AN EFFICIENT MULTIPLICATION ALGORITHM USING BINOMIAL RESIDUE REPRESENTATION

Yin Li
Institute of Information Security, Shanghai jiaotong University, Shanghai 200240, China

Christophe Negre
Team DALI/ELIAUS, University of Perpignan, France

Keywords: Finite Field, Multiplication, Montgomery, Binomial residue representation.

Abstract: In this paper, we propose an extension of the algorithm proposed by Bajard, Imbert and Negre in (Bajar et al., 2006), refered as BIN algorithm. We use binomial residue representation of field elements instead of the Lagrange representation of (Bajar et al., 2006). Specifically, every elements in $\mathbb{F}_{p^k}$ is represented by a set of residue modulo fixed binomials. We propose two versions of our algorithm, one in general form with a sub-quadratic complexity equal to $O(k^{1.5})$ operations in $\mathbb{F}_p$. The second one is optimized with the use of FFT. In this case the cost is $O(k\log(k))$ operations in $\mathbb{F}_p$. For fields $GF(p^k)$ suitable for elliptic curve cryptography our algorithm roughly improves the time delay of (Bajar et al., 2006) by 45%.

1 INTRODUCTION

Efficient implementation of finite field arithmetic is an important pre-requisite for cryptography and coding theory (Lidl and Niederreiter, 1994). Specifically this is the case for elliptic curve cryptography (ECC), proposed independently by Koblitz (Koblitz, 1987) and Miller (Miller, 1986). In ECC, the most used and also the most costly field operations is the multiplication.

During the past few years, more and more people believe that elliptic curve defined over $\mathbb{F}_{p^k}$ is better than $\mathbb{F}_{2^k}$ and $\mathbb{F}_p$ in efficiency point of view in software environment. Many works (Bailey and Paar, 1998; Lim and Hwang, 2000) have shown that $\mathbb{F}_{p^k}$ is a suitable choice for computer software implementation.

In 2006, Bajard, Imbert and Negre (Bajar et al., 2006) proposed an efficient multiplication algorithm for $\mathbb{F}_{p^k}$ using Lagrange representation (we will refer to it as the BIN algorithm). The BIN algorithm only needs $O(k)$ multiplications in $\mathbb{F}_p$. This algorithm is very efficient in hardware, but in software, it is not so efficient. Indeed it also needs $O(k^2\log k)$ additions in $\mathbb{F}_p$. When p has size of 32 bits which is an interesting case in software environment, multiplication in $\mathbb{F}_p$ is only twice slower than addition in software platform. So $O(k^2\log k)$ additions cause too much time delay in software implementation.

In this paper, we will use an extended form of Lagrange representation used in (Bajar et al., 2006), the binomial residue representation, for field representation.

Using a strategy similar to BIN algorithm through our binomial residue representation, we can reduce the number of addition balanced by the cost of increasing a few number of multiplications in $\mathbb{F}_p$. Specifically, in our algorithm multiplications and additions are only in sub-quadratic number. This ensures that our algorithm is faster in software implementation.

The rest of this paper is organized as follows. In Section 2, we briefly recall the Lagrange representation and then give the definition of binomial residue representation. In Section 3, we present the binomial residue multiplication algorithm in a general and optimized form. In Section 4 we evaluate the complexity of our approach and compare it with other methods for field multiplication. We then briefly conclude and give some further perspectives of this work.

Li Y. and Negre C. (2008).
AN EFFICIENT MULTIPLICATION ALGORITHM USING BINOMIAL RESIDUE REPRESENTATION.
In *Proceedings of the International Conference on Security and Cryptography*, pages 319-324
DOI: 10.5220/0001924503190324

2 BINOMIAL RESIDUE REPRESENTATION

A finite field $\mathbb{F}_{p^k}$ can be seen as the set of polynomials in $\mathbb{F}_p[x]$ with degree less than k. Arithmetic operation in $\mathbb{F}_{p^k}$, i.e., addition, multiplication or inversion, consists to perform it modulo an irreducible polynomial $P \in \mathbb{F}_p[x]$ of degree k. The most used operations in cryptographic protocol are addition and multiplication. The addition is just a simple addition of polynomial in $\mathbb{F}_p[x]$. To multiply two elements $A(x)$ and $B(x)$ modulo P, it requires to first compute the product $C = A \times B$ and then to reduce C modulo P. This operation is in general more costly than an addition, and thus it should be performed efficiently.

