J Cryptogr Eng (2015) 5:289–303
DOI 10.1007/s13389-015-0101-6

REGULAR PAPER

Some new results on binary polynomial multiplication

Murat Cenk[1] · **M. Anwar Hasan**[2]

Received: 19 December 2014 / Accepted: 3 May 2015 / Published online: 16 May 2015

Abstract This paper presents several methods for reducing the number of bit operations for multiplication of polynomials over the binary field. First, a modified Bernstein's 3-way algorithm is introduced, followed by a new 5-way algorithm. Next, a new 3-way algorithm that improves asymptotic arithmetic complexity compared to Bernstein's 3-way algorithm is introduced. This new algorithm uses three multiplications of one-third size polynomials over the binary field and one multiplication of one-third size polynomials over the finite field with four elements. Unlike Bernstein's algorithm, which has a linear delay complexity with respect to input size, the delay complexity of the new algorithm is logarithmic. The number of bit operations for the multiplication of polynomials over the finite field with four elements is also computed. Finally, all these new results are combined to obtain improved complexities.

Keywords Polynomial multiplication · Elliptic curve scalar multiplication · Binary fields · Karatsuba · Toom · Divide-and-conquer

1 Introduction

The design of algorithms for binary polynomial multiplication has long been of great interest to many researchers. Because of applications in a variety of areas, such as cryptography and coding theory, new techniques for improving polynomial multiplication have been presented in numerous papers, e.g., [1,4,5,7,8,13–18,20,23–25,27,28]. For cryptographic applications, arithmetic in the binary extension field $\mathbb{F}_{2^n}$ is often used and, of the basic operations in $\mathbb{F}_{2^n}$, multiplication contributes most to the total number of bit operations. For example, Bernstein [3] showed that a 251-bit scalar multiplication on a binary Edward curves entails 44,679,665 bit operations and that about 96.3 % of this computational cost is due to field multiplications. Multiplications in $\mathbb{F}_{2^n}$ can be performed in two steps: polynomial multiplication and polynomial reduction. The cost of reduction is $O(n)$ arithmetic operations, whereas the cost of multiplication is $O(n^{\omega})$, where $1 < \omega \leq 2$. The cost of reduction is, therefore, negligible with respect to polynomial multiplication for a large value of n.

Let $O(n^{\omega})$ be the arithmetic complexity, i.e., the number of bit operations for computing the product of two degree $(n-1)$ polynomials over the binary field. The classical or the school-book method of binary polynomial multiplication requires n^2 and $(n-1)^2$ bit level multiplications and additions, respectively. Using Karatsuba's algorithm [19], multiplication of two binary polynomials can be performed with three multiplications and four additions of half-size polynomials. Recursive use of the Karatsuba algorithm gives $\omega \leq 1.58$. More precisely, the Karatsuba algorithm requires $7n^{1.58} + O(n)$ operations.

The Karatsuba algorithm is based on the 2-way split, where the polynomials being multiplied are divided into two parts and the Karatsuba algorithm is then applied recursively. As an extension, the 3-way split version of the Karatsuba algorithm requires six multiplications of one-third size polynomials. In [26], the use of the Chinese remainder theorem resulted in sub-quadratic complexity for polynomial mul-

✉ Murat Cenk
mcenk@metu.edu.tr

M. Anwar Hasan
ahasan@uwaterloo.ca

1 Institute of Applied Mathematics, Middle East Technical University, Ankara, Turkey

2 Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada

tiplication algorithms with six multiplications. In [24] and [25], methods have been presented for 3-way splits with $6.33n^{1.63} + O(n)$ operations. More recently, this complexity has been improved to $6.27n^{1.63} + O(n)$ as reported in [11] and then to $5.8n^{1.63} + O(n)$ as described in [9].

At the CRYPTO 2009 conference, Bernstein proposed several algorithms, including 2-, 3- and 4-way split methods for polynomial multiplication over binary fields [3]. Bernstein's 2-way split algorithm improves the complexity of the Karatsuba algorithm to $6.5n^{1.58} + O(n)$. It should be noted that in [27], Zhou and Michalic also reported similar results for a 2-way split algorithm using a different approach. Bernstein's 2-way and 4-way split algorithms improve the additive complexity, while his 3-way split algorithm improves both the multiplicative and the additive complexity; specifically, the latter was reduced to $25.5n^{1.46} + O(n)$.

The approach used in [3] for reducing z complexity is to use the best possible algorithms in each recursion rather than the same algorithm in all recursions. For example, the product of degree five binary polynomials, (that is $n = 6$), requires 61 operations using the school-book method, but Bernstein reduced it to 57 operations by first using his 2-way split algorithm and then applying the school-book algorithm. The improved upper bounds are presented in [2]. This approach was also used in [25] and [13]. The best known results for almost all input sizes up to 1000 are listed in [2] using the 3-way and 4-way algorithms introduced in [3]. On the other hand, for values of $n = 11, 12, 15, 16, 18, 19$ and 20, the results reported in [6] are superior to those in [2].

1.1 Notation and model of computation

$\mathbb{F}_{q^n}$ is used for the finite field with q^n elements (where q is a prime power), and $\mathbb{F}_q[X]$ is employed for the ring of polynomials over $\mathbb{F}_q$. $M_q(n)$ represents the minimum number of bit operations required for the computation of the product of two polynomials of degree less than n over $\mathbb{F}_q$. $D_q(n)$ is used for the delay complexity of polynomial multiplication over $\mathbb{F}_q$, and D_A and D_X denote the delay of bit level multiplication and addition, respectively. Throughout this paper, the cost metric related to polynomial multiplication is taken as the number of bit operations (bit addition and bit multiplication) required for multiplying polynomials over $\mathbb{F}_2$ or $\mathbb{F}_4$, and since the computations are over characteristic two fields, addition and subtraction are equal.

1.2 Our contributions

The work presented in this paper represents the following contributions:

- A modification of Bernstein's 3-way algorithm offering improvements, albeit small but covering a wider range of polynomial degrees.
- An improved version of the 5-way algorithm introduced in [12] through an optimization of the number of additions.
- A new 3-way algorithm with a lower complexity than the ones described in [3,10,11]: it entails the asymptotic arithmetic complexity of $15.125n^{1.46} + O(n)$ and delay complexity $10\log_3(n)D_X + D_A$.
- New optimizations of algorithms for polynomial multiplication over $\mathbb{F}_4$.
- A new minimum number of bit operations for binary polynomial multiplication presented in [2] and [6].
- New results on the minimum number of bit operations for binary polynomial multiplication with logarithmic delay complexity.

1.3 Organization of paper

The remainder of the paper is organized as follows: Known algorithms related to our work are presented in the next section along with a description of the slight improvements that have been developed. The proposed improved algorithms over $\mathbb{F}_2$ are introduced in Sect. 3, and the reduced complexity of multiplication over $\mathbb{F}_4$ is explained in Sect. 4. Section 5 details how our improvements can enhance cryptographic applications, followed by a summary of our conclusions in Sect. 6.

2 Some known algorithms and their slight improvements

This section provides a brief review of a number of known efficient polynomial multiplication algorithms over $\mathbb{F}_2$ and presents methods of obtaining slight improvements in some of these algorithms. To save space, the details of the known algorithms are not included; only their complexities are discussed with appropriate references.

