Prime Field Multiplication in Adapated Modular Number System using Lagrange Multiplication

Christophe Negre, Thomas Plantard

Prime Field Multiplication in Adapted Modular Number System using Lagrange Representation

Abstract

In SAC'04 Bajard *et al.* introduced a new system of representation for integer arithmetic modulo a prime integer p, the Modular Number System. The multiplication in the MNS consists of a multiplication of two polynomials and a reduction of the coefficients. In this paper, we propose to use a *Lagrange Representation* to perform the polynomial multiplication in the MNS Algorithm. This method provides a multiplier with a complexity of n multiplications and $n(3\log_2(n)+2)$ additions of computer words. In practice, our method becomes better than usual methods when the size of the fields are larger than 500bits.

1 Introduction

For efficient implementation of cryptographic applications like Diffie-Hellman key-exchange protocol [7] or ECC [10, 9] its is necessary to have efficient modular integer arithmetic. Specially, for Diffie-Hellman key exchange the main operation is an exponentiation modulo a prime integer p: this operation is generally done using a chain of square and multiply modulo p. For ECC the main operation is the scalar multiplication which requires additions and multiplications modulo a prime integer p.

The multiplication modulo p consists to multiply two integers A and B and after that to compute the remainder modulo p. The methods to perform this operation differ if the integer p has a special form or not. If p is arbitrary, the most used methods are the method of Montgomery [11] and the method of Barrett [3]. The cost of these two methods is roughly equal to the cost of two integer multiplications.

If we choose the integer p with sparse binary representation [15] the reduction modulo p can be done really efficiently. This last case is, for now, the most efficient: to give a brief idea, the cost of the modular multiplication is mainly equal to one integer multiplication, followed by several additions/subtractions. Consequently Standards recommend this type of prime integer [13].

Recently Bajard, Imbert and Plantard [2] proposed a new method to perform modular arithmetic by using a new representation of the elements. An integer A modulo p is expressed as $A = \sum_{i=0}^{n-1} a_i\gamma^i$ with $0 \leq \gamma \leq p$ satisfies $\gamma^n \equiv c \mod p$ where c is equal to $\pm 1, \pm 2$. The coefficients a_i are small compared to p and γ.

In this representation the multiplication of A and B is done in two steps: the first step consists to multiply the polynomials A and B modulo $\gamma^n - c$, the second step consists to reduce the coefficients using a *small* MNS-representation of 2^k.

In this paper we will present a modified version of the multiplier of [2]. Specially, the polynomial multiplication is done modulo a small integer m such that m is bigger than the coefficients of the product of A and B. This approach get a real benefit when the polynomial $X^n - c$ splits totally modulo m: using a *Lagrange representation* the polynomial multiplication modulo $X^n - c$ becomes really efficient.

This article is organized as follows: in the section 2 and 3 we will briefly recall the AMNS representation and the original AMNS multiplier presented in [2]. Then, in section 4 we will recall the Lagrange representation [1], after that we will present the Lagrange form of our algorithm, with a refined version which uses the Fast Fourier Transform. We will give a small

example to illustrate our method. Finally we evaluate the complexity of our algorithm, and compare it with others methods, and finish by a brief conclusion.

2 Modular Number System

The efficiency of integer arithmetic is generally closely related to the system used to represent the elements. The most natural system used to represent positive integer is the Classical Number System.

Definition 1 (CNS). *A Classic Number System, is a couple of integer (β, n), such that for all positive integers $0 \leq A < \beta^n$ we can represent A by a vector $(a_i)_{i=0,\ldots,n-1}\mathbb{Z}^n$ such that*

$$\begin{array}{l} A = \sum_{i=0}^{n-1} a_i \beta^i \\ \forall 0 \leq i < n, \quad 0 \leq a_i < \beta \end{array} \tag{1}$$

The a_i are called the digits.

In the classic number system, n is the number of digits. β is the basis and also the upper bound of the digits.

Example 1. *We can verify that in the CNS $(4, 3)$, we can represent the integer $a = 22$ by the vector $A = [1, 1, 2]$.*

$$22 = 2 \times 4^0 + 1 \times 4^1 + 1 \times 4^2 \tag{2}$$

In a recent work [2], Bajard *et al.* proposed a modified system of representation to perform modular arithmetic: the Modular Number System. The Modular Number System is a system of representation derived from the CNS which includes the modulo p used in the modular arithmetic. In the MNS the condition $a_i \leq \beta$ is modified, the basis β can be as big as the prime p, but the a_i remains small.

