Efficient Modular Operation for A Class of Moduli

* Huapeng Wu, † M. Anwar Hasan, ‡ Ian F. Blake

November 14, 2001

Abstract

In this paper, we propose an efficient method to compute modular operation $X \pmod N$, where $X < N^2$. When the modulus N is in the form of $2^n + A$, where $-2^{n/2} \le A \le 2^{n/2}$, formulas which can significantly simplify modular operation are obtained. We show that the complexity in calculating modular operation depends on the Hamming weight in the non-adjacent form (NAF) of the modulus. When a modulus is a sum of three positive or negative powers of 2, the modular operation is equivalent to performing not more than six addition operations with the operands of about the same size as or less than the modulus.

Key Words:

Modular arithmetic, integer ring, elliptic curve cryptosystem, RSA cryptosystem.

1. INTRODUCTION

Modular reduction is essential in computation over an integer ring $\mathbb{Z}_n$. Consequently it has important applications in many public key cryptographic systems, such as RSA and elliptic curve system defined over GF(q), where q is a prime power and not divisible by 2. This operation can also be found in extensive use in digital signal processing, communication and computer engineering.

It is well known that integers of certain forms are more suited for modular reduction than others. The best examples are the Mersenne numbers and the Fermat numbers [3]. Families of the generalized Mersenne

*H. Wu is with the Centre for Applied Cryptographic Research, University of Waterloo, Waterloo, Canada. E-mail:h3wu@cacr.math.uwaterloo.ca .

†M. A. Hasan is with the Department of Electrical and Computer Engineering, University of Waterloo. E-mail: ahasan@ece.uwaterloo.ca .

‡I. F. Blake is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada. E-mail: ifblake@comm.utoronto.ca .

numbers and generalized Fermat numbers have also been discussed in many publications, for example, [4, 2, 1]. However, the methods they used are mostly *ad hoc*. In [5], the author proposed a method to efficiently compute modular operation which works for a number of families of primes.

In this paper, we propose an efficient method to compute modular operation $X \pmod{N}$, where $X < N^2$. When the modulus N is in the form of $2^n + A$, where $-2^{n/2} \leq A \leq 2^{n/2}$, formulas which can significantly simplify modular operation are obtained. We show it is particularly efficient when the non-adjacent form (NAF) of the modulus has a small number of nonzeros. For example, when the modulus is a sum of three positive or negative powers of 2, the modular operation is equivalent to performing not more than six addition operations with the operands of about the same size as or less than the modulus. The algorithm is an extended and improved version of the one proposed in [6].

We organize the paper as follows. In Section 2, the main result is presented in a theorem. A proof to this theorem is given in Sections 3 and 4. Finally, concluding remarks are made in Section 5.

2. Main Results

Before presenting a new method to obtain $X \pmod{N}$, where $X < N^2$ and X satisfies certain conditions, we first define certain notations.

1. Let a signed-digit binary representation of the modulus N be given by:[1]

$$N = 2^n + s'_{e_{w-2}} 2^{e_{w-2}} + s'_{e_{w-3}} 2^{e_{w-3}} + \cdots + s'_{e_1} 2^{e_1} + s'_{e_0} 2^{e_0}, \tag{1}$$

where $n > e_{w-2} > e_{w-3} > \cdots > e_1 > e_0 = 0$ and $s'_{e_i} \in \{-1, 1\}$, $i = 0, 1, \ldots, w-2$. Clearly the Hamming weight of the signed-digit representation of N is w. We write expression (1) as follows

$$2^n \bmod N \equiv s_{e_{w-2}} 2^{e_{w-2}} + s_{e_{w-3}} 2^{e_{w-3}} + \cdots + s_{e_0} 2^{e_0}, \tag{2}$$

where $s_j = -s'_j$ for $j = e_i$ and $i = 0, 1, \ldots, w-2$.

[1] If N is given in the NAF, the most efficiency using this method can be achieved.

