On Modular Reduction

Huapeng Wu*

June 11, 2000

Abstract

In this paper, an algorithm to partially evaluate modular reduction without division is proposed. A proof of the correctness of the algorithm is given. For a family of generalized Mersenne numbers $N = 2^n - 2^m - 1, 0 < m < \frac{n+1}{2}$, we show that the modular reduction operation $A \bmod N$, where $A < N^2$, can be reduced to

$$A \bmod N \equiv A_1 + A_2 + A_4 + 2^m(A_3 + A_4),$$

where $A \stackrel{\triangle}{=} A_1 + A_2 \times 2^n,\ 0 \leqslant A_1 \leqslant 2^n - 1$, and $A_2 \stackrel{\triangle}{=} A_3 + A_4 \times 2^{n-m},\ 0 \leqslant A_3 \leqslant 2^{n-m} - 1$. For another family of generalized Mersenne numbers $N = 2^n - 2^m - 2^{m_1} - 1, 0 < m_1 < m < \frac{n+1}{2}$, we find that the modular reduction operation $A \bmod N$, where $A < N^2$, can be partially solved as

$$\begin{aligned} A \bmod N \;\equiv\;& A_1 + A_2 + A_4 + A_6 + 2^m(A_3 + A_4 + A_6) + \\ & 2^{m_1}(A_3 + A_4 + A_6 + A_5 \times 2^{n-m}), \end{aligned}$$

where $A \stackrel{\triangle}{=} A_1 + A_2 \times 2^n,\ 0 \leqslant A_1 \leqslant 2^n - 1$, $A_2 \stackrel{\triangle}{=} A_3 + A_4 \times 2^{n-m},\ 0 \leqslant A_3 \leqslant 2^{n-m} - 1$, and $A_4 \stackrel{\triangle}{=} A_5 + A_6 \times 2^{m-m_1},\ 0 \leqslant A_5 \leqslant 2^{m-m_1} - 1$.

Key Word:

Modular arithmetic, public-key cryptosystems.

*H. Wu is with the Centre for Applied Cryptographic Research, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Canada N2L 3G1. E-mail:h3wu@cacr.math.uwaterloo.ca .

1. INTRODUCTION

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]. Other generalized Mersenne numbers and generalized Fermat numbers have also been discussed in many publications, for example, [4, 2, 1, 5].

In this paper, we propose an algorithm which can partially solve the modular reduction without division. As a result from this algorithm, we find that two families of generalized Mersenne numbers are suited for modular reduction. We note that this algorithm is similar to the one for polynomial modular reduction proposed by the same author in [6].

2. Algorithm for Modular Reduction without Division

Consider the modular operation $B \equiv A \bmod N$, where N is of binary length n and A is of binary length not greater than $n+t$. The basic idea of the algorithm can be as follows. For each bit position 2^i in the binary form of A, $i = 0, 1, \ldots, n+t-1$, we introduce a *weight list* l_i. First, the weight lists are initialized as $l_i = \langle a_i \rangle$, the bit at weight position 2^i in the binary expansion of A. Then those weight lists are updated based on the distribution of the Hamming weight in the binary expansion of $2^n \bmod N$. Finally, B can be obtained by performing certain operations on the updated weight lists.

The process of updating a weight list depends on the Hamming weight distribution of the binary expansion of $2^n \bmod N$. When there is a nonzero coefficient at weight 2^j in the binary expansion of $2^{n+i} \bmod N$, we append the term a_{n+i} to the weight list l_j. When we finish with all the terms in $2^{n+i} \bmod N, i = 0, 1, \ldots, t-1$, 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 value of A is not required. In the second part of the algorithm (Main Program), with the prepared weight lists and value of A available, we can obtain B 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 detailed algorithm is given below:

Algorithm 1 Partial Evaluation of Modular Reduction

Input: A and N, both in the binary form.

Output: $B \equiv A \bmod N$.

Part 1. Precomputation

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

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 binary expansion of $2^n \bmod N$:

$$2^n \bmod N \equiv 2^{e_{w-1}} + 2^{e_{w-2}} + \cdots + 2^{e_0},$$

where $n > e_{w-1} > e_{w-2} > \cdots > e_0 \geqslant 0$.

3. Compute the prepared weight lists:

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

For $j = 0$ To $w-1$

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

Part 2. Main Program

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

Output: B.