Optimal extension field. A strategy to simplify the reduction process consists to choose P as sparse as possible. For example in OEF (Bailey and Paar, 1998) they take P with binomial form. In this case the reduction modulo P consists simply to add the upper part to the lower part of C.

Lagrange representation approach (Bajar et al., 2006). Another interesting approach for modular multiplication is the Montgomery algorithm (cf. (Montgomery, 1985) for the integer version). Given two polynomials A and B of degree less than k, Montgomery algorithm computes $A \times B \times \Phi^{-1} \mod P$ where Φ is a constant polynomial often chosen as $\Phi = x^k$.

Algorithm 1: Montgomery Multiplication (Montgomery, 1985)

Require: Two polynomials $A(x), B(x)$ such that $\deg A, \deg B < k$
Other data : an irreducible degree k polynomial P, and $\Phi(x), \Phi'(x)$ such that $\deg \Phi, \deg \Phi' \geq k$ and P, Φ, Φ' are pairwise prime.
Ensure: $R = A \times B \times \Phi^{-1} \mod P$.
Step 1. $Q \leftarrow A \times B \times P^{-1} \mod \Phi$
Step 2. $R \leftarrow (A \times B - Q \times P) \times \Phi^{-1} \mod \Phi'$

In their paper, Bajard *et al.* proposed a version of Montgomery algorithm which use a Lagrange representation of the elements. Specifically they represent a polynomial $A(x)$ by its evaluation at $2k$ values $\alpha_i, \alpha'_i \in \mathbb{F}_p$ for $i = 1, \ldots, k$

$$A_{Lag} = (A(\alpha_1), \ldots, A(\alpha_k), A(\alpha'_1), \ldots, A(\alpha'_k))$$

As shown in (Bajar et al., 2006), the use of such representation is interesting to implement Montgomery algorithm when $\Phi = \prod_{i=1}^{k}(x - \alpha_i)$ and $\Phi' = \prod_{i=1}^{k}(x - \alpha'_i)$. Indeed, due to Chinese remainder theorem, the multiplication modulo Φ (or Φ') in Lagrange representation is a simple coefficient by coefficient multiplication. In other words it requires k independent multiplications in $\mathbb{F}_p$.

2.1 Binomial Residue Representation

In this subsection we present an extension of Lagrange representation of (Bajar et al., 2006). We use this representation to perform Montgomery multiplication using the same strategy as in (Bajar et al., 2006).

Let $\Phi(x) = \prod_{i=1}^{n} \phi_i(x)$ where ϕ_i are pairwise prime. Recall that Chinese remainder theorem (Bajar et al., 2006; Halbutoğullari and Ç. K. Koç, 2000) asserts that the following application is a ring isomorphism.

$$\begin{array}{rcl} \mathbb{F}_p[x]/(\Phi) & \tilde{\rightarrow} & \mathbb{F}_p[x]/(\phi_1) \times \cdots \times \mathbb{F}_p[x]/(\phi_n) \\ U & \mapsto & (U \mod \phi_1, \cdots, U \mod \phi_n). \end{array} \quad (1)$$

The CRT asserts also that if we know the residue $U_i = U \mod \phi_i$ we can get back to U by computing

$$U = \left(\prod_{i=1}^{n} |U_i \Phi_i^{-1}|_{\phi_i} \Phi_i\right) \mod \Phi$$
$$\text{where } \begin{cases} \Phi_i & = & \prod_{j=1, j\neq i}^{n} \phi_i \\ |\Phi_i^{-1}|_{\phi_i} & = & \Phi_i^{-1} \mod \phi_i \end{cases} \quad (2)$$

The operator $|\cdot|_{\phi_i}$ represents the reduction modulo ϕ_i.

If we consider ϕ_i with binomial form of fixed degree d, then we know that they are pairwise prime.