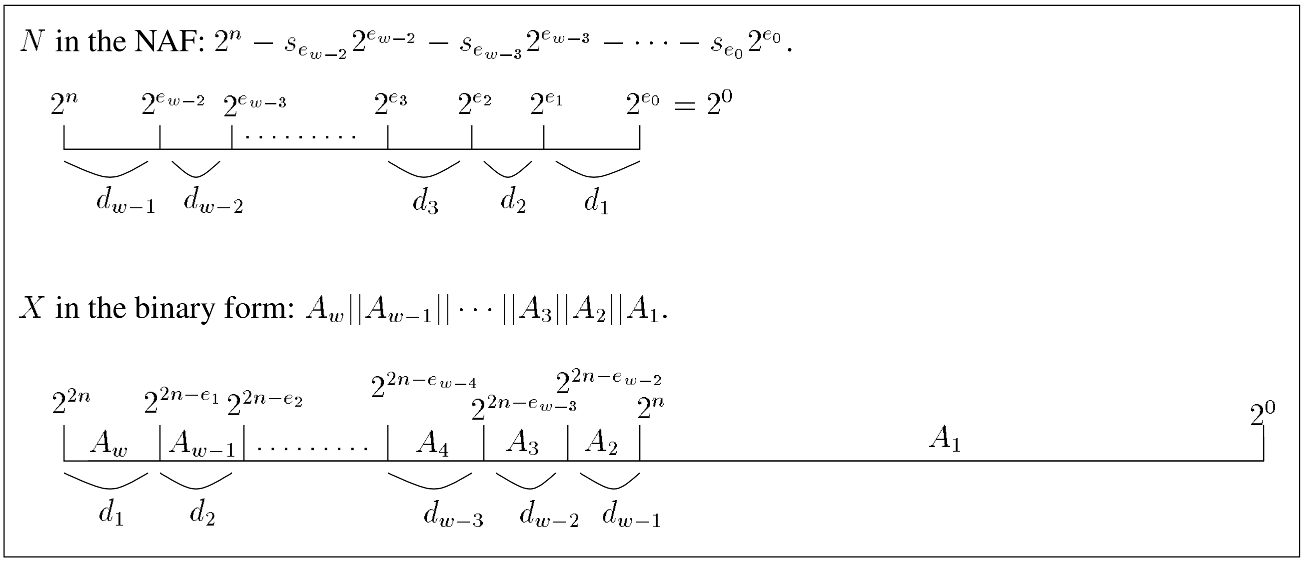

Figure 1: Diagram for partitioning N in the NAF and A in the binary form.

2. Let $d_i = e_i - e_{i-1}$ for $i = 1, 2, \ldots, w-1$, where $e_{w-1} = n$. Then the binary form of X can be partitioned and given by

$$\begin{aligned} X &= A_w \mid\mid A_{w-1} \mid\mid \cdots \mid\mid A_4 \mid\mid A_3 \mid\mid A_2 \mid\mid A_1 \\ &= (((\cdots(A_w \times 2^{d_2} + A_{w-1}) \times 2^{d_3} + \cdots A_4) \times 2^{d_{w-2}} + A_3) \times 2^{d_{w-1}} + A_2) \times 2^n + A_1, \end{aligned} \tag{3}$$

where the symbol "$||$" denotes a cascade operation. A diagram showing how X is partitioned according to the distribution of the nonzeros in N is given in Fig. 1.

Then a new method to obtain the modular operation $X \pmod N$ is described in the following theorem.

Theorem 1 *Let N be given in (1) or (2) and X be given in (3), respectively. Let $B_i = A_w \mid\mid A_{w-1} \mid\mid \cdots \mid\mid A_{i+1}$ for $i = 1, 2, \ldots, w-1$. If $e_{w-2} < \frac{n+1}{2}$, then the modular operation $X \pmod N$ can be solved with the following congruence.*

$$X \pmod N \equiv A_1 + \sum_{i=0}^{w-2} s_{e_i} 2^{e_i}[(A_{w-i} \mid\mid A_{w-(i+1)} \mid\mid \cdots \mid\mid A_2) + \sum_{i=1}^{w-2} s_{e_i} B_{w-i}]. \tag{4}$$

The proof of the theorem will be provided in the later sections.

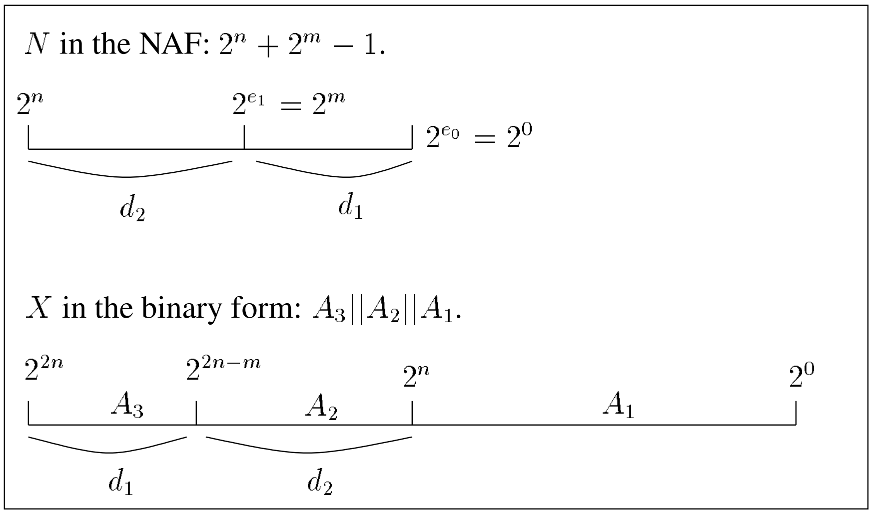

Figure 2: Diagram for partitioning N in the NAF and A in the binary form.

