Subquadratic Binary Field Multiplier in Double Polynomial System

Pascal Giorgi, Christophe Negre, Thomas Plantard

To cite this version:


HAL Id: hal-00140082
https://hal.science/hal-00140082
Submitted on 4 Apr 2007

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Subquadratic Binary Field Multiplier in Double Polynomial System

Pascal Giorgi\textsuperscript{1}, Christophe N`egre\textsuperscript{2}

\textit{Equipe DALI, LP2A, Universit`e de Perpignan}
\textit{avenue P. Alduy, F66860 Perpignan, France}
(1) pascal.giorgi@univ-perp.fr, (2) christophe.negre@univ-perp.fr

Thomas Plantard
\textit{Centre for Computer and Information Security Research}
\textit{School of Computer Science & Software Engineering}
\textit{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 sets elements as 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 $A(t, Y) = \sum_{a=0}^{n-1} a_i(t) Y^i$ 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 \((16\log_2(k) + 20)/T_X + 8T_A\) and a space complexity of \(O(k^{1.33})\), 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 \(k\)

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

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
  \[
  \gamma(t)^n \equiv \lambda(t) \mod P,
  \]
  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 \(A_B(t,Y) \in \mathbb{F}_2[t,Y]\) such that

\[
A_B(t,Y) = \sum_{i=0}^{n-1} a_i(t)Y^i \quad \text{with } \deg_r a_i(t) < r
\]

and \(A_B(t,\gamma(t)) \equiv A(t) \mod P\)

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 \(\mathbb{F}_{2^4}\).

| \(A(t)\) | \(0\) | \(t^2\) | \(t^3 + t^2 + t + 1\) | \(t^3 + t\) |
| \(A_B\) | \(0\) | \((t+1)Y^2\) | \(Y + 1\) | \(Y^2 + t + 1\) |

| \(A(t)\) | \(1\) | \(t^2 + 1\) | \(t^3 + t^2 + t\) | \(t^3 + t + 1\) |
| \(A_B\) | \(1\) | \((t + 1)Y^2 + 1\) | \(Y\) | \(Y^2 + t\) |

| \(A(t)\) | \(t\) | \(t^2 + t\) | \(t^3 + t^2\) | \(t^3 + 1\) |
| \(A_B\) | \(t + 1\) | \(Y^2 + Y\) | \(Y + t\) | \(Y^2 + 1\) |

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

**Remark 1.** The DPS can be seen as a generalization of the polynomial representation of double extensions \(\mathbb{F}_{2^m}\). Such extensions are usually constructed first as \(\mathbb{F}_{2^m} = \mathbb{F}_2[t]/(P(t))\) and then as \(\mathbb{F}_{2^m} = \mathbb{F}_{2^2}[Y]/(Y^n - \lambda)\) with \(\lambda \in \mathbb{F}_{2^2}\), see (Guajardo 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 \(C(t,\gamma) = A(t,\gamma)B(t,\gamma) \mod P(t)\) since \(E(\gamma) \equiv 0 \mod P(t)\) by definition of the DPS.

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

\[
R(t,\gamma) = A(t,\gamma)B(t,\gamma) \mod P(t) \text{ and } \deg_R(R) < r.
\]
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 $Y$:
   $C = AB \mod (Y^n - \lambda)$.
2. Coefficients reduction:
   $R = \text{RedCoeff}(C)$.

Algorithm 2 DPS Multiplication.

Input : $A, B \in \mathcal{B} = \{P; \gamma, n, r, \lambda\}$ with $E = Y^n - \lambda$
Data : $M$ such that $M(\gamma) \equiv 0 \mod P$, a polynomial $m \in F_2[t]$ and $M' = -M^{-1} \mod (E, m)$
Output: $R(t, \gamma)$ such that $R(t, \gamma) = A(t, \gamma)B(t, \gamma)m^{-1} \mod P$

begin
  $C \leftarrow A \times B \mod E$;
  $Q \leftarrow C \times M' \mod (E, m)$;
  $R \leftarrow (C + Q \times M \mod E) / m$;
end

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 $2(r-2) + \deg_e \lambda$. 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 $\deg R < r$. This reduction is only valid if $M(t, Y)$ is monic in $t$ and satisfies

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

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 $M(t, \gamma) \equiv 0 \mod P(t)$ is given.

