Polynomial multiplication over binary finite fields: new upper bounds

Alessandro De Piccoli[1], Andrea Visconti[1], Ottavio Giulio Rizzo[2]

[1] Department of Computer Science "Giovanni Degli Antoni",
Università degli Studi di Milano
via Comelico 39/41, 20135, Milano, Italy
Tel.: +39 02 50316361
ORCID: 0000-0002-6399-3164
alessandro.depiccoli@unimi.it
ORCID: 0000-0001-5689-8575
andrea.visconti@unimi.it
http://www.di.unimi.it/visconti
[2] Department of Mathematics "Federigo Enriques",
Università degli Studi di Milano
ottavio.rizzo@unimi.it

Abstract. When implementing a cryptographic algorithm, efficient operations have high relevance both in hardware and software. Since a number of operations can be performed via polynomial multiplication, the arithmetic of polynomials over finite fields plays a key role in real-life implementations — e.g. accelerating cryptographic and cryptanalytic software (pre- and post-quantum) [18]. One of the most interesting paper that addressed the problem has been published in 2009. In [5], Bernstein suggests to split polynomials into parts and presents a new recursive multiplication technique which is faster than those commonly used. In order to further reduce the number of bit operations [6] required to multiply n-bit polynomials, researchers adopt different approaches. In [19] a greedy heuristic has been applied to linear straight-line sequences listed in [6]. In 2013, D'angella, Schiavo and Visconti [21] skip some redundant operations of the multiplication algorithms described in [5]. In 2015, Cenk, Negre and Hasan [12] suggest new multiplication algorithms. In this paper, (a) we present a *"k-1"-level Recursion algorithm* that can be used to reduce the effective number of bit operations required to multiply n-bit polynomials; and (b) we use algebraic extensions of $\mathbb{F}_2$ combined with Lagrange interpolation to improve the asymptotic complexity.

Keywords: Polynomial multiplication, Karatsuba, Two-level Seven-way Recursion algorithm, binary fields, fast software implementations.

1 Introduction

Finite fields have applications in many areas of computer science and engineering, such as digital signal processing [29,9], coding theory [3,8], cryptography [30,2,10,31,25] and so on. Such applications usually need efficient implementations both in hardware [34,15,14,1,28,26] and software [5,21,19,12], thus a fast execution of arithmetic operations over finite fields is a crucial issue. In this paper particular attention is paid to binary fields, i.e., finite fields of characteristic 2, because they are very attractive for several cryptographic applications, especially for those who play with elliptic curves [4,7,5].

A binary field $\mathbb{F}_{2^n}$ is composed of binary polynomials modulo a n-degree irreducible polynomial. The multiplication between two elements of $\mathbb{F}_{2^n}$ is one of the most crucial low-level arithmetic operations. It consists of an ordinary polynomial multiplication and a modular reduction by an irreducible polynomial. Whereas the modular reduction is a relatively simple operation, the polynomial multiplication turns out to be a costly operation.

A real case scenario can help readers to understand the problem in details. In 2009, Bernstein show that, on a binary Edwards curve [5], a 251-bit single-scalar multiplication requires 44,679,665 bit operations, 43,011,084 of which (about 96%) are for field multiplications. That said, it is not difficult to understand why fast software implementations for polynomial multiplication over finite fields are desired.

It is well known that the naive polynomial multiplication algorithm — the so-called School-book algorithm — is not the optimal way to multiply two polynomials. If the polynomials involved in the product have the same degree, say n, the multiplication takes n^2 multiplications and $(n-1)^2$ additions. Thus, its complexity is $2n^2+\mathcal{O}(n)$. Many researchers have tried to improve this algorithm, following two main directions: (1) provide a better asymptotic estimation [34,16,35,24]; (2) reduce the effective number of bit operations [5,12,14,22,21].

A number of interesting approaches that improves the school-book algorithm have been published in literature — see for example Karatsuba [27], Toom [38], Cook [20], Schönhage and Strassen [37], Bernstein [5], and so on. More precisely, Karatsuba [27] achieves an asymptotic complexity of $7n^{1.58} + \mathcal{O}(n)$. Toom [38] and Cook [20] reduced the number of steps needed to multiply two polynomials introducing an algorithm with complexity $\mathcal{O}(n^{1+\epsilon})$, for arbitrary small $\epsilon > 0$. In [37] Schönhage and Strassen showed how to achieved complexity $\mathcal{O}(n \log n \log \log n)$ using a procedure based on the Fast Fourier Transform (FFT). In 2009, Bernstein [5] improves the Karatsuba identity (Three-way Recursion algorithm), obtaining the following asymptotic complexity $6.5n^{1.58} + \mathcal{O}(n)$. Cenk, Negre and Hasan in [12] suggest to change the field for the polynomials, getting an asymptotic complexity of $15.125n^{1.46} - 2.67n \log_3(n) + \mathcal{O}(n)$.

Notice that asymptotic estimations are not explicit bounds and real-world applications have to deal with issues of hardware and software implementations — e.g., hardware constraints, software speedups, and so on. Therefore, in order to get the minimum number of bit operations needed to multiply two n-bit polynomials — for sake of simplicity we call such a number $M(n)$ — researchers analyze, rearrange and modify the algorithms that provide interesting asymptotic estimations. Their aim is to improve bounds published in literature for specific value (small) of n, and these improvements that are not visible in the asymptotics. Consequently, a number of papers tries to reduce the effective number of bit operations [34,16,35,24]. As far as we know, the best explicit upper bounds for the polynomial multiplication appear in [6,19,21,12].

Karatsuba [27] was the first one who reduces the number of bit operations of the School-book algorithm. A different approach has been described by Bernstein in 2009 [5]. He refines the Karatsuba identity and suggests to use a polynomial multiplication technique which employs (recursively) different multiplication algorithms, picking, at each step, the best one. Moreover, he presents three new multiplication algorithms — i.e., Three-way Recursion, Five-way Recursion, and Two-level Seven-way Recursion algorithm — that are used to reduce the effective number of bit operations. The technique presented in [5] not only results in new software speed records [6], but also avoids well-known software side-channel attacks. Indeed, all computations are expressed as straight-line sequences of AND/XOR operations, thus they are data-independent. In [19] are published improvements for specific value of n obtained by applying Boyar-Peralta heuristic [11] on the linear part of straight-line sequences reported in [6]. In 2013, D'angella, Schiavo and Visconti [21] skip some redundant operations of the multiplication algorithms described in [5], reducing the number of bit operations for many values of n. The authors focus in particular on Five-way Recursion algorithm because such an algorithm is widely used. In 2015, Cenk, Negre and Hasan [12] present new multiplication algorithms which improve many of the explicit upper bounds previously described.

1.1 Our contributions

In this paper we investigate the possibility to (a) further reduce the effective number of bit operations required to multiply n-bit polynomials, and (b) improve the asymptotic complexity.

Firstly, we refine the Two-level Seven-way Recursion algorithm [5]. As shown in [5], it seems that Lagrange Interpolation is a useful tool to arrange the order of operations. Although in many cases this is true, in others it is not. Rearranging the operations in a different way, we present a *"k-1"-level Seven-way Recursion algorithm*, or *"k-1"-level Recursion* for short. We show that *Three-*, *Four-*, and *Five-level Recursion* can be used to improve the explicit upper bounds published in literature.

Secondly, we use algebraic extensions of $\mathbb{F}_2$ combined with Lagrange interpolation to improve the asymptotic complexity. We will show an interesting connection between this technique and the computation of the values of a polynomial in all of the field elements.

1.2 Organization of the paper

The remainder of the paper is organized as follows. In Section 2, we state definitions and some preliminary concepts that are useful to understand the following sections. In Section 3, starting with the classical school-book algorithm, we introduce some of the approaches currently adopted to multiply polynomials in an efficient way. Section 4 is the heart of this paper. We present our contribution, showing the new speed records achieved and explaining the techniques adopted. Finally, conclusions are drawn in Section 5.

2 Preliminaries

We restrict our analysis to polynomials over finite fields of characteristic 2, so we will not ever use the minus sign. If $F(t)$ and $G(t)$ are two of these polynomials, we will call their product $H(t)$.

To denote the cost of the multiplication in $\mathbb{F}_g$ between two polynomials of degree $n-1$ we will use $M_g(n)$.

2.1 Projective Lagrange Interpolation

As pointed out in [12], Lagrange Interpolation leads us to efficient multiplication algorithms. How does this technique work? Consider a field $\mathbb{K}$ and a polynomial $H \in \mathbb{K}[x]$,

$$H(x) = h_0 + h_1x + h_2x^2 + ... + h_nx^n$$

Algebra tells us that we need to fix the value of the polynomial in $n+1$ points in order to uniquely determine it. So, given a set of $n+1$ distinct points $\{k_0, \dots, k_n\} \subseteq \mathbb{K}$, we define the Lagrange polynomials as follows:

$$l_i(x) = \prod_{j \neq i} \frac{x - k_j}{k_i - k_j} \qquad i = 0, \dots, n$$

Notice that we have $l_i(k_i) = 1$ and $l_i(k_j) = 0$, $\forall j \neq i$. This feature allows us to exactly reconstruct any polynomial $H \in \mathbb{K}[x]$ as

$$H(x) = \sum_{i=0}^{n} H(k_i) \cdot l_i(x)$$

For our purposes, the above technique is not optimal. Given the same problem with only n points $\{k_0, ..., k_{n-1}\}$, define the degree $n-1$ polynomial

$$\overline{H} = \sum_{i=0}^{n-1} H(k_i) \cdot l_i(x)$$

We still have $\overline{H}(k_i) = H(k_i)$, for $i = 0, ..., n-1$. Let

$$l_\infty(x) = \prod_{j=0}^{n-1} (x - k_i)$$

and $H(\infty) = h_n$. Since $H(\infty) \cdot l_\infty$ vanishes at every k_i and has degree n, we can reconstruct H with the so-called Projective Lagrange Interpolation formula,

$$H(x) = \sum_{i=0}^{n-1} H(k_i) \cdot l_i(x) + H(\infty) \cdot l_\infty(x).$$

2.2 Which field?

Lagrange Interpolation requires $n+1$ points, but we just have two points in $\mathbb{F}_2$! Projective Lagrange Interpolation will do with n points since it makes use of the point at infinity: where can we find even more points? A possible answer is to consider finite algebraic extensions of $\mathbb{F}_2$, generated by a monic irreducible polynomial γ over $\mathbb{F}_2$ of degree d. Indeed, an extension $\mathbb{F}$ is a quotient $\mathbb{F}_2[X]/\langle\gamma(X)\rangle$, so the elements of $\mathbb{F}$ are all d-bit polynomials, i.e., the set of polynomials over $\mathbb{F}_2$ of degree at most $d-1$, and $\mathbb{F}$ has 2^d elements. If δ is another irreducible polynomial of degree d, there is a, non canonical, isomorphism $\mathbb{F}_2[X]/\langle\gamma(X)\rangle \simeq \mathbb{F}_2[X]/\langle\delta(X)\rangle$, so we will call such an extension $\mathbb{F}_{2^d}$.

$\mathbb{F}_{2^d}^\times$ is a cyclic group: let α be a fixed generator, we can see $\mathbb{F}_{2^d}$ as a vector space over $\mathbb{F}_2$ with basis $\{1, \alpha, \alpha^2, \dots, \alpha^{d-1}\}$. At last, note that $\mathbb{F}_{2^d}$ is the splitting field of $X^{2^d} + X$: its roots are all the elements of the field.