In the following we give an example to show how this result works for efficient computation of the modular operation. Let $N = 2^n + 2^m - 1, m < \frac{n+1}{2}$ and a partition of X is given by

$$\begin{aligned} X &= A_3 \mid\mid A_2 \mid\mid A_1 \\ &= (A_3 \times 2^{n-m} + A_2) \times 2^n + A_1, \end{aligned}$$

see Figure 2. Then we have $B_1 = A_3 \mid\mid A_2$ and $B_2 = A_3$. Since $w = 3$, $s_{e_1} = s_m = -1$ and $s_{e_0} = 1$, it follows from Theorem 1,

$$\begin{aligned} X \pmod N &\equiv A_1 + \sum_{i=0}^{1} s_{e_i} 2^{e_i} [(A_{w-i} \mid\mid A_{w-(i+1)} \mid\mid \cdots \mid\mid A_2) + \sum_{i=1}^{1} s_{e_i} B_{w-i}] \\ &\equiv A_1 + (A_3 \mid\mid A_2) - 2^m A_2 + (2^m - 1) B_2 \\ &\equiv A_1 + (A_3 \mid\mid A_2) - 2^m A_2 + (2^m - 1) A_3 \\ &\equiv A_1 + (A_3 \mid\mid A_2) - 2^m (A_2 - A_3) - A_3. \end{aligned} \tag{5}$$

We may also estimate the complexity of solving the modular operation by using (5). Obviously, four addition (subtraction) operations are needed to evaluate the expression on the right-hand side of (5). In the following we show that two more addition (subtraction) operations may be needed to keep the value of the sum of the four addition (subtraction) operations in the range $[0, N - 1]$.

The first term on the right-hand side of (5) $A_1 < 2^n < N$. The second term $(A_3 \mid\mid A_2)$ is of binary length $n+1$, which is the same as that of N. Since

$$N^2 = (2^n + 2^m - 1)^2 = 2^{2n} + 2^{n+m+1} + 2^{2m} - 2^{n+1} - 2^{m+1} + 1,$$

from $X = (A_3||A_2||A_1) < N^2$ and note $m < \frac{n+1}{2}$, we have $(A_3||A_2) < 2^n + 2^{m+1} - 1$. It follows $A_3 \le 2^m + 1$ and $0 \le A_2 < 2^{n-m}$. Then

$$\begin{aligned} X \pmod N &\equiv A_1 + (A_3 \mid\mid A_2) + (2^m - 1)A_3 - 2^m A_2 \\ &< 2^n + 2^n + 2^{m+1} - 1 + (2^m - 1)(2^m + 1) - 0 \\ &= 2^{n+1} + 2^{2m} + 2^{m+1} - 2 \\ &< 3 \cdot 2^n < 3N. \end{aligned} \tag{6}$$

On the other hand, write (5) as

$$\begin{aligned} X \pmod N &\equiv A_1 + (A_3 \mid\mid A_2) + (2^m - 1)A_3 - 2^m A_2 \\ &\ge (1 - 2^m)A_2 > -2^n > -N. \end{aligned} \tag{7}$$

It can be seen from (6) and (7) that two more addition (subtraction) operations may be needed to keep the value of the sum of the four addition (subtraction) operations in the range $[0, N-1]$. Thus, we can see from this example that the modular operation can be done by performing about six addition (subtraction) operations with operands of about the same size as or smaller than the modulus N. Similar arguments can be applied to the other three cases that the modulus N can be represented as a sum of three positive or negative powers of 2.

3. Partial Solution to the Modular Operation

A proof of Theorem 1 is given in three steps: Firstly we present an algorithm which can partially solve the modular operation. Then a proof of the correctness of the algorithm is given. Finally, we obtain the expression (4) in Theorem 1 by analyzing the output of the proposed algorithm. In this section we describe the first two steps and the final step is proposed in the next section.

3.1. A new algorithm for partial evaluation of modular operation

Consider the modular operation $Y \equiv X \bmod N$, where the NAF of N is in (1) or (2). Let X be an $n+t$-bit integer. The basic idea of the algorithm is as follows. For ith bit of $X = a_{n-1}a_{n-2}\ldots a_0$, $i = 0, 1, \ldots, n+t-1$, we introduce a *weight list* l_i. First, the weight lists are initialized as $l_i = \langle a_i \rangle$, where a_i is the ith bit of X.

From (2), it follows

$$2^{n+j} \bmod N \equiv s_{e_{w-2}} 2^{e_{w-2}+j} + s_{e_{w-3}} 2^{e_{w-3}+j} + \cdots + s_{e_0} 2^{e_0+j},\ j = 0, 1, \ldots, t-1. \tag{8}$$

Then the weight lists $l_i,\ i = 0, 1, \ldots, n+t-1$, are updated based on (8) in a manner that the final form of the weight lists has the following properties:

1. Each l_i contains a_i and several other a_j's or their negatives for $i = 0, 1, \ldots, n+t-1$.
2. Let S_{l_i} be the sum of all the numbers in the finally updated l_i. Then the value of $X \bmod N$ is decided only by $S_{l_0}, S_{l_2}, \ldots, S_{l_{n-1}}$. In fact, an incomplete solution for the modulation is

$$X \bmod N \equiv \sum S_{l_i} 2^i.$$

The process of updating a weight list depends on the distribution of nonzero bits in $2^n \bmod N$. When the jth bit of $2^{n+i} \bmod N$ is not a zero, we append the number a_{n+i} to the weight list l_j. When we finish the updating process with all the nonzero bits in $2^{n+i} \bmod N$, and for $i = t-1, t-2, \ldots, 1, 0$, the weight lists are referred to as the *prepared weight lists*.

The updating of weight lists can be considered as the *precomputation* part of the algorithm, where the value of X is not required. In the second part of the algorithm (Main Program), with the prepared weight lists and value of X available, we can obtain a partial solution to $X \pmod N$ by summing up the terms in the weight list l_i sequentially from $i = n+t-1$ to 0 and viewing the sum as the binary coefficient of B at weight position 2^i. The algorithm is given below:

Algorithm 1 Partial Evaluation of Modular Reduction
Input: X and N, both in the binary form.
Output: A partial solution: $Y \equiv X \bmod N$.