2.1 School-book algorithm

Let $A = \sum_{i=0}^{n-1} a_i X^i$, $B = \sum_{i=0}^{n-1} b_i X^i$ and $C = AB = \sum_{i=0}^{2n-2} c_i X^i$. The school-book algorithm computes the coefficients of the product of A and B as $C_i = \sum_{j+k=i}^{2n-2} a_j b_k X^i$ where $0 \le j, k < n$. The number of multiplications and additions required are n^2 and $(n-1)^2$, respectively. Moreover, one can easily derive the following:

$$\begin{cases} M_2(n+1) \le M_2(n) + 4n, \\ D_2(n+1) \le D_2(n) + D_X. \end{cases} \quad (1)$$

2.2 Karatsuba algorithm (with Bernstein's improvement)

Now, let A and B be degree $(2n-1)$ polynomials over $\mathbb{F}_2$ and C be their product. The improved Karatsuba algorithm splits A and B into two parts as $A(x) = A_0 + X^n A_1$ and $B(x) = B_0 + X^n B_1$ where $A_0 = \sum_{i=0}^{n-1} a_i X^i$, $A_1 = \sum_{i=0}^{n-1} a_{i+n} X^i$, $B_0 = \sum_{i=0}^{n-1} b_i X^i$, and $B_1 = \sum_{i=0}^{n-1} b_{i+n} X^i$. Bernstein proposed the following algorithm:

$$\begin{aligned}&(A_0 + X^n A_1)(B_0 + X^n B_1)\\&\quad = (1+X^n)(A_0B_0+X^nA_1B_1) + X^n(A_0 + A_1)(B_0 + B_1).\end{aligned}$$

The arithmetic complexity of the algorithm is as follows [3]:

$$\begin{cases} M_2(n+k) \leq 2M_2(n) + M_2(k) + 3n + 4k - 3, \\ \quad n/2 \leq k \leq n, \\ D_2(2n) \leq D_2(n) + 3D_X, \\ M_2(n) \leq 6.5n^{1.58} - 7n + 1.5, \\ D_2(n) \leq 3\log_2(n)D_X + D_A. \end{cases} \tag{2}$$

Remark 1 Assume that $k = n - \ell$ in (2) where $\ell = \{1, 2, 3\}$. In this case, it should be noted that the last ℓ terms of A_0B_0 and $(A_0 + A_1)(B_0 + B_1)$ are identical. Therefore, once A_0B_0 is computed, the cost of computing $(A_0 + A_1)(B_0 + B_1)$ is less than $M_2(n)$. The computation of the last ℓ terms is done using the school-book method, which yields the minimum values, and it is ℓ^2 for $\ell \in \{1, 2, 3\}$. Hence we have the following recursion:

$$\begin{aligned}&M_2(2n-\ell) \leq 2M_2(n)\\&\quad + M_2(n-\ell) + 7n - 4\ell - 3 - \ell^2, \quad 1 \leq \ell \leq 3.\end{aligned} \tag{3}$$

It should be noted that Bernstein obtained bounds by computing explicit algorithms and thus because of the detection of common operations, the bounds in [2] are less than the values obtained directly through the recursion. For $\ell > 3$, the number of common expressions might change depending on the value of n.

2.3 Bernstein's 3-way split algorithm

Let A and B be degree $(3n-1)$ polynomials over $\mathbb{F}_2$ and C be their product. This method splits A and B in three parts as follows: $A = A_0 + A_1X^n + A_2X^{2n}$, $B = B_0 + B_1X^n + B_2X^{2n}$ where $A_j = \sum_{i=0}^{n-1} a_{i+nj} X^i$ and $B_j = \sum_{i=0}^{n-1} b_{i+nj} X^i$ for $j = 0, 1, 2$. Bernstein's 3-way split algorithm is the following [3]:

$$\begin{cases} P_0 = A_0B_0,\ P_1 = (A_0 + A_1 + A_2)(B_0 + B_1 + B_2), \\ P_2 = (A_0 + A_1X + A_2X^2)(B_0 + B_1X + B_2X^2), \\ P_3 = \left((A_0 + A_1 + A_2) + (A_1X + A_2X^2)\right)\left((B_0 + B_1 + B_2)\right. \\ \quad \left.+(B_1X + B_2X^2)\right), \\ P_4 = A_2B_2,\ U = P_0 + (P_0 + P_1)X^n, \\ \quad V = P_2 + (P_2 + P_3)(X^n + X), \\ C = U + P_4(X^{4n} + X^n) + \dfrac{(U+V+P_4(X^4+X))(X^{2n}+X^n)}{X^2+X}. \end{cases} \tag{4}$$

The arithmetic complexity of the algorithm is as follows [3, 10, 11]:

$$\begin{cases} M_2(3n) \leq 3M_2(n) + 2M_2(n+2) + 35n - 12, \quad n \geq 2, \\ M_2(2n+k) \leq 2M_2(n) + M_2(k) + 2M_2(n+1) + 25n \\ \quad +10k - 12, \quad 1 \leq k \leq n-1, \\ D_2(3n) \leq D_2(n) + (3n+8)D_X, \\ M_2(n) \leq 25.5n^{1.46} - 25.5n + 1, \\ D_2(n) \leq (1.5n + 8\log_3(n) - 1.5)D_X + D_A. \end{cases} \tag{5}$$

The reason for the linear delay complexity is the division by $(X^2 + X)$ in the Eq. (4). This division requires $(n-2)$ bit additions and a delay of $(n-2)D_X$. A detailed explanation is in Section 2.3.2 of [11]. We also note that one can obtain a logarithmic delay for this type of exact division. However, in this case, the number of additions increases significantly.

Remark 2 It should be noted that in (4), the first term of each of P_0 and P_2 is a_0b_0, and the first term of each of P_1 and P_3 is $(a_0 + a_n + a_{2n})(b_0 + b_n + b_{2n})$. Two multiplications are thus saved here. As well, the last term of P_2 and that of P_4 are identical, which also saves a multiplication. Finally, the last two terms of P_2 and P_3 are likewise the same, which brings the savings up to five operations. It should also be noted that the first term of $P_0 + P_1$ and that of $P_2 + P_3$ are also the same. The result of all of the above observations is a total of nine common expressions for computing $M(3n)$. On the other hand, for $M_2(2n + k)$, $1 \leq k \leq n - 1$, one can observe three common multiplications in the first term of P_2 and P_0, the first term of P_3 and P_1, and the last term of P_2 and P_3. Furthermore, the first term of $P_0 + P_1$ and that of $P_2 + P_3$ are the same. Therefore, (5) can be rewritten as

$$\begin{cases} M_2(3n) \leq 3M_2(n) + 2M_2(n+2) + 35n - 12 - 9, \quad n \geq 2, \\ M_2(2n+k) \leq 2M_2(n) + M_2(k) + 2M_2(n+1) + 25n \\ \quad +10k - 12 - 4, \quad 1 \leq k \leq n-1. \end{cases} \tag{6}$$

One can also note that the number of common operations is actually greater than that indicated above. These observations were also reported in [3] and explicit algorithms are obtained by eliminating the common operations in [2]. The results in [2] are, therefore, better than the theoretical results detailed in [3].

2.4 Karatsuba-like improved 3-way split algorithm

Let $A, B, C, A_0, A_1, A_2, B_0, B_1$ and B_2 be as in Bernstein's 3-way algorithm presented above. This algorithm was obtained in [9] using a technique similar to that employed in [27]. The algorithm is as follows:

$$\begin{cases} P_0 = A_0B_0 = P_{0L} + P_{0H}X^n,\ P_1 = A_1B_1 = P_{1L} + P_{0H}X^n, \\ P_2 = A_2B_2 = P_{2L} + P_{2H}X^n, \\ P_3 = (A_1 + A_2)(B_1 + B_2) = P_{3L} + P_{3H}X^n, \\ P_4 = (A_0 + A_1)(B_0 + B_1) = P_{4L} + P_{4H}X^n, \\ P_5 = (A_0 + A_2)(B_0 + B_2) = P_{5L} + P_{5H}X^n, \\ R_0 = P_{0H} + P_{1L},\ R_1 = R_0 + P_{0L},\ R_2 = R_1 + P_{4L}, \\ R_3 = P_{1H} + P_{2L},\ R_4 = R_1 + R_3,\ R_5 = P_{4H} + P_{5L}, \\ R_6 = R_4 + R_5,\ R_7 = R_3 + P_{2H},\ R_8 = R_7 + R_0, \\ R_9 = R_8 + P_{3L}, R_{10} = R_9 + P_{5H},\ R_{11} = R_7 + P_{3H}, \\ C = P_{0L} + R_2X^n + R_6X^{2n} + R_{10}X^{3n} + R_{11}X^{4n} + P_{2H}X^{5n}. \end{cases}$$