3 Current approaches

3.1 School-book algorithm

Given two n-bit polynomials

$$F = f_0 + f_1 t + ... + f_n t^n \quad \text{and} \quad G = g_0 + g_1 t + ... + g_n t^n.$$

The steps of the algorithm are:

- Recursively multiply $f_0 + f_1 t + ... + f_{n-1}t^{n-1}$ by $g_0 + g_1 t + ... + g_{n-1}t^{n-1}$;
- Compute $(f_n g_0 + f_0 g_n)t^n + (f_n g_1 + f_1 g_n)t^{n+1} + ... + f_n g_n t^{2n}$. This takes $2n+1$ multiplications and n additions;
- Add the former to the latter. This takes $n-1$ additions for the coefficients of $t^n, ..., t^{2n-2}$; the other coefficients do not overlap.

We get the recursion formula $M(n+1) \leq M(n) + 4n$ and the best case bound $M(n) \leq 2n^2 - 2n + 1$. This algorithm is efficient only in low degrees. Indeed, as reported in [6], the cost of the school-book algorithm is too high from degree 14 on.

3.2 Karatsuba

Given two $2n$-bit polynomials F and G, write them as $F = F_0 + F_1 t^n$, $G = G_0 + G_1 t^n$ for some other n-bit polynomials F_0, F_1, G_0, G_1. The *Karatsuba* algorithm [27] can be described by the product.

$$\begin{aligned}&(F_0 + t^n F_1)(G_0 + t^n G_1)\\ &= (1 + t^n)F_0 G_0 + t^n (F_0 + F_1)(G_0 + G_1) + (t^n + t^{2n})F_1 G_1\end{aligned}$$

The operations involved are:

- $M_2(n)$: multiplication $F_0 G_0$
- $n-1$: sum $S_1 = (1 + t^n)F_0 G_0$
- $2n$: sums $F_0 + F_1$, $G_0 + G_1$
- $M_2(n)$: multiplication $(F_0 + F_1)(G_0 + G_1)$
- $M_2(n)$: multiplication $F_1 G_1$
- $n-1$: sum $S_2 = (t^n + t^{2n})F_1 G_1$
- $2n-1$: sum $S_3 = S_1 + t^n (F_0 + F_1)(G_0 + G_1)$
- $2n-1$: sum $S_3 + S_2$

Summing all costs, we get

$$M_2(2n) \leq 3M_2(n) + 8n - 4 \tag{1}$$

3.3 Bernstein

Bernstein improves the Karatsuba algorithm defining the so-called *Refined Karatsuba* algorithm [5]. As described in Section 3.2, we consider two $2n$-bit polynomials F, G and take F_0, G_0 as n-bit polynomials and F_1, G_1 as k-bit polynomials. The Refined Karatsuba algorithm can be described as follows.

$$\begin{aligned}&(F_0 + t^n F_1)(G_0 + t^n G_1)\\ &= (1 + t^n)F_0 G_0 + t^n (F_0 + F_1)(G_0 + G_1) + (t^n + t^{2n})F_1 G_1\\ &= (1 + t^n)F_0 G_0 + t^n (F_0 + F_1)(G_0 + G_1) + (1 + t^n)t^n F_1 G_1\\ &= (1 + t^n)(F_0 G_0 + t^n F_1 G_1) + t^n (F_0 + F_1)(G_0 + G_1)\end{aligned}$$

The cost estimation of the algorithm is

$$M_2(n+k) \leq 2M_2(n) + M_2(k) + 4k + 3n - 3 \qquad n/2 \leq k \leq n \tag{2}$$

This improves that of Karatsuba described in Section 3.2.

Moreover in [5] we can find another improvement but for higher degrees. In fact, Bernstein presents the so-called *Two-level Seven-way Recursion*. Consider the problem of multiplying two polynomial of $4n$ bits. Applying the Refined Karatsuba identity three times and factoring out $1+t^n$, we get

$$\begin{aligned}
&(F_0+t^nF_1+t^{2n}F_2+t^{3n}F_3)(G_0+t^nG_1+t^{2n}G_2+t^{3n}G_3)\\
=&(1+t^{2n})\Big((1+t^n)(F_0G_0+t^nF_1G_1+t^{2n}F_2G_2+t^{3n}F_3G_3)\\
&+t^n(F_0+F_1)(G_0+G_1)+t^{3n}(F_2+F_3)(G_2+G_3)\Big)\\
&+t^{2n}\Big(F_0+F_2+t^n(F_1+F_3)\Big)\Big(G_0+G_2+t^n(G_1+G_3)\Big)
\end{aligned}$$

The cost evaluation for polynomials with $3n+k$ coefficients, assuming $n/2 \le k \le n$, is

- $3M(n)$: multiplications F_0G_0, F_1G_1, F_2G_2.
- $M(k)$: multiplication F_3G_3.
- $3(n-1)$: sums $S_1 = F_0G_0 + t^nF_1G_1 + t^{2n}F_2G_2 + t^{3n}F_3G_3$.
- $2n+2k-1$: sum $(1+t^n)S_1$.
- $2n+M(n)$: multiplication $S_2 = (F_0+F_1)(G_0+G_1)$.
- $2k+M(n)$: multiplication $S_3 = (F_2+F_3)(G_2+G_3)$.
- $4n-2$: sums $S_4 = (1+t^n)S_1 + t^nS_2 + t^{3n}S_3$.
- $2n+2k+M(2n)$: multiplication $S_5 = (F_0+F_2+t^n(F_1+F_3))(G_0+G_2+t^n(G_1+G_3))$.
- $6n+2k-2$: sum $(1+t^{2n})S_4 + t^{2n}S_5$.

Hence, summing all the costs, we obtain

$$M(3n+k) \le M(2n) + 5M(n) + M(k) + 19n + 8k - 8 \qquad n/2 \le k \le n \tag{3}$$

3.4 Cenk, Negre, Hasan

In [12] and [13], the authors suggest to use a field bigger than $\mathbb{F}_2$ for Projective Lagrange Interpolation. They consider two $3n$-bit polynomials F and G, written as $F = F_0 + F_1t^n + F_2t^{2n}$, $G = G_0 + G_1t^n + G_2t^{2n}$ with F_0, F_1, F_2, G_0, G_1, G_2 n-bit polynomials. Then, they make computations using the elements of $\mathbb{F}_4$. If α is a generator of $\mathbb{F}_4^\times$, and assuming n odd, the new algorithm can be written as follows.

$$\begin{aligned}
&H(t) = (F_0+t^nF_1+t^{2n}F_2)(G_0+t^nG_1+t^{2n}G_2)\\
&H(0) = F_0G_0\\
&H(1) = (F_0+F_1+F_2)(G_0+G_1+G_2)\\
&H(\alpha) = (F_0+F_2+\alpha(F_1+F_2))(G_0+G_2+\alpha(G_1+G_2))\\
&H(\alpha+1) = (F_0+F_1+\alpha(F_1+F_2))(G_0+G_1+\alpha(G_1+G_2))\\
&H(\infty) = F_2G_2
\end{aligned}$$

$$\begin{aligned}
&(F_0+t^nF_1+t^{2n}F_2)(G_0+t^nG_1+t^{2n}G_2)\\
=&\Big(H(0)+t^nH(\infty)\Big)(1+t^{3n})\\
&+\Big(H(1)+(1+\alpha)(H(\alpha)+H(\alpha+1))\Big)(t^n+t^{2n}+t^{3n})\\
&+\alpha\Big(H(\alpha)+H(\alpha+1)\Big)t^{3n}+H(\alpha)t^{2n}+H(\alpha+1)t^n
\end{aligned} \tag{4}$$

Notice that if n is even, we just exchange the formulae for $H(\alpha)$ and $H(\alpha+1)$. As described in [12], the cost evaluation for the *CNH 3-way split algorithm* is

$$M_2(3n) \leq 2M_4(n) + 3M_2(n) + 29n - 12$$

An improvement of this algorithm is described in [12]: using two polynomials C_0 and C_1 to rearrange equations $H(\alpha)$ and $H(\alpha+1)$

$$\begin{aligned} H(\alpha) &= \Big(F_0 + F_2 + \alpha(F_1 + F_2)\Big)\Big(G_0 + G_2 + \alpha(G_1 + G_2)\Big) = C_0 + \alpha C_1 \\ H(\alpha+1) &= \Big(F_0 + F_1 + \alpha(F_1 + F_2)\Big)\Big(G_0 + G_1 + \alpha(G_1 + G_2)\Big) = (C_0 + C_1) + \alpha C_1 \end{aligned}$$

it is possible to redefine (4) as

$$\begin{aligned} &(F_0 + t^n F_1 + t^{2n} F_2)(G_0 + t^n G_1 + t^{2n} G_2) \\ &= H(\infty)t^{4n} + H(0) \\ &\quad + \Big(H(0) + H(1) + C_1\Big)t^{3n} + \Big(C_0 + H(1) + C_1\Big)t^{2n} + \Big(H(\infty) + H(1) + C_0\Big)t^n. \end{aligned}$$

The relative cost for this algorithm is

$$\begin{cases} M_2(3n) \leq 3M_2(n) + M_4(n) + 20n - 5 \\ M_4(3n) \leq 5M_4(n) + 56n - 19 \end{cases} \tag{5}$$

However, (5) does not perform well for low degrees as shown in [12] (see table 2). More encouraging is the best case bound, but it requires the following two results.

Result 1 *(From Master Theorem) Let a, b and i be positive integers and assume that $a \neq b$. Let $n = b^i$ and $a \neq 1$. The solution to the inductive relation*

$$\begin{cases} r_1 = e \\ r_n = a r_{n/b} + cn + d \end{cases}$$

is

$$r_n = \left(e + \frac{bc}{a-b} + \frac{d}{a-1}\right) n^{\log_b a} - \frac{bc}{a-b} n - \frac{d}{a-1}.$$

Proof. The proof is trivial. Substituting in the inductive relation the expression for r_n and $r_{n/b}$, we find an identity.

Result 2 *(From Master Theorem) Let a, b and i be positive integers. Let $n = b^i$, $a = b$, $a \neq 1$ and $\delta \neq 1$. The solution to the inductive relation*

$$\begin{cases} r_1 = e \\ r_n = a r_{n/b} + cn + f n^{\delta} + d \end{cases}$$

is

$$r_n = \left(e + \frac{f b^{\delta}}{a - b^{\delta}} + \frac{d}{a-1}\right) n - n^{\delta} \left(\frac{f b^{\delta}}{a - b^{\delta}}\right) + cn \log_b n - \frac{d}{a-1}.$$

Proof. Similar to the previous one

Going back to (5), we can apply the first lemma to the second inequality, getting

$$M_4(n) \leq 30.25n^{1.46} - 28n + 4.75$$

and replacing it in the first inequality, we obtain

$$M_2(3n) \leq 3M_2(n) + 30.25n^{1.46} - 8n - 0.25$$

Finally, using the second lemma we get the best case bound

$$M_2(n) \leq 15.125n^{1.46} - 14.25n - 2.67n \log_3 n + 0.125.$$

3.5 Find and Peralta

In [23], authors develop a new method based on Karatsuba algorithm. They consider kn-bit polynomials F and G, written as $F = F_0 + F_1 t^n + \ldots + F_{k-1} t^{(k-1)n}$ and $G = G_0 + G_1 t^n + \ldots + G_{k-1} t^{(k-1)n}$ for some n-bit polynomials F_i and G_i, $i = 0, \ldots, k-1$.

The sketch of their idea is the following: (a) compute all possible subsets of $\{F_0, F_1, \ldots, F_{k-1}\}$ and $\{G_0, G_1, \ldots, G_{k-1}\}$, excluding the emptyset; (b) take the sum of the elements in every subsets, thus having $2^k - 1$ sums for F and G respectively; (c) multiply the $2^k - 1$ sums for F by the corresponding sum for G — for example, $F_6 + F_8 + F_9$ will be multiplied by $G_6 + G_8 + G_9$ — obtaining H, a set of $2^k - 1$ elements; (d) a computer search gives a minimal subset $\mathcal{H} \subset H$, containing only the elements needed to multiply FG.