Definition 2 (MNS [2]). *A Modular Number System (MNS) $\mathcal{B}$, is a quadruple (p, n, γ, ρ), such that for all positive integers $0 \leq A < p$ there exists a polynomial $A_p(X)$ such that*

$$\begin{array}{l} A_p(\gamma) = A \bmod p \\ \deg(A_p) < n \\ \|A_p\|_\infty < \rho \end{array} \tag{3}$$

The polynomial A_p is a representation of A in $\mathcal{B}$. We note that $A_p = A_{\mathcal{B}}$. In the sequel we will generally use the same notation for the integer A and its MNS representation A_p.

Example 2. *In the table 1, we prove that the quadruplet $(17, 3, 7, 2)$ is a MNS.*

0	1	2	3	4	5
0	1	$-X^2$	$1 - X^2$	$-1 + X + X^2$	$X + X^2$

6	7	8	9	10	11
$-1 + X$	X	$1 + X$	$-X - 1$	$-X$	$-X + 1$

12	13	14	15	16
$-X - X^2$	$1 - X - X^2$	$-1 + X^2$	X^2	-1

Table 1: The elements of $\mathbb{Z}_{17}$ in $\mathcal{B} = MNS(17, 3, 7, 2)$

In particular, we can verify that if we evaluate $(-1 + X + X^2)$ in γ, we have $-1 + \gamma + \gamma^2 = -1 + 7 + 49 = 55 = 4 \bmod 17$. We have also $\deg(-1 + X + X^2) = 2 < 3$ and $\| - 1 + X + X^2\|_\infty = 1 < 2$.

The third definition below specializes a sub-family of the Modular Number Systems. Specially Bajard *et al.* use the possibility to choose freely the basis γ to have advantageous properties for the modular arithmetic. That's why we say that this system is adapted to the modular arithmetic: this is the *Adapted Modular Number System.*

Definition 3 (AMNS [2])**.** *A Modular Number System $\mathcal{B} = (p, n, \gamma, \rho)$ is called Adapted (AMNS) if there exists a small integer c such that*

$$\gamma^n = c \bmod p \tag{4}$$

We call E the polynomial $X^n - c$. γ is a root of the polynomial E in $\mathbb{Z}/p\mathbb{Z}$:

$$E(\gamma) \equiv 0 \pmod{p} \tag{5}$$

We also note $(p, n, \gamma, \rho)_E$ the Modular Number System (p, n, γ, ρ) which is adapted to the polynomial E.

3 AMNS Multiplication

In this section, we recall the AMNS multiplication algorithm proposed in [2]. This algorithm consists first to perform the product $C = A \times B \mod E$, and after that to reduce the coefficients c_i of C to get $c_i < \rho$.

Algorithm 1: AMNS Multiplication

Input : $\mathcal{B} = (p, n, \gamma, \rho)_c$ a AMNS
A, B two polynomials such that $A, B \in \mathcal{B}$
Output: R a polynomial such that $R \in \mathcal{B}$ and $R(\gamma) \equiv A(\gamma)B(\gamma) \pmod{p}$
begin
  $C \leftarrow A \times B \bmod E$;
  $R \leftarrow RedCoeff(C)$;
end

After the first step of the algorithm we get a polynomial C with degree $(n-1)$ such that $C(\gamma) \equiv A(\gamma)B(\gamma) \pmod{p}$. Indeed, by definition (Definition 3), we know that $E(\gamma) \equiv 0 \bmod p$ and that $C(X) = A(X)B(X) + Q(X)E(X)$. Thus we have

$$C(\gamma) = A(\gamma)B(\gamma) + Q(\gamma)E(\gamma) \equiv A(\gamma)B(\gamma) \pmod{p}.$$

But the polynomial C has coefficients which are bigger than ρ. Specially we know that $\|A\|_\infty, \|B\|_\infty < \rho$, thus the coefficients of C are such that $\|C\|_\infty < |c| n \rho^2$. That's why we have to reduce its coefficients to get the c_i smaller that ρ. We propose a simplified version of the Algorithm RedCoeff of Bajard *et al.* [2].

Algorithm 2: RedCoeff

Input : $\mathcal{B} = (p, n, \gamma, \rho)_c$ a AMNS
C a polynomial such that $\deg C < n$
Data : $\xi \in \mathcal{B}$ such that $2^k \equiv \xi(\gamma) \pmod{p}$ with $2^k \geq \rho, |c|\|\xi\|_1$
Output: R a polynomial such that $R \in \mathcal{B}$ and $R(\gamma) \equiv C(\gamma) \pmod{p}$
begin
  $R \leftarrow C$;
  while $\|R\|_\infty \geq \rho$ **do**
    $R = L + 2^k U$ with $\|L\|_\infty < 2^k$;
    $R \leftarrow L + \xi \times U \bmod E$;
  end
end

The basic idea of the algorithm consists to replace the "big" coefficient 2^k by an equivalent polynomial ξ (i.e., $\xi(\gamma) \equiv 2^k \pmod{p}$).

$$R = L + 2^k U \longrightarrow L + \xi \times U \bmod E. \tag{6}$$

This is interesting when ξ has small coefficients: in this case the coefficients of the polynomial in the right part are smaller than the original polynomial R. And if we repeat this process a number of times sufficiently large, we get a polynomial R with coefficients smaller than ρ.

