FINITE FIELD MULTIPLICATION IN LAGRANGE REPRESENTATION USING FAST FOURRIER TRANSFORM

Christophe Negre
Équipe DALI, LP2A, Université de Perpignan
avenue P. Alduy, 66 000 Perpignan, France

Keywords: Finite field arithmetic, FFT, multiplication.

Abstract: The multiplication in $\mathbb{F}_{p^n}$ can be performed using a polynomial version of Montgomery multiplication (Montgomery, 1985). In (Bajard et al., 2003) Bajard *et al.* improved this method by using a Lagrange representation: the elements of $\mathbb{F}_{p^n}$ are represented by their values at a fixed set of points. The costly operations in this new algorithm are the two changes of Lagrange representation which require $2r^2$ operations in $\mathbb{F}_p$ with $n \leq r \leq 2^{\lceil \log_2(n) \rceil}$. In this paper we present a new method to perform the change of Lagrange representation. This method uses Fast Fourier Transform and has a cost equal to $3rlog_2(r)$ operations in $\mathbb{F}_p$ with $r = 2^{\lceil \log_2(n) \rceil}$.

INTRODUCTION

Finite field arithmetic is used in many applications, for example in cryptography (Koblitz, 1987; Miller, 1986) and in error correcting code (Berlekamp, 1982). Getting efficient finite field arithmetic is an important challenge to make these applications as fast as possible. Specially in ECC essentially two field operations are required: multiplication and addition, and the multiplication is the most costly. Consequently, many works have been done to get an efficient multiplication in finite field $\mathbb{F}_{2^n}$ (Ning and Yin, 2001; Sunar and Koc, 1999), $\mathbb{F}_p$ and also for $\mathbb{F}_{p^n}$ (Bailey and Paar, 1998).

Our interest here concerns finite field extensions $\mathbb{F}_{p^n}$. Elements in these field can be seen as polynomial $A(X) \in \mathbb{F}_p[X]$ of degree $n-1$. The multiplication consist in a multiplication of polynomial following by a reduction modulo an irreducible polynomial $N(X) \in \mathbb{F}_p[X]$.

A method to perform arithmetic in $\mathbb{F}_{p^n}$ was proposed by Bajard *et al.* in (Bajard et al., 2003). This method is based on a generalization of the polynomial version of the Montgomery Algorithm (Montgomery, 1985). The arithmetic is done modulo *friable* polynomials Ψ and Ψ' with degree $r \geq n$; by *friable* we means they are product of polynomial of degree one. The arithmetic modulo such polynomials is very efficient when the elements are expressed in the Lagrange representation (Bajard et al., 2003) LR_Ψ and $LR_{\Psi'}$.

In this situtation the most costly operation is the change of representation from LR_Ψ to $LR_{\Psi'}$ which consists of a product of an $r \times r$ constant matrix with the vector $LR_\Psi(A)$. This requires roughly r^2 multiplications and r^2 additions. Here we will improve this method for well chosen Ψ and Ψ' and by using Fast Fourier Transform. The change of representation will require $4r\log(r)$ multiplications and $8r\log(r)$ additions .

This article is organized as follows: in the first section we will give some background on Lagrange representation of polynomials. In the section 2 we explain how the Lagrange algorithm (Bajard et al., 2003) of Bajard *et al.* works. The main contribution of the paper is in section 3 in which we present our new method to perform the change of Lagrange representation. At the end we evalutate the complexity of our algorithm and compare it to the original method of Bajard *et al.* and we conclude by a brief conclusion.

1 LAGRANGE REPRESENTATION

The Lagrange representation consists to represent a polynomial by its evalutation at n points. In an arithmetic point of view, this is related to the Chinese Remainder Theorem which asserts that the following application is an isomorphism.

$$\begin{array}{ccc} \mathbb{F}_p[X]/(\Psi) & \to & \mathbb{F}_p[X]/(X-e_1)\times\cdots\times\mathbb{F}_p[X]/(X-e_k) \\ A & \mapsto & (A \bmod (X-e_1),\ldots,A \bmod (X-e_k)), \end{array}$$

We remark that the computation of $A \mod (X - e_i)$ is simply the computation of $A(e_i)$. In other words the image of $A(X)$ by the isomorphism (1) is nothing else that the multi-points evaluation of A at the roots of $\Psi = \prod_{i=1}^{n}(X - e_i)$.