For example, if we consider $k = 4$ we get

$$F = F_0 + F_1 t^n + F_2 t^{2n} + F_3 t^{3n} \quad \text{and} \quad G = G_0 + G_1 t^n + G_2 t^{2n} + G_3 t^{3n}.$$

(a)-(b) After computing all possible subsets, the $2^4 - 1$ possible sums for F and G are

$$\begin{aligned}&\{F_0, F_1, F_2, F_3, F_0 + F_1, F_0 + F_2, F_0 + F_3, F_1 + F_2, F_1 + F_3, F_2 + F_3, F_0 + F_1 + F_2,\\ &F_0 + F_1 + F_3, F_0 + F_2 + F_3, F_1 + F_2 + F_3, F_0 + F_1 + F_2 + F_3\}\end{aligned}$$

$$\begin{aligned}&\{G_0, G_1, G_2, G_3, G_0 + G_1, G_0 + G_2, G_0 + G_3, G_1 + G_2, G_1 + G_3, G_2 + G_3, G_0 + G_1 + G_2,\\ &G_0 + G_1 + G_3, G_0 + G_2 + G_3, G_1 + G_2 + G_3, G_0 + G_1 + G_2 + G_3\}\end{aligned}$$

(c) It is straightforward and give us

$$H = \{H_0, H_1, H_2, H_3, H_{01}, H_{02}, H_{03}, H_{12}, H_{13}, H_{23}, H_{123}, H_{023}, H_{013}, H_{012}, H_{0123}\}$$

where $H_{i_1 \ldots i_k} = (F_{i_1} + \ldots + Fi_k)(G_{i_1} + \ldots + Gi_k)$.
(d) Now, a computer search will give the following

$$\mathcal{H} = \{H_0, H_1, H_{01}, H_2, H_{02}, H_3, H_{13}, H_{23}, H_{0123}\},$$

The elements of $\mathcal{H}$ are the only ones needed to multiply FG. Then, the authors split each $H_{i_1 \ldots i_k}$ in three parts, say H_L, H_M and H_H, and find the SLPs that compute $f(x) = (H_M)x$ and $f(x) = (H_L, H_H)x$ over $GF(2)$.

Calling $M_\wedge(\mathcal{C})$ the cardinality of $\mathcal{H}$, $s(T)$ the number of operations needed to compute all the sums of the form $F_{i_1} + \ldots + Fi_k$, $s(R)$ the number of operations of a SLP that computes $f(x) = (H_M)x$ over $GF(2)$, and $s(E)$ the number of operations of a SLP that computes $f(x) = (H_L, H_H)x$ over $GF(2)$, the general estimate for multiplying two kn-bit polynomials will be

$$M(kn) \leq n_\wedge M(n) + 2n \cdot s(T) + (n-1) \cdot s(E) + s(R).$$

Setting $k = 4, 5, 6, 7, \ldots$, we get

$$\begin{aligned} &M(4n) \leq 9M(n) + 34n - 12 \\ &M(5n) \leq 13M(n) + 54n - 19 \\ &M(6n) \leq 17M(n) + 85n - 29 \\ &M(7n) \leq 22M(n) + 107n - 33 \\ &\ldots \end{aligned} \tag{6}$$

Notice that finding the number of operations of a SLP that computes $f(x) = (H_M)x$ and $f(x) = (H_L, H_H)x$ over $GF(2)$ may require heavy use of HW resources.

4 Our contribution

In this section, we define a more efficient algorithm rearranging the order of operations and improve the general complexity through best case bounds. In the sequel, we will denote these two approaches with **(I)** and **(II)** respectively.

4.1 Improvements of Two-level Seven-way (I)

We can now give an improvement of the preceding algorithm for higher degrees. In fact, we consider polynomials of $8n$ bits and apply the same technique of the Two-level Seven-way Recursion. We can collect t^{4n}, apply the Refined Karatsuba and apply Two-level Seven-way Recursion for inner multiplication. We will call the following algorithm *Three-level Recursion.*

$$\begin{aligned}
&\left(\textstyle\sum_{i=0}^{7} t^{in} F_i\right)\left(\textstyle\sum_{i=0}^{7} t^{in} G_i\right) \\
&= \left(\textstyle\sum_{i=0}^{3} t^{in} F_i + t^{4n}\sum_{i=0}^{3} t^{in} F_{i+4}\right)\left(\textstyle\sum_{i=0}^{3} t^{in} G_i + t^{4n}\sum_{i=0}^{3} t^{in} G_{i+4}\right) \\
&= (1+t^{4n})\Bigg(\left(\textstyle\sum_{i=0}^{3} t^{in} F_i\right)\left(\textstyle\sum_{i=0}^{3} t^{in} G_i\right) + t^{4n}\left(\textstyle\sum_{i=0}^{3} t^{in} F_{i+4}\right)\left(\textstyle\sum_{i=0}^{3} t^{in} G_{i+4}\right)\Bigg) + \\
&\quad t^{4n}\left(\textstyle\sum_{i=0}^{3} t^{in} F_i + \sum_{i=0}^{3} t^{in} F_{i+4}\right)\left(\textstyle\sum_{i=0}^{3} t^{in} G_i + \sum_{i=0}^{3} t^{in} G_{i+4}\right) \\
&= (1+t^{4n})\Bigg(\left(\textstyle\sum_{i=0}^{3} t^{in} F_i\right)\left(\textstyle\sum_{i=0}^{3} t^{in} G_i\right) + t^{4n}\left(\textstyle\sum_{i=0}^{3} t^{in} F_{i+4}\right)\left(\textstyle\sum_{i=0}^{3} t^{in} G_{i+4}\right)\Bigg) + \\
&\quad t^{4n}\left(\textstyle\sum_{i=0}^{3} t^{in} (F_i + F_{i+4})\right)\left(\textstyle\sum_{i=0}^{3} t^{in} (G_i + G_{i+4})\right)
\end{aligned}$$

$$
\begin{aligned}
&= (1+t^{4n})\Bigg((1+t^{2n})\Bigg((1+t^n)\left(\textstyle\sum_{i=0}^{7} t^{in}F_iG_i\right) + \\
&\textstyle\sum_{j=0}^{3} t^{(2j+1)n}(F_{2j}+F_{2j+1})(G_{2j}+G_{2j+1})\Bigg) + \\
&+t^{2n}(F_0+F_2+(F_1+F_3)t^n)(G_0+G_2+(G_1+G_3)t^n) + \\
&+t^{6n}(F_4+F_6+(F_5+F_7)t^n)(G_4+G_6+(G_5+G_7)t^n)\Bigg) + \\
&t^{4n}\left(\textstyle\sum_{i=0}^{3} t^{in}(F_i+F_{i+4})\right)\left(\textstyle\sum_{i=0}^{3} t^{in}(G_i+G_{i+4})\right)
\end{aligned}
$$

The cost evaluation for polynomials with $7n+k$ coefficients, assuming $n/2 \leq k \leq n$, is

- $7M(n)$: multiplication F_iG_i, for $i=0,\dots,6$
- $M(k)$: multiplication F_7 by G_7
- $7(n-1)$: sum $S_1 = \sum_{i=0}^{7} t^{in}F_iG_i$
- $6n+2k-1$: sum $S_2 = (1+t^n)S_1$
- $3(2n+M(n))$: multiplication $(F_{2j}+F_{2j+1})(G_{2j}+G_{2j+1})$, for $j=0,1,2$
- $2k+M(n)$: multiplication $(F_6+F_7)(G_6+G_7)$
- $4(2n-1)$: sum $S_3 = S_2 + \sum_{j=0}^{3} t^{(2j+1)n}(F_{2j}+F_{2j+1})(G_{2j}+G_{2j+1})$
- $6n+2k-1$: sum $S_4 = (1+t^{2n})S_3$
- $4n+M(2n)$: multiplication $S_5 = (F_0+F_2+(F_1+F_3)t^n)(G_0+G_2+(G_1+G_3)t^n)$
- $2n+2k+M(2n)$: multiplication $S_6 = (F_4+F_6+(F_5+F_7)t^n)(G_4+G_6+(G_5+G_7)t^n)$
- $2(4n-1)$: sum $S_7 = S_4 + t^{2n}S_5 + t^{6n}S_6$
- $6n+2k-1$: sum $S_8 = (1+t^{4n})S_7$
- $6n+2k+M(4n)$: multiplication $S_9 = \left(\sum_{i=0}^{3} t^{in}(F_i+F_{i+4})\right)\left(\sum_{i=0}^{3} t^{in}(G_i+G_{i+4})\right)$
- $8n-1$: sum $S_8 + t^{4n}S_9$

Hence, summing all the costs, we get

$$
M(7n+k) \leq M(4n) + 2M(2n) + 11M(n) + M(k) + 67n + 12k - 17 \qquad n/2 \leq k \leq n
$$

One could continue in the same fashion of the *Three-level*, consider polynomials of $2^k n$ bits, collect $t^{2^{k-1}n}$, apply the *Refined Karatsuba* and the *"k-1"-level Recursion*. We are going to see that this is not a totally right way.

We want to see which kind of improvements are given from algorithms of the Section 3.3. They are of two types: the best case bound (for n large enough) and concrete (only on low degree). Lemma 1 will help us to state the best case bounds.

If we go back to the recursion (2), we see that, when k is equal to n, it could be rewritten as

$$
M(2n) \leq 3M(n) + 7n - 3 \tag{7}
$$

so, also as

$$
M(n) \leq 3M(n/2) + \frac{7}{2}n - 3.
$$

We can now apply Lemma 1, finding

$$
M(n) \leq 6.5n^{\log_2 3} - 7n + 1.5
$$

What about (3)? If we state $k = n$, we get

$$M(4n) \leq M(2n) + 6M(n) + 27n - 8$$

so, we cannot apply Lemma 1, but if we substitute $M(2n)$ with the recursion formula (7), we find

$$M(4n) \leq 9M(n) + 34n - 11 \tag{8}$$

finally, we obtain

$$M(n) \leq 6.43n^{\log_2 3} - 6.8n + 1.38$$

Notice that (8) is not the best known, in fact, in [23] we can find

$$M(4n) \leq 9M(n) + 34n - 12 \tag{9}$$

so, for higher levels of recursion we will use (9) instead of (8).

To enable an easy comparison of different algorithms, in Table 1 we present the the best case bounds. Notice that the first and the third coefficients of each estimation are decreasing, instead the second one is growing.

Algorithm	Best case bound	Number of bits
[5]*Refined Karatsuba*	$M(n) \leq 6.50n^{\log_2 3} - 7.00n + 1.50$	$n = 2^x$
[5]*Two-level Seven-way*	$M(n) \leq 6.43n^{\log_2 3} - 6.80n + 1.38$	$n = 4^x$
[23]*4-way split*	$M(n) \leq 6.30n^{\log_2 3} - 6.80n + 1.50$	$n = 4^x$
Three-level	$M(n) \leq 6.34n^{\log_2 3} - 6.68n + 1.35$	$n = 8^x$
Four-level	$M(n) \leq 6.30n^{\log_2 3} - 6.62n + 1.31$	$n = 16^x$
Five-level	$M(n) \leq 6.28n^{\log_2 3} - 6.57n + 1.30$	$n = 32^x$

Table 1. Best case bounds: comparison of different algorithms

By exploiting the recursion formulae, we can also improve the cost of the multiplication between two polynomials of low degree (see Table 2).