Example 2. We consider the field $F_{24}$, with the DPS $\mathcal{B} = \{P = t^3 + t^3 + t^2 + t, \gamma = t^3 + t^2 + t, n = 3, r = 2, \lambda = t\}$. In Table 2, we give an example of trace of DPS multiplication.

<table>
<thead>
<tr>
<th>Operations</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$tY^2 + tY$</td>
</tr>
<tr>
<td>$B$</td>
<td>$(t + 1)Y + t$</td>
</tr>
<tr>
<td>$M$</td>
<td>$tY^2 + Y + t$</td>
</tr>
<tr>
<td>$M'$</td>
<td>$(1 + t)Y^2 + (1 + t)Y + 1$</td>
</tr>
<tr>
<td>$m$</td>
<td>$t^2$</td>
</tr>
<tr>
<td>$C$</td>
<td>$tY^2 + t^2Y + t^2 + t$</td>
</tr>
<tr>
<td>$Q$</td>
<td>$tY^2$</td>
</tr>
<tr>
<td>$Q \times M$</td>
<td>$(t^2 + t)Y^2 + t^3Y + t^2$</td>
</tr>
<tr>
<td>$C + Q \times M$</td>
<td>$t^2Y^2 + (t^3 + t^2)Y + t^3$</td>
</tr>
<tr>
<td>$R$</td>
<td>$Y^2 + (t + 1)Y + t$</td>
</tr>
</tbody>
</table>

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

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

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

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

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

$$C(t, \gamma) + Q(t, \gamma)M(t, \gamma) \equiv C(t, \gamma) \equiv A(t, \gamma)B(t, \gamma) \mod P.$$

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

$$R' \equiv C + C \times (-M^{-1} \times M) \mod (E, m) \equiv (C - C) \mod (E, m) \equiv 0 \mod (E, m).$$

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 $R(t,γ) = A(t,γ)B(t,γ)m^{-1} \mod P$ 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 $\mathcal{B} = (P, γ, n, r, λ)$ a Double Polynomial System, $M$ be a polynomial of $\mathcal{B}$ such that $M(γ) \equiv 0 \mod P$ and $σ = deg_γ(M)$. Let $A, B$ be two elements expressed in the DPS $\mathcal{B}$. If $r$ and the polynomial $m$ satisfy

$$r > σ + deg_t(λ) \quad \text{and} \quad deg_t(m) > deg_t(λ) + r \quad (3)$$

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 $\mathcal{B} = (P, γ, n, r, λ)$ if $deg_t R < n$ and if $deg_t(R) < r$. The fact that $deg_t R < n$ is easy to see since all the computation in the Algorithm 2 are done modulo $E = Y^n - λ$.

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

$$deg_t R = deg_t((A × B + Q × M) \mod E) / m ≤ \max(deg_tA + deg_tB, deg_tQ + deg_tM) + deg_tλ - deg_t m ≤ \max(2r, σ + deg_t m) + deg_tλ - deg_t m.$$ 

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

\hfill $\Box$

### 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 $M(t, γ) \equiv 0 \mod P$ and $deg_t M$ is small. This polynomial belongs to the set

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

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 $\mathbb{F}_2[t] :$

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

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.

$$\Omega = \begin{pmatrix} P & 0 & 0 & \cdots & 0 \\ -γ & 1 & 0 & \cdots & 0 \\ -γ^2 & 0 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ -γ^{r-1} & 0 & 0 & \cdots & 1 \end{pmatrix} \quad (\leq P$$