Assume that A and B are degree $2n + k - 1$ polynomials, where $1 \le k \le n$. $A_0,\ A_1,\ B_0$ and B_1 are then degree $(n-1)$ polynomials, and A_2 and B_2 are degree $(k-1)$ polynomials. Therefore, $P_{0L},\ P_{1L}$, and P_{2L} are degree $(n-1)$ polynomials, and P_{0H} and P_{1H} are $(n-2)$ polynomials. On the other hand, P_{2L} is a degree $(n-1)$ polynomial, P_{2H} is a degree $(2k - n - 1)$ polynomial for $n/2 < k \le n$, P_{2L} is a degree $(2k - 2)$ polynomial, and $P_{2H} = 0$ for $k \le n/2$. Note that $(A_0 + A_1)$ and $(B_0 + B_1)$ each require n additions, $(A_0 + A_2),\ (A_1 + A_2),\ (B_0 + B_2)$, and $(B_1 + b_2)$ each require k additions; $R_0,\ R_3,\ R_5,\ R_{10}$, and R_{11} each require $(n-1)$ additions; $R_1,\ R_2,\ R_4,\ R_6,\ R_8$, and R_9 each require n additions and R_7 requires $(2k-n-1)$ additions for $n/2 < k \le n$. For $k \le n/2$, R_7 requires no additions. Therefore, we obtain the following recursions [9]:

$$\begin{cases} M_2(3n) \le 6M_2(n) + 18n - 6, \\ M_2(2n+k) \le 5M_2(n) + M_2(k) + 12n + 6k - 6, \\ \quad n/2 < k \le n, \\ M_2(2n+k) \le 5M_2(n) + M_2(k) + 13n + 4k - 5, \quad k \le n/2, \\ D_2(3n) \le D_2(n) + 4D_X, \\ M_2(n) \le 5.8n^{1.63} - 6n + 1.2, \\ D_2(n) \le 4\log_3(n)D_X + D_A. \end{cases} \tag{7}$$

Remark 3 Assume that $k = n - \ell$ for $1 \le \ell \le 2$. The last ℓ terms of the products A_0B_0 and $(A_0 + A_2)(B_0 + B_2)$ are then the same, and the last ℓ terms of the products A_1B_1 and $(A_1 + A_2)(B_1 + B_2)$ are also the same. Therefore, we can obtain the following bound using the school-book method:

$$M_2(3n - \ell) \le 5M_2(n) + M_2(n - \ell) + 18n - 6\ell - 6 - 2\ell^2, \quad 1 \le \ell \le 2. \tag{8}$$

2.5 Bernstein's 4-way split algorithm

Let A and B be two degree $(4n - 1)$ polynomials over $\mathbb{F}_2$ and C be their product. This method splits A and B into four parts as $A = A_0 + A_1X^n + A_2X^{2n} + A_3X^{3n}$, $B = B_0 + B_1X + B_2X^{2n} + B_3X^{3n}$ where $A_j = \sum_{i=0}^{n-1} a_{i+nj}X^i$ and $B_j = \sum_{i=0}^{n-1} b_{i+nj}X^i$ for $j = 0, 1, 2, 3$. Bernstein's 4-way algorithm is the following:

$$\begin{cases} AB = (1+X^{2n})((1+X^n)(A_0B_0 + X^nA_1B_1 + X^{2n}A_2B_2 + X^{3n}A_3B_3) \\ \quad + X^n(A_0 + A_1)(B_0 + B_1) + X^{3n}(A_2 + A_3)(B_2 + B_3)) \\ \quad + X^{2n}(A_0 + A_2 + (A_1 + A_3)X^n)(B_0 + B_2 + (B_1 + B_3)X^n). \end{cases}$$

The arithmetic complexity of the algorithm is as follows [3,9]:

$$\begin{cases} M_2(4n) \le M_2(2n) + 6M_2(n) + 27n - 8, \\ M_2(3n + k) \le M_2(2n) + 5M_2(n) + M_2(k) + 19n + 8k - 8, \\ \quad n/2 \le k \le n, \\ D_2(4n) \le D_2(n) + 5D_X, \\ M_2(n) \le 6.425n^{1.58} - 6.8n + 1.375, \\ D_2(n) \le 5\log_4(n)D_X + D_A. \end{cases} \tag{9}$$

Remark 4 It should be noted that if $k = n - \ell$ in (9) for $1 \le \ell \le 3$, then A_2B_2 and $(A_2 + A_3)(B_2 + B_3)$ have the same last ℓ terms. Similarly, $(A_0 + A_2 + (A_1 + A_3)X^n)(B_0 + B_2 + (B_1 + B_3)X^n)$ and A_1B_1 have the same last ℓ terms. Therefore, once A_2B_2 and A_1B_1 are computed using the school-book method, the cost of computing $(A_2 + A_3)(B_2 + B_3)$ and $(A_0 + A_2 + (A_1 + A_3)X^n)(B_0 + B_2 + (B_1 + B_3)X^n)$ is less than or equal to $M_2(n) - \ell^2$ and $M_2(2n) - \ell^2$, respectively. Thus, we get the following recursion:

$$M_2(4n - \ell) \le M_2(2n) + 5M_2(n) + M_2(n - \ell) + 27n - 8\ell - 8 - 2\ell^2, \quad 1 \le \ell \le 3. \tag{10}$$

2.6 CNH 3-way split algorithm

Let $A, B, C, A_0, A_1, A_2, B_0, B_1$, and B_2 be defined as in Bernstein's 3-way algorithm. In [10,11], Cenk, Negre, and Hasan proposed the following algorithm for computing $C = AB$, where α is the generator of $\mathbb{F}_4$:

$$\begin{cases} P_0 = A_0B_0,\ P_1 = (A_0 + A_1 + A_2)(B_0 + B_1 + B_2), \\ P_2 = (A_0 + A_2 + \alpha(A_1 + A_2))(B_0 + B_2 + \alpha(B_1 + B_2)), \\ P_3 = (A_0 + A_1 + \alpha(A_1 + A_2))(B_0 + B_1 + \alpha(B_1 + B_2)), \\ P_4 = A_2B_2, \\ C = (P_0 + X^nP_4)(1 + X^{3n}) + (P_1 + (1 + \alpha)(P_2 + P_3)) \\ \quad (X^n + X^{2n} + X^{3n}) + \alpha(P_2 + P_3)X^{3n} + P_2X^{2n} + P_3X^n \end{cases} \tag{11}$$

Table 1 Cost of polynomial multiplication over $\mathbb{F}_2$

Algorithm	Split	$M_2(n)$	$D_2(n)$
Bernstein [3]	2	$6.5n^{1.58} - 7n + 1.5$	$3\log_2(n)D_X + D_A$
Bernstein [3]	3	$25.5n^{1.46} - 25.5n + 1$	$(1.5n + 8\log_3(n) - 1.5)D_X + D_A$
CNH [9]	3	$5.8n^{1.63} - 6n + 1.2$	$4\log_3(n)D_X + D_A$
CNH [10,11]	3	$30.25n^{1.46} - 28n + 4.75$	$10\log_3(n)D_X + D_A$
Proposed (24)	3	$15.125n^{1.46} - 2.67n\log_3(n) - 14.25n + 0.125$	$10\log_3(n)D_X + D_A$
Bernstein [3]	4	$6.425n^{1.58} - 6.8n + 1.375$	$5\log_4(n)D_X + D_A$
Proposed (17)	5	$6.46n^{1.58} - 6.877n + 1.42$	$13\log_5(n)D_X + D_A$

The complexities of the algorithm are computed in [10,11] as follows:

$$\begin{cases} M_2(3n) \le 2M_4(n) + 3M_2(n) + 29n - 12, \\ M_4(3n) \le 5M_4(n) + 58n - 21, \\ D_2(n) \le D_4(n/3) + 8D_X, \\ D_4(n) \le D_4(n/3) + 10D_X. \end{cases} \quad (12)$$

