Subquadratic Binary Field Multiplier in Double Polynomial System

Pascal Giorgi, Christophe Negre, Thomas Plantard

Subquadratic Binary Field Multiplier in Double Polynomial System

Pascal Giorgi[1], Christophe Nègre[2]
Équipe DALI, LP2A, Unversité de Perpignan
avenue P. Alduy, F66860 Perpignan, France
(1) pascal.giorgi@univ-perp.fr, (2) christophe.negre@univ-perp.fr

Thomas Plantard
Centre for Computer and Information Security Research
School of Computer Science & Software Engineering
University of Wollongong, Australia
thomaspl@uow.edu.au

Keywords: Binary Field Multiplication, Subquadratic Complexity, Double Polynomial System, Lagrange Representation, FFT, Montgomery reduction, .

Abstract: We propose a new space efficient operator to multiply elements lying in a binary field $\mathbb{F}_{2^k}$. Our approach is based on a novel system of representation called *Double Polynomial System* which set elements as a bivariate polynomials over $\mathbb{F}_2$. Thanks to this system of representation, we are able to use a Lagrange representation of the polynomials and then get a logarithmic time multiplier with a space complexity of $O(k^{1.31})$ improving previous best known method.

1 Introduction

Efficient hardware implementation of finite field arithmetic, and specifically of binary field $\mathbb{F}_{2^k}$, is often required in cryptography and in coding theory (Berlekamp, 1982). For example in elliptic curve cryptosystem (Koblitz, 1987; Miller, 1986), the main operation is the scalar multiplication on the curve, which necessitates thousands of multiplications and additions over a finite field. Similarly, hundreds of multiplications over a binary field are required for the Diffie-Hellman Key exchange protocol (Diffie and Hellman, 1976).

Previously to this work, several architectures have already been proposed to efficiently implement the arithmetic in $\mathbb{F}_{2^k}$. These architectures are mostly dedicated to the multiplication since this operation is extensively used and is often the most expensive. Each of them takes advantage of a special representation of the field. In particular, one of them uses polynomial basis or shifted polynomial basis (Mastrovito, 1991; Koc and Sunar, 1999; Fan and Dai, 2005) while another uses normal basis (Gao, 1993; Koc and Sunar, 1998). The latter providing a really efficient squaring in the field since in this basis the squaring is just a cyclic shift of the coefficients.

In these representations the main approach to perform the multiplication consists to express the operation as a matrix-vector product with binary entries. Parallel architectures are thus capable to perform this product within logarithmic time. However, these architectures still achieve a space complexity of k^2. According to the recent improvements proposed in (Fan and Hasan, 2007), one can still perform the matrix-vector product in logarithmic time but with a space complexity of $k^{1.56}$ or $k^{1.63}$. This has been made possible thanks to structured matrices such as Toeplitz ones and a divide-and-conquer approach for the products.

In this paper we propose a new approach which reduces the exponent in the space complexity to 1.31 while keeping a logarithmic time complexity. First, we introduce a novel system of representation, the Double Polynomial System. In this representation, elements of $\mathbb{F}_{2^k}$ are polynomials in two variables $A(t,Y) = \sum_{i=0}^{n-1} a_i(t)Y^i$ where $a_i(t)$ have degree strictly less than r.

Therefore, as in classical polynomial representation, the multiplication can be performed in two steps : a polynomial multiplication, and then a reduction phase to reduce the degrees in Y and in t.

The reduction in Y is simple due to the definition of DPS. The same is not true for the reduction in t. Here, we use a Montgomery-like reduction approach in order to perform this reduction with few polynomial multiplications, this enabling us to easily use

the Fast Fourier Transform. Therefore, our multiplier fully benefits from the FFT process which is highly parallelizable and provides a subquadratic space complexity.

Hence, we propose a binary field multiplier which has a delay of $(16\log_3(k)+20)T_X + 8T_A$ and a space complexity of $O(k^{1.31})$, where T_X and T_A correspond respectively to the delay of one XOR gate and one AND gate.

Let us briefly give the outline of the paper. We first introduce the DPS representation for binary fields $\mathbb{F}_{2^k}$ (Section 2). We present the DPS multiplication in Section 3 and discuss the problem of finding a suitable polynomial to achieve our Montgomery-like coefficient reduction in Section 4. Then, we present in Section 5 a modified version of our multiplication introducing Lagrange basis. We recall in Section 6 some basic facts on the architecture design of a ternary FFT on which we rely for our multiplier. We finally conclude this paper by a detailed explanation of the complete architecture for our DPS-Lagrange multiplier and its complexity analysis and comparison (Section 7).

2 DPS Representation

A binary field $\mathbb{F}_{2^k}$ is generally constructed as the set of polynomials modulo an irreducible polynomial $P \in \mathbb{F}_2[t]$ of degree k

$$\begin{aligned}\mathbb{F}_{2^k} &= \mathbb{F}_2[t]/(P(t)) \\ &= \{A(t) \in \mathbb{F}_2[t] \text{ s.t. } \deg A(t) < k\}\end{aligned}$$

We introduce a novel binary field representation, the Double Polynomial System (DPS), inspired from AMNS number system of Bajard *et al.* (J.-C. Bajard, 2005).