Definition 1 (Lagrange representation) *Let $A \in \mathbb{F}_p[X]$ with $\deg A < n$, and $\Psi = \prod_{i=1}^{r}(X - e_i)$, where $e_i \in \mathbb{F}_p$ for $1 \le i \le r$ and $e_i \ne e_j$ for $i \ne j$. If $a_i = A(e_i)$ for $1 \le i \le r$, the Lagrange representation (LR) of $A(X)$ modulo Ψ is defined by*

$$\mathrm{LR}_\Psi(A(X)) = (a_1, \ldots, a_r). \tag{1}$$

The advantage of the LR representation to perform operations modulo Ψ is a consequence of the Chinese remainder theorem. Specially the arithmetic modulo Ψ in classical polynomial representation can be costly if Ψ 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 $(X - e_i)$. But arithmetic modulo $(X - e_i)$ is the arithmetic modulo p since the product of two degree zero polynomials is just the product modulo p of the two constant coefficients.

2 MONGOMERY MULTIPLICATION IN LANGRANGE REPRESENTATION

Montgomery in (Montgomery, 1985) proposed an algorithm to perform integer modular multiplications. The polynomial version of the Montgomery algorithm can be used to perform polynomial modular arithmetic in $\mathbb{F}_p[X]$. Here we will present only the generalized version of this algorithm.

We consider an irreducible polynomial $N \in \mathbb{F}_p[X]$ of degree n and two polynomials $\Psi, \Psi' \in \mathbb{F}_p[X]$ such that

$$\gcd(\Psi, \Psi') = \gcd(\Psi, N) = \gcd(\Psi', N) = 1,$$

and $\deg \Psi, \deg \Psi' \ge n$. In this situation, the generalized polynomial version of Montgomery Algorithm computes $AB\Psi^{-1} \bmod N$.

The advantage of Montgomery multiplication is to avoid euclidean division to compute AB modulo N. This division is replaced by exactly 5 multiplications

Algorithm 1 Generalized Montgomery Multiplication.

```
Require: $A, B \in \mathbb{F}_p[X]$, with $\deg A, \deg B \le n-1$;
    a monic irreducible polynomial $N \in \mathbb{F}_p[X]$, with
    $\deg N = n$; $\Psi, \Psi'$, with $\deg \Psi = \deg \Psi' = r \ge n$, and $\gcd(\Psi, \Psi') = \gcd(\Psi, N) = 1$
Ensure: $AB\Psi^{-1} \bmod N$
 1: $Q \leftarrow A \times B \times N^{-1} \bmod \Psi$
 2: $R \leftarrow (A \times B + Q \times N) \times \Psi^{-1} \bmod \Psi'$
```

modulo Ψ or Ψ' (2 in step 1 and 3 in step 2). But this makes sense only if the arithmetic operation modulo Ψ and Ψ' can be done efficiently. In their original paper Bajard *et al.* proposed to use Ψ and Ψ' which split totally in $\mathbb{F}_p[X]$, i.e.,

$$\Psi = \prod_{i=1}^{r}(X - e_i) \text{ and } \Psi' = \prod_{i=1}^{r}(X - e'_i).$$

In section 1 we noticed that the arithmetic modulo polynomials Ψ and Ψ' can be done efficiently using Lagrange representation. The use Lagrange representation LR_Ψ and $LR_{\Psi'}$ provides some complications in the Generalized Montgomery Algorithm: between step 1 and step 2 we have to perform some conversions from Lagrange representation of Q relatively to Ψ to Lagrange representation of Q relatively to Ψ'. Similarly after the step 2 we should compute the Lagrange representation of R relatively to Ψ.