Lemma 1. *Two binomials $\phi(x) = x^d + \alpha, \phi'(x) = x^d + \alpha'$ with $\alpha, \alpha' \in \mathbb{F}_p$ are relatively prime if and only if $\alpha \neq \alpha'$.*

Proof. It is well known that for any $\phi(x), \phi'(x)$

$$\gcd(\phi(x), \phi'(x)) = \gcd(\phi(x), \phi'(x) - \phi(x)).$$

But in the situation of the Lemma, we have $\phi'(x) - \phi(x) = \alpha' - \alpha$. This implies

$$\gcd(\phi(x), \phi'(x)) = \gcd(\phi(x), \phi'(x) - \phi(x)) = 1$$

if and only if $\alpha' \neq \alpha$. □

So if we take $\Phi = \prod_{i=1}^{n} \phi_i(x)$ where ϕ_i are distinct binomials of degree d, then equation (1) holds. This means that we can represent a polynomial U of degree less than nd by its n residues modulo ϕ_i.

Definition 1 (Binomial Residue Representation)**.** *Let $\phi_1(x) = x^d - \alpha_1, \cdots, \phi_n(x) = x^d - \alpha_n$ be n relatively prime binomials of degree d. Let $U \in \mathbb{F}_p[x]$ with $\deg U < n \cdot d$. The residue representation $U_{BR\Phi}$ of U relatively to $\Phi = \prod_{i=1}^{n} \phi_i$ is defined as the n remainders modulo $\phi_1, \phi_2, \ldots$ and ϕ_n*

$$U_{BR\Phi} = (U_1, \cdots, U_n) \text{ where } U_i(x) = U(x) \text{ mod } \phi_i(x)$$

The arithmetic modulo Φ is advantageous in BR representation since the arithmetic split into n independent arithmetic units which perform arithmetic operations (addition and multiplication) in $\mathbb{F}_p[x]/(\phi_i)$. Based on this feature, we construct an efficient Montgomery Multiplication in the following section.

3 MONTGOMERY MULTIPLICATION USING BINOMIAL RESIDUE REPRESENTATION

Let $\phi_1(x),\cdots,\phi_n(x)$ and $\phi'_1(x),\ldots,\phi'_n(x)$ be $2n$ distinct binomials of degree $d>0$ and let

$$\Phi(x)=\prod_{i=1}^{n}\phi_i(x),\quad \Phi'(x)=\prod_{i=1}^{n}\phi'_i(x).$$

Using this type of polynomial for Φ and Φ' in Montgomery Algorithm 1 we can perform Step 1 and Step 2 in BR representation. The major drawback of this approach is that it requires two BR representation systems : one relatively to Φ and the other to Φ'. Consequently we must include two conversion operation to convert Q from $BR\Phi$ to $BR\Phi'$ and the other to convert R from $BR\Phi'$ to $BR\Phi$. We will explained in subsection 3.1 how to perform these conversions.

In the sequel we note Γ the $BR\Phi'$ representation of $\Phi^{-1} \mod \Phi'$, and we note P' the inverse of P modulo Φ (such polynomials exists since Φ,Φ' and P are pairwise prime).

Algorithm 2: BR multiplication

Require: $A_{BR\Phi},A_{BR\Phi'}$ and $B_{BR\Phi},B_{BR\Phi'}$ the BR representation of $A(x)$ and $B(x)$. of degree $< k$.
Other data : an irreducible degree k polynomial P, and $\Phi(x),\Phi'(x)$ such that $\deg\Phi,\deg\Phi' \geq k$ and P,Φ,Φ' are pairwise prime.
Ensure: $Q_{BR\Phi},Q_{BR\Phi'}$ where $Q = A\times B\times P^{-1} \mod \Phi$
Step 1. $Q_{BR\Phi} \leftarrow A_{BR\Phi}\times B_{BR\Phi}\times P'_{BR\Phi}$
Step 2. $Q_{BR\Phi'} \leftarrow Convert_{BR\Phi\rightarrow BR\Phi'}(Q_{BR\Phi})$
Step 3. $R_{BR\Phi'} \leftarrow (A_{BR\Phi'}\times B_{BR\Phi'} - Q_{BR\Phi'}\times P_{BR\Phi'})\times\Gamma$
Step 4. $R_{BR\Phi'} \leftarrow Convert_{BR\Phi'\rightarrow BR\Phi}(R_{BR\Phi'})$

Remark 1. *We notice that Algorithm 2 requires $2n$ binomials with degree d. But, due to Lemma 1 there are at most p distinct binomials of degree d in $\mathbb{F}_p[x]$. So we must have $p\geq 2n$.*

3.1 Conversion Operation

In this subsection, we focus on the conversion operation in Step 2 and Step 4 in Algorithm 2. There are several methods for completing the transformation.