Remark 5 We can improve this algorithm by observing the common additions in $(P_1 + (1+\alpha)(P_2 + P_3))(X^n + X^{2n} + X^{3n})$. Assume that the inputs are from $\mathbb{F}_4[X]$. For simplicity let $R = (P_1 + (1+\alpha)(P_2 + P_3))$. Since R is a degree $(2n-2)$ polynomial, we can write $R = R_0 + R_1X^n$ where R_0 is a degree $(n-1)$ polynomial and R_1 is a degree $(n-2)$ polynomial. We have then

$$R(X^n + X^{2n} + X^{3n}) \\ = X^nR_0 + X^{2n}(R_0 + R_1) + X^{3n}(R_0 + R_1) + X^{4n}R_1,$$

requiring $2(n-1)$ $\mathbb{F}_4$ additions for $R_0 + R_1$ which improves the original computation cost $2(2n-2)$. It should be noted that this technique does not change the delay complexity. The complexity for degree $(2n+k)$ polynomials can be easily be obtained for $1 \le k \le n$ since, in this case, $(A_1 + A_2)$, $(B_1 + B_2)$, $((A_0 + A_1) + A_2)$, and $((B_0 + B_1) + B_2)$ each require $8k$ additions. As well, $(P_0 + X^nP_4)$ needs $(n-1)$ additions if $k > n/2$ and $(2k-1)$ additions if $k < n/2$. The following are thus the new complexities for polynomial multiplication over $\mathbb{F}_4$:

$$\begin{cases} M_4(3n) \le 5M_4(n) + 56n - 19, \quad M_4(1) = 7, \\ M_4(2n+k) \le 4M_4(n) + M_4(k) + 48n + 8k - 19, \\ \quad n/2 \le k \le n, \\ M_4(2n+k) \le 4M_4(n) + M_4(k) + 46n + 12k - 19, \\ \quad 1 \le k < n/2, \\ D_4(n) \le D_4(n/3) + 10D_X, D_4(1) = 2D_X + D_A \\ M_4(n) \le 30.25n^{1.46} - 28n + 4.75, \\ D_4(n) \le (10\log_3(n) + 2)D_X + D_A. \end{cases} \quad (13)$$

Similarly, the complexities over $\mathbb{F}_2$ are obtained as follows:

$$\begin{cases} M_2(n) \le 2M_4(n/3) + 3M_2(n/3) + 29n - 12, \quad M_2(1) = 1, \\ D_2(n) \le D_4(n/3) + 8D_X, \quad D_2(1) = D_A, \\ M_2(n) \le 30.25n^{1.46} - 9.27n\log_3(n) - 27.5n + 0.75, \\ D_2(n) \le 10\log_3(n)D_X + D_A. \end{cases} \quad (14)$$

3 New improved algorithms over $\mathbb{F}_2$

This section presents a method that yields better complexities than the Bernstein 3-way algorithm. Moreover, a new 5-way split algorithm for binary polynomial multiplication resulting from improvements to the one described in [12] is introduced, and a new 3-way split algorithm with improved complexity is also proposed. The complexity comparisons of the methods introduced in this section are included in Table 1.

3.1 A new split method for Bernstein's 3-way split algorithm

Let $A(X) = \sum_{i=0}^{3n-1} a_iX^i$ and $B(X) = \sum_{i=0}^{3n-1} b_iX^i$ be two polynomials of degree $3n-1$. In this method, we compute $(XA(X))(XB(X))$ instead of $A(X)B(X)$ using Bernstein's 3-way split algorithm. Note that $XA(X) = \sum_{i=0}^{3n-1} a_iX^{i+1}$ and $XB(X) = \sum_{i=0}^{3n-1} b_iX^{i+1}$ are degree $3n$ polynomials with first terms zero. We now apply Bernstein's 3-way split algorithm by assuming that $XA(X)$ and $XB(X)$ are degree $3n+2$ polynomials. Here, we take the coefficients of X^{3n+1} and X^{3n+2} of both $XA(X)$ and $XB(X)$ as zero, and thus we have:

$$XA(X) = A_0 + A_1X^{n+1} + A_2X^{2n+2}, \\ XB(X) = B_0 + B_1X^{n+1} + B_2X^{2n+2},$$

where each of A_i and B_i for $0 \le i \le 2$ are degree n polynomials. However, it should be noted that the first term of A_0 and B_0 is zero and that the last two terms of A_2 and B_2

are zero. Therefore, we can say that this method splits $3n$-term polynomials as $(n, n+1, n-1)$ rather than (n, n, n) where the i-th value in the triples for $i = 1, 2, 3$ shows the number of terms of A_i and B_i. The computational cost of Bernstein's 3-way algorithm for this splitting approach is as follows:

- $4n - 2$: Computing $A_0 + A_1 + A_2$ and $B_0 + B_1 + B_2$. These are degree n polynomials.
- $2n-2$: Computing $A_1X + A_2X^2$ and $B_1X + B_2X^2$. These are degree $(n + 1)$ polynomials with the constant term being zero.
- $2n$: Computing $A_0 + (A_1X + A_2X^2)$ and $B_0 + (B_1X + B_2X^2)$. These are degree $(n + 1)$ polynomials with the constant term being zero.
- $2n$: Computing $A_0 + A_1 + A_2 + (A_1X + A_2X^2)$ and $B_0 + B_1 + B_2 + (B_1X + B_2X^2)$. These are degree $(n+1)$ polynomials.
- $M_2(n)$: Computing $P_0 = A_0B_0$ where P_0 is a degree $2n$ polynomial with the constant term and the coefficient of X as zero.
- $M_2(n+1)$: Computing $P_1 = (A_0+A_1+A_2)(B_0+B_1+B_2)$ where P_1 is a degree $2n$ polynomial.
- $M_2(n+1)$: Computing $P_2 = (A_0+A_1X+A_2X^2)(B_0+B_1X + B_2X^2)$ where P_2 is a degree $2n + 2$ polynomial with the constant term and the coefficient of X being zero.
- $M_2(n + 2) - 1$: Computing $P_3 = (A_0 + A_1 + A_2 + A_1X + A_2X^2)(B_0 + B_1 + B_2 + B_1X + B_2X^2)$ where P_3 is a degree $2n + 2$ polynomial and the last term is the same as that of P_2.
- $M_2(n-1)$: Computing $P_4 = A_2B_2$ where P_4 is a degree $2n - 4$ polynomial.
- $2n$: Computing $S = P_2 + P_3$ where S is a degree $(2n+1)$ polynomial because the last terms of P_2 and P_3 are equal.
- $3n - 1$: Computing $U = P_0 + (P_0 + P_1)X^{n+1}$ where U is a degree $3n + 1$ polynomial and the first two terms are zero.
- $3n + 3$: Computing $V = P_2 + S(X^{n+1} + X)$ where V is a degree $3n + 2$ term with the first term being zero.
- $7n - 6$: Computing $W = U + V + P_4(X^4 + X)$ where W is a degree $3n + 2$ polynomial with the first term as zero.
- $3n$: Computing $W' = W/(X(X + 1))$ where W' is a degree $3n$ polynomial.
- $2n$: Computing $W'' = W'(X^{2n+2} + X^{n+1})$ where W'' is a degree $5n + 2$ polynomial with first n terms being zero.
- $5n-3$: Computing $C = U + P_4(X^{4n+4} + X^{n+1}) + W''$. This is the product polynomial $X^2A(X)B(X)$.

It should also be noted that the original algorithm is better for $(3n - 1)$ terms polynomials. However, for $(2n + k)$ term polynomials with $1 \leq k \leq n - 2$, the proposed splitting approach yields better results than the original recursion. For example, the method introduced above splits $(3n - 2)$ term polynomials as $(n-1, n, n-1)$ instead of $(n, n, n-2)$. The recursions for the above computations for a $3n$-term and a similar computations for $(3n - 2)$ term polynomials can be summed up as follows:

$$\begin{cases} M_2(3n) \leq M_2(n) + 2M_2(n+1) + M(n+2) \\ \quad +M(n-1) + 35n - 12, \\ M_2(3n-2) \leq 2M_2(n) + M_2(n+1) + 2M(n-1) \\ \quad +35n - 13. \end{cases} \tag{15}$$