Definition 1 (DPS representation). *A Double Polynomial System (DPS) is a quintuplet $\mathcal{B} = (P, \gamma, n, r, \lambda)$ such that*

- *$P(t) \in \mathbb{F}_2[t]$ is an irreducible polynomial of degree k,*
- *$\gamma(t), \lambda(t) \in \mathbb{F}_2[t]/(P(t)))$ satisfy*

$$\gamma(t)^n \equiv \lambda(t) \bmod P,$$

and $\lambda(t)$ has a low degree in t.

A DPS representation of an element $A(t) \in \mathbb{F}_2[t]/(P)$ is a polynomial $A_{\mathcal{B}}(t,Y) \in \mathbb{F}_2[t,Y]$ such that

$$A_{\mathcal{B}}(t,Y) = \sum_{i=0}^{n-1} a_i(t) Y^i \text{ with } \deg_t a_i(t) < r$$

and $A_{\mathcal{B}}(t,\gamma(t)) \equiv A(t) \bmod P$

In the sequel we will often omit the subscript $\mathcal{B}$ to denote the DPS form of an element A. In some cases, when it is clear from the context, we may discard the variables t, Y to define the DPS representation of an element. We will also denote by E the polynomial $E = Y^n - \lambda$.

Example 1. Let us consider the field $\mathbb{F}_{2^4}$, then the quintuplet $\mathcal{B} = (P = t^4+t^3+t^2+t+1, \gamma = t^3+t^2+t, n = 3, r = 2, \lambda = t)$ is a DPS for this field. We can check this with Table 1 which gives the DPS expression of each element in $\mathbb{F}_{2^4}$.

TAB. 1: Elements of $\mathbb{F}_{2^4}$ in $\mathcal{B}$.

$A(t)$	0	t^2	t^3+t^2+t+1	t^3+t
$A_{\mathcal{B}}$	0	$(t+1)Y^2$	$Y+1$	Y^2+t+1

$A(t)$	1	t^2+1	t^3+t^2+t	t^3+t+1
$A_{\mathcal{B}}$	1	$(t+1)Y^2+1$	Y	Y^2+t

$A(t)$	t	t^2+t	t^3+t^2	t^3+1
$A_{\mathcal{B}}$	t	Y^2+Y+1	$Y+t$	Y^2

$A(t)$	$t+1$	t^2+t+1	t^3+t^2+1	t^3
$A_{\mathcal{B}}$	$t+1$	Y^2+Y	$Y+t+1$	Y^2+1

In particular, we can verify that if we evaluate $(t+1)Y^2+1$ in γ, we get $(t+1)\gamma^2+1 = (t+1)(t^3+t^2+t)^2+1 = t^2+1 \mod P$, as expected. One can also see that $\deg_Y((t+1)Y^2+1) = 2 < 3 = n$ and $\deg_t((t+1)Y^2+1) = 1 < 2 = r$. ◊

Remark 1. The DPS can be seen as a generalization of the polynomial representation of double extensions $\mathbb{F}_{2^{rn}}$. Such extensions are usually constructed first as $\mathbb{F}_{2^r} = \mathbb{F}_2[t]/(P(t))$ and then as $\mathbb{F}_{2^{rn}} = \mathbb{F}_{2^r}[Y]/(Y^n - \lambda))$ with $\lambda \in \mathbb{F}_{2^r}$, see (Guajardjo and Paar, 1997). However, this construction is not possible when the degree k of the field $\mathbb{F}_{2^k}$ is prime. DPS provides an alternative for double extension in this situation.

Remark 2. As in classical polynomial representation, the addition in DPS is just a parallel bitwise XOR on the coefficients.

We proceed now by considering the problem of the multiplication of two elements expressed in a DPS. This can be done in two steps as described in Algorithm 1.

The first step of the algorithm consists of a classical polynomial multiplication modulo the binomial $E(Y) = (Y^n - \lambda)$. The resulting polynomial $C(t,Y)$ satisfies $C(t,\gamma) = A(t,\gamma)B(t,\gamma) \bmod P(t)$ since $E(\gamma) \equiv 0 \bmod P(t)$ by definition of the DPS.

The second step computes an element $R(t,Y)$ such that it becomes a valid DPS representation of $A \times B$:

$$R(t,\gamma) = A(t,\gamma)B(t,\gamma) \bmod P(t) \text{ and } \deg_t(R) < r.$$

Algorithm 1 DPS multiplication scheme.

Input : $A, B \in \mathcal{B} = (P, \gamma, n, r, \lambda)$
Output : $C = A \times B \in \mathcal{B}$
1. Polynomial multiplication in Y :
   $C = AB \bmod (Y^n - \lambda)$.
2. Coefficients reduction :
   $R = RedCoeff(C)$.

It is clear from the DPS system and from the multiplication modulo a binomial $Y^n - \lambda$ that C has coefficients $c_i(t)$ with degree in t bounded by $2(r-2) + \deg_t \lambda$. Therefore, these coefficients must be reduced to get the result of the multiplication expressed in the DPS representation.

