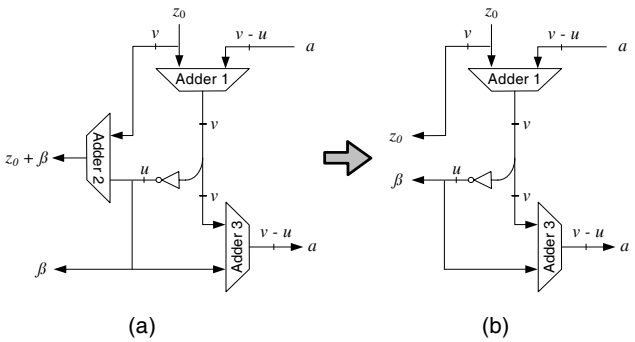

Figure 4. Parameter Generation Logic.

$$\begin{aligned} 5: &\ \beta := -(z_0 + \alpha) \bmod b \\ 6: &\ \alpha := (z_0 + \alpha + \beta)/b \\ 7': &\ Z(t) := Z(t) + \beta \cdot (N(t) - 1(t)) \bmod q \\ 8': &\ Z(t) := Z(t) - z_0(t) \bmod q \end{aligned}$$

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 4.b. A similar modification is possible for MSMP by pre-computing and storing

$$\{N_1(t) - 1(t), N_2(t) - 1(t), \ldots, N_u(t) - 1(t)\}$$

instead of $\{N_1(t), N_2(t), \ldots, N_u(t)\}$. With these adjustments the cost of PGL becomes equivalent to the delay of the first adder which is a single two operand addition.

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. Therefore, here we have to engage a u-bit CPA on the critical path. On the contrary, the addition of most significant $v - u$ bits is the carry to the next run which is not in the critical path.

Once the second adder has been discarded the delay of PGL becomes equivalent 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 does not produce any round around carry, hence this adder is realized as a regular integer adder having $2 \log u + 5$ delay for $\frac{3}{2} u \log u + 7u$ area.

The **Processing Engine** is the most resource-consuming stage and it corresponds to Steps 7, 8, and 9. In Figure 5 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 are modulo q adders. The shift operation at the bottom of the figure corresponds to Step 9 of the SMP core. As we pick ω as a power of 2, the

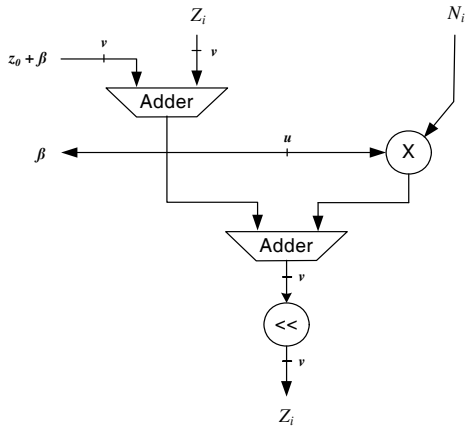

Figure 5. A Processing unit.

multiplications with the coefficients of $\Gamma(t)$ correspond to a constant $d-1-i$ bit circular shifts for $i = 1, 2, \ldots, d-1$.

Notice that, Step 8 can be buried into the Wallace tree of the multiplication of Step 7. Therefore, we simply need a $u+1$ operand Wallace tree network which needs $7vu$ area for $4\theta(u+1)+4$ delay in Mersenne rings and $9uv+20v$ area for $4\theta(u+2)+7$ delay in Fermat rings.

After adding everything up described in previous paragraphs, in Table 5 we give the entire complexity of MSMP. A similar analysis for SMP can be found in [8].

ring	area	delay
M	$8v^2d + 7uvd +$ $\frac{7}{2}v^2 + 7vd - 28v$ $+\frac{3}{2}u\log u + 7u$	$4\theta(v) + 1 + 2d\log(u)$ $+9d + 4d(\theta(d)$ $+\theta(\frac{v}{2}) + \theta(u+1))$
F	$9v^2d + 9uvd +$ $4v^2 + 31vd + 4v$ $+\frac{3}{2}u\log u + 7u$	$4\theta(v+1) + 4 + 2d\log(u)$ $+18d + 4d(\theta(d+1) +$ $\theta(\frac{v}{2}+1) + \theta(u+2))$
F $d = 2^r$	$9v^2d + 9uvd +$ $+31vd + 2v$ $+\frac{3}{2}u\log u + 7u$	$4\theta(v+1) + 4 + 2d\log(u)$ $+15d + 4d(\theta(d+1)$ $+\theta(u+2))$

Table 5. MSMP performance analysis.

5 Conclusions

We proposed new techniques for performing modular multiplication and exponentiation. Our initial motivation was to obtain modular arithmetic algorithms working completely in the spectrum in order to fully utilize the convolution property. For carrying out modular arithmetic operations, one has to deal with the computations of modular reduction. After defining spectral reduction and related concepts, we introduced a spectral reduction algorithm using the linearity and shifting property of DFT, and therefore, spectral modular multiplication and spectral modular exponentiation algorithms are obtained quite naturally. To avoid the round-off errors of the complex transforms, we employ finite ring spectrum in our method.

We carefully analyzed and stated the conditions for which our algorithms work without overflows. Working in the spectrum, we can exploit a vast amount of parallelism in our computations. Therefore, our algorithms yield highly parallel hardware architectures which we described.

We are currently working on creating VHDL implementations of the spectral modular exponentiation; such implementation will yield a more detailed analysis of our method and its advantages and disadvantages over algorithms and architectures which do not employ spectral techniques.

Acknowledgments

The authors would like to thank to Thomas Schmidt for his valuable comments and suggestions.

References

[1] H. B. A.V. Curiger and H. Kaeslin. Regular VLSI architectures for multiplication modulo (2^n+1). *IEEE J. Solid State Circuits*, 26(7):990–994, July 1991.

[2] R. E. Blahut. *Fast Algorithms for Digital Signal Processing*. Addison-Wesley publishing Company, 1985.

[3] N. Koblitz. *A Course in Number Theory and Cryptography*. Springer, Berlin, Germany, Second edition, 1994.

[4] Ç. K. Koç. High-Speed RSA Implementation. Technical Report TR 201, RSA Laboratories, 73 pages, November 1994.

[5] P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, Apr. 1985.

[6] H. J. Naussbaumer. *Fast Fourier Transform and Convolution Algorithms*. Springer, Berlin, Germany, 1982.

[7] J. M. Pollard. Implementation of number theoretic transform. *Electronics Letters*, 12(15):378–379, July 1976.

[8] G. Saldamlı. *Spectral Modular Arithmetic*. PhD thesis, Department of Electrical and Computer Engineering, Oregon State University, May 2005.

[9] A. Schönhage and V. Strassen. Schnelle multiplikation grosser zahlen. *Computing*, 7:281–292, 1971.

[10] A. Skavantzos and P. B. Rao. New multipliers modulo $2^n - 1$. *IEEE Transactions on Computers*, 41(8):957–961, Aug. 1992.

[11] G. A. J. Z. Wang and W. C. Miller. An efficient tree architecture for modulo $2^n + 1$ multiplication. *J. VLSI Signal Processing Systems*, 14(3):241–248, Dec. 1996.

[12] R. Zimmermann. Efficient VLSI implementation of modulo $(2^n \pm 1)$ addition and multiplication. In *Proceedings of the 14th IEEE Symposium on Computer Architecture*, pages 158–167, 1999.