The first method is mentioned in (Bajar et al., 2006), it consists to apply Newton interpolation. It first computes $Q(x)$and then compute the remainders of $Q(x)$ modulo ϕ'_i for $i=1,\ldots,n$. This method requires roughly $O(\frac{3}{2}n^2d)$ operations in $\mathbb{F}_p$.

This second method consists to apply an interpolation similar to the original Lagrange interpolation. This method is also mentioned in (Bajar et al., 2006) but they showed that it is more complicated than Newton interpolation. Here we will show that this method is well suited to our Binomial Residue representation.

Let Q be a polynomial of degree less than k where k is the degree of P the polynomial which defines the field $\mathbb{F}_{p^k}$. Let $Q_{BR\Phi}=(Q_1,\ldots,Q_n)$ be its binomial representation relatively to Φ. Using equation (2) we can get back to the polynomial form of Q

$$Q=\left(\sum_{i=1}^{n}Q_i|\Phi_i^{-1}|_{\phi_i}\Phi_i\right) \mod \Phi,$$

where

$$\Phi_i=\prod_{j=1,j\neq i}^{n}\phi_j.$$

To get the $BR\Phi'$ representation of Q we have to compute the remainder modulo ϕ'_j, for $j=1,\ldots,n$, of the previous expression of Q. In the following lemma we give the resulting expression of these residues.

Lemma 2. *Let Φ and Φ' be as follow*

$$\Phi=\prod_{i=0}^{n}(x^d-\alpha_i) \text{ and } \Phi'=\prod_{i=0}^{n}(x^d-\alpha'_i),$$

with $\alpha_i,\alpha'_i\in\mathbb{F}_p$ all distinct. Let $Q_{BR\Phi}=(Q_1,\ldots,Q_n)$ the BR representation relatively to Φ, then the BR representation of Q relatively to Φ' can be computed as

$$Q_{BR\Phi'}=\begin{bmatrix}\omega_{1,1}&\omega_{1,2}&\cdots&\omega_{1,n}\\ \omega_{2,1}&\omega_{2,2}&\cdots&\omega_{2,n}\\ \vdots&\vdots&\ddots&\vdots\\ \omega_{n,1}&\omega_{n,2}&\cdots&\omega_{n,n}\end{bmatrix}\begin{bmatrix}Q_1\\Q_2\\\vdots\\Q_n\end{bmatrix} \quad (3)$$

where

$$\omega_{i,j}=\prod_{\ell=1,\ell\neq i}^{n}\frac{\alpha'_j-\alpha_\ell}{\alpha_i-\alpha_\ell}$$

Proof. Let us go back the the following equation

$$Q=\left(\underbrace{\sum_{i=0}^{n}Q_i|\Phi_i^{-1}|_{\phi_i}\Phi_i}_{(\star)}\right) \mod \Phi.$$

$$\Omega = \frac{1}{2^r} \cdot \underbrace{\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \beta^2 & \cdots & \beta^{2n-2} \\ 1 & \beta^4 & \cdots & \beta^{4n-4} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \beta^{2n-2} & \cdots & \beta^{(2n-2)(2n-2)} \end{bmatrix}}_{(\star)} \cdot \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & \beta & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \beta^{n-1} \end{bmatrix} \cdot \underbrace{\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \beta^2 & \cdots & \beta^{2n-2} \\ 1 & \beta^4 & \cdots & \beta^{4n-4} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \beta^{2n-2} & \cdots & \beta^{(2n-2)(2n-2)} \end{bmatrix}}_{(\star)}$$

Figure 1: DFT form of equation (3).

We first show that in our context, no final reduction modulo Φ is needed in this previous expression of Q. Indeed, we show that the polynomials $|\Phi_i^{-1}|_{\phi_i}$ for $i = 1,\ldots,n$ are constant polynomials, and this will prove that the polynomial $(\star)$ has a degree at most $nd-1$.