Let us check that the Algorithm 2 works properly. We will show that the algorithm stops and that the output R satisfies $\|R\|_\infty < \rho$. For this we use the norm $\|A\|_1 = \sum_{i=0}^{n-1} |a_i|$ of a polynomial A. We will show that when $\|R\|_\infty \geq \rho$ and if we apply the reduction described in equation (6) the norm $\|R\|_1$ strictly decreases. This means that there is a finite number of loop WHILE, because there is only a finite number of value between 0 and $\|C\|_1$.

So we suppose that $\|R\|_\infty \geq \rho$. In this situation, we have $U > 0$ since, by assumption, $\rho \geq 2^k$. We compare the norm of $L + 2^k U$ and $L + \xi \times U \bmod E$ to show that this norm decreases. First, we evaluate $\|L + 2^k U\|_1$

$$\|L + 2^k U\|_1 = \|L\|_1 + \|2^k U\|_1 = \|L\|_1 + 2^k \|U\|_1$$

Next, we evaluate the norm of $\|L + \xi \times U \bmod E\|_1$

$$\|L + \xi \times U \bmod E\|_1 \leq \|L\|_1 + \|\xi \times U \bmod E\|_1 \leq \|L\|_1 + |c|\|\xi\|_1 \|U\|_1.$$

By assumption the polynomial ξ satisfies $2^k \geq |c|\|\xi\|_1$, this implies that

$$\|L + \xi \times U \bmod E\|_1 < \|L\|_1 + 2^k \|U\|_1 \leq \|L + 2^k U\|_1$$

This means that the norm $\|R\|_1$ strictly decreases as required.

In practice, we choose ξ which has a small number of non-zero coefficients, and such that these coefficients are equal to ± 1. With such ξ, we have only to perform two passing through the loop WHILE of the Algorithm 2.

4 Improved AMNS Algorithm

The AMNS multiplication (Algorithm 1) requires one polynomial multiplication modulo $E = X^n - c$. There is different strategy to perform this operation efficiently: the polynomial multiplication can be done with Karatsuba or FFT algorithm, followed by a reduction modulo E.

Here we will study a modified version of Algorithm 1 by using a Lagrange representation of the polynomials. Our method performs the polynomial multiplication and the reduction modulo E at the same time. We begin by a brief review on Lagrange Representation [1].

4.1 Lagrange Representation

The Lagrange representation consists to represent a polynomial by its values at n points, the roots of $E = \prod_{i=1}^{n}(X - \alpha_i)$ modulo an integer m. In an arithmetic point of view, this is related to the Chinese remainder theorem which asserts that the following application is an isomorphism.

$$\begin{array}{ccc} \mathbb{Z}/m\mathbb{Z}[X]/(E) & \longrightarrow & \mathbb{Z}/m\mathbb{Z}[X]/(X-\alpha_1) \times \cdots \times \mathbb{Z}/m\mathbb{Z}[X]/(X-\alpha_n) \\ A & \longmapsto & (A \bmod (X-\alpha_1), \ldots, A \bmod (X-\alpha_n)). \end{array} \tag{7}$$

We remark that the computation of $A \bmod (X - \alpha_i)$ is simply the computation of $A(\alpha_i)$. In other words the image of $A(X)$ by the isomorphism (7) is nothing else that the multi-points evaluation of A at the roots of E.

Definition 4 (Lagrange representation)**.** *Let $A \in \mathbb{Z}[X]$ with $\deg A < n$, and $\alpha_1, \ldots, \alpha_n$ the n distinct roots modulo m of $E(X)$.*

$$E(X) = \prod_{i=1}^{r} (X - \alpha_i) \mod m$$

If $a_i = A(\alpha_i) \mod m$ for $1 \leq i \leq k$, the Lagrange representation (LR) of $A(X)$ modulo m is defined by

$$\mathrm{LR}(A(X), m) = (a_1, \ldots, a_n). \tag{8}$$

The advantage of the LR representation to perform operations modulo E is a consequence of the Chinese remainder theorem. Specially the arithmetic modulo E in classical polynomial representation can be costly if E has a high degree, in LR representation this arithmetic is decomposed into n independent arithmetic units, each consists of arithmetic modulo a very simple polynomial $(X - \alpha_i)$. But arithmetic modulo $(X - \alpha_i)$ is the arithmetic modulo m since the product of two degree zero polynomials is just the product modulo m of the two constant coefficients.