Part 1. Precomputation

Input: N (binary length n) and the upper bound $n+t$ of the binary length of X.

Output: Prepared weight lists $l_0, l_1, \ldots, l_{n+t-1}$.

1. Initialization of weight lists:

   $$2^j : \; l_j = \langle a_j \rangle, \; j = 0, 1, \ldots, n+t-1.$$

2. Obtain the minimal weight SD representation of $2^n \bmod N$:

   $$2^n \bmod N = s_{e_{w-2}} 2^{e_{w-2}} + s_{e_{w-3}} 2^{e_{w-3}} + \cdots + s_{e_0} 2^{e_0},$$

   where $n > e_{w-2} > e_{w-3} > \cdots > e_0 \geqslant 0$ and $s_{e_i} \in \{-1, 1\}$.

3. Compute the prepared weight lists:

   For $i = t-1$ To 0, Step -1

   For $j = 0$ To $w-2$

   Append $s_{e_j} a_{n+i}$ as one element to the weight list l_{i+e_j} .

Part 2. Main Program

Input: The binary weights of X, and the prepared weight lists $l_0, l_1, \ldots, l_{n+t-1}$.

Output: Partial solution $Y \equiv X \pmod{N}$.

1. For $i = n+t-1$ To 0, Step -1

   $a_i \Leftarrow$ the sum of all the elements in l_i;

2. $Y \Leftarrow a_0$; $d \Leftarrow 1$;

   For $i = 1$ To $n-1$,

   $d \Leftarrow d \times 2$;

   $Y \Leftarrow Y + d \times a_i$;

3.2. A proof for the algorithm

A proof of the correctness of Algorithm 1 is given as follows.

Proof: Let

$$X = \sum_{i=0}^{n+t-1} a_i 2^i.$$

To partially reduce $X \bmod N$ and note that N is of binary length n, we have

$$\begin{aligned}
\sum_{i=0}^{n-1} a_i 2^i &= a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + \cdots + a_1 2 + a_0 \\
a_n 2^n \bmod N &\equiv a_n s_{e_{w-2}} 2^{e_{w-2}} + a_n s_{e_{w-3}} 2^{e_{w-3}} + \cdots + a_n s_{e_0} 2^{e_0} \\
a_{n+1} 2^{n+1} \bmod N &\equiv a_{n+1} s_{e_{w-2}} 2^{e_{w-2}+1} + a_{n+1} s_{e_{w-3}} 2^{e_{w-3}+1} + \cdots + a_{n+1} s_{e_0} 2^{e_0+1} \\
a_{n+2} 2^{n+2} \bmod N &\equiv a_{n+2} s_{e_{w-2}} 2^{e_{w-2}+2} + a_{n+2} s_{e_{w-3}} 2^{e_{w-3}+2} + \cdots + a_{n+2} s_{e_0} 2^{e_0+2} \\
\vdots \quad &\vdots \quad \vdots \\
a_{n+t-2} 2^{n+t-2} \bmod N &\equiv a_{n+t-2} s_{e_{w-2}} 2^{e_{w-2}+t-2} + a_{n+t-2} s_{e_{w-3}} 2^{e_{w-3}+t-2} + \cdots + a_{n+t-2} s_{e_0} 2^{e_0+t-2} \\
a_{n+t-1} 2^{n+t-1} \bmod N &\equiv a_{n+t-1} s_{e_{w-2}} 2^{e_{w-2}+t-1} + a_{n+t-1} s_{e_{w-3}} 2^{e_{w-3}+t-1} + \cdots + a_{n+t-1} s_{e_0} 2^{e_0+t-1}
\end{aligned} \tag{9}$$

It can be seen that there are totally $tw + n$ terms on the right-hand side of the above $t + 1$ expressions (9). In order to evaluate $X \bmod N$, one way we can need do is to perform modular reduction on each of the terms that have weight equal to or higher than 2^n. When all the terms' weights have been reduced to 2^{n-1} or less, then we sum them up. Let us rewrite these the $tw + n$ terms in the following array of $t + 1$ rows with the first column being the indices. Clearly the sum of the $tw + n$ terms is congruent to X modulo N. Now let us start with the term of the highest weight.

$$\begin{array}{rllll}
(-1): & a_{n-1}2^{n-1} & a_{n-2}2^{n-2} & \cdots & a_0 \\
(0): & a_n s_{e_{w-2}} 2^{e_{w-2}} & a_n s_{e_{w-3}} 2^{e_{w-3}} & \cdots & a_n s_{e_0} 2^{e_0} \\
(1): & a_{n+1} s_{e_{w-2}} 2^{e_{w-2}+1} & a_{n+1} s_{e_{w-3}} 2^{e_{w-3}+1} & \cdots & a_{n+1} s_{e_0} 2^{e_0+1} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
(i): & a_{n+i} s_{e_{w-2}} 2^{e_{w-2}+i} & a_{n+i} s_{e_{w-3}} 2^{e_{w-3}+i} & \cdots & a_{n+i} s_{e_0} 2^{e_0+i} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
(t-2): & a_{n+t-2} s_{e_{w-2}} 2^{e_{w-2}+t-2} & a_{n+t-2} s_{e_{w-3}} 2^{e_{w-3}+t-2} & \cdots & a_{n+t-2} s_{e_0} 2^{e_0+t-2} \\
(t-1): & a_{n+t-1} s_{e_{w-2}} 2^{e_{w-2}+t-1} & a_{n+t-1} s_{e_{w-3}} 2^{e_{w-3}+t-1} & \cdots & a_{n+t-1} 2^{e_0} 2^{e_0+t-1}
\end{array} \tag{10}$$