Let us prove that $|\Phi_i^{-1}|_{\phi_i}$ is a constant. We have

$$|\Phi_i|_{\phi_i} = \left(\prod_{j=1, j\neq i}^{n} (x^d - \alpha_j) \right) \mod (x^d - \alpha_i)$$

Bur, since $x^d = \alpha_i \mod (x^d - \alpha_i)$, we can replace x^d by α_i we get $|\Phi_i|_{\phi_i} = \prod_{j=1, j\neq i}^{n} (\alpha_i - \alpha_j)$ and also

$$|\Phi_i^{-1}|_{\phi_i} = \frac{1}{\prod_{j=1, j\neq i}^{n} (\alpha_i - \alpha_j)}.$$

This proves that $|\Phi_i^{-1}|_{\phi_i}$ is a constant polynomial and that the following equation holds

$$Q = \sum_{i=0}^{n} Q_i |\Phi_i^{-1}|_{\phi_i} \Phi_i.$$

Now we compute the $BR\Phi'$ representation of Q by computing the remainder of the later expression of Q modulo ϕ'_j for $j = 1,\ldots,n$. We have

$$Q'_j = \sum_{i=0}^{n} \left| Q_i |\Phi_i^{-1}|_{\phi_i} \Phi_i \right|_{\phi'_j}. \quad (4)$$

where Q'_j is the j-th polynomial in $Q_{BR\Phi'}$. To simplify this expression (4) we use the binomial form of the polynomials ϕ_i and ϕ'_j.

We first remark that

$$\left| Q_i |\Phi_i^{-1}|_{\phi_i} \Phi_i \right|_{\phi'_j} = \left| Q_i |\Phi_i^{-1}|_{\phi_i} |\Phi_i|_{\phi'_j} \right|_{\phi'_j} \quad (5)$$

But we know that

$$|\Phi_i^{-1}|_{\phi_i} = \frac{1}{\prod_{j=1, j\neq i}^{n} (\alpha_i - \alpha_j)}.$$

and we can obtain using a similar method that

$$|\Phi_i|_{\phi'_j} = \prod_{\ell=1, \ell\neq i}^{n} (\alpha'_j - \alpha_\ell).$$

If we replace these expressions of $|\Phi_i^{-1}|_{\phi_i}$ and $|\Phi_i|_{\phi'_j}$ in (5) we finaly get

$$Q'_j = \sum_{i=0}^{n} Q_i \omega_{i,j} \text{ where } \omega_{i,j} = \prod_{\ell=1, \ell\neq i}^{n} \frac{\alpha'_j - \alpha_\ell}{\alpha_i - \alpha_\ell}$$

□

The main advantage of the conversion expression of equation (3), is that the matrix Ω has its coefficients in $\mathbb{F}_p$. Consequently, if we perform the change of representation using (3) with a direct computation of the matrix vector product, the conversion has a cost of $n^2 d$ multiplications and $n(n-1)d$ additions in $\mathbb{F}_p$.

3.2 Optimization

In some special cases, we can perform the conversions more efficiently. This is possible with the strategy used by Negre (Negre, 2006) to improve Bajard *et al.* approach. Negre showed that under some conditions on α_i and α'_j and p the matrix vector product in equation (3) can be done through two FFT evaluation.

Specifically, assume that $n = 2^r$ and that $2^{r+1} | (p-1)$ then there exists a primitive 2^{r+1}-th root of unity $\beta \in \mathbb{F}_p$. The 2^{r+1} elements β^i for $i = 0,\ldots,2^{r+1}-1$ are distinct elements. We take Φ and Φ' as

$$\Phi = \prod_{i=0}^{n-1} (x^d - \beta^{2i}) \text{ and } \Phi' = \prod_{i=0}^{n} (x^d - \beta^{2i+1})$$

In this situation the matrix $\Omega = [\omega_{i,j}]_{i,j=1,\ldots,n}$ expressed in Lemma 2 can be written as in Figure 1 (cf. Negre (Negre, 2006) for detailed explanation).