4.2 Improved AMNS algorithm using Lagrange representation

Let us go back to the Algorithm 1 et let us see how to use Lagrange representation to perform polynomial arithmetic in the AMNS multiplication Algorithm.

In view to use Lagrange representation, we select an integer m such that the polynomial $E = (X^n - c)$ splits in $\mathbb{Z}/(m\mathbb{Z})[X]$

$$E = \prod_{i=1}^{n} (X - \alpha_i) \mod m.$$

We can then represent the polynomials A and B in Lagrange representation, and in the Algorithm 1, we can do the multiplication $C = A(X) \times B(X) \mod E$ in Lagrange representation.

Using this strategy, we have to deal with some troubleshooting: it is not possible to perform the reduction of the coefficients in Lagrange representation. Consequently we have to reconstruct the polynomial form of C and then reduce the coefficient. So if we call *LRtoPol* and *PoltoLR* the two subroutines which performs the change of representation between Polynomial and Lagrange representation, we get the following modified form of the Algorithm 1.

Algorithm 3: Lagrange-AMNS Multiplication

Input : $LR(A, m), LR(B, m)$ the Lagrange representation modulo m
Output: $LR(R, m)$ such that $R \in \mathcal{B}$ and $R(\gamma) = A(\gamma)B(\gamma) \bmod p$
begin
  $LR(C, m) \leftarrow LR(A, m) \times LR(B, m)$;
  $C \leftarrow LRtoPol(LR(C, m))$;
  $R \leftarrow RedCoeff(C)$;
  $LR(R, m) \leftarrow PoltoLR(R)$;
end

4.3 The *PoltoLR* routine

We deal with the computation of the Lagrange representation $LR(A, m)$ from the polynomial representation of A. Recall that $E = X^n - c$, consequently the roots α_j of E modulo m are of the form $\alpha_j = \Omega\omega^j$ where Ω is an arbitrary roots of E modulo m and ω is a primitive n-th roots of unity.

To compute $A(\Omega\omega^j)$ for $j = 1, \ldots, n$ we first determine

$$\widetilde{A}(X) = A(\Omega X) = \sum_{i=0}^{n-1} a_i \Omega^i X^i.$$

After that we get

$$LR(A,m) = (\widetilde{A}(1), \widetilde{A}(\omega), \ldots, \widetilde{A}(\omega^{n-1})) = DFT(m,n,A,\omega).$$

So the *PoltoLR* works as follows

Algorithm 4: PoltoLR

Input : $A(X)$ a polynomial
Data : A root Ω of $E = X^n - c$ and ω a primitive n-th roots of unity in $\mathbb{Z}/m\mathbb{Z}$
Output: $LR(A,m)$
begin
  $\widetilde{A} \leftarrow A(\Omega X)$;
  $LR(A,m) \leftarrow DFT(\widetilde{A})$;
end

4.4 The *LRtoPol* routine

For the reverse problem which consists to reconstruct the polynomial $A(X)$ from its Lagrange representation $LR(A,m)$ we simply reverse the previous process:

1. we first compute $\widetilde{A} = DFT^{-1}(m,n,A,\omega)$,
2. and after that $A(X) = \widetilde{A}(\Omega^{-1}X) = \sum_{i=0}^{n-1} \widetilde{a}_i \Omega^{-i} X^i$.

Algorithm 5: LRtoPol

Input : $LR(A,m)$
Data : A root Ω of $E = X^n - c$ and ω a primitive n-th roots of unity in $\mathbb{Z}/m\mathbb{Z}$
Output: $A(X)$
begin
  $\widetilde{A}(X) \leftarrow DFT^{-1}(LR(A,m))$;
  $A(X) = \widetilde{A}(\Omega^{-1}X)$;
end

Finally the cost of each change of representation is mainly reduced to the computation of the DFT or the reverse DFT^{-1}. This is really interesting when the integer n is a power of 2 since in this case we can use the Fast Fourier Transform.