Obviously, $a_{n+t-1} s_{e_{w-2}} 2^{e_{w-2}+t-1}$ is the term of the highest weight among the above $tw+n$ terms. From Step 3, Part 1 of Algorithm 1, it can be seen that the prepared weight lists $l_{e_{w-2}+t-1} = \langle a_{e_{w-2}+t-1}, s_{e_{w-2}} a_{n+t-1} \rangle$, and $l_{e_{w-2}+t+i} = \langle a_{e_{w-2}+t+i} \rangle$, $i \geqslant 0$. Subsequently, in Step 1 of Main Program, the first addition operation is performed on

$$a_{e_{w-2}+t-1} \Leftarrow a_{e_{w-2}+t-1} + s_{e_{w-2}} a_{n+t-1}. \tag{11}$$

For the purpose of avoiding ambiguousness, we put a prime on the updated coefficient $a_{e_{w-2}+t-1}$ and rewrite (11) as follows

$$a'_{e_{w-2}+t-1} \Leftarrow a_{e_{w-2}+t-1} + s_{e_{w-2}} a_{n+t-1}. \tag{12}$$

This step is equivalent to removing the term $a_{n+t-1}2^{e_{w-2}+t-1}$ from the array (10), and updating all the terms with coefficient $a_{e_{w-2}+t-1}$. That is, to change the $(e_{w-2}+t-n-1)^{\text{st}}$ row in the array into

$$a'_{e_{w-2}+t-1}2^{2e_{w-2}-n+t-1} + a'_{e_{w-2}+t-1}2^{e_{w-2}+e_{w-3}-n+t-1} + \cdots +$$

$$a'_{e_{w-2}+t-1}2^{e_{w-2}+e_1-n+t-1} + a'_{e_{w-2}+t-1}2^{e_{w-2}+e_0-n+t-1},$$

where $a'_{e_{w-2}+t-1} = a_{e_{w-2}+t-1} + s_{e_{w-2}}a_{n+t-1}$. Thus, it can be seen that the highest weight in the array is now reduced to $e_{w-2}+t-2$. Again, in Step 3, Part 1, it can be seen that the prepared weight list

$$l_{e_{w-2}+t-2} = \begin{cases} \langle a_{e_{w-2}+t-2}, s_{e_{w-2}}a_{n+t-2}\rangle & \text{if } e_{w-2} > e_{w-3}+1, \\ \langle a_{e_{w-2}+t-2}, s_{e_{w-2}}a_{n+t-2}, s_{e_{w-3}}a_{n+t-1}\rangle & \text{if } e_{w-2} = e_{w-3}+1, \end{cases}$$

We first discuss the case of $e_{w-2} > e_{w-3}+1$.

1. If $e_{w-2} > e_{w-3}+1$, then, in Step 1, Part 2, it performs

   $$a'_{e_{w-2}+t-2} \Leftarrow a_{e_{w-2}+t-2} + s_{e_{w-2}}a_{n+t-2}.$$

   It is equivalent to removing $a_{n+t-2}2^{e_{w-2}+t-2}$ from the array (10) and updating all the terms with coefficient $a_{e_{w-2}+t-2}$ by changing the coefficient into $a'_{e_{w-2}+t-2} = a_{e_{w-2}+t-2} + a_{n+t-2}$. When this is done, the highest weight in the array (10) is reduced to $e_{w-2}+t-3$.

2. If $e_{w-2} = e_{w-3}+1$, in Step 1, Part 2, it performs

   $$a'_{e_{w-2}+t-2} \Leftarrow a_{e_{w-2}+t-2} + s_{e_{w-2}}a_{n+t-2} + s_{e_{w-3}}a_{n+t-1}.$$

   It is equivalent to removing two terms $a_{n+t-2}s_{e_{w-2}}2^{e_{w-2}+t-2}$ and $a_{n+t-1}s_{e_{w-3}}2^{e_{w-3}+t-1}$ from the array (10), and updating all the terms with coefficient $a_{e_{w-2}+t-2}$ by changing the coefficient into $a'_{e_{w-2}+t-2} = a_{e_{w-2}+t-2} + s_{e_{w-2}}a_{n+t-2} + s_{e_{w-3}}a_{n+t-1}$. When it is done the highest weight in the array (10) is $e_{w-2}+t-3$.

Clearly, this removing and updating process does not alter the fact that the sum of all the terms in the updating array is congruent to Y modulo N. Then above removing and updating process continues until all the terms of weight higher than $n-1$ have been removed from the array (10). Let us divide Step 1, Part 2, into two sub-steps as follows:

1. For $\; i = n+t-1 \;$ To $\; n, \;$ Step $\; -1$

$a_i \Leftarrow$ the sum of all the elements in l_i;

2. For $i = n-1$ To 0, Step -1

   $a_i \Leftarrow$ the sum of all the elements in l_i.