Each of the polynomials $V(t, γ)$ defined by the rows of $\Omega$ satisfy $V(t, γ) \equiv 0$, 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 $(f_1, f_2, \ldots, f_n)$ of $\mathcal{M}'$ which satisfies the following degree properties:

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

$$deg_t f_1 ≤ deg_t f_2 ≤ \ldots ≤ deg_t f_n \quad (5)$$

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

$$deg_t M ≤ (deg_t P) / n \quad (6)$$

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

Besides 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 $O(n^2k^2)$ binary operations with Algorithm *WeakPopovForm* of (Mulders and Storjohann, 2003) or with an asymptotic complexity of $O(n^2k \log k)$ 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

\[
\mathcal{R}[Y]/(E(Y)) \rightarrow \prod_{i=1}^{n} \mathcal{R}[Y]/(Y - \alpha_i) \quad (7)
\]

\( A \leftrightarrow (A \mod (Y - \alpha_i))_{i \in \{1, \ldots, n\}} \).

The computation of \( A \mod (Y - \alpha_i) \) 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 \( E(Y) \).

\[
E(Y) = \prod_{i=1}^{n} (Y - \alpha_i) \mod m
\]

If \( \alpha_i = A(\alpha_i) \) for \( 1 \leq i \leq n \), the Lagrange representation (LR) of \( A(Y) \) is defined by \( LR(A(Y)) = (\alpha_1, \ldots, \alpha_n) \).

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 \( m_1(t) \):

\[
E = \prod_{i=1}^{n} (Y - \alpha_i) \mod m_1(t),
\]

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

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

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 \( \alpha_i \) modulo \( m_1(t) \)
- \( \overline{A} \) the Lagrange representation in \( \alpha'_i \) modulo \( m_2(t) \).

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

**Algorithm 3 DPS-LR Multiplication.**

**Input:** \( \overline{A}, \overline{B}, \overline{R} \)

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

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

begin

\[
\overline{Q} \leftarrow \overline{A} \times \overline{B} \times \overline{M}';
\]

\[
\overline{Q} \leftarrow \text{Convert}_{m_1 \rightarrow m_2}(\overline{Q});
\]

\[
\overline{R} \leftarrow (\overline{Q} \times \overline{B}) + \overline{Q} \times \overline{M} \times m_2^{-1};
\]

\[
\overline{R} \leftarrow \text{Convert}_{m_2 \rightarrow m_1}(\overline{R});
\]

end

The operations to compute \( \overline{Q} \) and \( \overline{R} \) 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

\[ \alpha_j = \mu_1 \omega_1^j \mod m_1 \]

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

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

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

where \( DFT_{m_1}(\hat{A}, n, \omega_1) \) is the evaluations of the polynomial \( \hat{A} \) in the \( n \)-th roots of unity \( \omega_1 \).

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

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

**Algorithm 4** Convert\(m_1\rightarrow m_2\).

**Input**: \( \hat{A} \)

**Output**: \( \hat{A} \)

\[
\begin{align*}
\hat{A}(Y) & \leftarrow DFT_{m_1}^{-1}(\hat{A}, n, \omega_1) \; ; \\
A(Y) & \leftarrow \hat{A}(\mu_1Y^{-1}) \mod m_1 \; ; \\
\hat{A}(Y) & \leftarrow DFT_{m_2}(A(Y), n, \omega_2) \mod m_2 \; ; \\
\hat{A} & \leftarrow DFT_{m_2}(\hat{A}(Y), n, \omega_2) ;
\end{align*}
\]