3 Multiplication in DPS

A straightforward method for the reduction phase in t of Algorithm 1 is to perform an Euclidean division $C = Q \times M + R$ where $\deg_t R < r$. This reduction is only valid if $M(t, Y)$ is monic in t and satisfies

$$M(t, \gamma) \equiv 0 \bmod P(t) \text{ with } \deg_t(M) = r. \quad (1)$$

Generally, one can easily compute a polynomial M satisfying equation (1), e.g. Section 4, but ensuring monicity is difficult.

In order to avoid monicity attached to a division strategy, we adapt the Montgomery trick (Montgomery, 1985) to our DPS system. The idea is to replace the Euclidean division by few multiplications and one exact division. This corresponds to annihilating the lower part of the $c_i(t)$ instead of the higher ones.

This method is given in Algorithm 2 assuming a polynomial $M(t, Y)$ satisfying $M(t, \gamma) \equiv 0 \bmod P(t)$ is given.

Algorithm 2 DPS Multiplication.

Input : $A, B \in \mathcal{B} = (P, \gamma, n, r, \lambda)$ with $E = Y^n - \lambda$
Data : M such that $M(\gamma) \equiv 0 \bmod P$, a polynomial $m \in \mathbb{F}_2[t]$ and $M' = -M^{-1} \bmod (E, m)$
Output: R such that $R(t, \gamma) = A(t, \gamma)B(t, \gamma)m^{-1} \bmod P$
begin
  $C \leftarrow A \times B \bmod E$;
  $Q \leftarrow C \times M' \bmod (E, m)$;
  $R \leftarrow (C + Q \times M \bmod E)/m$;
end

Example 2. We consider the field $\mathbb{F}_{2^4}$, with the DPS $\mathcal{B} = (P = t^4 + t^3 + t^2 + t + 1, \gamma = t^3 + t^2 + t, n = 3, r = 2, \lambda = t)$. In Table 2, we give an example of trace of DPS multiplication.

TAB. 2: DPS multiplication trace.

Operations	*Results*
A	$tY^2 + tY$
B	$(t+1)Y + t$
M	$tY^2 + Y + t + 1$
M'	$(1+t)Y^2 + (1+t)Y + 1$
m	t^2
C	$tY^2 + t^2Y + t^3 + t$
Q	tY^2
$Q \times M$	$(t^2 + t)Y^2 + t^3Y + t^2$
$C + Q \times M$	$t^2Y^2 + (t^3 + t^2)Y + t^3$
R	$Y^2 + (t+1)Y + t$

We can check that $R(t, \gamma) \equiv t^2 + t \mod P$ is equal to $A(t, \gamma)B(t, \gamma)t^{-2} \mod P$. ◊

Lemma 1. *Algorithm 2 is correct.*

Proof. We need to demonstrate that the output R of the algorithm satisfies the following equation

$$R(t, \gamma) = A(t, \gamma)B(t, \gamma)m^{-1} \bmod P. \quad (2)$$

From the definition 1 of DPS representation, we know that $E(\gamma) \equiv 0 \bmod P$. Thus, we have

$$C(t, \gamma) \equiv A(t, \gamma)B(t, \gamma) \bmod P.$$

By definition of M, we have $M(t, \gamma) \equiv 0 \bmod P$ and consequently

$$\begin{aligned} C(t, \gamma) + Q(t, \gamma)M(t, \gamma) &\equiv C(t, \gamma) \\ &\equiv A(t, \gamma)B(t, \gamma) \bmod P \end{aligned}$$

We now need to prove that the division by m is exact. This is equivalent to prove the following equivalence $C + Q \times M \bmod E \equiv 0 \bmod m$. By definition, we have $Q = C \times M' \bmod E$ and $M' = -M^{-1} \bmod (E, m)$. We consider $R' = C + Q \times M \bmod (E, m)$, then the following equivalences hold

$$\begin{aligned} R' &\equiv C + C \times (-M^{-1} \times M) \bmod (E, m) \\ &\equiv (C - C) \bmod (E, m) \\ &\equiv 0 \bmod (E, m). \end{aligned}$$

Thus, division by m is exact. Hence, the algorithm is correct since an exact division (the division by m) is equal to the multiplication by an inverse modulo P. □

At this level, we know that the resulting polynomial R of the previous algorithm satisfies the equation $R(t,\gamma) = A(t,\gamma)B(t,\gamma)m^{-1} \bmod P$ but we do not know whether it is expressed in the DPS, i.e., if the coefficients of R have degree in t smaller than r. This is the goal of the following theorem.

Theorem 1. *Let $\mathcal{B} = (P,\gamma,n,r,\lambda)$ a Double Polynomial System, M be a polynomial of $\mathcal{B}$ such that $M(\gamma) \equiv 0 \bmod P$ and $\sigma = deg_t(M)$. Let A,B be two elements expressed in the DPS $\mathcal{B}$. If r and the polynomial m satisfy*

$$r > \sigma + \deg_t(\lambda) \quad \textit{and} \quad \deg_t(m) > \deg_t(\lambda) + r \quad (3)$$

then the polynomial R output by the Algorithm 2 is expressed in the DPS $\mathcal{B}$.

Proof. From the Definition 1, the polynomial R belongs to the DPS $\mathcal{B} = (P,\gamma,n,r,\lambda)$ if $\deg_Y R < n$ and if $\deg_t(R) < r$. The fact that $\deg_Y R < n$ is easy to see since all the computation in the Algorithm 2 are done modulo $E = Y^n - \lambda$.