4.2 Product in finite fields: general case (II)

There are several approaches that can be adopted to multiply two polynomials, say F and G, in an efficient way. In this section we provide a new one. In doing so, we make some useful assumptions. We take d a non negative integer and the factors F and G of the form

$$F(t) = \sum_{i=0}^{2^{d-1}} F_i(t)t^{in} \qquad \text{with } F_i \in \mathbb{F}_2[t],\ \deg F_i \leq n-1$$

In order to simplify notation, given a factor $F(t)$ of the above form, we define

$$\widetilde{F}(x) = \sum_{i=0}^{2^{d-1}} F_i(t)x^i$$

We are now ready to suggest a new efficient algorithm.

n	Best known	Our contribution	Gates gained	Depth best known	Depth of our contribution	Depth gained	Algorithm used
24	702 [12]	697	5	10	9	1	3-lev
32	1156 [12]	1148	8	11	10	1	3-lev
40	1703 [23]	1700	3	14	13	1	3-lev
47	2228 [23]	2214	14	13	11	2	4-lev
48	2259 [23]	2238	21	13	11	2	4-lev
63	3626 [23]	3612	14	14	12	2	4-lev
64	3673 [23]	3640	23	13	12	1	4-lev
72	4510 [23]	4510	0	25	15	10	3-lev
79	5329 [23]	5313	16	16	15	1	4-lev
80	5366 [23]	5345	21	16	15	1	4-lev
95	7073 [23]	6978	95	15	13	2	5-lev
96	7110 [23]	7006	104	16	13	3	5-lev
120	10438 [5]	10294	144	130	17	113	3-lev
127	11447 [5]	11277	170	17	14	3	5-lev
128	11466 [12]	11309	157	16	14	2	5-lev

Table 2. Improvements of $M(1) - M(128)$: we apply Three-, Four-, and Five-level Recursion algorithm

Let's start with an observation. There is an interesting connection between $x^{2^d} + x$ and Lagrange polynomials. Indeed, we can prove the following three equalities:

1. $l_0(x) = \dfrac{x^{2^d} + x}{x} = x^{2^d-1} + 1$
2. $l_{\alpha^i}(x) = \dfrac{x^{2^d} + x}{x + \alpha^i} \qquad i = 0, 1, \ldots, 2^d - 2$
3. $l_\infty = x^{2^d} + x = x(x^{2^d-1} + 1) = x \cdot l_0(x)$

We now rewrite the interpolation law as follows:

$$\widetilde{H}(x) = \widetilde{H}(0) \cdot l_0(x) + \sum_{i=0}^{2^d-2} \widetilde{H}(\alpha^i) \cdot l_{\alpha^i}(x) + \widetilde{H}(\infty) \cdot l_\infty(x)$$

$$\widetilde{H}(x) = \widetilde{H}(0) \cdot l_0(x) + \sum_{i=0}^{2^d-2} \widetilde{H}(\alpha^i) \cdot l_{\alpha^i}(x) + x\widetilde{H}(\infty) \cdot l_0(x)$$

$$\widetilde{H}(x) = \widetilde{H}(0) \cdot (1 + x^{2^d-1}) + \sum_{i=0}^{2^d-2} \widetilde{H}(\alpha^i) \frac{x^{2^d} + x}{x + \alpha^i} + x\widetilde{H}(\infty) \cdot (1 + x^{2^d-1})$$

$$\widetilde{H}(x) = (1 + x^{2^d-1})(\widetilde{H}(0) + x\widetilde{H}(\infty)) + \sum_{i=0}^{2^d-2} \widetilde{H}(\alpha^i) \frac{x^{2^d} + x}{x + \alpha^i} \tag{10}$$

Notice that fractions $\frac{x^{2^d}+x}{x+\alpha^i}$ of Equation (10) are Lagrange polynomials of $\mathbb{F}_{2^d}^\times$. Using the naive division algorithm, we obtain

$$l_{\alpha^i}(x) = \frac{x^{2^d} + x}{x + \alpha^i} = \sum_{j=1}^{2^d-1} (\alpha^i)^{(j-1)} x^{2^d-j} \tag{11}$$

and replacing Equation (11) in (10), we get

$$\widetilde{H}(x) = (1 + x^{2^d-1})(\widetilde{H}(0) + x\widetilde{H}(\infty)) + \sum_{i=0}^{2^d-2} \widetilde{H}(\alpha^i) \sum_{j=1}^{2^d-1} \alpha^{i(j-1)} x^{2^d-j}$$

$$\widetilde{H}(x) = \underbrace{(1 + x^{2^d-1})(\widetilde{H}(0) + x\widetilde{H}(\infty))}_{S_A} + \underbrace{\sum_{j=1}^{2^d-1} \left(\sum_{i=0}^{2^d-2} \alpha^{i(j-1)} \widetilde{H}(\alpha^i) \right) x^{2^d-j}}_{S_B} \tag{12}$$

We will now discuss the costs of this algorithm.

Consider S_A: it will always be the same in every field $\mathbb{F}_{2^d}$. The cost of the operations in S_A is:

- $M_2(n)$: multiplication $\widetilde{H}(0) = F_0 G_0$
- $M_2(n)$: multiplication $\widetilde{H}(\infty) = F_{2^{d-1}} G_{2^{d-1}}$
- $n - 1$: sum $\widetilde{H}(0) + x\widetilde{H}(\infty)$
- 0: sum $(1 + x^{2^d-1})(\widetilde{H}(0) + x\widetilde{H}(\infty))$

The last estimate holds only for $d \neq 1$, otherwise polynomials $\widetilde{H}(0)+x\widetilde{H}(\infty)$ and $x(\widetilde{H}(0)+x\widetilde{H}(\infty))$ overlap on some bits and it becomes $2n - 1$.

Consider now the sum $S_A + S_B$. The degree of S_A is $(2^d + 2)n - 2$, but its structure lacks many powers. Indeed, S_A is a polynomial that has two parts, the first with powers whose degrees are running from 0 to $3n - 2$, the second from $(2^d - 1)n$ to $(2^d + 2)n - 2$. This is very useful because S_B has powers with degrees from n to $(2^d + 1)n - 2$, so, S_A and S_B overlaps only in two parts. The first in $(3n - 2) - n + 1 = 2n - 1$ bits and the second in $(2^d + 1)n - 2 - (2^d - 1)n + 1 = 2n - 1$. Since the cost of $S_A + S_B$ does not depend on the field, it is

- $4n - 2$: sum $H(t) = S_A + S_B$

Finally, consider the sums in S_B. Supposing that the internal summation has been computed, the external one is conducted over $2^d - 1$ polynomials. These polynomials have powers from cn to $cn + 2n - 2$, with $c = 1, \ldots, 2^d - 1$ and each one overlaps the following on $n - 1$ bit. Therefore, the cost of the external sum in S_B is

- $(2^d - 2)(n - 1)$: sum $S_1 x + S_2 x^2 + \cdots + S_{2^d-1} x^{2^d-1}$

We are left to compute the internal sums in S_B. We will show that we do not need to compute all $\widetilde{H}(\alpha^i)$.

Firstly, we start with showing that if $i = 2^q i'$ for some q, then there will be a connection between the coefficients of $\widetilde{H}(\alpha^i)$ and $\widetilde{H}(\alpha^{i'})$.

Theorem 1. *If we take integers i and i' such that $i' = 2^q i$ for some q, then we can express the coefficients of $\widetilde{H}(\alpha^{i'})$ as a linear combination of the coefficients of $\widetilde{H}(\alpha^i)$.*

Proof. We have

$$\widetilde{H}(\alpha^i) = \widetilde{F}(\alpha^i) \cdot \widetilde{G}(\alpha^i) = \sum_{j=0}^{2^{d-1}} F_j \alpha^{ij} \sum_{k=0}^{2^{d-1}} G_k \alpha^{ik} = \sum_{l=0}^{2^d} \left(\sum_{\substack{j+k=l \\ 0 \leq j,k \leq 2^{d-1}}} F_j G_k \right) (\alpha^i)^l.$$

We define

$$H_l = \sum_{\substack{j+k=l \\ 0\leq j,k\leq 2^{d-1}}} F_j G_k$$

thus

$$\widetilde{H}(\alpha^i) = \sum_{l=0}^{2^d} H_l \alpha^{il} \tag{13}$$

Remember that the field $\mathbb{F}_{2^d}$ can be viewed as vector space over $\mathbb{F}_2$. So, we can write every power of α as a linear combination of the elements of the basis $\{1, \alpha, \alpha^2, \ldots, \alpha^{d-1}\}$

$$\alpha^{il} = \sum_{b=0}^{d-1} c_{b,il}\alpha^b \tag{14}$$

and substitute (14) in (13), getting

$$\widetilde{H}(\alpha^i) = \sum_{l=0}^{2^d} H_l \sum_{b=0}^{d-1} c_{b,il}\alpha^b = \sum_{b=0}^{d-1}\left(\sum_{l=0}^{2^d} H_l c_{b,il}\right)\alpha^b$$

Take now $\widetilde{H}(\alpha^{iw})$ with $w > 1$, from (14) we have

$$\alpha^{ilw} = (\alpha^{il})^w = \left(\sum_{b=0}^{d-1} c_{b,il}\alpha^b\right)^w .$$

In order to write coefficients of $\widetilde{H}(\alpha^{iw})$ as linear combinations of the coefficients of $\widetilde{H}(\alpha^i)$, we need the following equality:

$$\left(\sum_{b=0}^{d-1} c_{b,il}\alpha^b\right)^w = \sum_{b=0}^{d-1} c_{b,il}\alpha^{bw} \tag{15}$$

Suppose it holds, then

$$\widetilde{H}(\alpha^{iw}) = \sum_{l=0}^{2^d} H_l \left(\sum_{b=0}^{d-1} c_{b,il}\alpha^b\right)^w = \sum_{l=0}^{2^d} H_l \sum_{b=0}^{d-1} c_{b,il}\alpha^{bw} = \sum_{b=0}^{d-1}\left(\sum_{l=0}^{2^d} H_l c_{b,il}\right)\alpha^{bw}.$$

Finally, using (14), we obtain

$$\widetilde{H}(\alpha^{iw}) = \sum_{b=0}^{d-1}\left(\sum_{l=0}^{2^d} H_l c_{b,il}\right)\alpha^{bw} = \sum_{b=0}^{d-1}\left(\sum_{l=0}^{2^d} H_l c_{b,il}\right)\sum_{t=0}^{d-1} c_{t,bw}\alpha^t =$$
$$= \sum_{t=0}^{d-1}\left(\sum_{b=0}^{d-1} c_{t,bw}\left(\sum_{l=0}^{2^d} H_l c_{b,il}\right)\right)\alpha^t$$

Let's go back to (15): since we are in characteristic two, the equality holds when $w = 2^q$, for some q.

Secondly, we have to remember that $\alpha^{2^d} = \alpha$. So, for every $\widetilde{H}(\alpha^i)$, with $i \not\equiv 0 \bmod 2^d - 1$, there are at most d different evaluations of $\widetilde{H}$ that can be computed with $\widetilde{H}(\alpha^i)$. They are the following set:

$$P_i = \{\widetilde{H}(\alpha^i), \widetilde{H}(\alpha^{2i}), \widetilde{H}(\alpha^{2^2 i}), \ldots, \widetilde{H}(\alpha^{2^{d-1} i})\}$$

We can count the number of P_i for every algebraic extension of $\mathbb{F}_2$, because it depends only on the degree d.

Theorem 2. *The number of different P_i is*

$$P = -1 + \frac{1}{d} \sum_{k=0}^{d-1} \gcd(2^k - 1, 2^d - 1)$$

In particular, if $2^d - 1$ is prime, $P = (2^d - 2)/d$.