Then it can be seen that the first sub-step corresponds to the removing job, while the second sub-step is responsible for updating the terms of weight lower than 2^n and summing them up to yield the output Y. Therefore, we have Y is congruent to X modulo N, or $Y \equiv X \mod N$.

Since there may be multiple terms with value of either 1 or -1 at the weight position 2^i, the finally updated coefficient a_i can be out of the proper range of $\{0, 1\}$, where $i = n-1, n-1, \ldots, 0$. Consequently, the output Y can exceed the modulus N or have a negative value. $\square$

4. Analyzing the Output of the Proposed Algorithm

Before investigating the general case, we first analyze the case of $w = 3$.

4.1. Family of moduli of form $N = 2^n \pm 2^m \pm 1$

Let N be given as in (2) for $w = 3$ and $e_1 = m$, and X be given in (3). Then we proceed with Algorithm 1 when it takes inputs of N and X.

Part 1. Precomputation: preparing the weight lists.

1. Initialize the weight lists:

$$2^i: \quad l_i = \langle a_i \rangle, \quad i = 0, 1, \ldots, 2n-1.$$

2. Obtain the binary expansion of $2^n \mod N$:

$$2^n \mod N = s_m 2^m + s_0,\ s_0, s_m \in \{-1, 1\}.$$

3. Compute the prepared weight lists:

   It is easy to see that the prepared weight lists are as follows:

$$
\begin{array}{lclll}
l_{2n-1} & = & \langle a_{2n-1} \rangle & & \\
\vdots & \vdots & \vdots & & \\
l_{n+m} & = & \langle a_{n+m} \rangle & & \\
l_{n+m-1} & = & \langle a_{n+m-1}, & s_m a_{2n-1} \rangle & \\
l_{n+m-2} & = & \langle a_{n+m-2}, & s_m a_{2n-2} \rangle & \\
\vdots & \vdots & \vdots & \vdots & \\
l_n & = & \langle a_n, & s_m a_{2n-m} \rangle & \\
l_{n-1} & = & \langle a_{n-1}, & s_m a_{2n-m-1}, & s_0 a_{2n-1} \rangle \\
l_{n-2} & = & \langle a_{n-2}, & s_m a_{2n-m-2}, & s_0 a_{2n-2} \rangle \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
l_m & = & \langle a_m, & s_m a_n, & s_0 a_{n+m} \rangle \\
l_{m-1} & = & \langle a_{m-1}, & & s_0 a_{n+m-1} \rangle \\
\vdots & \vdots & \vdots & & \vdots \\
l_0 & = & \langle a_0, & & s_0 a_n \rangle
\end{array}
$$

Part 2. Main Program

We can divide the first step of Main Program into three sub-steps as follows:

1. For $i = n+m-1$ To n, Step -1,

   $a_i \Leftarrow a_i + s_m a_{n-m+i}$;

2. For $i = n-1$ To m, Step -1,

   $a_i \Leftarrow a_i + s_m a_{n-m+i}$;

3. For $i = n-1$ To 0, Step -1,

   $a_i \Leftarrow a_i + s_0 a_{n+i}$.

Let

$$
\begin{aligned}
X &= A_3 \,||\, A_2 \,||\, A_1 \\
&= (A_3 \times 2^{n-m} + A_2) \times 2^n + A_1,
\end{aligned}
$$

and $B_1 = A_3 \,||\, A_2$ and $B_2 = A_3$. Then it can be seen that Sub-step 1 performs $B_1 \Leftarrow B_1 + s_m A_3$ if $m < \frac{n+1}{2}$. This is because, otherwise, say that $m = \frac{n+1}{2}$, the value of a_{2n-m} would not be an original bit of X when performing $a_n \Leftarrow a_n + s_m a_{2n-m}$, since it has been changed when $a_{n+m-1} \Leftarrow a_{n+m-1} + s_m a_{2n-1}$ is performed. Similar argument applies to the case that $m > \frac{n+1}{2}$. Sub-step 2 performs $A_1 \Leftarrow A_1 + s_m 2^m (A_2 + s_m A_3)$ while Sub-step 3 finishes with $A_1 \Leftarrow A_1 + s_0 (B_1 + s_m A_3)$. Note the order that each sub-step is performed. Then we have

$$\begin{aligned}
X \pmod N &\equiv A_1 + s_m 2^m A_2 + 2^m A_3 + s_0 (A_2 + A_3 \times 2^{n-m}) + s_0 s_m A_3 \\
&\equiv A_1 + (s_m 2^m + s_0) A_2 + (2^m + s_0 2^{n-m} + s_0 s_m) A_3 \\
&\equiv A_1 + s_0 B_1 + s_m 2^m A_2 + (2^m + s_0 s_m) A_3.
\end{aligned}$$

We summarize the results in the following lemma.