4.5 Fast Fourier Transform.

We briefly recall how the FFT works. The Fast Fourier transform is a recursive algorithm which computes the DFT of a polynomial A at the $n = 2^\ell$ roots of unity mod m. Let us denote ω a primitive n-th root of unity, and $\widehat{a}_i = A(\omega^i)$ the i-th coefficient of the DFT of A. The FFT is thus based on the following property

$$\begin{array}{rcl} \hat{a}_i & = & A(\omega^i) = A_1((\omega^2)^i) + \omega^i A_2((\omega^2)^i), \\ \hat{a}_{i+(n/2)} & = & A(\omega^{i+n/2}) = A_1((\omega^2)^i) - \omega^i A_2((\omega^2)^i), \end{array} \tag{9}$$

where the polynomial $A_1(X)$ is the even part of A and $A_2(X)$ is the odd part of A

$$A_1(X) = \sum_{i=0}^{n/2} a_{2i} X^i, \quad A_2(X) = \sum_{i=0}^{n/2} a_{2i+1} X^i.$$

The computation of $A_1((\omega^2)^i)$ and $\omega^i A_2((\omega^2)^i)$ are thus common to $\hat{a}_i$ and $\hat{a}_{i+(n/2)}$.

Consequently to compute $FFT(m,n,\omega,A)$ we recursively compute $FFT(m,n/2,\omega^2,A_1)$ and $FFT(m,n/2,\omega^2,A_2)$ and after that we deduce $FFT(m,n,\omega,A)$ by using the relations (9). The FFT algorithm is given in appendix (Algorithm 6).

Brassard *et al.* in [4] used such algorithm to multiply two integers using a modified version of the Schonhage-Strassen Algorithm. The major improvement of Brassard *et al.* was to use roots of unity in $\mathbb{Z}/m\mathbb{Z}$ instead of complex roots of unity to perform the FFT.

Moreover they proposed to use special m, typically they set $n = 2^\ell$ and they proposed to take m as a Fermat number $m = 2^{n/2} + 1$. They showed that using such m, $\mathbb{Z}/m\mathbb{Z}$ admit n-th roots of unity which are $\omega_i = 2^i$ for $i = 0, \ldots, n$.

The use of such m is interesting since the multiplication of any integer a by these roots consists of a simple shift of the binary representation of a. Consequently, the multiplications modulo m by the roots of unity in the FFT algorithm 6 can be done as follows

$$\begin{aligned} c &= a\omega^i \mod m \\ &= ((a \uparrow i) \mod 2^{n/2}) - ((a \uparrow i)/2^{n/2}), \end{aligned}$$

where $(a \uparrow i)$ is the integer a shifted by i places. Consequently any multiplication by ω^i has cost which is equal to one addition.

Consequently in this situation, every multiplication in the algorithm of FFT is replaced by an addition, and we get a real improvement of the efficiency of the FFT.

5 Example

In this section we present a complete example of the Lagrange-AMNS multiplication. This example concerns the multiplication in the AMNS $\mathcal{B}$:

$$\mathcal{B} = (p = 72057595648540673, n = 4, \gamma = 54044296180953088, \rho = 2^{14})$$

In this AMNS, we have $c = -1$:

$$E = X^4 + 1$$

with $E(\gamma) \equiv 0 \pmod{p}$.

In the algorithm RedCoeff (Algorithm 2), we use the polynomial $\xi = X + X^2 + X^3$ with $2^{14} \equiv \xi(\gamma) \pmod{p}$ and in the Lagrange-AMNS multiplication (Algorithm 3), we use the moduli $m = 2^{32} + 1$. In this situation the polynomial E splits as

$$E = \prod_{i=0}^{n-1} (X - \Omega\omega^i) \bmod m$$

where $\Omega = -2^8, \omega = 2^{16}$. We will apply the Lagrange-AMNS multiplication to the two polynomials below expressed in this AMNS

$$\begin{aligned} A &= 8932 + 13274X - 1171X^2 - 2557X^3, \\ B &= -10984 + 11764X - 9934X^2 + 11677X^3. \end{aligned}$$

We compute the Lagrange representation of A and B

$$\begin{aligned} LR(A, m) &= [4164503771, 714940440, 4271963375, 3733530033], \\ LR(B, m) &= [1006887238, 849360966, 1985988843, 452686314]. \end{aligned}$$

Now, we can begin the Lagrange-AMNS multiplication algorithm (Algorithm 3).