Hence, we have only to prove that $\deg_t R < r$. Since by definition $\deg_t A, \deg_t B < r$ we have the following inequalities

$$\begin{aligned} \deg_t R &= \deg_t((A \times B + Q \times M) \bmod E)/m) \\ &\leq \max(\deg_t A + \deg_t B, \deg_t Q + \deg_t M) \\ &\quad + \deg_t \lambda - \deg_t m \\ &\leq \max(2r, \sigma + \deg_t m) + \deg_t \lambda - \deg_t m. \end{aligned}$$

According to our hypothesis in the equation (3), we have both $2r + \deg_t \lambda - \deg_t m < r$ and $\sigma + \deg_t m + \deg_t \lambda - \deg_t m < r$. Hence, we get $\deg_t(R) < r$ as required. □

4 Construction of the polynomial M

The result of this section uses mathematical structures involving module over the polynomial ring $\mathbb{F}_2[t]$ in order to prove existence of a suitable polynomial M. The remaining of the paper is independent from this section and readers who are not familiar with such mathematical structure can skip this section without misunderstanding.

Our goal is to construct a polynomial M such that $M(t,\gamma) \equiv 0 \bmod P$ and $\deg_t M$ is small. This polynomial belongs to the set

$$\mathcal{M} = \{A(t,Y)) \in \mathbb{F}_2[t,Y] \text{ with } \deg_Y A < n\}.$$

The set $\mathcal{M}$ has a natural structure of $\mathbb{F}_2[t]$ module. Recall that an $\mathbb{F}_2[t]$-module $\mathcal{M}$ is an (additive) abelian group, with a scalar multiplication over $\mathbb{F}_2[t]$:

$$\mathbb{F}_2[t] \times \mathcal{M} \to \mathcal{M}.$$

In order to calculate the element M with low degree in t, we will use a sub-module $\mathcal{M}'$ of $\mathcal{M}$ spanned by the following linearly independent vectors.

$$\Omega = \begin{pmatrix} P & 0 & 0 & \dots & 0 \\ -\gamma & 1 & 0 & \dots & 0 \\ -\gamma^2 & 0 & 1 & \dots & 0 \\ \vdots & & & \ddots & \vdots \\ -\gamma^{n-1} & 0 & 0 & \dots & 1 \end{pmatrix} \begin{matrix} \leftarrow P \\ \leftarrow Y - \gamma \\ \leftarrow Y^2 - \gamma^2 \\ \vdots \\ \leftarrow Y^{n-1} - \gamma^{n-1} \end{matrix}$$

Each of the polynomials $V(t,Y)$ defined by the rows of Ω satisfy $V(t,\gamma) \equiv 0$, and any $\mathbb{F}_2[t]$-linear combination of these polynomials satisfies also this property. Therefore, one way to construct M consists to compute a *minimal basis* of $\mathcal{M}'$ and define M as the basis element with the smaller degree in t. The notion of minimality is related to the degree in t of the basis elements.

According to polynomial matrix properties, one can find a minimal basis of Ω by computing its matrix reduced form called the Popov form (Mulders and Storjohann, 2003). In particular, the properties of the Popov form (Villard, 1996, §1.2) tell us that it exist a minimal basis $(f_1, f_2, \dots, f_n)$ of $\mathcal{M}'$ which satisfies the following degree properties :

$$\sum_{i=1}^{n} \deg_t f_i = \deg_t(\det(\Omega)) \quad (4)$$

$$\deg_t f_1 \leq \deg_t f_2 \leq \dots \leq \deg_t f_n \quad (5)$$

If we set $M = f_1$ then the degree in t of M is minimal and satisfies the degree bound

$$\deg_t M \leq (\deg_t P)/n \quad (6)$$

Indeed, according to equations (4) and (5), we have $n \times \deg_t M < \sum_{i=1}^{n} \deg_t f_i$ and since $\det(\Omega) = P(t)$ we get the announced bound.

Beside the fact that the calculation of M is only needed once at the construction of the DPS representation, one would need to efficiently compute such polynomial. This can be achieve within a complexity of $O(n^3k^2)$ binary operations with Algorithm *WeakPopovForm* of (Mulders and Storjohann, 2003) or with an asymptotic complexity of $O(n^3 k \log k)$ binary operations with Algorithm *ColumnReduction* of (Giorgi et al., 2003).

5 DPS multiplication in Lagrange representation

In this section, we present a version of Algorithm 2 using a Lagrange representation of the DPS elements.

5.1 Lagrange representation

Let $\mathcal{R}$ a ring, and $\mathcal{R}[Y]$ the polynomial ring over $\mathcal{R}$. The Lagrange representation of a polynomial of degree $n-1$ in $\mathcal{R}[Y]$ is given by its values at n distinct points. For us, these n points will be the roots of a polynomial $E = \prod_{i=1}^{n}(Y-\alpha_i) \in \mathcal{R}[Y]$. From an arithmetic point of view, this is related to the Chinese Remainder Theorem which asserts that the following application is an isomorphism

$$\begin{aligned} \mathcal{R}[Y]/(E(Y)) &\xrightarrow{\sim} \prod_{i=1}^{n}\mathcal{R}[Y]/(Y-\alpha_i) \qquad (7)\\ A &\longmapsto (A \bmod (Y-\alpha_i))_{i\in\{1,\ldots,n\}}. \end{aligned}$$

The computation of $A \bmod (Y-\alpha_i)$ is simply the computation of $A(\alpha_i)$. In other words, the image of $A(Y)$ by the isomorphism (7) is nothing else than the multi-points evaluation of A at the roots of E. This fact motivates the following Lagrange representation of the polynomials.