The matrix $(\star)$ is $2^r \times 2^r$ matrix of Discrete Fourier Transform , so we can compute the corresponding matrix vector product with FFT algorithm.

Consequently a conversion between two BR representations require two FFT computation and the same is true for the reverse transformation. We deduce that a conversion as a cost of $2n\log_2(n)d$ additions and $n\log_2(n)d + nd$ multiplications in $\mathbb{F}_p$.

4 COMPLEXITY AND COMPARISON

In this section we evaluate the complexity of Algorithm 2. The complexity is given by the number of addition, multiplication and multiplication by a constant of $\mathbb{F}_p$ elements. We denote by M the cost of one multiplication, by A the cost of an addition and CM the cost of one multiplication by a constant in $\mathbb{F}_p$.

4.1 Classical Approach

This approach consists to perform Algorithm 2 using no optimization given by FFT approach. This is the case for example when no 2^{r+1} root of unity lie in $\mathbb{F}_p$. We evaluate in Table 1 the cost of each step of Algorithm 2 separately.

- In Step 1 and Step 3, we have to perform respectively $2n$ multiplications of polynomial of degree d modulo binomials (for Step 3, Γ lie in $\mathbb{F}_p^n$). These multiplications are done using school book method and direct reduction modulo the binomials ϕ_i and ϕ'_j. Each of these multiplications in $\mathbb{F}[x]/(\phi_i)$ has a cost of $d^2M + d(d-1)A + (d-1)CM$.
- Step 2 and Step 4, is done through a matrix vector product as expressed in Lemma 2.

  We indicate the cost of each case in Table 1.

Table 1: Complexity of BR Montgomery Multiplication.

Step	M	A	CM
1	$2nd^2$	$2nd(d-1)$	$2n(d-1)$
2	-	$n(n-1)d$	n^2d
3	$2nd^2$	$2nd(d-1)+nd$	$2n(d-1)+nd$
4	-	$n(n-1)d$	n^2d
Total	$4nd^2$	$4nd^2+2n^2d-5nd$	$2n^2d+5nd-4n$

By adding the complexity of each step and obtain the total complexity of Algorithm 2.

4.2 Optimized BR Multiplication

We suppose now that there exists an integer r such that $2^{r+1}|(p-1)$ and we take $n = 2^r \geq d$. Then, we can use the FFT optimized version of Algorithm 2 .

- For Step 1 and Step 3, we have to perform operation modulo ϕ_i or ϕ'_j. Since $2^{r+1} \geq 2d$, we can perform the multiplication modulo ϕ_i and ϕ'_j using FFT method (cf. (von zur Gathen and Gerhard, 1999)). Each of them require $2d$ multiplications, $(4d\log(d)+d)$ additions and $(2d\log(d)+2d)$ constant multiplications in $\mathbb{F}_p$.
- Step 2 and Step 4 are computed as explained in subsection 3.2. Their cost are thus equal to $2n\log_2(n)d$ additions and $n\log_2(n)d + nd$ constant multiplications in $\mathbb{F}_p$.

  The resulting complexity is given in Table 2.

Table 2: Complexity of Optimized BR Multiplication.

Step	M	A	CM
1	$4nd$	$2nd(4\log_2(d)+1)$	$4nd(\log_2(d)+1)$
2	-	$2nd\log_2(n)$	$nd(\log_2(n)+1)$
3	$4nd$	$2nd(4\log_2(d)+1)+nd$	$4nd(\log_2(d)+1)+nd$
4	-	$2nd\log_2(n)$	$nd(\log_2(n)+1)$
Tot.	$8nd$	$4nd\log_2(d^4n)+5nd$	$2nd\log_2(d^4n)+11nd$

4.3 Comparison

For practical fields $\mathbb{F}_{p^k)}$, care must be taken for the selection of algorithm parameters n and d. The product of n and d should be close to k (if $nd \gg k$, our algorithm have high complexity).

To simplify, let us assume that $nd = k$. The best choice for n and k to get the best complexity is when $n = 2d$. In this situation we get the Complexities of Table 3. We gave also in this table the complexity of the approaches presented in (Bajar et al., 2006) and (Negre, 2006).

Table 3: Comparison.