1. First, we compute $LR(C, m)$, we perform the multiplication of $LR(A, m)$ and $LR(B, m)$ in Lagrange representation:

$$LR(C, m) = [756168373, 3163466426, 341208378, 3390444409].$$

2. In a second time, we compute the polynomial C using the *LRtoPol* algorithm (Algorithm 5), we get

$$C = -234661752 - 52453039X + 110145201X^2 - 13254508X^3.$$

3. Now, we reduce the coefficients of C using the RedCoeff algorithm (Algorithm 2). We decompose

$$\begin{array}{rcll} C & = & L + 2^{14}U & \text{with} \quad L = 6280 + 8529X + 11953X^2 + 148X^3 \\ & & & \text{and} \quad U = -14323 - 3202X + 6722X^2 - 809X^3 \end{array}$$

We compute $\xi U \bmod E$

$$\begin{array}{rcl} \xi U \bmod E & = & (X + X^2 + X^3)(-14323 - 3202X + 6722X^2 - 809X^3) \bmod (E) \\ & = & -20236X - 16716X^2 - 10803X^3 - 2711 \bmod (E). \end{array}$$

We obtain

$$\begin{array}{rl} R & = L + \xi U \bmod E \\ & = (6280 + 8529X + 11953X^2 + 148X^3) + (-2711 - 20236X - 16716X^2 - 10803X^3) \\ & = 3569 - 11707X - 4763X^2 - 10655X^3 \end{array}$$

We can stop the reduction of the coefficients because we have $\|R\|_\infty = 11707 < \rho = 16384$.

We can verify that the result is exact. We have

$$\begin{array}{rcl} A(\gamma) & = & 32371859214075011 \bmod p, \\ B(\gamma) & = & 65633143240000727 \bmod p, \\ R(\gamma) & = & 57944334546866439 \bmod p, \end{array}$$

and $R(\gamma) \equiv A(\gamma)B(\gamma) \pmod p$

4. The Lagrange-AMNS algorithm finish by computing the Lagrange representation of R using the PoltoLR algorithm

$$LR(R, m) = [2358429896, 3452218564, 1312248603, 1467051807].$$

6 Complexity

In this part, we evaluate the complexity of the Lagrange-AMNS multiplication (Algorithm 3). In the table 2, we give the cost of the different steps of our method. The cost of this algorithm is expressed in term of multiplications and additions of computer word.

Let n the degree of the polynomial E, we evaluate the number of word operation on word which have the length of m. In the previous example, m is about 2^{32}.

Table 2: Complexity of Lagrange-AMNS multiplication

Operation	#Mult.	#Add.
Pol. Mult.	n	n
LRtoPol	-	$(n/2)(3\log_2(n) - 1)$
RedCoeff	-	$n\|\xi\|_1$
PoltoLR	-	$(n/2)(3\log_2(n) - 1)$
Total	n	$n(3\log_2(n) + \|\xi\|_1)$

In practice ξ is such that $\|\xi\|_1 \leq 3$, i.e., really small.

Let us compare our method to usal methods to perform arithmetic modulo a prime integer p. There is two strategies:

1. First, there exists some generalist algorithm like Montgomery [11] or Barrett [3] methods which perform integer multiplication modulo an arbitrary prime p. These two techniques for modular multiplication have a cost bigger than two integer multiplications.

2. In the other hand, there exists some particular moduli like Mersenne number [8], pseudo Mersenne number [6], generalized Mersenne number [15], and More generalized Mersenne number [5]. The multiplication modulo this type of prime number is done by first computing the multiplication of two integer, and after that by computing the remainder modulo p. The modular reduction have at least the cost of one addition due to the special form of p.

Consequently, all these techniques require to perform at least one integer multiplication. This integer multiplication can be done using the following classical techniques

- Classical school book method,
- Karatsuba,
- Schonhage-Strassen.

The Schonhage-Strassen-like method ([14, 4]) use the Fast Fourier Transform. Specially, if the integer are stored in n computer words, the Schonhage-Strassen-like method require to evaluate two polynomials at 2n points with a FFT. Which is exactly twice bigger than in our method where we evaluate two polynomials at only n points. Consequently the cost of these methods [14, 4] is equal to $2n$ word multiplications and $2n(3\log_2(n) + \|\xi\|_1)$ word additions. Our method is thus two times better than any method which uses a Schonhage-Strassen-like Algorithm to perform integer multiplication.

Consequently the complexity of our method is very competitive: we know that bigger his the length of moduli, better is the cost of our method compared to other methods.

Now, the important point is to know when our method become better than classical methods when they use Classical method or Karatsuba to perform integer multiplication.

To compare our method to these methods we evaluate the complexity for different size of prime. We used the algorithm proposed in [2] to build efficient AMNS for the Lagrange-AMNS algorithm. With this Algorithm, we find several efficient AMNS. In the the appendix B, we give a complete definition of different AMNS that we propose for modular arithmetic. In the tables 3 and 4, we compare the cost of the modular multiplication in these different AMNS to the cost of a classic integer multiplication. In [12], the GNU MP proposed to use classic algorithm for multiplication on n computer words when $n < 10$ and Karatsuba Algorithm when $10 \leq n < 300$.