Definition 2 (Lagrange representation). *Let $A \in \mathcal{R}[Y]$ with $\deg A < n$, and $\alpha_1,\ldots,\alpha_n$ be the n distinct roots of a polynomial $E(Y)$.*

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

If $a_i = A(\alpha_i)$ for $1 \le i \le n$, the Lagrange representation (LR) of $A(Y)$ is defined by $\mathrm{LR}(A(Y)) = (a_1,\ldots,a_n)$.

Lagrange representation is advantageous to perform operations modulo E : this is a consequence of the Chinese Remainder Theorem. Specifically 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 does arithmetic modulo a very simple polynomial $(X-\alpha_i)$. Furthermore, arithmetic modulo $(X-\alpha_i)$ is the arithmetic in $\mathcal{R}$ since the product of two zero degree polynomials is just the product of the two constant coefficients.

5.2 Multiplication algorithm

Let us go back to the Algorithm 2 and see how to use Lagrange representation to perform polynomial arithmetic in each step. The first two steps can be done in Lagrange representation modulo $m_1(t)$ such that E split modulo $m_1(t)$:

$$E = \prod_{i=1}^{n}(Y-\alpha_i) \bmod m_1(t),$$

The third step must be done modulo a second polynomial $m_2(t)$, which also splits E,

$$E = \prod_{i=1}^{n}(Y-\alpha'_i) \bmod m_2(t),$$

since the division by m_1 cannot be performed modulo the polynomial $m_1(t)$.

We then need to represent the polynomials A and B in Algorithm 2 with both their Lagrange representations modulo $m_1(t)$ and $m_2(t)$.

Notation 1. We will use in the sequel the following notation. For a polynomial A of degree $n-1$ in Y we will denote

- $\overline{A}$ the Lagrange representation in α_i modulo $m_1(t)$
- $\overline{\overline{A}}$ the Lagrange representation in α'_i modulo $m_2(t)$.

Hence, we can do the following modifications to the Algorithm 2 :

Algorithm 3 DPS-LR Multiplication.

Input : $\overline{A},\overline{\overline{A}},\overline{B},\overline{\overline{B}}$

Data : $\overline{\overline{M}}$ such that $M(t,\gamma) \equiv 0 \bmod P$, $\overline{M'}$ such that $M' = -M^{-1} \pmod{E, m_1}$.

Output: $\overline{R},\overline{\overline{R}}$ such that $R \in \mathcal{B}$ and $R(t,\gamma) = A(t,\gamma)B(t,\gamma)m_1^{-1} \bmod P(t)$

begin

$\overline{Q} \leftarrow \overline{A} \times \overline{B} \times \overline{M'}$;
$\overline{\overline{Q}} \leftarrow Convert_{m_1 \to m_2}(\overline{Q})$;
$\overline{\overline{R}} \leftarrow (\overline{\overline{A}} \times \overline{\overline{B}}) + \overline{\overline{Q}} \times \overline{\overline{M}}) \times m_1^{-1}$;
$\overline{R} \leftarrow Convert_{m_2 \to m_1}(\overline{\overline{R}})$;

end

The operations to compute $\overline{Q}$ and $\overline{\overline{R}}$ are performed in Lagrange representation and then can be easily parallelized. It consists of n independent multiplications in $\mathbb{F}_2[t]/(m_1(t))$ and $\mathbb{F}_2[t]/(m_2(t))$.

The major drawback of this algorithm is the conversions between Lagrange representations modulo m_1 and m_2. It is necessary to perform these operations efficiently in order to get a multiplier yielding our announced space complexity.

5.3 Lagrange representations conversion

In order to provide an efficient implementation of conversions between Lagrange representations modulo m_1 and m_2, we rely on the binomial form of

$E = Y^n - \lambda$. Indeed, if $\mu_1 = \alpha_1$ is a root of E modulo m_1 then all others roots can be written

$$\alpha_j = \mu_1 \omega_1^i \bmod m_1$$

where ω_1 is a n-th primitive root of unity in $\mathbb{F}_2[t]/(m_1)$. This property comes from the fact that $(\alpha_j/\mu_1)^n = 1 \bmod m_1$ and thus there exists an integer i such that $\alpha_j/\mu_1 = \omega_1^i \bmod m_1$. This is still true modulo m_2.

Thus, the multi-point evaluation of the polynomial $A(Y)$ in α_i modulo m_1 can be done as follow :

1. set $\widetilde{A}(Y) = A(\mu_1^{-1}Y) = \sum_{i=0}^{n-1} a_i \mu_1^{-i} Y^i$
2. compute $\overline{A} = DFT_{m_1}(\widetilde{A}, n, \omega_1)$,

where $DFT_{m_1}(\widetilde{A}, n, \omega_1)$ is the evaluations of the polynomial $\widetilde{A}$ in the n-th roots of unity ω_1^i.