3.2 Improved 5-way split algorithm

This section presents a new improvement to the 5-way split algorithm described in [12]. Let $A = \sum_{i=0}^{5n-1} a_iX^i$ and $B = \sum_{i=0}^{5n-1} b_iX^i$ two degree $(5n - 1)$ polynomials over $\mathbb{F}_2$ and $C = \sum_{i=0}^{10n-2} c_iX^i$ be their product. This method splits A and B in five parts as $A = A_0 + A_1X^n + A_2X^{2n} + A_3X^{3n} + A_4X^{4n}$, $B = B_0 + B_1X^n + B_2X^{2n} + B_3X^{3n} + B_4X^{4n}$, where $A_j = \sum_{i=0}^{n-1} a_{i+nj}X^i$ and $B_j = \sum_{i=0}^{n-1} b_{i+nj}X^i$ for $j = 0, 1, 2, 3, 4$. Then we can write $C = \sum_{i=0}^{8} C_iX^{in}$. Cenk and Özbudak proposed the following algorithm in [12]:

$$\begin{cases} m_1 = A_0B_0,\ m_2 = A_1B_1,\ m_3 = A_2B_2,\ m_4 = A_3B_3, \\ m_5 = A_4B_4, m_6 = (A_0 + A_1)(B_0 + B_1), \\ m_7 = (A_0 + A_2)(B_0 + B_2),\ m_8 = (A_2 + A_4)(B_2 + B_4), \\ m_9 = (A_3 + A_4)(B_3 + B_4), \\ m_{10} = (A_0 + A_2 + A_3)(B_0 + B_2 + B_3), \\ m_{11} = (A_1 + A_2 + A_4)(B_1 + B_2 + B_4), \\ m_{12} = (A_0 + A_3 + A_1 + A_4)(B_0 + B_3 + B_1 + B_4), \\ m_{13} = (A_0+A_1+A_2+A_3+A_4)(B_0+B_1+B_2+B_3+B_4), \\ C_0 = m_1,\ C_1 = m_6+m_1+m_2,\ C_2 = m_7+m_1+m_3+m_2, \\ C_3 = m_1 + m_{13} + m_{12} + m_{10} + m_8 + m_3 + m_5 + m_4, \\ C_4 = m_6+m_1+m_2+m_{13}+m_{10}+m_{11}+m_9+m_5+m_4, \\ C_5 = m_7 + m_1 + m_3 + m_2 + m_{13} + m_{11} + m_{12} + m_5, \\ C_6 = m_8+m_3+m_5+m_4,\ C_7 = m_9+m_4+m_5,\ C_8 = m_5. \end{cases} \tag{16}$$

The improvement to this algorithm is based on the use of the method described in [27]. To this end, we divide each m_i for $1 \leq i \leq 13$ into two parts as $m_i = p_{2i-1} + p_{2i}X^n$, where p_{2i-1} is a degree $(n-1)$ polynomial, p_{2i} is a degree $(n-2)$ polynomial, and $n \geq 2$. We substitute the new decompositions of the m_i's into C_i's and let the new representation of C be $C = \sum_{i=1}^{10} U_iX^{(i-1)n}$. The explicit new algorithm is as follows:

$$\begin{cases} t_1 = p_1 + p_2,\ t_2 = t_1 + p_3,\ t_3 = t_2 + p_{11}, \\ t_4 = p_4 + p_5,\ t_5 = p_{12} + p_{13}, t_6 = t_4 + t_5,\ t_7 = t_2 + t_6, \\ t_8 = t_1 + t_4,\ t_9 = p_6 + p_7,\ t_{10} = t_8 + t_9, \\ t_{11} = t_{10} + p_9,\ t_{12} = p_{14} + p_{15},\ t_{13} = t_{11} + t_{12}, \\ t_{14} = p_{19} + p_{23},\ t_{15} = t_{14} + p_{25}, t_{16} = t_{13} + t_{15}, \\ t_{17} = p_8 + p_9,\ t_{18} = t_{17} + p_{10},\ t_{19} = t_{18} + p_{18}, \\ t_{20} = t_{18} + t_9, t_{21} = p_{16} + p_{17},\ t_{22} = t_{20} + t_{21}, \\ t_{23} = t_{22} + t_3,\ t_{24} = p_{20} + p_{21}, t_{25} = p_{24} + p_{25}, \\ t_{26} = p_{19} + p_{24},\ t_{27} = t_{24} + t_{25},\ t_{28} = t_{27} + t_{26}, \\ t_{29} = t_{28} + t_{23}, t_{30} = t_7 + t_{19},\ t_{31} = t_{27} + t_{30}, \\ t_{32} = p_{22} + p_{23},\ t_{33} = t_{31} + t_{32},\ t_{34} = t_{11} + p_1, \\ t_{35} = t_{34} + p_{10},\ t_{36} = t_{35} + t_{12},\ t_{37} = t_{36} + p_{22}, \\ t_{38} = t_{37} + p_{24},\ t_{39} = t_{38} + p_{26}, \\ U_1 = p_1,\ U_2 = t_3,\ U_3 = t_7,\ U_4 = t_{16},\ U_5 = t_{29}, \\ U_6 = t_{33},\ U_7 = t_{39}, U_8 = t_{22},\ U_9 = t_{19},\ U_{10} = p_{10}, \end{cases} \tag{17}$$

The cost of (17) is $(39n - 17)$ additions. The cost of linear combinations of A_i's and the linear combinations of B_i's can be computed with a total of $16n$ additions. The following recursion is thus obtained:

$$M_2(5n) \leq 13M_2(n) + 55n - 17. \tag{18}$$

When the input sizes are $(4n + k)$ for $1 \leq k \leq n$, the sizes of A_4 and B_4 are then k bits and the cost of $(A_2 + A_4)$, $(A_3 + A_4)$, $(B_2 + B_4)$, and $(B_3 + B_4)$ is $4k$ rather than $4n$. On the other hand, the size of $m_5 = A_4 B_4 = p_9 + p_{10} X^n$ is a $2k - 1$. It should be noted that p_9 is an n-bit polynomial, p_{10} is a $(2k - n - 1)$-bit polynomial for $n/2 \leq k \leq n$, p_9 is a $(2k - 1)$-bit polynomial, and p_{10} is the 0 polynomial for $1 \leq k < n/2$. When the cost of t_{11}, t_{17}, t_{18}, and t_{35} in (17) is re-computed, the following recursion is obtained:

$$M_2(4n + k) \leq 12M_2(n) + M_2(k) + 47n + 8k - 17. \tag{19}$$

An additional remark can be made regarding the case of $k = n - \ell$ for $1 \leq \ell \leq 3$. Here, the last ℓ terms of m_4 and m_9 are identical, and similarly the last ℓ terms of m_3 and m_8 are identical. We can, therefore, write

$$M_2(5n - \ell) \leq 12M_2(n) + M_2(n - \ell) + 55n - 8\ell - 17 - \ell^2. \tag{20}$$

The delay complexity can be computed as

$$D_2(5n) \leq D_2(n) + 13D_X. \tag{21}$$

The complexities are summarized as follows:

$$\begin{cases} M_2(5n) \leq 13M_2(n) + 55n - 17, \\ M_2(4n + k) \leq 12M_2(n) + M_2(k) + 47n + 8k - 17, \\ \quad 1 \leq k \leq n, \\ D_2(5n) \leq D_2(n) + 13D_X. \end{cases} \tag{22}$$

Asymptotic complexities of this algorithm are the following:

$$\begin{cases} M_2(n) \leq 13M_2(n/5) + 55n/5 - 17,\ M_2(1) = 1, \\ M_2(n) \leq 6.46n^{1.58} - 6.87n + 1.42, \\ D_2(n) \leq D_2(n/5) + 13D_X,\ D_2(1) = D_A, \\ D_2(n) \leq 13\log_5(n)D_X + D_A. \end{cases} \tag{23}$$

3.3 New improved 3-way algorithm

This section presents a process for improving the algorithm discussed in Sect. 2.6 by about 50%. The enhancement is obtained by analyzing the products P_2 and P_3 in (11). Let A, B, C, A_0, A_1, A_2, B_0, B_1, and $B_2 \in \mathbb{F}_2[X]$ be defined as in the explanation of the CNH algorithm in Sect. 2. It should be noted that if

$$P_2 = (A_0 + A_2 + \alpha(A_1 + A_2))(B_0 + B_2 + \alpha(B_1 + B_2)) \\ = P_{2,0} + \alpha P_{2,1},$$

then one can compute

$$P_3 = (A_0 + A_1 + \alpha(A_1 + A_2))(B_0 + B_1 + \alpha(B_1 + B_2)) \\ = (P_{2,0} + P_{2,1}) + \alpha P_{2,1}.$$