	M	A	CM
This paper	$4k^{1.5}$	$6k^{1.5}-5k$	$2k^{1.5}+5k$
BIN	$2k$	$O(k^2\log(k))$	$(4k-1)$
This paper	$8k$	$10k\log_2(k)-k$	$5k\log_2(k)+8k$
Negre	$4k$	$4k\log_2(k)+k$	$2k\log_2(k)+2k$

Comparison of general approaches. Compare to the BIN algorithm, our algorithm requires more multiplications but less additions, both multiplications and additions are in sub-quadratic number. For software implementation with p of size less than a word computer, BIN algorithm is less appropriate to this case than BR multiplication.

Indeed, we denote T_M the time delay of one multiplication in $\mathbb{F}_p$ and we denote T_A the time delay of one addition in $\mathbb{F}_p$. We will assume that T_M is about two times the value of T_A, as this is the case in software environment when p has the size of a computer word. The total time delay of Algorithm 2 for software implementation is $(18k^{1.5}+5k)T_A$. On the other hand the multiplication algorithm of (Bajar et al., 2006) actually has a time delay $(3k^2\log k + 3k^2 + 12k)T_A$.

Table 4: Explicit comparison.

p	k	$n \times d$	algorithm 1 $\#T_A$	BIN(Bajar et al., 2006) $\#T_A$	accelerate
59	29	10×3	2950	7292	59%
67	29	10×3	2950	7292	59%
73	29	10×3	2950	7292	59%
127	23	8×3	2072	4024	48%
257	23	8×3	2072	4024	48%
503	19	8×2	1620	2692	39%
521	19	8×2	1620	2692	39%
8191	13	7×2	933	1227	23%
131071	11	6×2	732	873	16%

In Table 4 we give several fields with cryptographic size and the corresponding time delay required for BIN multiplier and BR multiplier.

Comparison of FFT approaches. For FFT approach we see that our algorithm is slower by a factor between 2 and 4. In the case $2^r > k$, Negre's Algorithm should be preferred to BR-FFT multiplication. But when there are not enough roots of unity, Negre's approach cannot be used, in this case we get benefit of our algorithm.

5 CONCLUSIONS

In this paper, we have proposed a multiplication algorithm for field $\mathbb{F}_{p^k}$. This algorithm extends the previous work done in (Bajar et al., 2006). We study different strategies to implement our algorithm (general approach with Lagrange conversion and optimized approach with Fast Fourier Transform). We get two multipliers : one with sub-quadratic complexity $O(k^{1.5})$ which works in general and the other with complexity $O(k\log_2(k))$ which works only in special situations.

In software platform our general method seems to be better than original BIN (Bajar et al., 2006) algorithm. Compared to the BIN algorithm for several fields of cryptographic size our algorithm should accelerated the multiplication by an average ratio of 30%.

REFERENCES

Bailey, D. V. and Paar, C. (1998). Optimal extension field for fast arithmetic in public key algorithm. In *Advances in Cryptology-CRYPTO'98*, volume 1462 of *LNCS*, pages 472–485. Springer-Verlag.

Bajar, J. C., Imbert, L., and Negre, C. (2006). Arithmetic operation in finite fields of medium prime characteristic using the Lagrange representation.

Halbutoğullari, A. and Ç. K. Koç (2000). Parallel multipliers using polynomial residue arithmetic. *Designs, Codes and Cryptography*, pages 155–173.

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

Lidl, R. and Niederreiter, H. (1994). *Introduction to Finite Fields and Their Applications*. Cambridge Univ. Press.

Lim, C. H. and Hwang, H. S. (2000). Fast implementation of elliptic curve arithmetic in GF(p^n). *Public Key Cryptography*, 1751:405–421.

Miller, V. (1986). Uses of elliptic curve in cryptography. In *Advances in Cryptology, Proc. CRYPTO'8*, pages 417–428.

Montgomery, P. L. (1985). Modular multiplication without trial division. *Mathematics of Computation*, pages 519–521.

Negre, C. (2006). Finite field multiplication in lagrange representation using fast fourier transform. In *International Conference on Security and Cryptography, SECRYPT 2006*.

von zur Gathen, J. and Gerhard, J. (1999). *Modern computer algebra*. Cambridge University Press, New York, NY, USA.