Similarly the Lagrange interpolation which compute $A(Y)$ from $\overline{A}$ can be done by reversing the previous process.

By gluing together this two processes we get the following algorithm to perform conversion between Lagrange representations.

Algorithm 4 *Convert*$_{m_1 \to m_2}$.

Input : $\overline{A}$

Output: $\overline{\overline{A}}$

$\widetilde{A}(Y) \leftarrow DFT_{m_1}^{-1}(\overline{A}, n, \omega_1)$;
$A(Y) \leftarrow \widetilde{A}(\mu_1^{-1}Y) \bmod m_1$;
$\widetilde{\overline{A}}(Y) \leftarrow A(\mu_2 Y) \bmod m_2$;
$\overline{\overline{A}} \leftarrow DFT_{m_2}(\widetilde{\overline{A}}(Y), n, \omega_2)$;

As a consequence, the conversion has a the cost of two Discrete Fourier Transforms. This can be done efficiently by using FFT algorithm (Gathen and Gerhard, 1999, §8.2). Moreover, such FFT computations can be done even faster when special moduli $m_i(t)$ such that $m_i(t) = t^{2\times 3^s} + t^{3^s} + 1$ are used (Schönhage, 1977). These moduli can be seen as a polynomial analog of Fermat numbers.

6 Architecture for FFT computation

We present an architecture to perform the FFT calculation of a polynomial $A(Y) \in \mathcal{R}[Y]$ of degree $n-1$, keeping in mind our targeted Lagrange conversion algorithm. We consider the ring $\mathcal{R} = \mathbb{F}_2[t]/(m(t))$ where $m(t) = t^{2n/3} + t^{n/3} + 1$ and $n = 3^s$. Note that the FFT process needs to be performed using ternary method since binary one is not feasible over characteristic 2 rings (Schönhage, 1977).

Let us denote ω a primitive n-th root of unity modulo $m(t)$ and $\theta = \omega^{n/3}$ a 3rd root of unity. The ternary FFT process is based on the following three-way splitting of A

$$\begin{aligned} A_1 &= \textstyle\sum_{j=0}^{n/3-1} a_{3j} Y^{3j}, \\ A_2 &= \textstyle\sum_{j=0}^{n/3-1} a_{3j+1} Y^{3j}, \\ A_3 &= \textstyle\sum_{i=0}^{n/3-1} a_{3j+2} Y^{3j}, \end{aligned}$$

such that $A = A_1 + YA_2 + Y^2 A_3$.

Let $\widehat{A}[i] = A(\omega^i)$ be the i-th coefficient of $DFT_m(A, n, \omega)$. Let us also denote by $\hat{A}_1[i], \hat{A}_2[i]$ and $\hat{A}_3[i]$ the coefficients of the DFT of order $n/3$ of respectively A_1, A_2 and A_3.

The following relations can be obtained by evaluating $A = A_1 + YA_2 + Y^2 A_3$ in $\omega^i, \omega^{i+n/3}$ and $\omega^{i+2n/3}$:

$$\begin{aligned} \hat{A}[i] &= \hat{A}_1[i] + \omega^i \hat{A}_2[i] + \omega^{2i} \hat{A}_3[i], \\ \hat{A}[i+n/3] &= \hat{A}_1[i] + \theta\omega^i \hat{A}_2[i] + \theta^2 \omega^{2i} \hat{A}_3[i], \quad (8) \\ \hat{A}[i+2n/3] &= \hat{A}_1[i] + \theta^2\omega^i \hat{A}_2[i] + \theta\omega^{2i} \hat{A}_3[i]. \end{aligned}$$

This operation is frequently called the butterfly operation. It can be performed efficiently if we compute modulo $m(t)(t^{n/3}+1) = t^n + 1$ instead of $m(t)$. Indeed, in this case $\omega = t$ and a multiplication $a(t) \times \omega^i$ modulo $t^n + 1$ is a simple cyclic shift. The butterfly circuit (Figure 1) is a consequence of this remark and the relations given in (8). In Figure 1, the $\gg$ blocks refer to a simple shift operations by the given value and the $\bigoplus$ blocks refer to XOR operator. When no value is given, then shift operation is not performed.

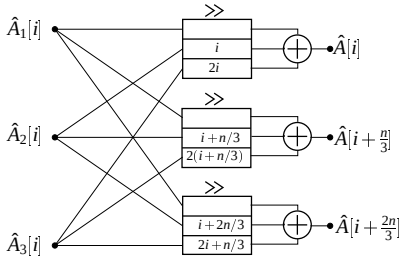

FIG. 1: Ternary butterfly operator.

Within the FFT, the computations of $\hat{A}_1, \hat{A}_2$ and $\hat{A}_3$ are done in the same way. These polynomials are split in three parts and butterfly operations are applied again. This process is done recursively until constant polynomial are reached. If we entirely develop this recursive process we obtain the schematized architecture in Figure 2.