We define an action of the (additive) group $\mathbb{Z}$ on $\mathbb{Z}/(2^d - 1)\mathbb{Z}$ as $k \cdot i = 2^k i$. Since d acts trivially, this action induces an action of $\mathbb{Z}/d\mathbb{Z}$ on $\mathbb{Z}/(2^d - 1)\mathbb{Z}$: if $O(i)$ is the orbit of $i \in \mathbb{Z}/(2^d - 1)\mathbb{Z}$, then $P_i = \{\widetilde{H}(\alpha^j) : j \in O(i)\}$. We have a trivial orbit $O(0) = \{0\}$ which would correspond to the set $P_0 = \{\widetilde{H}(1)\}$ which we will not count. In order to prove the Theorem 2, we need a couple of additional lemmata.

Lemma 1 (Burnside's Lemma). *If the finite group G acts on the finite set X, then the number of orbits is*

$$\frac{1}{\# G} \sum_{g \in G} \# \operatorname{Fix}(g)$$

where $\operatorname{Fix}(g) = \{x \in X : g \cdot x = x\}$.

Proof. See [36], chapter 3.

Lemma 2. *Fix an integer N and let $x \in \mathbb{Z}/N\mathbb{Z}$. Then*

$$\#\{y \in \mathbb{Z}/N\mathbb{Z} : xy = 0\} = \gcd(x, N)$$

Proof. Let $\mathcal{Z} = \{y \in \mathbb{Z}/N\mathbb{Z} : xy = 0\}$: it is not empty since it includes 0 and it is straightforward to verify that $\mathcal{Z}$ is an ideal in $\mathbb{Z}/N\mathbb{Z}$, thus $\mathcal{Z} = \langle d \rangle$ where d is a divisor of N and $\mathcal{Z}$ has N/d elements. Let $D = \gcd(x, N)$, $\nu = N/D$ and define $\tilde{x}$ as the smallest positive integer such that $\tilde{x} \equiv x \bmod N$. Since

$$\nu x = \frac{N}{D} x \equiv N \frac{\tilde{x}}{D} \equiv 0 \bmod N$$

we have that $\nu \in \mathcal{Z}$. Viceversa, if $y \in \mathcal{Z}$ and $\tilde{y}$ is the smallest positive integer such that $\tilde{y} \equiv y \bmod N$, we have that $\tilde{y}\tilde{x} = kN$ for some integer $k \geq 0$. Thus

$$\tilde{y} \frac{\tilde{x}}{D} = k \frac{N}{D} = k\nu; \quad \text{i.e.,} \quad \tilde{y} \frac{\tilde{x}}{D} \equiv 0 \bmod \nu$$

Since $\tilde{x}/D$ and $\nu = N/D$ are relatively prime, this implies $\tilde{y} \equiv 0 \bmod \nu$, i.e., ν divides $\tilde{y}$, thus $y \in \langle \nu \rangle$. This shows that $\mathcal{Z} = \langle \nu \rangle$, hence that $\#\mathcal{Z} = N/\nu = \gcd(x, N)$.

Proof (Theorem 2). Fix $k \in \mathbb{Z}/d\mathbb{Z}$: we want to compute $\operatorname{Fix}(k) = \{x \in \mathbb{Z}/(2^d - 1)\mathbb{Z} : k \cdot x = x\}$. If $x \in \operatorname{Fix}(k)$ then $2^k x = x$, that is $(2^k - 1)x = 0$; and, viceversa, if $(2^k - 1)x = 0$ then $k \cdot x = x$. Hence, $\operatorname{Fix}(k) = \{x \in \mathbb{Z}/(2^d - 1)\mathbb{Z} : (2^k - 1)x = 0\}$ has, by the previous lemma, $\gcd(2^k - 1, 2^d - 1)$ elements.

The thesis now follows from Burnside's Lemma.

Let's sum up the costs of Equation (12).

- $M_2(n)$: multiplication $\widetilde{H}(0) = F_0G_0$
- $M_2(n)$: multiplication $\widetilde{H}(\infty) = F_{2^{d-1}}G_{2^{d-1}}$
- $n-1$: sum $\widetilde{H}(0) + x\widetilde{H}(\infty)$
- 0: sum $(1 + x^{2^d-1})(\widetilde{H}(0) + x\widetilde{H}(\infty))$
- $4n-2$: sum $H(t) = S_A + S_B$
- $(2^d-2)(n-1)$: sums $S_1x + S_2x^2 + \cdots + S_{2^d-1}x^{2^d-1}$
- Δ_1: evaluation $\widetilde{F}(\alpha^i)$, $\widetilde{G}(\alpha^i)$
- $M_2(n)$: multiplication $\widetilde{H}(1)$
- $PM_{2^d}(n)$: multiplications $\widetilde{H}(\alpha^i)$
- Δ_2: sums S_i, $i = 1, \ldots, 2^d-1$

Some of the previous costs are left blank, in particular Δ_1 and Δ_2, since the evaluation of F, G and the sums S_i depends on the polynomial used to generate the field $\mathbb{F}_{2^d}$. Roughly speaking, we can say that $\Delta_1 = An$ and $\Delta_2 = B(2n-1)$, obtaining the following estimation:

$$M((2^{d-1}+1)n) \leq 3M_2(n) + PM_{2^d}(n) + \underbrace{(2^d + 3 + A + 2B)}_{Q_1} n + \underbrace{(-1 - 2^d - B)}_{Q_2}$$

$$M((2^{d-1}+1)n) \leq 3M_2(n) + PM_{2^d}(n) + Q_1n + Q_2 \tag{16}$$

Now, we want to apply the following result.

Result 3 *(From Master Theorem) Let a and b be positive real numbers with $a \geq 1$ and $b \geq 2$. Let $T(n)$ be defined by*

$$T(n) = \begin{cases} aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + f(n) & n > 1 \\ d & n = 1 \end{cases}$$

Then

1. *if $f(n) = \Theta(n^c)$ where $\log_b a < c$, then $T(n) = \Theta(n^c) = \Theta(f(n))$,*
2. *if $f(n) = \Theta(n^c)$ where $\log_b a = c$, then $T(n) = \Theta(n^{\log_b a} \log_b n)$,*
3. *if $f(n) = \Theta(n^c)$ where $\log_b a > c$, then $T(n) = \Theta(n^{\log_b a})$.*

The same results apply with ceilings replaced by floors.

Proof. See [32], Section 5.2.

We cannot apply Theorem 3 to (16) since both M_2 and M_{2^d} appear: we will have to move everything down to $\mathbb{F}_2$-operations.

4.3 Bit operations and asymptotic estimation (II)

As seen in Section 4.2, we need to evaluate an $\mathbb{F}_{2^d}$-polynomial $\widetilde{F}$ of degree $2^d - 1$. Recall that the field $\mathbb{F}_{2^d}$ can be seen as an $\mathbb{F}_2$-vector space of dimension d. Thus, for all i, we can evaluate $\widetilde{F}(\alpha^i)$ as follows:

$$\widetilde{F}(\alpha^i) = \sum_{j=0}^{d-1} F_j\alpha^j \qquad F_j \in \mathbb{F}_2[t]$$

To compute $\widetilde{H}(\alpha^i)$ we need to multiply the two evaluations of $\widetilde{F}$ and $\widetilde{G}$.

$$\widetilde{H}(\alpha^i) = \widetilde{F}(\alpha^i)\widetilde{G}(\alpha^i) = \sum_{j=0}^{d-1} F_j \alpha^j \sum_{k=0}^{d-1} G_k \alpha^k = \sum_{l=0}^{2d-2} \underbrace{\left(\sum_{\substack{j+k=l \\ 0\le j,k\le d-1}} F_j G_k \right)}_{H_l} \alpha^l$$

We want now to compute H_l. We take care only of multiplications. If we look at H_l, we note that it is formed by the sum of the products between F_j and G_k such that $j+k=l$. We separate the two cases: $j=k$ and $j\neq k$. If $j=k$, we need the multiplication F_jG_j. If $j\neq k$, we need two multiplications, which are F_jG_k and F_kG_j. For the latter, we exchange one multiplication with four sums, since $\operatorname{char}\mathbb{F}_{2^d}=2$ and we have already computed F_jG_j.

$$F_jG_k + F_kG_j = (F_j+F_k)(G_j+G_k) + F_jG_j + F_kG_k$$

The required multiplications are

$$d + \binom{d}{2} = d + \frac{d(d-1)}{2} = \frac{d^2+d}{2}.$$

Now, we can write the estimation for bit calculations over $\mathbb{F}_{2^d}$, assuming a generic estimate for the number of bit additions

$$M_{2^d}(n) \le \frac{d^2+d}{2} M_2(n) + Cn + D \tag{17}$$

Substituting (17) in the estimation (16), we obtain a formula which we can apply Theorem 3 to:

$$M_2((2^{d-1}+1)n) \le 3M_2(n) + P\left(\frac{d^2+d}{2}M_2(n) + Cn + D\right) + Q_1 n + Q_2$$

$$M_2((2^{d-1}+1)n) \le \left(3 + \frac{P(d^2+d)}{2}\right) M_2(n) + (Q_1 + CP)n + (Q_2 + DP)$$

Applying the third case of Theorem 3, we get:

$$M_2(n) = \Theta\left(n^E\right), \quad \text{where} \quad E = \frac{\log\left(3 + \frac{P(d^2+d)}{2}\right)}{\log(2^d+1)}$$

If we compute the exponent E for $1\le d\le 20$, it is not difficult to see that E decreases from 1.58 to 1.17.

4.4 Case d=2 (II)

Using Equation (12), we are able to find a better best case bound than that presented in [12] (see CNH 3-way split algorithm (24)). Indeed,

- $M_2(n)$: multiplication $\widetilde{H}(0) = F_0G_0$
- $M_2(k)$: multiplication $\widetilde{H}(\infty) = F_2G_2$
- $2k$: sums $S_1 = F_0 + F_2$, $S_2 = G_0 + G_2$
- $2k$: sums $S_3 = F_1 + F_2$, $S_4 = G_1 + G_2$
- $2n$: sums $S_5 = S_1 + F_1$, $S_6 = S_2 + G_1$

- 0: multiplications $P_1 = \alpha S_3$, $P_2 = \alpha S_3$
- 0: sums $S_7 = S_1 + P_1$, $S_8 = S_2 + P_2$
- $M_2(n)$: multiplication $\widetilde{H}(1) = S_5 S_6$
- $M_4(n)$: multiplication $\widetilde{H}(\alpha) = S_7 S_8 (= C_0 + C_1\alpha)$
- $2n-1$: sum $S_9 = \widetilde{H}(1) + C_1$
- $2n-1$: sum $S_{10} = S_9 + C_0$
- $2n-1$: sum $S_{11} = S_{10} + C_1$
- $2(n-1)$: sums $S_{12} = S_9 x^3 + S_{10} x^2 + S_{11} x$
- $n-1$: sum $S_{13} = \widetilde{H}(0) + x\widetilde{H}(\infty)$
- 0: sum $S_{14} = (1 + x^3) S_{13}$
- $4n-2$: sum $H = S_{14} + S_{12}$

Summing all the costs, we obtain

$$\begin{cases} M(2n+k) \leq 2M_2(n) + M_2(k) + M_4(n) + 15n + 4k - 8 & n/2 \leq k \leq n \\ M(3n) \leq 3M_2(n) + M_4(n) + 19n - 8 & k = n \end{cases} \tag{18}$$