If we note $Change\_Rep_{\Psi\to\Psi'}$ the sub-routine which computes the Lagrange representation of an element A relatively to Ψ' from its Lagrange representation relatively to Ψ, the generalized Montgomery's Algorithm 1 becomes

Algorithm 2 LR Modular Multiplication.

```
Require: $A, B \in \mathbb{F}_p[X]$, with $\deg A, \deg B \le k-1$;
    a monic irreducible polynomial $N \in \mathbb{F}_p[X]$, with
    $\deg N = n$; $\Psi, \Psi'$, with $\deg \Psi = \deg \Psi' = r$,
    and $\gcd(\Psi, \Psi') = \gcd(\Psi, N) = 1$
 1: $LR_\Psi(Q) \leftarrow LR_\Psi(A) \times LR_\Psi(B) \times LR_\Psi(N)^{-1}$
 2: $LR_{\Psi'}(Q) \leftarrow Change\_Rep_{\Psi,\Psi'}(LR_\Psi(Q))$
 3: $LR_{\Psi'}(R) \leftarrow (LR_{\Psi'}(A) \times LR_{\Psi'}(B) - LR_{Psi'}(Q) \times LR_{\Psi'}(N)) \times LR_{\Psi'}(\Psi)^{-1}$
 4: $LR_{\Psi'}(R) \leftarrow Change\_Rep_{\Psi',\Psi}(LR_{\Psi'}(R)$
```

In (Bajard et al., 2003) Bajard *et al.* the change of Lagrange representation is done with a product matrix-vector $\Omega \cdot LR_\Psi(Q)$ where the matrix $\Omega = [\omega_{i,j}]_{i,j=1,\ldots,r}$ is a $r \times r$ matrix and has the following coefficients

$$\omega_{i,j} = \prod_{\ell=1}^{r} \frac{e'_i - e_\ell}{e_j - e_\ell}$$

This matrix-vector product is a costly operation: it requires r^2 multiplications and $r(r-1)$ additions.

In this paper we study a different strategy to perform the change of Lagrange representation. Our strategy was to express the change of representation in term of Discrete Fourrier Transform. This becomes interisting when this DFT can be computed with the FFT, since the FFT is really efficient.

3 CHANGE OF LAGRANGE REPRESENTATION WITH FFT

For now, we will take the prime integer p such that $2^{k+1}|p-1$ with $k \geq 1$. In this case there exists an element $\alpha \in \mathbb{F}_p$ such that the 2^{k+1} elements α^i for $i = 0, \ldots, 2^{k+1}-1$ are the 2^k distinct roots of $X^{2^{k+1}}-1$.

Let $r = 2^k$, in this situation we choose the two polynomials Ψ and Ψ' as follows

$$\begin{array}{rcl} \Psi & = & \prod_{i=0}^{r}(X-\alpha^{2i}) = X^r - 1 \\ \text{and} \quad \Psi' & = & \prod_{i=0}^{r}(X-\alpha^{2i+1}) = X^r + 1. \end{array}$$

We are going to express the change between two Lagrange representations LR_Ψ and $LR_{\Psi'}$ in term of to Discrete Fourier Transform. But before we recall some basic fact on DFT and FFT.

3.1 Background On Dft and FFT

Let $\beta = \alpha^2$, the evaluation of a polynomial $A(X) \in \mathbb{F}_p[X]$ at the elements β^i for $i = 0, \ldots, r$ corresponds to the Discrete Fourier Transformation of A

$$\begin{array}{rcl} DFT_r(A) & = & (\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_r) \\ & = & (A(1), A(\beta), A(\beta^2), \ldots, A(\beta^{r-1})) \end{array}$$

The DFT_r of A can be computed by using the FFT_r Algorithm. This algorithm is based on the following identities

$$\begin{array}{rcl} \hat{a}_i & = & A_e((\beta^2)^i) + \beta^i A_o((\beta^2)^i) \\ \hat{a}_{i+(r/2)} & = & A_e((\beta^2)^i) - \beta^i A_o((\beta^2)^i), \end{array} \qquad (2)$$

where the polynomial $A_e(X)$ is the even part of A and $A_o(X)$ is the odd part of A

$$A_e(X) = \sum_{i=0}^{r/2} a_{2i}X^i, \quad A_o(X) = \sum_{i=0}^{r/2} a_{2i+1}X^i.$$