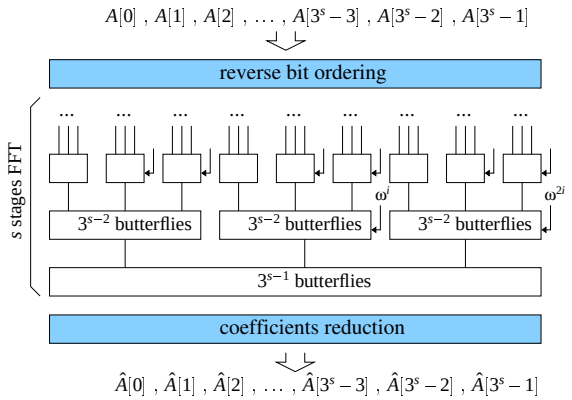

FIG. 2: Ternary FFT circuit.

Let us now evaluate the complexity of this architecture. It is composed of $\log_3(n)$ stages where each stage consists of $n/3$ butterfly operations. Each of these butterfly operations requires $6n$ XOR gates, and has a delay of $2T_X$, where T_X is the delay of one XOR gate.

Consequently, this architecture has a space complexity of

$$\mathcal{S}(FFT_{m(t)}) = (2n\log_3(n)+n)\ \text{XOR} \qquad (9)$$

and a delay of

$$\mathcal{D}(FFT_{m(t)}) = (2\log_3(n)+1)T_X. \qquad (10)$$

7 Architecture and Complexity

We now present an hardware architecture associated to Algorithm 3 in the special case where m_1 and m_2 are chosen as

$$m_1 = t^{2n}+t^n+1 \text{ and } m_2 = t^{2n/3}+t^{n/3}+1.$$

This choice enables us to use the FFT circuit presented in the previous section. The architecture of our binary field multiplier is given in Figure 3. It is constituted of FFT blocks and multipliers modulo $m_1(t)$ and $m_2(t)$.

TAB. 3: Complexity of multipliers modulo m_1 and m_2.

	Mul_{m1}	
	Space	**Time**
#AND	$3n^{\log_3(6)}$	1
#XOR	$\frac{72}{5}n^{\log_3(6)}-9n-7/5$	$3\log_3(n)+3$

	Mul_{m2}	
	Space	**Time**
#AND	$\frac{1}{2}n^{\log_3(6)}$	1
#XOR	$\frac{36}{15}n^{\log_3(6)}-n/5+n-1$	$3\log_3(n)$

These multipliers are referenced by blocks Mul_{m_1} and Mul_{m_2} in our architecture. Because of the special form of $m_1(t)$ and $m_2(t)$ we can use the multiplier of Fan and Hasan (Fan and Hasan, 2007) to perform this operation. Therefore, the complexity (cf. Table 3) of these blocks are easily deduced from (Fan and Hasan, 2007, Table 1).

The FFT blocks are designed using ternary method presented in previous section. Therefore, their complexity are those given in (9) and (10).

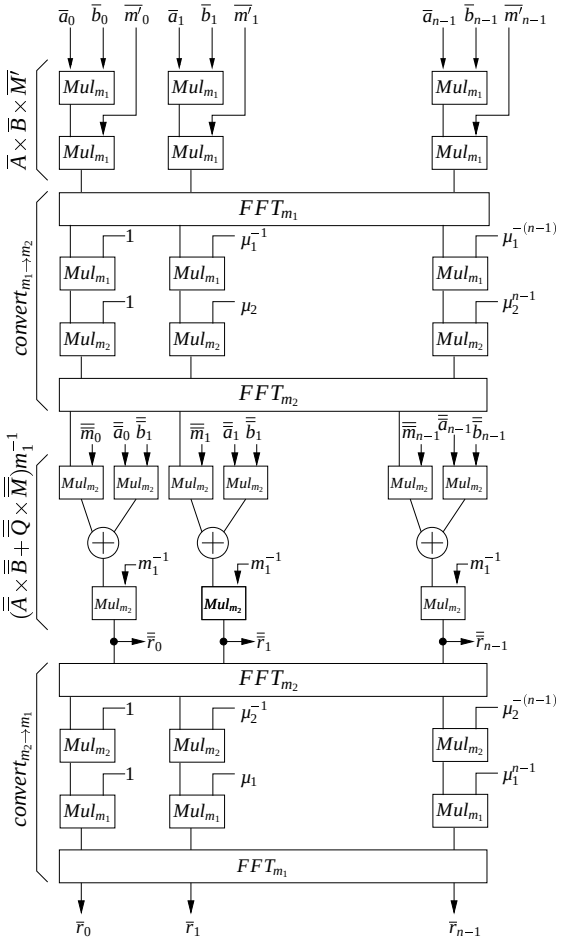

FIG. 3: DPS-Lagrange Multiplier.

The complexity of our multiplier can be evaluated with respect to the numbers of each blocks and their corresponding space complexity denoted $\mathcal{S}$, and time complexity denoted $\mathcal{D}$. For the space complexity this gives $4n\mathcal{S}(Mul_{m_1}) + 5n\mathcal{S}(Mul_{m_2}) + 2\mathcal{S}(FFT_{m_1}) + 2\mathcal{S}(FFT_{m_2}) + 2n^2/3$ XOR. Similarly, the critical path of this architecture gives the delay $4\mathcal{D}(Mul_{m_1}) + 4\mathcal{D}(Mul_{m_2}) + 2\mathcal{D}(FFT_{m_1}) + 2\mathcal{D}(FFT_{m_2}) + T_X$.