But this is not enough. In order to get the best case bound, we have to compute the costs for the same algorithm that uses polynomials over $\mathbb{F}_4$. In this case, we cannot deduce the expression for $\widetilde{H}(\alpha+1)$ from $\widetilde{H}(\alpha)$. In addition, from equation

$$\alpha(a_0 + a_1\alpha) = a_1 + (a_0 + a_1)\alpha$$

we have that the cost of the multiplication by α is 1, and from

$$(a_0 + a_1\alpha) + (b_0 + b_1\alpha) = (a_0 + b_0) + (a_1 + b_1)\alpha$$

we have that the cost of the sum between two polynomials is doubled. Thus,

- $M_4(n)$: multiplication $\widetilde{H}(0) = F_0 G_0$
- $M_4(n)$: multiplication $\widetilde{H}(\infty) = F_2 G_2$
- $4n$: sums $S_1 = F_0 + F_1$, $S_2 = G_0 + G_1$
- $4n$: sums $S_3 = F_1 + F_2$, $S_4 = G_1 + G_2$
- $2n$: multiplications $P_1 = \alpha S_3$, $P_2 = \alpha S_4$
- $4n$: sums $S_5 = S_1 + P_1$, $S_6 = S_2 + P_2$
- $4n$: sums $S_7 = S_5 + S_3$, $S_8 = S_6 + S_4$
- $4n$: sums $S_9 = S_1 + F_2$, $S_{10} = S_2 + G_2$
- $M_4(n)$: multiplication $\widetilde{H}(1) = S_9 S_{10}$
- $M_4(n)$: multiplication $\widetilde{H}(\alpha) = S_7 S_8$
- $M_4(n)$: multiplication $\widetilde{H}(\alpha+1) = S_5 S_6$
- $8n-4$: sum $S_{13} = \widetilde{H}(1) + \widetilde{H}(\alpha) + \widetilde{H}(\alpha+1)$
- $10n-5$: sum $S_{14} = \widetilde{H}(1) + \widetilde{H}(\alpha+1) + \alpha(\widetilde{H}(\alpha) + \widetilde{H}(\alpha+1))$
- $4n-2$: sum $S_{15} = \widetilde{H}(1) + \widetilde{H}(\alpha) + \alpha(\widetilde{H}(\alpha) + \widetilde{H}(\alpha+1))$
- $4(n-1)$: sums $S_{16} = S_{13} x^3 + S_{14} x^2 + S_{15} x$
- $2(n-1)$: sum $S_{17} = \widetilde{H}(0) + x\widetilde{H}(\infty)$
- 0: sum $S_{18} = (1 + x^3) S_{17}$
- $8n-4$: sum $H = S_{18} + S_{16}$

The sum of the costs in $\mathbb{F}_4$ is

$$M_4(3n) \le 5M_4(n) + 58n - 21$$

We observe that this is not good as

$$M_4(3n) \le 5M_4(n) + 56n - 19 \tag{19}$$

which can be found in [12]. Applying Lemma 1 to (19), we get

$$M_4(n) \le 30.25n^{1.46} - 28n + 4.75$$

Then, we substitute the preceding inequality to the second of (18) obtaining

$$M_2(3n) \le 3M_2(n) + 30.25n^{1.46} - 9n - 3.25$$

Finally, to get the best case bound, we apply Lemma 2:

$$M_2(n) \le 15.125n^{1.46} - 3n \log_3 n - 15.75n + 1.625.$$

5 Conclusions

In this paper, we presented a new algorithm to multiply two n-bit polynomials. We showed how this new approach can be used to (a) reduce the effective number of bit operations and (b) improve the asymptotic estimations.

The idea described in this paper can be easily implemented to speed up cryptographic software implementations. Notice that further improvements might be obtained avoiding some redundant XOR operations involved in the multiplication algorithms [5]. For example, it is possible to apply a greedy heuristic [33,11,39] to a straight-line sequence such as the one provided in Appendix A. Unfortunately, this approach is computational expensive and often it does not provide a useful result in an acceptable amount of time.

References

1. Abdulrahman, E.A.H., Reyhani-Masoleh, A.: High-speed hybrid-double multiplication architectures using new serial-out bit-level mastrovito multipliers. IEEE Transactions on Computers 65(6), 1734–1747 (2016)
2. Agnew, G.B., Beth, T., Mullin, R.C., Vanstone, S.A.: Arithmetic operations in $GF(2^m)$. Journal of Cryptology 6(1), 3–13 (1993)
3. Berlekamp, E.R.: Algebraic coding theory, vol. 111. McGraw-Hill New York (1968)
4. Bernstein, D.J.: Curve25519: New diffie-hellman speed records. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) Public Key Cryptography - PKC 2006: 9th International Conference on Theory and Practice in Public-Key Cryptography, New York, NY, USA, April 24-26, 2006. Proceedings, pp. 207–228. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
5. Bernstein, D.J.: Batch binary edwards. In: Halevi, S. (ed.) Advances in Cryptology - CRYPTO 2009: 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings, pp. 317–336. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)

6. Bernstein, D.J.: High-speed cryptography in characteristic 2: Minimum number of bit operations for multiplication (2009), `http://binary.cr.yp.to/m.html`
7. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. Journal of Cryptographic Engineering 2(2), 77–89 (2012)
8. Blahut, R.E.: Theory and practice of error control codes, vol. 126. Addison-Wesley Reading Massachusetts (1983)
9. Blahut, R.E.: Fast algorithms for digital signal processing. Addison-Wesley Longman Publishing Co., Inc. (1985)
10. Blake, I., Seroussi, G., Smart, N.: Elliptic curves in cryptography, vol. 265. Cambridge university press (1999)
11. Boyar, J., Peralta, R.: A new combinational logic minimization technique with applications to cryptology. In: Festa, P. (ed.) Experimental Algorithms: 9th International Symposium, SEA 2010, Ischia Island, Naples, Italy, May 20-22, 2010. Proceedings, pp. 178–189. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
12. Cenk, M., Hasan, M.A.: Some new results on binary polynomial multiplication. Journal of Cryptographic Engineering 5(4), 289–303 (2015)
13. Cenk, M., Negre, C., Hasan, M.A.: Improved three-way split formulas for binary polynomial multiplication. In: Selected areas in cryptography. pp. 384–398. Springer (2011)
14. Cenk, M., Negre, C., Hasan, M.A.: Improved three-way split formulas for binary polynomial and toeplitz matrix vector products. IEEE Transactions on Computers 62(7), 1345–1361 (2013)
15. Chakraborty, D., Mancillas-Lpez, C., Rodrguez-Henrquez, F., Sarkar, P.: Efficient hardware implementations of brw polynomials and tweakable enciphering schemes. IEEE Transactions on Computers 62(2), 279–294 (2013)
16. Chang, N.S., Kim, C.H., Park, Y.H., Lim, J.: A non-redundant and efficient architecture for Karatsuba-Ofman algorithm. In: Information Security, 8th International Conference, ISC 2005, Singapore, pp. 288–299. Springer (2005)
17. Chen, D.D., Yao, G.X., Cheung, R.C.C., Pao, D., Ko, .K.: Parameter space for the architecture of fft-based montgomery modular multiplication. IEEE Transactions on Computers 65(1), 147–160 (2016)
18. Chou, T.: Accelerating pre-and post-quantum cryptography. Ph.D. thesis, Technische Universiteit Eindhoven (2016)
19. CMT: Circuit minimization work, `http://www.cs.yale.edu/homes/peralta/CircuitStuff/CMT.html`
20. Cook, S.A.: On the minimum computation time of functions. PhD thesis, Harvard University (1966)
21. D'angella, D., Schiavo, C.V., Visconti, A.: Tight upper bounds for polynomial multiplication. In: Applied Computing Conference, 2013. ACC'13. WEAS. pp. 31–37. WEAS (2013)
22. Fan, H., Sun, J., Gu, M., Lam, K.Y.: Overlap-free karatsuba–ofman polynomial multiplication algorithms. IET Information security 4(1), 8–14 (2010)
23. Find, M.G., Peralta, R.: Better circuits for binary polynomial multiplication. IEEE Transactions on Computers 68(4), 624–630 (April 2019)
24. von zur Gathen, J., Shokrollahi, J.: Fast arithmetic for polynomials over F_2 in hardware. In: Information Theory Workshop, 2006. ITW'06 Punta del Este. IEEE. pp. 107–111. IEEE (2006)
25. Homma, N., Saito, K., Aoki, T.: Toward formal design of practical cryptographic hardware based on galois field arithmetic. IEEE Transactions on Computers 63(10), 2604–2613 (2014)
26. Imana, J.L.: Fast bit-parallel binary multipliers based on type-i pentanomials. IEEE Transactions on Computers PP(99), 1–1 (2017)
27. Karatsuba, A., Ofman, Y.: Multiplication of multidigit numbers on automata. In: Soviet physics doklady. vol. 7, pp. 595–596 (1963)
28. Li, Y., Ma, X., Zhang, Y., Qi, C.: Mastrovito form of non-recursive karatsuba multiplier for all trinomials. IEEE Transactions on Computers 66(9), 1573–1584 (2017)
29. McClellen, J.H., Rader, C.M.: Number theory in digital signal processing. Prentice Hall Professional Technical Reference (1979)

30. McEliece, R.J.: Finite fields for computer scientists and engineers, vol. 23. Kluwer Academic Publishers Boston (1987)
31. Menezes, A.J., Van Oorschot, P.C., Vanstone, S.A.: Handbook of applied cryptography. CRC press (1997)
32. Orellana, R.: Course notes in discrete mathematics in computer science, `https://math.dartmouth.edu/archive/m19w03/public_html/book.html`
33. Paar, C.: Optimized arithmetic for reed-solomon encoders. In: Proceedings of IEEE International Symposium on Information Theory. pp. 250– (1997)
34. Peter, S., Langendorfer, P.: An efficient polynomial multiplier in $GF(2^m)$ and its application to ECC designs. In: Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE'07. pp. 1–6. IEEE (2007)
35. Rodrıguez-Henrıquez, F., Koç, Ç.: On fully parallel Karatsuba multipliers for $GF(2^m)$. In: International Conference on Computer Science and Technology (CST 2003), Cancun, Mexico. pp. 405–410 (2003)
36. Rotman, J.J.: An introduction to the theory of groups, vol. 148. Springer Science & Business Media (2012)
37. Schönhage, D.D.A., Strassen, V.: Schnelle multiplikation grosser zahlen. Computing 7(3-4), 281–292 (1971)
38. Toom, A.L.: The complexity of a scheme of functional elements realizing the multiplication of integers. In: Soviet Mathematics Doklady. vol. 3, pp. 714–716 (1963)
39. Visconti, A., Schiavo, C.V., Peralta, R.: Improved upper bounds for the expected circuit complexity of dense systems of linear equations over GF(2). Information Processing Letters 137, 1–5 (2018)

6 Appendix A

We present $M(24)$, the straight-line sequence of bit operations, or straight-line program (SLP), needed to multiply two 24-bit polynomials. This SLP has been obtained by applying *Three-level Recursion* algorithm.

$$F(x)G(x) = \sum_{i=0}^{23} f[i]x^i \sum_{j=0}^{23} g[j]x^j = \sum_{k=0}^{46} h[k]x^k = H(x)$$