The FFT_r recursively computes $FFT_{r/2}(A_e)$ and $FFT_{r/2}(A_o)$ and then deduces $FFT_r(A) = (A(\beta^0)A(\beta^2), \ldots, A(\beta^{(r-1)}))$ using equations (2). For a complete description of the FFT algorithm see (von zur Gathen and Gerhard, 1999)

In our situation the FFT is a powerful algorithm which enables us

- to compute the Lagrange representation relatively to Ψ of a polynomial $A(X)$.
- to compute the polynomial from its LR representations relatively to Ψ since the reverse operation of the FFT is $FFT_r^{-1} = \frac{1}{r}FFT_r$.

3.2 The $Change\_Rep$ Routines

Let us see how use the FFT in the change of representation between LR_Ψ and $LR_{\Psi'}$. We know only how to reconstruct a polynomial $A(X)$ from its LR_Ψ representation

$$A(X) = FFT_r^{-1}(LR_\Psi(A)).$$

But if now we compute $\widetilde{A}(X) = A(\alpha X)$ and if after that we compute $FFT_r(\widetilde{A})$, we obtain the $r = 2^k$ elements

$$\widetilde{A}(\beta^i) = A(\alpha\alpha^{2i}) = A(\alpha^{2i+1}),$$

for $i = 0, \ldots, r$. This means that $LR_{\Psi'}(A) = FFT_r(\widetilde{A}))$. Consequently the change of Lagrange representation $Change\_Rep_{\Psi\to\Psi'}$ can be done with the following algorithm.

Algorithm 3 $Change\_Rep_{\Psi\to\Psi'}$.

```
1: $A(X) \leftarrow FFT^{-1}(LR_\Psi(A))$
2: $\widetilde{A}(X) \leftarrow A(\alpha X)$
3: $LR_{\Psi'}(A) \leftarrow FFT(\widetilde{A})$
```

The reverse of the change of representation is done using the reverse process. First we get back to $\widetilde{A}(X)$ by computing $FFT_r^{-1}(LR_{\Psi'}(A)$, then we compute to $A(X)$ with $A(X) = \widetilde{A}(\alpha^{-1}X)$, and finally we get $LR_\Psi(A)$ by computing $LR_\Psi(A) = FFT_r(A(X))$.

Algorithm 4 $Change\_Rep_{\Psi'\to\Psi}$.

```
1: $\widetilde{A}(X) \leftarrow FFT^{-1}(LR_{\Psi'}(A))$
2: $A(X) \leftarrow \widetilde{A}(\alpha^{-1}X)$
3: $LR_\Psi(A) \leftarrow FFT(A(X))$
```

It is natural to wonder if it is possible to merge the three step of the Algorithm 3 and 4 in a FFT-like recursive Algorithm since the main computation are done with FFT_r and FFT_r^{-1}. Specially the computation of the polynomial representation of $A(X)$ and $\widetilde{A}(X)$ seems to be superfluous, it thus could be avoided. For now we could not obtain such recursive algorithm, but it will be interesting in the future to get such recursive method to improve the performance of the Algorithm 3 and 4.

4 COMPLEXITY COMPARISON

In this section we evalutate the complexity of the modified form of Lagrange multiplication of Bajard *et al.*. First we focus on the theoretic complexity by evaluating the number of field operations $\mathbb{F}_p$ (additions and multiplications). In the table 4 we give the cost of each step of the Algorithm 2 used with $Change\_Rep$ of section 3. For an explicit evaluation of the cost of the FFT we refer to (von zur Gathen and Gerhard, 1999).