This calculation shows that P_3 can be obtained from P_2. Note that this method works because $A_i, B_i \in \mathbb{F}_2[X]$ for $0 \leq i \leq 2$. By using $P_3 = (P_{2,0} + P_{2,1}) + \alpha P_{2,1}$, we propose the following algorithm:

$$\begin{cases} P_0 = A_0 B_0,\ P_1 = (A_0 + A_1 + A_2)(B_0 + B_1 + B_2), \\ P_4 = A_2 B_2, \\ P_2 = (A_0 + A_2 + \alpha(A_1 + A_2))(B_0 + B_2 + \alpha(B_1 + B_2)) \\ \quad = P_{2,0} + \alpha P_{2,1}, \\ C = P_4 X^{4n} + (P_0 + P_1 + P_{2,1})X^{3n} + (P_{2,0} \\ \quad + P_1 + P_{2,1})X^{2n} + (P_4 + P_1 + P_{2,0})X^n + P_0 \end{cases} \tag{24}$$

Now we can compute the complexity of this algorithm where A_0, B_0, A_1, and B_1 are degree $(n - 1)$ polynomials and A_2 and B_2 are degree $(k - 1)$ polynomials. Assume that $1 \leq k \leq n$. Each of $(A_1 + A_2)$ and $(A_0 + A_2)$ then requires k additions, and $(A_0 + (A_1 + A_2))$ requires n additions. Since the polynomials are over $\mathbb{F}_2$, $(A_0 + A_2 + \alpha(A_1 + A_2))$ does not require any additions. Similarly, the right-hand side, i.e., B_i's, require $(n + 2k)$ additions. On the other hand, each of $(P_1 + P_{2,1})$, $(P_0 + (P_1 + P_{2,1}))$, $(P_{2,0} + (P_1 + P_{2,1}))$ and

$(P_1 + P_{2,0})$ requires $(2n - 1)$ additions, and $(P_4 + (P_1 + P_{2,0}))$ requires $(2k - 1)$ additions. Finally, the overlaps of the coefficients of X^0, X^n, X^{2n}, and X^{3n} require $(3n - 3)$ additions, and the cost of the overlapping of the coefficient of X^{4n} with the other terms is $(n - 1)$ if $n/2 \le k \le n$, and $(2k - 1)$ if $1 \le k < n/2$. On the other hand, the delay complexity can be computed as described in [11] and we obtain the complexities as follows:

$$\begin{cases} M_2(3n) \le 3M_2(n) + M_4(n) + 20n - 5, \\ M_2(2n+k) \le 2M_2(n) + M_2(k) + M_4(n) + 14n + 6k - 5, \\ \quad n/2 \le k \le n, \\ M_2(2n+k) \le 2M_2(n) + M_2(k) + M_4(n) + 13n + 8k - 11, \\ \quad 1 \le k < n/2. \\ D_2(3n) \le D_4(n) + 7D_X, \end{cases} \tag{25}$$

In order to compute $M_2(n)$, we need $M_4(n)$. By using the results in (13), one can obtain asymptotic complexities of this algorithm as follows:

$$\begin{cases} M_2(n) \le 3M_2(n/3) + M_4(n/3) + 20n/3 - 5, \ M_2(1) = 1, \\ M_2(n) \le 15.125n^{1.46} - 14.25n - 2.4274\log_3(n) + 0.125, \\ D_2(n) \le D_4(n/3) + 8D_X, \ D_2(1) = D_A, \\ D_2(n) \le 10\log_3(n)D_X + D_A. \end{cases} \tag{26}$$

3.4 Comparison of complexities

To enable an easy comparison, the complexity results are presented in Table 1. As it can be seen, the 2-way algorithm is the Karatsuba algorithm with Bernstein's improvement. On the other hand, the proposed 3-way algorithm is far superior to the 3-way split algorithms. Bernstein's 4-way split and the proposed 5-way split algorithms that yield improvements are also included in the table. It should also be noted that Negre has reported [21,22] about improvements in the 3-way splits algorithm of [9] with a complexity $4.68n^{1.63} + O(n)$ and in the 4-way split algorithm of [3] with a complexity $5.25n^{1.58} + O(n)$.

4 Minimum number of bit operations for $M_4(n)$

The algorithm presented in Sect. 3.3 entails the multiplication of polynomials over $\mathbb{F}_4$. Efficient algorithms for multiplication over $\mathbb{F}_4$ are, therefore, needed to obtain better complexity results over $\mathbb{F}_2$. We can use the multiplication algorithms over $\mathbb{F}_2$ presented in the previous sections for multiplications over $\mathbb{F}_4$. However, it should be noted that the addition of $\mathbb{F}_4$ elements requires two-bit additions and that the multiplication of $\mathbb{F}_4$ elements requires seven-bit operations, i.e., four multiplications and three additions (using the school-book algorithm). The determination of the cost of multiplications over $\mathbb{F}_4$, therefore, requires the following modifications to the recursions presented in the previous sections: $M_2(n)$ is converted to $M_4(n)$, and the number of additions over $\mathbb{F}_2$ is multiplied by two. If the algorithm includes bit multiplications (as in the case of the school-book algorithm), then the number of bit multiplications is multiplied by seven, which is the cost of multiplication in $\mathbb{F}_4$. As an illustration, the school-book algorithm for the multiplication of polynomials over $\mathbb{F}_4$ can be modified as follows: Let A and B be degree n polynomials over $\mathbb{F}_4$. We can write $A = A_0 + X^n a_n$ and $B = B_0 + X^n b_n$, where A_0 and B_0 are degree $(n-1)$ polynomials over $\mathbb{F}_4$, and a_n and b_n are in $\mathbb{F}_4$. Then

$$A \cdot B = A_0B_0 + X^n(A_0b_n + a_nB_0) + X^{2n}a_nb_n.$$

The costs of A_0B_0, $(A_0b_n + a_nB_0)$ and a_nb_n are $M_4(n)$, $2nM_4(1) + 2n$, and $M_4(1)$, respectively. The final overlap needs $2(n-1)$ additions. Using $M_4(1) \le 7$, we obtain the following:

$$\begin{cases} M_4(n+1) \le M_4(n) + 18n + 5, \\ D_4(n+1) \le D_4(n) + D_X. \end{cases} \tag{27}$$

Similarly, the improved Karatsuba algorithm presented in Sect. 2 has the following recursion for $\mathbb{F}_4$ multiplications:

$$\begin{cases} M_4(n+k) \le 2M_4(n) + M_4(k) + 6n + 8k - 6, \\ \quad n/2 \le k \le n, \\ D_4(2n) \le D_4(n) + 3D_X. \end{cases} \tag{28}$$

On the other hand, the 3-way algorithm discussed in Sect. 2 has the following recursion for multiplications over

$$\begin{cases} M_4(2n+k) \le 5M_4(n) + M_4(k) + 24n + 12k - 12, \\ \quad n/2 < k \le n, \\ D_4(3n) \le D_4(n) + 4D_X. \end{cases} \tag{29}$$

Bernstein's 4-way split algorithm presented in Sect. 2 can be used for multiplication over $\mathbb{F}_4$ using the following recursion:

$$\begin{aligned} M_4(3n+k) \le\ & M_4(2n) \\ & + 5M_4(n) + M_4(k) + 38n + 16k - 16, \quad n/2 \le k \le n. \end{aligned} \tag{30}$$

The recursive equation for the new 5-way split algorithm introduced in Sect. 3.2 can be used for multiplications over $\mathbb{F}_4$ by applying the following recursion:

$$\begin{cases} M_4(4n+k) \le 12M_4(n) + M_4(k) + 96n + 16k - 36, \\ \quad 1 \le k \le n, \\ D_4(4n) \le D_4(n) + 5D_X. \end{cases} \tag{31}$$

The next step is to describe a general method for multiplying polynomials over $\mathbb{F}_4$. Let α be the generator of $\mathbb{F}_4$, $A = \sum_{i=0}^{n-1} a_i X^i$, $B = \sum_{i=0}^{n-1} B_i X^i$ and $C = AB = \sum_{i=0}^{2n-2} C_i X^i$ be polynomials over $\mathbb{F}_4$. We can write, $A = A_0 + \alpha A_1$ and $B = B_0 + \alpha B_1$ where A_0, A_1, B_0, and B_1 are degree $n-1$ polynomials over $\mathbb{F}_2$. We then have

$$\begin{aligned} AB &= (A_0 + \alpha A_1)(B_0 + \alpha B_1) \\ &= A_0B_0 + A_1B_1 + ((A_0 + A_1)(B_0 + B_1) + A_0B_0)\alpha. \end{aligned} \tag{32}$$

The complexity of this formula can be computed as

$$\begin{cases} M_4(n) \le 3M_2(n) + 6n - 2. \\ D_4(n) \le D_2(n) + 2D_X. \end{cases} \tag{33}$$

As a final step, we can then use the CNH 3-way algorithm discussed in Sect. 2. The recursion of this algorithm is the following:

$$\begin{cases} M_4(3n) \le 5M_4(n) + 56n - 19, \\ M_4(2n+k) \le 4M_4(n) + M_4(k) + 48n + 8k - 19, \\ \quad n/2 \le k \le n, \\ D_4(n) \le D_4(n/3) + 10D_X. \end{cases} \tag{34}$$

5 Improved upper bounds over $\mathbb{F}_2$

This section presents the new upper bounds on the minimum number of operations for binary polynomial multiplications with the use of the algorithms discussed in the previous sections.