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 \( m_1(t) \) such that \( m_1(t) = t^{2x3} + t^3 + 1 \) 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 \( A(Y) \in \mathbb{F}[Y] \) of degree \( n-1 \), keeping in mind our targeted Lagrange conversion algorithm. We consider the ring \( \mathbb{F} = \mathbb{F}_2[i]/(m(t)) \) where \( m(t) = t^{3n} + t^3 + 1 \) and \( n = 3^t \). 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 \( \omega \) a primitive \( n \)-th root of unity modulo \( m(t) \) and \( \theta = \omega^{3^t} \) a 3rd root of unity. The ternary FFT process is based on the following three-way splitting of \( A \):

\[
\begin{align*}
A_1 & = \sum_{j=0}^{n/3-1} a_{3j} Y^{3j} , \\
A_2 & = \sum_{j=0}^{n/3-1} a_{3j+1} Y^{3j} , \\
A_3 & = \sum_{j=0}^{n/3-1} a_{3j+2} Y^{3j} ,
\end{align*}
\]

such that \( A = A_1 + Y A_2 + Y^2 A_3 \).

Let \( \tilde{A}[i] = A(\omega^i) \) be the \( i \)-th coefficient of \( DFT_m(A, n, \omega) \). Let us also denote by \( \tilde{A}_1[i], \tilde{A}_2[i] \) and \( \tilde{A}_3[i] \) 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 \( A = A_1 + Y A_2 + Y^2 A_3 \) in \( \omega, \omega + \omega^2, 3 \omega^2 \):

\[
\begin{align*}
\tilde{A}[i] & = \tilde{A}_1[i] + \omega \tilde{A}_2[i] + \omega^2 \tilde{A}_3[i] , \\
\tilde{A}[i + n/3] & = \tilde{A}_1[i] + \theta \omega \tilde{A}_2[i] + \theta^2 \omega^2 \tilde{A}_3[i] , \\
\tilde{A}[i + 2n/3] & = \tilde{A}_1[i] + \theta^2 \omega \tilde{A}_2[i] + \theta^3 \omega^2 \tilde{A}_3[i] ,
\end{align*}
\]

This operation is frequently called the butterfly operation. It can be performed efficiently if we compute modulo \( m(t)(\omega^{n/3} + 1) = t^n + 1 \) instead of \( m(t) \). Indeed, in this case \( \omega = \theta \) and a multiplication by \( \omega \) 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 \( \oplus \) blocks refer to XOR operator. When no value is given, then shift operation is not performed.

![Ternary butterfly operator](image.png)

**Fig. 1:** Ternary butterfly operator.
These multipliers are referenced by blocks $\text{Mul}_{m_1}$ and $\text{Mul}_{m_2}$ 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).

### 7 Architecture and Complexity

Let us now evaluate the complexity of this architecture. It is composed of $\log_2(n)$ 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

$$S(FFT_{m(t)}) = (2n\log_2(n) + n) \text{ XOR}$$

and a delay of

$$D(FFT_{m(t)}) = (2\log_2(n) + 1)T_X.$$  

#### 7.1 Tab. 3: Complexity of multipliers modulo $m_1$ and $m_2$.

<table>
<thead>
<tr>
<th>$\text{Mul}_{m_1}$</th>
<th>Space</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>#AND</td>
<td>$3n^{log_2(10)}$</td>
<td>1</td>
</tr>
<tr>
<td>#XOR</td>
<td>$\frac{2}{3}n^{log_2(10)} - 9n - 7/5$</td>
<td>$3\log_2(n) + 3$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\text{Mul}_{m_2}$</th>
<th>Space</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>#AND</td>
<td>$\frac{1}{5}n^{log_2(10)}$</td>
<td>1</td>
</tr>
<tr>
<td>#XOR</td>
<td>$\frac{2}{5}n^{log_2(10)} - n/5 + n - 1$</td>
<td>$3\log_2(n)$</td>
</tr>
</tbody>
</table>

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

Using these expressions, (9),(10) and Table 3, we
can compute the complexity with respect to the number of XOR and AND gates and their corresponding delay $T_A$ and $T_X$.

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 $F_{2^k}$.

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 $F_2[m]/(m_1)$ and $F_2[m]/(m_2)$ 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


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