	Multiplication	Additions
Step 1	2r	0
Step 2	$rlog_2(r)$	$2rlog_2(r)$
Step 3	$3r$	r
Step 4	$(2rlog_2(r)+r-1)$	$4rlog_2(r)$

The global cost of the algorithm is thus equal to $(4rlog_2(r)+7r-2)$ multiplications and $(8rlog_2(r)+r)$ additions. We get a clearly improvement compared to original LR multiplication, since the complexity was equal to $(2r(r-1)+r)$ addtions and $2r^2+5r$ multiplications in $\mathbb{F}_p$.

Hardware implementation. For hardware implementations, the FFT have the nice property to be parallelizable. Specially, we can compute $FFT_r(A)$ with $r/2$ multipliers and r adders in parallel. The delay of the architecture to perform one FFT_r is then equal to $\log_2(r)T_M+\log_2(r)T_A$. For a precise explanation of this fact we refer to (Johnson et al., 2000) .

But the other computations in the Algorithm 2 require also at most only r multipliers in parallel and r adders in parallel. This is clear for step 1 and step 3, for step 2 and 4, we need r adders and multipliers for the FFT parts, and also r multipliers for the computation of $\widetilde{Q}$ and $R(X)$. We can use at each time the same r adders and multipliers.

Consequently the Algorithm 2 can be implemented in hardware with an architecture using r multipliers and r adders in parallel.

Let us evaluate the delay of such architecture. The total delay is equal to the sum of the delay of each step of Algorithm 2. If we note T_M the time for a multiplication in $\mathbb{F}_p$, the step 1 has a delay of $2T_M$ and the step 2 has a delay of $3T_M+T_A$. In step 2 and 4 the delay is equal to the delay of two FFT_r plus the delay of two multiplications ,i.e. (one T_M for the multiplications by r^{-1} and a second for the computation of $\widetilde{A}$), each step has a delay of $(\log_2(r)+2)T_M+(\log_2(r)T_A$.

Finally, the global delay is equal to $(\log_2(r)+5)T_M+(\log_2(r)+1)T_A$.

5 CONCLUSION

In this paper we have presented a modified version of the Algorithm of Bajard *et al.* (Bajard et al., 2003) computes the product of two elements of $\mathbb{F}_{p^n}$. We have modified only a part of the algorihtm: precisely we modified the changes of representation in a way to use FFT algorithm. We thus obtain an algorithm which have a sub-quadratic complexity: a multiplications requires $(4rlog_2(r)+7r-2)$ multiplications and $(8rlog_2(r)+r)$ additions in $\mathbb{F}_p$ instead of $(2r(r-1)+r)$ addtions and $2r^2+5r$ multiplications in the original work of Bajard *et al.* (Bajard et al., 2003).

We are greatefull to N. Louvet for helpfull comments on a preliminary version of this paper.

REFERENCES

Bailey, D. and Paar, C. (1998). Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms. *Lecture Notes in Computer Science*, 1462:472.

Bajard, J.-C., Imbert, L., Negre, C., and Plantard, T. (2003). Efficient Multiplication in GF(p^k) for Elliptic Curve Cryptography. In *ARITH 16, 16th IEEE Symposium on Computer Arithmetic June 15-18, 2003 Santiago de Compostela, SPAIN.*

Berlekamp, E. (1982). Bit-serial Reed-Solomon encoder. *IEEE Transaction on Information Theory*, IT-28(6).

Johnson, J., Kumhom, P., and Nagvajara, P. (2000). Design, optimization, and implementation of a universal fft processor. In *13th IEEE International ASIC/SOC Conference, Washington, DC.*

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

Miller, V. (1986). Use of elliptic curves in cryptography. *Advances in Cryptology, proceeding's of CRYPTO'85*, 218:417–426.

Montgomery, P. (1985). Modular multiplication without trial division. *Mathematic of computation*, 44(170).

Ning, P. and Yin, Y. (2001). Efficient Software Implementation for Finite Field Multiplication in Normal Basis. *Lecture Notes in Computer Science*, 2229:177.

Sunar, B. and Koc, C. (1999). Mastrovito Multiplier for All Trinomials. *IEEE Transaction on Computers*.

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