$t1 = f[2] * g[2]$
$t2 = f[2] * g[0]$
$t3 = f[2] * g[1]$
$t4 = f[0] * g[2]$
$t5 = f[1] * g[2]$
$t6 = f[1] * g[1]$
$t7 = f[1] * g[0]$
$t8 = f[0] * g[1]$
$t9 = f[0] * g[0]$
$t10 = t8 + t7$
$t11 = t6 + t4$
$t12 = t11 + t2$
$t13 = t5 + t3$
$t14 = f[5] * g[5]$
$t15 = f[5] * g[3]$
$t16 = f[5] * g[4]$
$t17 = f[3] * g[5]$
$t18 = f[4] * g[5]$
$t19 = f[4] * g[4]$
$t20 = f[4] * g[3]$
$t21 = f[3] * g[4]$
$t22 = f[3] * g[3]$
$t23 = t21 + t20$
$t24 = t19 + t17$
$t25 = t24 + t15$
$t26 = t18 + t16$
$t27 = f[8] * g[8]$
$t28 = f[8] * g[6]$
$t29 = f[8] * g[7]$
$t30 = f[6] * g[8]$
$t31 = f[7] * g[8]$
$t32 = f[7] * g[7]$
$t33 = f[7] * g[6]$
$t34 = f[6] * g[7]$
$t35 = f[6] * g[6]$
$t36 = t34 + t33$
$t37 = t32 + t30$
$t38 = t37 + t28$
$t39 = t31 + t29$
$t40 = f[11] * g[11]$
$t41 = f[11] * g[9]$
$t42 = f[11] * g[10]$
$t43 = f[9] * g[11]$
$t44 = f[10] * g[11]$
$t45 = f[10] * g[10]$
$t46 = f[10] * g[9]$
$t47 = f[9] * g[10]$
$t48 = f[9] * g[9]$
$t49 = t47 + t46$
$t50 = t45 + t43$
$t51 = t50 + t41$
$t52 = t44 + t42$
$t53 = f[14] * g[14]$
$t54 = f[14] * g[12]$
$t55 = f[14] * g[13]$
$t56 = f[12] * g[14]$
$t57 = f[13] * g[14]$
$t58 = f[13] * g[13]$
$t59 = f[13] * g[12]$
$t60 = f[12] * g[13]$
$t61 = f[12] * g[12]$
$t62 = t60 + t59$
$t63 = t58 + t56$
$t64 = t63 + t54$
$t65 = t57 + t55$
$t66 = f[17] * g[17]$
$t67 = f[17] * g[15]$
$t68 = f[17] * g[16]$
$t69 = f[15] * g[17]$
$t70 = f[16] * g[17]$
$t71 = f[16] * g[16]$
$t72 = f[16] * g[15]$
$t73 = f[15] * g[16]$
$t74 = f[15] * g[15]$
$t75 = t73 + t72$
$t76 = t71 + t69$
$t77 = t76 + t67$
$t78 = t70 + t68$
$t79 = f[20] * g[20]$
$t80 = f[20] * g[18]$
$t81 = f[20] * g[19]$
$t82 = f[18] * g[20]$
$t83 = f[19] * g[20]$
$t84 = f[19] * g[19]$
$t85 = f[19] * g[18]$
$t86 = f[18] * g[19]$
$t87 = f[18] * g[18]$
$t88 = t86 + t85$
$t89 = t84 + t82$
$t90 = t89 + t80$
$t91 = t83 + t81$
$t92 = f[23] * g[23]$
$t93 = f[23] * g[21]$
$t94 = f[23] * g[22]$
$t95 = f[21] * g[23]$
$t96 = f[22] * g[23]$
$t97 = f[22] * g[22]$
$t98 = f[22] * g[21]$
$t99 = f[21] * g[22]$
$t100 = f[21] * g[21]$
$t101 = t99 + t98$
$t102 = t97 + t95$
$t103 = t102 + t93$
$t104 = t96 + t94$
$t105 = t13 + t22$
$t106 = t1 + t23$
$t107 = t26 + t35$
$t108 = t14 + t36$
$t109 = t39 + t48$
$t110 = t27 + t49$
$t111 = t52 + t61$
$t112 = t40 + t62$
$t113 = t65 + t74$
$t114 = t53 + t75$
$t115 = t78 + t87$
$t116 = t66 + t88$
$t117 = t91 + t100$
$t118 = t79 + t101$
$t119 = t105 + t9$
$t120 = t106 + t10$
$t121 = t25 + t12$
$t122 = t107 + t105$
$t123 = t108 + t106$
$t124 = t38 + t25$
$t125 = t109 + t107$
$t126 = t110 + t108$
$t127 = t51 + t38$
$t128 = t111 + t109$
$t129 = t112 + t110$
$t130 = t64 + t51$
$t131 = t113 + t111$
$t132 = t114 + t112$
$t133 = t77 + t64$
$t134 = t115 + t113$
$t135 = t116 + t114$
$t136 = t90 + t77$
$t137 = t117 + t115$
$t138 = t118 + t116$
$t139 = t103 + t90$
$t140 = t104 + t117$
$t141 = t92 + t118$
$t142 = f[0] + f[3]$
$t143 = f[1] + f[4]$
$t144 = f[2] + f[5]$
$t145 = g[0] + g[3]$
$t146 = g[1] + g[4]$
$t147 = g[2] + g[5]$
$t148 = t144 * t147$
$t149 = t144 * t145$
$t150 = t144 * t146$
$t151 = t142 * t147$
$t152 = t143 * t147$
$t153 = t143 * t146$
$t154 = t143 * t145$
$t155 = t142 * t146$
$t156 = t142 * t145$
$t157 = t155 + t154$
$t158 = t153 + t151$
$t159 = t158 + t149$
$t160 = t152 + t150$
$t161 = f[6] + f[9]$
$t162 = f[7] + f[10]$
$t163 = f[8] + f[11]$
$t164 = g[6] + g[9]$
$t165 = g[7] + g[10]$
$t166 = g[8] + g[11]$
$t167 = t163 * t166$
$t168 = t163 * t164$
$t169 = t163 * t165$
$t170 = t161 * t166$
$t171 = t162 * t166$
$t172 = t162 * t165$
$t173 = t162 * t164$
$t174 = t161 * t165$
$t175 = t161 * t164$
$t176 = t174 + t173$
$t177 = t172 + t170$
$t178 = t177 + t168$
$t179 = t171 + t169$
$t180 = f[12] + f[15]$
$t181 = f[13] + f[16]$
$t182 = f[14] + f[17]$
$t183 = g[12] + g[15]$
$t184 = g[13] + g[16]$
$t185 = g[14] + g[17]$
$t186 = t182 * t185$
$t187 = t182 * t183$
$t188 = t182 * t184$
$t189 = t180 * t185$
$t190 = t181 * t185$
$t191 = t181 * t184$
$t192 = t181 * t183$
$t193 = t180 * t184$
$t194 = t180 * t183$
$t195 = t193 + t192$
$t196 = t191 + t189$
$t197 = t196 + t187$
$t198 = t190 + t188$
$t199 = f[18] + f[21]$
$t200 = f[19] + f[22]$
$t201 = f[20] + f[23]$
$t202 = g[18] + g[21]$
$t203 = g[19] + g[22]$
$t204 = g[20] + g[23]$
$t205 = t201 * t204$
$t206 = t201 * t202$
$t207 = t201 * t203$
$t208 = t199 * t204$
$t209 = t200 * t204$
$t210 = t200 * t203$
$t211 = t200 * t202$
$t212 = t199 * t203$
$t213 = t199 * t202$
$t214 = t212 + t211$
$t215 = t210 + t208$
$t216 = t215 + t206$
$t217 = t209 + t207$
$t218 = t119 + t156$
$t219 = t120 + t157$
$t220 = t121 + t159$
$t221 = t122 + t160$
$t222 = t123 + t148$
$t223 = t125 + t175$
$t224 = t126 + t176$
$t225 = t127 + t178$
$t226 = t128 + t179$
$t227 = t129 + t167$
$t228 = t131 + t194$
$t229 = t132 + t195$
$t230 = t133 + t197$
$t231 = t134 + t198$
$t232 = t135 + t186$
$t233 = t137 + t213$
$t234 = t138 + t214$
$t235 = t139 + t216$
$t236 = t140 + t217$
$t237 = t141 + t205$
$t238 = t221 + t9$
$t239 = t222 + t10$
$t240 = t124 + t12$
$t241 = t223 + t218$
$t242 = t224 + t219$
$t243 = t225 + t220$
$t244 = t226 + t221$
$t245 = t227 + t222$
$t246 = t130 + t124$
$t247 = t228 + t223$
$t248 = t229 + t224$
$t249 = t230 + t225$
$t250 = t231 + t226$
$t251 = t232 + t227$
$t252 = t136 + t130$
$t253 = t233 + t228$
$t254 = t234 + t229$
$t255 = t235 + t230$
$t256 = t236 + t231$
$t257 = t237 + t232$
$t258 = t103 + t136$
$t259 = t104 + t233$
$t260 = t92 + t234$
$t261 = f[0] + f[6]$
$t262 = f[1] + f[7]$
$t263 = f[2] + f[8]$
$t264 = f[3] + f[9]$
$t265 = f[4] + f[10]$
$t266 = f[5] + f[11]$
$t267 = g[0] + g[6]$
$t268 = g[1] + g[7]$
$t269 = g[2] + g[8]$
$t270 = g[3] + g[9]$
$t271 = g[4] + g[10]$
$t272 = g[5] + g[11]$
$t273 = t263 * t269$
$t274 = t263 * t267$
$t275 = t263 * t268$
$t276 = t261 * t269$
$t277 = t262 * t269$
$t278 = t262 * t268$
$t279 = t262 * t267$
$t280 = t261 * t268$
$t281 = t261 * t267$
$t282 = t280 + t279$
$t283 = t278 + t276$
$t284 = t283 + t274$
$t285 = t277 + t275$
$t286 = t266 * t272$
$t287 = t266 * t270$
$t288 = t266 * t271$
$t289 = t264 * t272$
$t290 = t265 * t272$
$t291 = t265 * t271$
$t292 = t265 * t270$
$t293 = t264 * t271$
$t294 = t264 * t270$
$t295 = t293 + t292$
$t296 = t291 + t289$
$t297 = t296 + t287$
$t298 = t290 + t288$
$t299 = t267 + t270$
$t300 = t268 + t271$
$t301 = t269 + t272$
$t302 = t261 + t264$
$t303 = t262 + t265$
$t304 = t263 + t266$
$t305 = t304 * t301$
$t306 = t304 * t299$
$t307 = t304 * t300$
$t308 = t302 * t301$
$t309 = t303 * t301$
$t310 = t303 * t300$
$t311 = t303 * t299$
$t312 = t302 * t300$
$t313 = t302 * t299$
$t314 = t312 + t311$
$t315 = t310 + t308$
$t316 = t315 + t306$
$t317 = t309 + t307$
$t318 = t285 + t294$
$t319 = t273 + t295$
$t320 = t313 + t318$
$t321 = t314 + t319$
$t322 = t316 + t297$
$t323 = t317 + t298$
$t324 = t305 + t286$
$t325 = t320 + t281$
$t326 = t321 + t282$
$t327 = t322 + t284$
$t328 = t323 + t318$
$t329 = t324 + t319$
$t330 = f[12] + f[18]$
$t331 = f[13] + f[19]$
$t332 = f[14] + f[20]$
$t333 = f[15] + f[21]$
$t334 = f[16] + f[22]$
$t335 = f[17] + f[23]$
$t336 = g[12] + g[18]$
$t337 = g[13] + g[19]$
$t338 = g[14] + g[20]$
$t339 = g[15] + g[21]$
$t340 = g[16] + g[22]$
$t341 = g[17] + g[23]$
$t342 = t332 * t338$
$t343 = t332 * t336$
$t344 = t332 * t337$
$t345 = t330 * t338$
$t346 = t331 * t338$
$t347 = t331 * t337$
$t348 = t331 * t336$
$t349 = t330 * t337$
$t350 = t330 * t336$
$t351 = t349 + t348$
$t352 = t347 + t345$
$t353 = t352 + t343$
$t354 = t346 + t344$
$t355 = t335 * t341$
$t356 = t335 * t339$
$t357 = t335 * t340$
$t358 = t333 * t341$
$t359 = t334 * t341$
$t360 = t334 * t340$
$t361 = t334 * t339$
$t362 = t333 * t340$
$t363 = t333 * t339$
$t364 = t362 + t361$
$t365 = t360 + t358$