Using these expressions, (9),(10) and Table 3, we

TAB. 4: Complexity comparison.

	Space Complexity		Time Complexity	
Method	# AND	# XOR	T_A	T_X
This paper	$14.5k^{1.31}$	$69.6k^{1.31} - 31k + k^{0.5}(8\log_3(k) + 39)$	8	$16\log_3(k) + 20$
FH^* binary	$k^{1.58}$	$5.5k^{1.58} - 5k - 0.5$	1	$2\log_2(k) + 1$
FH^* ternary	$k^{1.63}$	$4.8k^{1.63} - 4k - 0.8$	1	$3\log_3(k) + 1$

FH^* = (Fan and Hasan, 2007) ;

can compute the complexity with respect to the number of XOR and AND gates and their corresponding delay T_X and T_A.

Let r be the degree in t of the coefficients in the DPS representation then $\deg_t(m_2)$ must satisfy $\deg_t(m_2) \geq r$. Therefore, this implies that $k \leq r \times n = 2n^2/3$ and thus leads to use $n \approx \sqrt{k}$, where k is the degree of the field $\mathbb{F}_{2^k}$.

Finally, we obtain the complexity of the DPS-Lagrange multiplier stated in Table 4. We also give in this table the complexity of the best known method, regarding space and time complexity, to perform binary field multiplication.

One can remark that our approach decrease the space complexity from $k^{1.58}$ to $k^{1.31}$, while it is slower by a factor roughly equals to 5.3.

For the sake of simplicity, we have presented our algorithm with special moduli m_1 and m_2. As a consequence, we have an increase in the degree of m_1 which has an influence on the final complexity. Another approach would be to use only m_2 as a special Fermat polynomial and takes m_1 such that $\mathbb{F}_2[t]/(m_1)$ and $\mathbb{F}_2[t]/(m_2)$ are isomorphic. Therefore, operations modulo m_1 could be performed modulo m_2 through isomorphism transformations, and then decrease the space complexity by a factor at least 3.

8 Conclusion

In this paper we have presented a novel algorithm to perform multiplication in binary field, using a Double Polynomial System of representation. This system enables the use of Fast Fourier Transform in the multiplication according to Lagrange representation. The resulting multiplier still achieve a logarithmic time complexity, but asymptotically improve the space complexity from $O(k^{1.58})$ to $O(k^{1.31})$,

Our method is a first approach to reduce the space complexity of binary field multiplier. In particular, some optimizations can be done to reduce the constant factors in the complexity. For example, lot of multiplications by a constant are counted as full multiplication in the current complexity evaluation.

Furthermore, one can also reduce the exponent in the space complexity by replacing Fan and Hasan multipliers with a quasi-linear approach (e.g. Schönhage's technique (Schönhage, 1977)).

Références

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

Diffie, W. and Hellman, M. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, 24 :644–654.

Fan, H. and Dai, Y. (2005). Fast bit-parallel GF(2^n) multiplier for all trinomials. *IEEE Trans. on Comp.*, 54(4) :485–490.

Fan, H. and Hasan, A. (2007). A new approach to subquadratic space complexity parallel multipliers for extended binary fields. *IEEE Trans. Comput.*, 56(2) :224–233.

Gao, S. (1993). *Normal Basis Over Finite Field*. PhD thesis, University of Waterloo.

Gathen, J. v. and Gerhard, J. (1999). *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA.

Giorgi, P., Jeannerod, C.-P., and Villard, G. (2003). On the complexity of polynomial matrix computations. In *Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation, Philadelphia, Pennsylvania, USA*, pages 135–142. ACM Press.

Guajardjo, J. and Paar, C. (1997). Efficient algorithms for elliptic curve cryptosystems. In *Advances in Cryptology, Proceedings of Eurocrypt'97*, volume 1233 of *LNCS*, pages 342–356. Springer-Verlag.

J.-C. Bajard, L.Imbert, T. P. (2005). Modular number systems : Beyong the mersenne family. In *Selected Area in Cryptography, procee-*

dings of SAC2004,Waterloo, Canada, August 9-10 2004, volume 3357 of *LNCS*, pages 159–169. Springer-Verlag.

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

Koc, C. and Sunar, B. (1998). Low Complexity Bit-Parallel Canonical and Normal Basis Multiplier For a Class of Finite Fields. *IEEE Transaction on Computers*, 47 :353–356.

Koc, C. and Sunar, B. (1999). Mastrovito Multiplier for All Trinomials. *IEEE Transaction on Computers*, 48(5) :522–52.

Mastrovito, E. (1991). *VLSI architectures for computations in Galois fields*. PhD thesis, Dep.Elec.Eng.,Linkoping Univ.

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

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

Mulders, T. and Storjohann, A. (2003). On lattice reduction for polynomial matrices. *Journal of Symbolic Computation*, 35(4) :377–401.

Schönhage, A. (1977). Schnelle multiplikation von polynomen über körpern der charakteristik 2. *Acta Informatica*, 7 :395–398.

Villard, G. (1996). Computing Popov and Hermite forms of polynomial matrices. In *Proceedings fo the 1996 International Symposium on Symbolic and Algebraic Computation, Zurich, Suisse*, pages 250–258. ACM Press.