Table 3: Cost of Lagrange-AMNS multiplication with $m = 2^{32} + 1$

	Lagrange-AMNS Multiplication		Classic Multiplication			
			Classic Method		Karatsuba Method	
$\|p\|$	$\#M_{32}$	$\#A_{32}$	$\#M_{32}$	$\#A_{32}$	$\#M_{32}$	$\#A_{32}$
111	8	88	16	32	9	44
207	16	224	49	98	25	150
416	32	576	169	338	71	475
768	64	1344	576	1152	189	1333

We can see in the tables 3 and 4 that when p is about 2^{400} our method becomes faster than any other methods. We know that this advantage grow when the length of p become bigger.

7 Conclusion

In this paper we have presented a novel Algorithm to perform integer modular arithmetic. Primarily we gave a polynomial formulation of our algorithm which use the AMNS [2] representation of integer and a Montgomery-like method to reduce the coefficients. Secondly

Table 4: Cost of Lagrange-AMNS multiplication with $m = 2^{64} + 1$

	Lagrange-AMNS Multiplication		Classic Multiplication			
			Classic Method		Karatsuba Method	
$\|p\|$	$\#M_{64}$	$\#A_{64}$	$\#M_{64}$	$\#A_{64}$	$\#M_{64}$	$\#A_{64}$
223	8	88	16	32	9	44
240	8	96	16	32	9	44
464	16	240	64	128	27	165
926	32	544	225	450	256	542

we modified this algorithm in view to use a Lagrange representation to speed up the polynomial multiplication part of the algorithm. We obtain an algorithm which has a complexity of n and $n(3\log_2(n) + \|\xi\|_1)$ additions of computer words. This complexity is twice better than classical method which use Schonhage-Strassen to perform integer multiplication. In practice, our method should be better than usual method for prime p bigger than 2^{500}.

References

[1] J.-C. Bajard, L. Imbert, C. Negre, and T. Plantard. Efficient multiplication in GF (p^k) for elliptic curve cryptography. In *ARITH'16: IEEE Symposium on Computer Arithmetic*, pages 181–187, June 2003.

[2] J.-C. Bajard, L. Imbert, and T. Plantard. Modular number systems: Beyond the Mersenne family. In *SAC'04: 11th International Workshop on Selected Areas in Cryptography*, pages 159–169, August 2004.

[3] P. Barrett. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In A. M. Odlyzko, editor, *Advances in Cryptology – CRYPTO '86*, volume 263 of *LNCS*, pages 311–326. Springer-Verlag, 1986.

[4] G Brassard, S Monet, and D Zuffellato. Algorithms for very large integer arithmetic. *Tech. Sci. Inf.*, 5(2):89–102, 1986.

[5] J. Chung and A. Hasan. More generalized Mersenne numbers. In M. Matsui and R. Zuccherato, editors, *Selected Areas in Cryptography – SAC 2003*, volume 3006 of *LNCS*, Ottawa, Canada, August 2003. Springer-Verlag. (to appear).

[6] R. Crandall. Method and apparatus for public key exchange in a cryptographic system. U.S. Patent number 5159632, 1992.

[7] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[8] D. E. Knuth. *The Art of Computer Programming, Vol. 2. Seminumerical Algorithms.* Addison-Wesley, Reading, Mass., 1981.

[9] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, January 1987.

[10] V. Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology, proceeding's of CRYPTO'85*, volume 218 of *LNCS*, pages 417–426. Springer-Verlag, 1986.

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

[12] GNU MP. The gnu multiple precision arithmetic librairy, May 2006.

[13] National Institute of Standards and Technology. *FIPS PUB 197: Advanced Encryption Standard (AES).* FIPS PUB. National Institute for Standards and Technology, November 2001.

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

[15] J. Solinas. Generalized Mersenne numbers. Research Report CORR-99-39, Center for Applied Cryptographic Research, University of Waterloo, Waterloo, ON, Canada, 1999.

A Appendix: FFT

The Algorithm below is the Algorithm given in the paper of Brassard *et al.* wich performs the Fast Fourier Transform of a polynomial in $\mathbb{Z}/m\mathbb{Z}[X]$.