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

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

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

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

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

$d \Leftarrow d \times 2;$

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

Proof: Let

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

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

$$\begin{array}{rcl} \sum_{i=0}^{n-1} a_i 2^i & = & a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + \cdots + a_1 2 + a_0 \\ a_n 2^n \bmod N & \equiv & a_n 2^{e_{w-1}} + a_n 2^{e_{w-2}} + \cdots + a_n 2^{e_0} \\ a_{n+1} 2^{n+1} \bmod N & \equiv & a_{n+1} 2^{e_{w-1}+1} + a_{n+1} 2^{e_{w-2}+1} + \cdots + a_{n+1} 2^{e_0+1} \\ a_{n+2} 2^{n+2} \bmod N & \equiv & a_{n+2} 2^{e_{w-1}+2} + a_{n+2} 2^{e_{w-2}+2} + \cdots + a_{n+2} 2^{e_0+2} \\ \vdots & \vdots & \vdots \\ a_{n+t-2} 2^{n+t-2} \bmod N & \equiv & a_{n+t-2} 2^{e_{w-1}+t-2} + a_{n+t-2} 2^{e_{w-2}+t-2} + \cdots + a_{n+t-2} 2^{e_0+t-2} \\ a_{n+t-1} 2^{n+t-1} \bmod N & \equiv & a_{n+t-1} 2^{e_{w-1}+t-1} + a_{n+t-1} 2^{e_{w-2}+t-1} + \cdots + a_{n+t-1} 2^{e_0+t-1} \end{array} \quad (1)$$

It can be seen that there are totally $tw+n$ terms on the right-hand side of the above $t+1$ expressions (1). In order to evaluate $A \bmod N$, what we need to do is to perform modular reduction on each of the $tw+n$ terms, and then sum them up. Let us rewrite these terms in the following array of $t+1$ rows (with the first column being the indices):

$$\begin{array}{rllllll} (-1): & a_{n-1}2^{n-1} & a_{n-2}2^{n-2} & \cdots & a_1 2 & a_0 \\ (0): & a_n 2^{e_{w-1}} & a_n 2^{e_{w-2}} & \cdots & a_n 2^{e_1} & a_n 2^{e_0} \\ (1): & a_{n+1}2^{e_{w-1}+1} & a_{n+1}2^{e_{w-2}+1} & \cdots & a_{n+1}2^{e_1+1} & a_{n+1}2^{e_0+1} \\ \vdots\ & \vdots & \vdots & \vdots & \vdots & \vdots \\ (i): & a_{n+i}2^{e_{w-1}+i} & a_{n+i}2^{e_{w-2}+i} & \cdots & a_{n+i}2^{e_1+i} & a_{n+i}2^{e_0+i} \\ \vdots\ & \vdots & \vdots & \vdots & \vdots & \vdots \\ (t-2): & a_{n+t-2}2^{e_{w-1}+t-2} & a_{n+t-2}2^{e_{w-2}+t-2} & \cdots & a_{n+t-2}2^{e_1+t-2} & a_{n+t-2}2^{e_0+t-2} \\ (t-1): & a_{n+t-1}2^{e_{w-1}+t-1} & a_{n+t-1}2^{e_{w-2}+t-1} & \cdots & a_{n+t-1}2^{e_1+t-1} & a_{n+t-1}2^{e_0+t-1} \end{array} \quad (2)$$

Clearly, $a_{n+t-1}2^{e_{w-1}+t-1}$ is the term with 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 $l_{e_{w-1}+t-1} = \langle a_{e_{w-1}+t-1}, a_{n+t-1}\rangle$, and $l_{e_{w-1}+t+i} = \langle a_{e_{w-1}+t+i}\rangle$, $i \geqslant 0$. Subsequently, in Step 1 of Main Program, the first addition operation is performed on

$$a_{e_{w-1}+t-1} \Leftarrow a_{e_{w-1}+t-1} + a_{n+t-1}. \quad (3)$$

For the purpose of avoiding ambiguousness, we put a prime on the updated coefficient $a_{e_{w-1}+t-1}$ and rewrite (3) as follows

$$a'_{e_{w-1}+t-1} \Leftarrow a_{e_{w-1}+t-1} + a_{n+t-1}. \tag{4}$$

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

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

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

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

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

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

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

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

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

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

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

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

The above process continues until all the terms of weight higher than $n-1$ have been removed from the array (2). This is being done by the first sub-step of Step 1, Part 2, if we divide this step 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.