$t366 = t365 + t356$
$t367 = t359 + t357$
$t368 = t336 + t339$
$t369 = t337 + t340$
$t370 = t338 + t341$
$t371 = t330 + t333$
$t372 = t331 + t334$
$t373 = t332 + t335$
$t374 = t373 * t370$
$t375 = t373 * t368$
$t376 = t373 * t369$
$t377 = t371 * t370$
$t378 = t372 * t370$
$t379 = t372 * t369$
$t380 = t372 * t368$
$t381 = t371 * t369$
$t382 = t371 * t368$
$t383 = t381 + t380$
$t384 = t379 + t377$
$t385 = t384 + t375$
$t386 = t378 + t376$
$t387 = t354 + t363$
$t388 = t342 + t364$
$t389 = t382 + t387$
$t390 = t383 + t388$
$t391 = t385 + t366$
$t392 = t386 + t367$
$t393 = t374 + t355$
$t394 = t389 + t350$
$t395 = t390 + t351$
$t396 = t391 + t353$
$t397 = t392 + t387$
$t398 = t393 + t388$
$t399 = t238 + t281$
$t400 = t239 + t282$
$t401 = t240 + t284$
$t402 = t241 + t325$
$t403 = t242 + t326$
$t404 = t243 + t327$
$t405 = t244 + t328$
$t406 = t245 + t329$
$t407 = t246 + t297$
$t408 = t247 + t298$
$t409 = t248 + t286$
$t410 = t250 + t350$
$t411 = t251 + t351$
$t412 = t252 + t353$
$t413 = t253 + t394$
$t414 = t254 + t395$
$t415 = t255 + t396$
$t416 = t256 + t397$
$t417 = t257 + t398$
$t418 = t258 + t366$
$t419 = t259 + t367$
$t420 = t260 + t355$
$t421 = t405 + t9$
$t422 = t406 + t10$
$t423 = t407 + t12$
$t424 = t408 + t218$
$t425 = t409 + t219$
$t426 = t249 + t220$
$t427 = t410 + t399$
$t428 = t411 + t400$
$t429 = t412 + t401$
$t430 = t413 + t402$
$t431 = t414 + t403$
$t432 = t415 + t404$
$t433 = t416 + t405$
$t434 = t417 + t406$
$t435 = t418 + t407$
$t436 = t419 + t408$
$t437 = t420 + t409$
$t438 = t235 + t249$
$t439 = t236 + t410$
$t440 = t237 + t411$
$t441 = t103 + t412$
$t442 = t104 + t413$
$t443 = t92 + t414$
$t444 = f[0] + f[12]$
$t445 = f[1] + f[13]$
$t446 = f[2] + f[14]$
$t447 = f[3] + f[15]$
$t448 = f[4] + f[16]$
$t449 = f[5] + f[17]$
$t450 = f[6] + f[18]$
$t451 = f[7] + f[19]$
$t452 = f[8] + f[20]$
$t453 = f[9] + f[21]$
$t454 = f[10] + f[22]$
$t455 = f[11] + f[23]$
$t456 = g[0] + g[12]$
$t457 = g[1] + g[13]$
$t458 = g[2] + g[14]$
$t459 = g[3] + g[15]$
$t460 = g[4] + g[16]$
$t461 = g[5] + g[17]$
$t462 = g[6] + g[18]$
$t463 = g[7] + g[19]$
$t464 = g[8] + g[20]$
$t465 = g[9] + g[21]$
$t466 = g[10] + g[22]$
$t467 = g[11] + g[23]$
$t468 = t455 * t467$
$t469 = t455 * t465$
$t470 = t455 * t466$
$t471 = t453 * t467$
$t472 = t454 * t467$
$t473 = t454 * t466$
$t474 = t454 * t465$
$t475 = t453 * t466$
$t476 = t453 * t465$
$t477 = t475 + t474$
$t478 = t472 + t470$
$t479 = t452 * t464$
$t480 = t452 * t462$
$t481 = t452 * t463$
$t482 = t450 * t464$
$t483 = t480 + t482$
$t484 = t451 * t464$
$t485 = t451 * t463$
$t486 = t451 * t462$
$t487 = t450 * t463$
$t488 = t450 * t462$
$t489 = t487 + t486$
$t490 = t484 + t481$
$t491 = t449 * t461$
$t492 = t449 * t459$
$t493 = t449 * t460$
$t494 = t447 * t461$
$t495 = t448 * t461$
$t496 = t448 * t460$
$t497 = t448 * t459$
$t498 = t447 * t460$
$t499 = t447 * t459$
$t500 = t498 + t497$
$t501 = t495 + t493$
$t502 = t446 * t458$
$t503 = t446 * t456$
$t504 = t446 * t457$
$t505 = t444 * t458$
$t506 = t445 * t458$
$t507 = t445 * t457$
$t508 = t445 * t456$
$t509 = t444 * t457$
$t510 = t444 * t456$
$t511 = t509 + t508$
$t512 = t506 + t504$
$t513 = t512 + t499$
$t514 = t502 + t500$
$t515 = t501 + t488$
$t516 = t491 + t489$
$t517 = t490 + t476$
$t518 = t479 + t477$
$t519 = t462 + t465$
$t520 = t463 + t466$
$t521 = t464 + t467$
$t522 = t450 + t453$
$t523 = t451 + t454$
$t524 = t452 + t455$
$t525 = t524 * t521$
$t526 = t524 * t519$
$t527 = t524 * t520$
$t528 = t522 * t521$
$t529 = t523 * t521$
$t530 = t523 * t520$
$t531 = t523 * t519$
$t532 = t522 * t520$
$t533 = t522 * t519$
$t534 = t532 + t531$
$t535 = t529 + t527$
$t536 = t456 + t459$
$t537 = t457 + t460$
$t538 = t458 + t461$
$t539 = t444 + t447$
$t540 = t445 + t448$
$t541 = t446 + t449$
$t542 = t541 * t538$
$t543 = t541 * t536$
$t544 = t541 * t537$
$t545 = t539 * t538$
$t546 = t540 * t538$
$t547 = t540 * t537$
$t548 = t540 * t536$
$t549 = t539 * t537$
$t550 = t539 * t536$
$t551 = t549 + t548$
$t552 = t546 + t544$
$t553 = t550 + t510$
$t554 = t514 + t511$
$t555 = t515 + t513$
$t556 = t516 + t542$
$t557 = t517 + t533$
$t558 = t518 + t516$
$t559 = t478 + t517$
$t560 = t468 + t525$
$t561 = t553 + t513$
$t562 = t554 + t551$
$t563 = t555 + t552$
$t564 = t556 + t514$
$t565 = t557 + t515$
$t566 = t558 + t534$
$t567 = t559 + t535$
$t568 = t560 + t518$
$t569 = t459 + t465$
$t570 = t460 + t466$
$t571 = t461 + t467$
$t572 = t456 + t462$
$t573 = t457 + t463$
$t574 = t458 + t464$
$t575 = t447 + t453$
$t576 = t448 + t454$
$t577 = t449 + t455$
$t578 = t444 + t450$
$t579 = t445 + t451$
$t580 = t446 + t452$
$t581 = t580 * t574$
$t582 = t580 * t572$
$t583 = t580 * t573$
$t584 = t578 * t574$
$t585 = t579 * t574$
$t586 = t579 * t573$
$t587 = t579 * t572$
$t588 = t578 * t573$
$t589 = t578 * t572$
$t590 = t588 + t587$
$t591 = t585 + t583$
$t592 = t577 * t571$
$t593 = t577 * t569$
$t594 = t577 * t570$
$t595 = t575 * t571$
$t596 = t576 * t571$
$t597 = t576 * t570$
$t598 = t576 * t569$
$t599 = t575 * t570$
$t600 = t575 * t569$
$t601 = t599 + t598$
$t602 = t596 + t594$
$t603 = t572 + t569$
$t604 = t573 + t570$
$t605 = t574 + t571$
$t606 = t578 + t575$
$t607 = t579 + t576$
$t608 = t580 + t577$
$t609 = t608 * t605$
$t610 = t608 * t603$
$t611 = t608 * t604$
$t612 = t606 * t605$
$t613 = t607 * t605$
$t614 = t607 * t604$
$t615 = t607 * t603$
$t616 = t606 * t604$
$t617 = t606 * t603$
$t618 = t616 + t615$
$t619 = t613 + t611$
$t620 = t591 + t600$
$t621 = t581 + t601$
$t622 = t617 + t589$
$t623 = t618 + t590$
$t624 = t619 + t602$
$t625 = t609 + t592$
$t626 = t622 + t620$
$t627 = t623 + t621$
$t628 = t624 + t620$
$t629 = t625 + t621$
$t630 = t589 + t510$
$t631 = t590 + t511$
$t632 = t565 + t561$
$t633 = t566 + t562$
$t634 = t567 + t563$
$t635 = t568 + t564$
$t636 = t478 + t602$
$t637 = t468 + t592$
$t638 = t630 + t563$
$t639 = t631 + t564$
$t640 = t632 + t626$
$t641 = t633 + t627$
$t642 = t634 + t628$
$t643 = t635 + t629$
$t644 = t636 + t565$
$t645 = t637 + t566$
$t646 = t469 + t471$
$t647 = t473 + t646$
$t648 = t503 + t505$
$t649 = t507 + t648$
$t650 = t483 + t485$
$t651 = t492 + t494$
$t652 = t496 + t651$
$t653 = t647 + t650$
$t654 = t649 + t652$
$t655 = t526 + t528$
$t656 = t530 + t655$
$t657 = t653 + t656$
$t658 = t543 + t545$
$t659 = t547 + t658$
$t660 = t654 + t659$
$t661 = t582 + t584$
$t662 = t586 + t661$
$t663 = t593 + t595$
$t664 = t597 + t663$
$t665 = t650 + t654$
$t666 = t662 + t665$
$t667 = t652 + t653$
$t668 = t664 + t667$
$t669 = t657 + t660$
$t670 = t662 + t610$
$t671 = t612 + t614$
$t672 = t664 + t671$
$t673 = t672 + t670$
$t674 = t673 + t669$
$t675 = t421 + t510$
$t676 = t422 + t511$
$t677 = t423 + t649$
$t678 = t424 + t561$
$t679 = t425 + t562$
$t680 = t426 + t660$
$t681 = t427 + t638$
$t682 = t428 + t639$
$t683 = t429 + t666$
$t684 = t430 + t640$
$t685 = t431 + t641$
$t686 = t432 + t674$
$t687 = t433 + t642$
$t688 = t434 + t643$
$t689 = t435 + t668$
$t690 = t436 + t644$
$t691 = t437 + t645$
$t692 = t438 + t657$
$t693 = t439 + t567$
$t694 = t440 + t568$
$t695 = t441 + t647$
$t696 = t442 + t478$
$t697 = t443 + t468$
$h[0] = t9$
$h[1] = t10$
$h[2] = t12$
$h[3] = t218$
$h[4] = t219$
$h[5] = t220$
$h[6] = t399$
$h[7] = t400$
$h[8] = t401$
$h[9] = t402$
$h[10] = t403$
$h[11] = t404$
$h[12] = t675$
$h[13] = t676$
$h[14] = t677$
$h[15] = t678$
$h[16] = t679$
$h[17] = t680$
$h[18] = t681$
$h[19] = t682$
$h[20] = t683$
$h[21] = t684$
$h[22] = t685$
$h[23] = t686$
$h[24] = t687$
$h[25] = t688$
$h[26] = t689$
$h[27] = t690$
$h[28] = t691$
$h[29] = t692$
$h[30] = t693$
$h[31] = t694$
$h[32] = t695$
$h[33] = t696$
$h[34] = t697$
$h[35] = t415$
$h[36] = t416$
$h[37] = t417$
$h[38] = t418$
$h[39] = t419$
$h[40] = t420$
$h[41] = t235$
$h[42] = t236$
$h[43] = t237$
$h[44] = t103$
$h[45] = t104$
$h[46] = t92$