Algorithm 6: Fast Fourier Transform [4]

Input : Two integers m and $n = 2^\ell$, ω a primitive n^{th} root of unity modulo m, and $A = [a_0, a_1, \ldots, a_{n-1}]$ the coefficients of a degree $n-1$ polynomial in $\mathbb{Z}[X]$.
Output: $\widehat{A} = (\widehat{a}_0, \ldots, \widehat{a}_{n-1})$ the DFT of A in $\{1, \omega, \omega^2, \ldots, \omega^{n-1}\}$
if $n = 0$ **then**
  $FFT(m, n, \omega, A) \leftarrow A$
else
  $A_1 \leftarrow [a_0, a_2, \ldots, a_{2(n-1)}]$;
  $R_1 \leftarrow FFT(m, n/2, \omega^2, A_1)$;
  $A_2 \leftarrow [a_1, a_3, \ldots, a_{2n-1}]$;
  $R_2 \leftarrow FFT(m, n/2, \omega^2, A_2)$;
  for $i = 0, \ldots, n/2 - 1$ **do**
    $\lambda \leftarrow \omega^i R_2[i] \mod m$;
    $R[i] \leftarrow (R_1[i] + \lambda \mod m)$;
    $R[i + n/2] \leftarrow (R_1[i] - \lambda \mod m)$;
  end
end

B Appendix: AMNS

In this section we give a list of practical AMNS basis associated to prime integer.

B.1 AMNS with 32bits digit

B.1.1 111bits modulo

1. $p = 2596148467953040258123756591841281$
2. $n = 8$
3. $\gamma = 84309851953528350105183947308107 1$
4. $\rho = 2^{14}$
5. $\xi = X^1 + X^4$
6. $m = 2^{32} + 1$

B.1.2 207bits modulo

1. $p = 205688094185080687937563657719079685397477172542110018664726529$
2. $n = 16$
3. $\gamma = 33405783449899356776149843245645372735825774648390014355904674$
4. $\rho = 2^{13}$
5. $\xi = X^3 + X^8$
6. $m = 2^{32} + 1$

B.1.3 416bits modulo

1. $p = 168570521293320685225250332101994914278690400043010070371320270686591074038166717$
   $778527995073687025652394187385473000450301697$
2. $n = 32$
3. $\gamma = 121833832973826529775071731437626530125799994126995466989252307821168182754343543$
   $302226994446050014814489559979204590274925745 6$
4. $\rho = 2^{13}$
5. $\xi = 1 + X + X^{11}$
6. $m = 2^{32} + 1$

B.1.4 768bits modulo

1. $p = 152844561450046858090486331989628783016245328788248849880240943409507021526800203$
   $803196109709554142411552343109684432033506660611828237802265386141634718393094819606809230$
   $856258195523359949352498520115064791450164428098633332643289 7$
2. $n = 64$
3. $\gamma = 827251465040671895170593654922786175533014713593037709264952321236603859767846334$
   $093697523348503291182628927068258330739625778413222092520061427071011561016524449016726892$
   $693682176321088384219159099445808437600756053427538701149540$
4. $\rho = 2^{12}$
5. $\xi = 1 + X^4 + X^{27}$
6. $m = 2^{32} + 1$

B.2 AMNS with 64bits digit

B.2.1 223bits modulo

1. $p = 13479973333575319897333507543509820529115276003593724674453077491713$
2. $n = 8$
3. $\gamma = 566385478070800442705852402027281060151850784783130045358570508872$
4. $\rho = 2^{28}$
5. $\xi = X^2 + X^5$
6. $m = 2^{64} + 1$

B.2.2 240bits modulo

1. $p = 176684706477838434797324399113358486132140007936646886333434580916450099 3$
2. $n = 8$
3. $\gamma = 110427941240116917154820642115569949581402602648218323880853675119214591$
4. $\rho = 2^{30}$
5. $\xi = X + X^4 + X^7$
6. $m = 2^{64} + 1$

B.2.3 464bits modulo

1. $p = 476341012158300296208621682932821057556196476279544379403225322850924005619267209$
   $159987170490190995693172801308864032578102213459155353603 53$
2. $n = 16$
3. $\gamma = 315425170274030140664175282670289214830164278211803918141082846974356540767338607$
   $114907160862852966540803812100908119170837780037536472100 94$

4. $\rho = 2^{29}$
5. $\xi = 1 + X + X^{10}$
6. $m = 2^{64} + 1$

B.2.4 926bits modulo

1. $p = 56725193347083399307177066732402822880983741823554781305533289311063410078445601407820446567388776801691252536879344262493777887836135900209740756822332418153004763098696952514972549418254907314012592849936514122572047774879547817657433625792302790004999465743590775109480716697 7$
2. $n = 32$
3. $\gamma = 38976392291401862683587844546009404668285230879381807092106844587555496189663348807893773408860533535874020701251689282498045386465086289049175774828890218085046964497222677718167246091433558935962766733841264823458995903786639075014558537507775596664295203925980856559105440273$
4. $\rho = 2^{29}$
5. $\xi = X + X^{11}$
6. $m = 2^{64} + 1$