Lemma 1 *Let N and X be given in (2) and (3), respectively. Let $w = 3$ and $e_1 = m < \frac{n+1}{2}$. Then the modular operation $X \pmod N$ can be partially solved as follows:*

1. *When $N = 2^n - 2^m - 1$,*

$$X \bmod N \equiv A_1 + (A_3 || A_2) + 2^m A_2 + (2^m + 1) A_3.$$

2. *When $N = 2^n - 2^m + 1$,*

$$X \bmod N \equiv A_1 - (A_3 || A_2) + 2^m A_2 + (2^m - 1) A_3.$$

3. *When $N = 2^n + 2^m - 1$,*

$$X \bmod N \equiv A_1 + (A_3 || A_2) - 2^m A_2 + (2^m - 1) A_3.$$

4. *When $N = 2^n + 2^m + 1$,*

$$X \bmod N \equiv A_1 - (A_3 || A_2) - 2^m A_2 + (2^m - 1) A_3.$$

4.2. Modulus of general form

Let the modulus N be given by a binary SD representation in (1) or (2). Let X be given in (3) and assume that $X^2 < N$, hence, $t \le n$. In the following we proceed with Algorithm 1 when it takes inputs of N and A given above.

Part 1. Precomputation: preparing the weight lists.

1. Initialize the weight lists:

$$2^i: \quad l_i = \langle a_i \rangle, \quad i = 0, 1, \ldots, 2n-1.$$

2. Obtain the binary expansion of $2^n \bmod N$:

$$2^n \bmod N = s_{e_{w-2}} 2^{e_{w-2}} + s_{e_{w-3}} 2^{e_{w-3}} + \cdots + s_{e_1} 2^{e_1} + s_{e_0} 2^{e_0}.$$

3. Compute the prepared weight lists:

   The prepared weight lists are given in the Appendix A.

Part 2. Main Program

We can divide the first step of Main Program into w sub-steps as follows:

(1). For $j = w-2$ To 0, Step -1,

For $i = n + e_j - 1$ To n, Step -1,

$a_i \Leftarrow a_i + s_{e_j} a_{n-e_j+i}$;

(2). For $i = n-1$ To e_{w-2}, Step -1,

$a_i \Leftarrow a_i + s_{e_{w-2}} a_{n-e_{w-2}+i}$;

(3). For $i = n-1$ To e_{w-3}, Step -1,

$a_i \Leftarrow a_i + s_{e_{w-3}} a_{n-e_{w-3}+i}$.

(k). $3 <$k$\leqslant w$

For $i = n-1$ To e_{w-k+1}, Step -1,

$a_i \Leftarrow a_i + s_{e_{w-k+1}} a_{n-e_{w-k+1}+i}$.

(w). For $i = n-1$ To 0, Step -1,

$a_i \Leftarrow a_i + s_0 a_{n+i}$.

We partition X as in (3): $X = A_w||A_{w-1}||\cdots||A_2||A_1$, and $B_i = A_w||A_{w-1}||\cdots||A_{i+1}$ for $i = 1, 2, \ldots, w-1$. Then the w sub-steps performed in Part 2 of Algorithm 1 are equivalent to performing

(1).

$$B_1 \Leftarrow B_1 + \Delta,$$

where $\Delta = \sum_{i=1}^{w-2} s_{e_i} B_{w-i}$.

(2).

$$A_1 \Leftarrow A_1 + s_{e_{w-2}} 2^{e_{w-2}} (A_2 + \Delta).$$

(3).

$$A_1 \Leftarrow A_1 + s_{e_{w-3}} 2^{e_{w-3}} (A_2 + 2^{n-e_{w-2}} A_3 + \Delta).$$

(k).

$$A_1 \Leftarrow A_1 + s_{e_{w-k}} 2^{e_{w-k}} (A_2 + 2^{n-e_{w-2}} A_3 + \cdots + 2^{n-e_{w-k+1}} A_k + \Delta).$$

($w-1$).

$$A_1 \Leftarrow A_1 + s_{e_1} 2^{e_1} (A_2 + 2^{n-e_{w-2}} A_3 + \cdots + 2^{n-e_2} A_{w-1} + \Delta).$$

(w).

$$A_1 \Leftarrow A_1 + s_{e_0} (B_1 + \Delta).$$

Then it follows

$$\begin{aligned} X \pmod N \equiv\ & A_1 + s_{e_{w-2}} 2^{e_{w-2}} (A_2 + \Delta) + s_{e_{w-3}} 2^{e_{w-3}} [(A_3||A_2) + \Delta] + \cdots \\ & + s_{e_1} 2^{e_1} [(A_{w-1}||A_{w-2}||\cdots||A_2) + \Delta] + s_{e_0} (B_1 + \Delta) \\ =\ & A_1 + \sum_{i=0}^{w-2} s_{e_i} 2^{e_i} [(A_{w-i}||A_{w-(i+1)}||\cdots||A_2) + \Delta] \\ =\ & A_1 + \sum_{i=0}^{w-2} s_{e_i} 2^{e_i} [(A_{w-i}||A_{w-(i+1)}||\cdots||A_2) + \sum_{i=1}^{w-2} s_{e_i} (A_w||A_{w-1}||\cdots||A_{w-i+1})] \end{aligned} \tag{13}$$

Clearly, Theorem 1 follows from the above expression.

5. Concluding Remarks

In this paper we have proposed a method to compute modular operation for a large family of the moduli. The complexities of the proposed method depends on the Hamming weight of the NAF of the modulus N. When N has a Hamming weight of 3, the modular operation is shown to be equivalent to not more than six addition operations with operands of the same size as or smaller size than that of N. It is expected that the results can be used for efficient computation of modular reduction and consequently speed up the computation of many cryptosystems.