The first improvement is for $n = 9$. The improved 3-way algorithm presented in Sect. 2 yields $M_2(9) \le 126$ whereas this bound is reported as 132 in [2]. On the other hand, the new 5-way algorithm results in $M_2(15) \le 317$, which is better than the 326 arrived at [6]. Explicit algorithms for $n = 9$ and $n = 15$ are presented in the appendix. Similarly, we obtain $M_2(18) \le 438$, which is better than that reported in [6]. For $n = 11, 12$, we were unable to obtain improvements on the upper bounds compared to the results described in [6]. However, for almost all values of n greater than 20, we have obtained improved bounds and tabulated new bounds for some specific values of n, which are used in cryptographic applications. Details are included in the appendix.

We also note that although improvements in the number of bit operations can be obtained primarily through modifications to Bernstein's 3-way algorithm, the corresponding level of delay complexities is significantly higher because Bernstein's 3-way algorithm entails a linear delay complexity in input size. For this reason, we have also searched the minimum number of bit operations with a logarithmic delay. In this respect, the new 3-way algorithm introduced in Sect. 3.3 produces the best results. It should be noted that although the numbers of operations increase slightly, delay complexities decrease significantly since the new 3-way split algorithm is associated with a logarithmic delay. The results are summarized in Table 2 that includes four different complexities. Column A shows the known best bounds reported in [2] and [6] before the current work. The improved minimum numbers of bit operations over $\mathbb{F}_2$ and $\mathbb{F}_4$ are listed in columns B and C, respectively, and the best possible minimum number of bit operations with logarithmic delay complexities are indicated in column D. In addition to $M_2(n)$ and $M_4(n)$, the table also provides the name of the algorithm along with the new size of the polynomial after splitting.

The numbers in the column entitled Alg. of Table 2 represent the following algorithms: 1 is the school-book, 2 is the Karatsuba with Bernstein's improvement, 2.1 is the Karatsuba with Bernstein's improvement with input size $2n-1$, 2.2 is the Karatsuba with Bernstein's improvement with input size $2n-2$, 2.3 is the Karatsuba with Bernstein's improvement with input size $2n-3$, 3 is Karatsuba-like 3-way split, 5 is Bernstein's 3-way split, 5.1 is modified Bernstein's 3-way split algorithm with input size $3n$, 5.2 is modified Bernstein's 3-way split algorithm with input size $3n-2$, 6 is Bernstein's 4-way split with input size $4n$, 6.1 is Bernstein's 4-way split with input size $4n-1$, 6.2 is for Bernstein's 4-way split with input size $4n-2$, 7 is for the improved 5-way split for input size $5n$, 7.1 is improved 5-way split for input size $5n-1$, 8 is for the method referring in [6], 9 is the general method described in Sect. 4, 10 is the Karatsuba algorithm with Bernstein's improvements for $\mathbb{F}_4$, 14 is the improved CNH 3-way split algorithm over $\mathbb{F}_4$ in Sect. 2, 15 is Bernstein's 4-way for polynomials over $\mathbb{F}_4$, and finally 16 is the improved 5-way split for polynomials over $\mathbb{F}_4$.

For example, for $n = 15$ in column B, it can be seen that the new 5-way algorithm is used, and the new size of the polynomials becomes five. To verify the complexity, one should then use the $M_2(5)$. It must also be noted that special care should be given in those cases in which the size of the polynomials after splitting may be different, as in the case of $M_2(17)$, which contains a multiplication of size nine and a multiplication of size eight. An additional remark is related to the modified Bernstein's algorithm. If

Table 2 New upper bounds on $M_2(n)$, $D_2(n)$, $M_4(n)$ and $D_4(n)$ where A, B, and C present minimum number of bit operations; and D presents minimum number of bit operations with logarithmic delay

n	A	B				C				D			
	$M_2(n)$	$M_2(n)$	$D_2(n)$	Alg.	Split	$M_4(n)$	$D_4(n)$	Alg.	Split	$M_2(n)$	$D_2(n)$	Alg.	Split
2	5	5	2	1	1	25	4	9	2	5	2	1	1
3	13	13	3	1	2	55	5	9	3	13	3	1	2
4	25	25	4	1	3	97	6	9	4	25	4	1	3
5	41	41	5	1	4	151	7	9	5	41	5	1	4
6	57	57	6	2	3	201	8	10	3	57	6	2	3
7	81	81	7	1	6	283	9	9	7	81	7	1	6
8	100	100	7	2	4	339	11	15	2	100	7	2	4
9	132	**126**	7	3	3	424	15	14	3	126	7	3	3
10	155	155	8	2	5	513	17	16	2	155	8	2	5
11	186	186	7	8	0	616	11	10	6	186	7	8	0
12	207	207	7	8	0	677	13	15	3	207	7	8	0
13	255	255	8	8	0	841	10	9	13	255	8	8	0
14	289	289	10	2	7	941	12	10	7	289	10	2	7
15	326	**317**	16	7	3	1015	18	16	3	317	16	7	3
16	349	349	8	8	0	1121	16	15	4	349	8	8	0
17	413	**407**	10	2.1	9	1264	18	14	6	407	10	2.1	9
18	454	**438**	10	2	9	1322	18	14	6	438	10	2	9
19	498	498	11	2.1	10	1569	20	10	10	498	11	2.1	10
20	527	527	8	8	0	1673	20	10	10	527	8	8	0
21	602	**596**	11	2.1	11	1788	19	14	7	596	11	2.1	11
22	641	**632**	10	2	11	1970	21	14	8	632	10	2	11
23	678	**676**	10	2.1	12	2060	21	14	8	676	10	2.1	12
24	704	**702**	10	2	12	2124	21	14	8	702	10	2	12
25	800	**791**	18	7	5	2448	25	14	9	791	18	7	5
26	856	**853**	11	2	13	2512	25	14	9	853	11	2	13
27	922	**912**	11	3	9	2605	25	14	9	912	11	3	9
28	956	956	15	6	7	2916	27	14	10	956	15	6	7
29	1044	**1020**	19	2.1	15	3009	27	14	10	1020	19	2.1	15
30	1085	**1053**	19	2	15	3106	27	14	10	1053	19	2	15
31	1129	**1119**	19	2.1	16	3460	21	10	16	1119	19	2.1	16
32	1158	**1156**	11	2	16	3566	27	14	11	1156	11	2	16
33	1286	**1274**	13	2.1	17	3677	21	14	11	1274	13	2.1	17
34	1358	**1335**	13	2.2	18	3858	27	14	12	1335	13	2.2	18
35	1441	**1393**	15	6.1	9	3969	23	14	12	1393	15	6.1	9
36	1483	**1429**	15	6	9	4038	23	14	12	1429	15	6	9
37	1585	**1559**	14	2.1	19	4673	21	14	13	1559	14	2.1	19
38	1636	**1616**	13	2.2	20	4742	23	14	13	1616	13	2.2	20
39	1687	**1680**	13	6.1	10	4914	20	14	13	1680	13	6.1	10
40	1720	**1718**	11	2	20	5190	23	14	14	1718	11	2	20
41	1871	**1858**	14	2.1	21	5362	22	14	14	1858	14	2.1	21
42	1950	**1929**	13	2.2	22	5470	22	14	14	1929	13	2.2	22
43	2020	**1996**	15	6.1	11	5706	28	14	15	1996	15	6.1	11
44	2064	**2037**	15	6	11	5814	28	14	15	2037	15	6	11
45	2150	**2116**	20	7	9	5896	28	14	15	2116	20	7	9