The second sub-step is responsible for summing up all the terms whose weight has been already reduced to $n-1$ or less. Since there may be multiple terms with weight of i, the finally updated coefficient a_i can be larger than one, where $i = n-1, n-1, \ldots, 0$. Subsequently, the sum (B) obtained in Step 2, Part 2 of Algorithm 1 can exceed the modulus N. $\square$

Note that after Step 1 of Main Program in Algorithm 1, the expression $(a_{n-1}a_{n-2}\ldots a_0)$ can be viewed as a *special* binary representation of B, since, as discussed in the last paragraph in the proof, the value of its digit a_i can be *larger than one*. Consequently, the output $B \equiv A \bmod N$ may not be in the proper range $[0, N-1]$. Hence, one more step may be needed to reduce B to the proper range. Nevertheless, Algorithm 1 can be used as a tool to exploit possibly simple relation between A and B when the modulus N is of some particular form.

As an example, we use Algorithm 1 to develop such a simple relation between A and B for the modulus being a Mersenne number. Let $N = 2^n - 1$, it can be seen that $2^n \bmod 2^n - 1 = 1$. And the prepared weight lists can be obtained as follows:

$$\begin{array}{rcl} l_{n+t-1} & = & \langle a_{n+t-1} \rangle \\ \vdots & \vdots & \vdots \\ l_t & = & \langle a_t \rangle \\ l_{t-1} & = & \langle a_{t-1}, a_{n+t-1} \rangle \\ l_{t-2} & = & \langle a_{t-2}, a_{n+t-2} \rangle \\ \vdots & \vdots & \vdots \\ l_0 & = & \langle a_0, a_n \rangle \end{array}$$

When $t \leqslant n$ as it is required in the common case that $A < N^2$, Step 1 of Main Program in Algorithm 1 computes

$$a_i \Leftarrow a_i + a_{n+i} \text{ for } i = 0, 1, \dots, n-1. \tag{5}$$

And the final value of a_i is the weight at 2^i of a binary expansion of B. If we let $A \triangleq A_1 + A_2 \times 2^n$, where $0 \leqslant A_1 \leqslant 2^n - 1$, then it can be seen from (5) that

$$A \bmod 2^n - 1 \equiv B = A_1 + A_2.$$

When $t > n$, we have

$$a_{n+i} \Leftarrow a_{n+i} + a_{2n+i} \text{ for } i = 0, 1, \dots, t-n-1; \tag{6a}$$

$$a_i \Leftarrow a_i + a_{n+i} \quad \text{ for } i = 0, 1, \dots, n-1. \tag{6b}$$

Then by defining that $A \triangleq A_1 + A_2 \times 2^n$ and $A_2 \triangleq A_3 + A_4 \times 2^n$, where $0 \leqslant A_1, A_3 \leqslant 2^n - 1$, it can be seen from (6a) and (6b) that

$$A \bmod 2^n - 1 \equiv A_1 + A_3 + A_4.$$

3. Classes of Generalized Mersenne Numbers Used as Moduli

Let the modulus be given by $N = 2^n - 2^m - 1, 0 < m < n$. Assume that $A^2 < N$, hence, $t = n$. 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 $2^n \bmod N$:

$$2^n \bmod N = 2^m + 1.$$

3. Compute the prepared weight lists:

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

$$\begin{array}{rcl}
l_{2n-1} & = & \langle a_{2n-1} \rangle \\
\vdots & \vdots & \vdots \\
l_{n+m} & = & \langle a_{n+m} \rangle \\
l_{n+m-1} & = & \langle a_{n+m-1}, a_{2n-1} \rangle \\
l_{n+m-2} & = & \langle a_{n+m-2}, a_{2n-2} \rangle \\
\vdots & \vdots & \vdots \\
l_{n} & = & \langle a_{n}, a_{2n-m} \rangle \\
l_{n-1} & = & \langle a_{n-1}, a_{2n-m-1}, a_{2n-1} \rangle \\
l_{n-2} & = & \langle a_{n-2}, a_{2n-m-2}, a_{2n-2} \rangle \\
\vdots & \vdots & \vdots \\
l_{m} & = & \langle a_{m}, a_{n}, a_{n+m} \rangle \\
l_{m-1} & = & \langle a_{m-1}, a_{n+m-1} \rangle \\
\vdots & \vdots & \vdots \\
l_{0} & = & \langle a_{0}, a_{n} \rangle
\end{array}$$

Part 2. Main Program

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

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

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

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

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

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

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

and let

$$A \triangleq A_1 + A_2 \times 2^n,\ 0 \leqslant A_1 \leqslant 2^n - 1, \tag{7}$$

and

$$A_2 \triangleq A_3 + A_4 \times 2^{n-m},\ 0 \leqslant A_3 \leqslant 2^{n-m} - 1. \tag{8}$$

Then it can be seen that Sub-step 1 performs $A_2 + A_4$ 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 A when performing $a_n \Leftarrow a_n + a_{2n-m}$, since it has been changed when $a_{n+m-1} \Leftarrow a_{n+m-1} + a_{2n-1}$ is performed, and similar argument applies to the case that $m > \frac{n+1}{2}$. It is also clear that Sub-steps 2 and 3 are quivalent to performing $A_1 + 2^m(A_3 + A_4) + (A_2 + A_4)$.