References

[1] R. E. Crandall. Method and apparatus for public key exchange in a cryptographic system. U.S. Patent No.5159632, 1992.

[2] S. W. Golomb. Properties of the sequence $3 \cdot 2^n + 1$. *Math. Comp.*, 30(135):657–663, 1976.

[3] D. E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*. Addison-Wesley Publishing Company, Reading, MA, 1981.

[4] J. M. Pollard. Implementation of numer-theoretic transforms. *Electronic Letters*, 12(15):378–379, 1976.

[5] J. A. Solinas. Generalized Mersenne numbers. Technical report, The centre for applied cryptographic research, University of Waterloo, 1999. CORR 99-39, online at http://www.cacr.uwaterloo.ca/.

[6] H. Wu. On modular reduction. Technical report, The centre for applied cryptographic research, University of Waterloo, 2000. CORR 2000-31, online at http://www.cacr.uwaterloo.ca/.

A The Weight List for N of The General Form.

$$
\begin{array}{lllllllll}
l_{2n-1} & = & \langle a_{2n-1} \rangle & & & & & & \\
\vdots & \vdots & \vdots & & & & & & \\
l_{n+e_{w-2}} & = & \langle a_{n+e_{w-2}} \rangle & & & & & & \\
l_{n+e_{w-2}-1} & = & \langle a_{n+e_{w-2}-1}, & s_{e_{w-2}} a_{2n-1} \rangle & & & & & \\
l_{n+e_{w-2}-2} & = & \langle a_{n+e_{w-2}-2}, & s_{e_{w-2}} a_{2n-2} \rangle & & & & & \\
\vdots & \vdots & \vdots & \vdots & & & & & \\
l_{n+e_{w-3}} & = & \langle a_{n+e_{w-3}}, & s_{e_{w-2}} a_{2n-e_{w-2}+e_{w-3}} \rangle & & & & & \\
l_{n+e_{w-3}-1} & = & \langle a_{n+e_{w-3}-1}, & s_{e_{w-2}} a_{2n-e_{w-2}+e_{w-3}-1} & s_{e_{w-3}} a_{2n-1} \rangle & & & & \\
\vdots & \vdots & \vdots & \vdots & \vdots & & & & \\
l_{n+e_1} & = & \langle a_{n+e_1}, & s_{e_{w-2}} a_{2n-e_{w-2}+e_1}, & s_{e_{w-3}} a_{2n-e_{w-3}+e_1} & \cdots & s_{e_2} a_{2n-e_2+e_1} \rangle & & \\
l_{n+e_1-1} & = & \langle a_{n+e_1-1}, & s_{e_{w-2}} a_{2n-e_{w-2}+e_1-1}, & s_{e_{w-3}} a_{2n-e_{w-3}+e_1-1} & \cdots & s_{e_2} a_{2n-e_2+e_1-1} & s_{e_1} a_{2n-1} \rangle & \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\
l_n & = & \langle a_n, & s_{e_{w-2}} a_{2n-e_{w-2}} & s_{e_{w-3}} a_{2n-e_{w-3}} & \cdots & s_{e_2} a_{2n-e_2} & s_{e_1} a_{2n-e_1} \rangle & \\
l_{n-1} & = & \langle a_{n-1}, & s_{e_{w-2}} a_{2n-e_{w-2}-1} & s_{e_{w-3}} a_{2n-e_{w-3}-1} & \cdots & s_{e_2} a_{2n-e_2-1} & s_{e_1} a_{2n-e_1-1} & s_{e_0} a_{2n-1} \rangle \\
l_{n-2} & = & \langle a_{n-2}, & s_{e_{w-2}} a_{2n-e_{w-2}-2} & s_{e_{w-3}} a_{2n-e_{w-3}-2} & \cdots & s_{e_2} a_{2n-e_2-2} & s_{e_1} a_{2n-e_1-2} & s_{e_0} a_{2n-2} \rangle \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
l_{e_{w-2}} & = & \langle a_{e_{w-2}}, & s_{e_{w-2}} a_n & s_{e_{w-3}} a_{n-e_{w-3}+e_{w-2}}, & \cdots & s_{e_2} a_{n-e_2+e_{w-2}} & s_{e_1} a_{n-e_1+e_{w-2}} & s_{e_0} a_{n+e_{w-2}} \rangle \\
l_{e_{w-2}-1} & = & \langle a_{e_{w-2}-1}, & & s_{e_{w-3}} a_{n-e_{w-3}+e_{w-2}-1} & \cdots & s_{e_2} a_{n-e_2+e_{w-2}-1} & s_{e_1} a_{n-e_1+e_{w-2}-1} & s_{e_0} a_{n+e_{w-2}-1} \rangle \\
\vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\
l_1 & = & \langle a_1, & & & & & & s_{e_0} a_{n+1} \rangle \\
l_0 & = & \langle a_0, & & & & & & s_{e_0} a_n \rangle
\end{array}
$$

Table 1: Weigth lists when the modulus has a binary signed-digit representation $N = 2^n - \sum_{i=0}^{w-2} s_{e_i} 2^{e_i}$.