We summarize the above result in the following theorem:

Theorem 1 *For $N = 2^n - 2^m - 1$, $0 < m < \frac{n+1}{2}$, and $A < N^2$, the following expression holds:*

$$A \bmod N \equiv A_1 + A_2 + A_4 + 2^m(A_3 + A_4), \tag{9}$$

where $A \triangleq A_1 + A_2 \times 2^n,\ 0 \leqslant A_1 \leqslant 2^n - 1$, and $A_2 \triangleq A_3 + A_4 \times 2^{n-m},\ 0 \leqslant A_3 \leqslant 2^{n-m} - 1$.

The following property is easy to obtain.

Property 1 *Let $S = A_1 + A_2 + A_4 + 2^m(A_3 + A_4)$, where $A_i, i = 1, 2, 3, 4$, are given in Theorem 1. Then $S < 4N$, and an estimate of the average value of S is given by $\overline{S} < 2N$.*

For another family of generalized Mersenne numbers of form $N = 2^n - 2^m - 2^{m_1} - 1$, $0 < m_1 < m < \frac{n+1}{2}$, and $A < N^2$, by applying to it Algorithm 1 and the similar analysis as the above, we conclude

Theorem 2 *For $N = 2^n - 2^m - 2^{m_1} - 1,\ 0 < m_1 < m < \frac{n+1}{2}$, and $A < N^2$, we can partially solve the modular reduction of $A \bmod N$ as*

$$\begin{aligned} A \bmod N \ \equiv\ & A_1 + A_2 + A_4 + A_6 + 2^m(A_3 + A_4 + A_6) + \\ & 2^{m_1}(A_3 + A_4 + A_6 + A_5 \times 2^{n-m}), \end{aligned}$$

where $A \triangleq A_1 + A_2 \times 2^n,\ 0 \leqslant A_1 \leqslant 2^n - 1,\ A_2 \triangleq A_3 + A_4 \times 2^{n-m},\ 0 \leqslant A_3 \leqslant 2^{n-m} - 1$, and $A_4 \triangleq A_5 + A_6 \times 2^{m-m_1},\ 0 \leqslant A_5 \leqslant 2^{m-m_1} - 1$.

For the generalized Mersenne number of general form

$$N = 2^n - 2^{e_{w-1}} - 2^{e_{w-2}} - \cdots - 2^{e_1} - 2^{e_0}, \tag{10}$$

where $\frac{n+1}{2} > e_{w-1} > e_{w-2} > \cdots e_1 > e_0 = 0$, and $A < N^2$, we have

Theorem 3 *For N given by (10) and $A < N^2$, we can partially solve the modular reduction of $A \bmod N$ as*

$$A \bmod N \ \equiv\ A_1 + \sum_{i=1}^{w} A_{2i} + (A_3 + \sum_{i=2}^{w} A_{2i}) \sum_{i=1}^{w-1} 2^{e_i} + 2^{n-e_{w-1}} \sum_{i=2}^{w-1} \sum_{j=1}^{w-i} A_{2i+1} 2^{e_j},$$

where

$$A \triangleq A_1 + A_2 \times 2^n,\ 0 \leqslant A_1 \leqslant 2^n - 1,$$

$$A_2 \triangleq A_3 + A_4 \times 2^{n-e_{w-1}},\ 0 \leqslant A_3 \leqslant 2^{n-e_{w-1}} - 1,$$

and

$$A_{2i} \triangleq A_{2i+1} + A_{2i+2} \times 2^{e_{w-i+1}-e_{w-i}},\ 0 \leqslant A_{2i+1} \leqslant 2^{e_{w-i+1}-e_{w-i}} - 1, i = 2, 3, \ldots, w-1.$$

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.

[6] H. Wu. On computation of polynomial modular reduction. Technical report, The centre for applied cryptographic research, University of Waterloo, 2000. CORR 2000-31.