Table 2 continued

n	A	B				C				D			
	$M_2(n)$	$M_2(n)$	$D_2(n)$	Alg.	Split	$M_4(n)$	$D_4(n)$	Alg.	Split	$M_2(n)$	$D_2(n)$	Alg.	Split
46	2192	**2182**	15	6.2	12	6286	26	14	16	2182	15	6.2	12
47	2239	**2229**	15	6.1	12	6368	28	14	16	2229	15	6.1	12
48	2268	**2260**	15	6	12	6482	26	14	16	2260	15	6	12
49	2460	**2451**	21	2.1	25	6988	28	14	17	2451	21	2.1	25
50	2572	**2545**	21	2	25	7102	28	14	17	2545	21	2	25
51	2677	**2668**	16	6.1	13	7253	28	14	17	2668	16	6.1	13
52	2735	**2726**	16	6	13	7382	28	14	18	2726	16	6	13
53	2881	**2858**	14	2.1	27	7533	28	14	18	2858	14	2.1	27
54	2948	**2922**	14	2	27	7599	28	14	18	2922	14	2	27
55	3017	**3006**	20	7	11	8569	30	14	19	3006	20	7	11
56	3060	**3060**	20	6	14	8635	30	14	19	3060	20	6	14
57	3239	**3191**	22	2.1	29	8890	30	14	19	3191	22	2.1	29
58	3320	**3256**	22	2.2	30	9099	30	14	20	3256	22	2.2	30
59	3406	**3304**	20	7.1	12	9354	30	14	20	3304	20	7.1	12
60	3456	**3334**	20	7	12	9466	30	14	20	3334	20	7	12
61	3552	**3500**	22	2.1	31	9862	30	14	21	3500	22	2.1	31
62	3595	**3571**	22	2	31	9974	30	14	21	3571	22	2	31
63	3651	**3632**	21	6.1	16	10,097	29	14	21	3632	21	6.1	16
64	3682	**3674**	16	6	16	10,750	31	14	22	3674	16	6	16
65	3938	**3927**	16	2.1	33	10,873	31	14	22	3927	16	2.1	33
66	4050	**4040**	86	5.1	22	11,063	31	14	22	4048	16	2.2	34
67	4134	**4110**	88	5.2	23	11,281	31	14	23	4159	18	2.3	35
68	4183	**4167**	88	5	23	11,462	31	14	24	4228	18	6	17
69	4403	**4296**	97	5.1	23	11,569	31	14	23	4356	18	2.3	36
70	4452	**4374**	99	5.2	24	11,775	31	14	24	4420	20	6.2	18
71	4499	**4476**	99	5	24	11,873	31	14	24	4494	20	6.1	18
72	4642	**4535**	20	6	18	11,945	31	14	24	4535	20	6	18
73	4828	**4701**	101	5.2	25	13,217	35	14	25	4798	18	2.1	37
74	4864	**4839**	101	5	25	13,289	35	14	25	4892	29	7.1	15
75	5097	**4929**	29	7	15	13,521	35	14	26	4929	29	7	15
76	5133	**5097**	103	5.2	26	13,593	35	14	26	5109	18	6	19
77	5239	**5205**	101	5	26	13,925	35	14	26	5241	16	2.1	39
78	5322	**5297**	16	6.2	20	13,997	35	14	26	5297	16	6.2	20
79	5384	**5359**	29	7.1	16	14,345	35	14	27	5359	29	7.1	16
80	5420	**5400**	21	7	16	14,417	35	14	27	5400	21	7	16
81	5740	**5630**	110	5.1	27	14,518	35	14	27	5713	17	2.1	41
82	5799	**5723**	112	5.2	28	15,709	37	14	28	5854	16	2.2	42
83	5875	**5818**	112	5	28	15,810	37	14	28	5983	18	2.3	43
84	5996	**5929**	113	5.1	28	16,129	37	14	29	6064	18	6	21
85	6158	**6007**	115	5.2	29	16,230	37	14	29	6209	23	7	17
86	6202	**6091**	115	5	29	16,549	37	14	29	6284	20	6.2	22
87	6353	**6204**	116	5.1	29	16,650	37	14	29	6369	20	6.1	22
88	6397	**6302**	118	5.2	30	16,985	37	14	30	6415	20	6	22
89	6495	**6388**	118	5	30	17,086	37	14	30	6576	23	2.1	45
90	6568	**6500**	117	5	30	17,191	37	14	30	6660	23	2	45

Table 2 continued

n	A	B				C				D			
	$M_2(n)$	$M_2(n)$	$D_2(n)$	Alg.	Split	$M_4(n)$	$D_4(n)$	Alg.	Split	$M_2(n)$	$D_2(n)$	Alg.	Split
91	6666	**6572**	120	5.2	31	18,550	37	14	31	6794	23	2.1	46
92	6717	**6662**	120	5	31	18,655	37	14	31	6851	20	6	23
93	6991	**6831**	120	5.1	31	19,017	31	14	31	6944	23	2.3	48
94	7043	**6931**	122	5.2	32	19,127	37	14	32	7013	18	2	47
95	7096	**7073**	120	5	32	19,489	37	14	32	7076	20	6.1	24
96	7132	**7112**	20	6	24	19,603	37	14	32	7112	20	6	24
97	7516	**7337**	121	5.2	33	19,981	31	14	33	7496	21	1	96
98	7574	**7503**	121	5	33	20,095	37	14	33	7684	24	2.2	50
99	7870	**7636**	124	5.1	33	20,214	31	14	33	7859	26	6.1	25
100	7909	**7766**	126	5	34	20,867	37	14	34	7934	21	7	20
101	8047	**7894**	126	5	34	20,986	37	14	34	8230	24	2.1	51
102	8184	**7979**	129	5	35	21,175	37	14	34	8345	24	2.2	52
103	8322	**8097**	129	5.2	35	21,478	33	14	35	8466	23	6.1	26
104	8404	**8178**	129	5	35	21,667	37	14	35	8538	21	6	26
105	8635	**8358**	129	5.1	35	21,786	33	14	35	8805	19	2.1	53
106	8717	**8450**	131	5.2	36	21,991	37	14	36	8932	19	2.2	54
107	8810	**8603**	131	5	36	22,110	33	14	36	8998	31	4	36
108	8959	**8758**	131	5	36	22,187	33	14	36	9040	31	4	36
109	9141	**8874**	133	5.2	37	24,154	34	17	108	9311	23	2.1	55
128	11,486	**11,466**	21	6	32	30,675	38	14	43	11,466	21	6	32
135	12,453	**12,309**	163	5.1	45	31,981	38	14	45	13,077	23	6.1	34
136	12,499	**12,422**	165	5.2	46	33,499	38	14	46	13,148	23	6	34
137	12,595	**12,522**	163	5	46	33,589	38	14	46	13,415	21	2.1	69
163	16,923	**16,828**	194	5.2	55	43,939	39	17	162	17,919	24	2.3	83
189	20,985	**20,671**	218	5.1	63	53,994	39	14	63	21,766	25	6.3	48
191	21,104	**21,048**	218	5	64	56,654	41	14	64	21,919	25	6.1	48
233	29,354	**29,156**	274	5	79	74,254	45	14	78	31,381	43	4	78
251	33,096	**32,604**	376	5	84	84,147	47	14	85	34,748	29	6.1	63
256	34,079	**33,397**	383	5.2	86	87,106	47	14	86	35,230	26	6	64
269	36,086	**35,656**	399	5	90	90,863	47	14	90	38,876	45	4	90
270	36,266	**35,832**	400	5.1	90	90,976	47	14	90	38,966	45	4	90
271	36,409	**35,978**	402	5.2	91	95,859	48	17	270	40,046	46	1	270
272	36492	**36127**	402	5	91	96,460	47	14	91	40344	28	6	68
273	37,084	**36,400**	403	5.1	91	96,815	47	14	92	40,747	45	4	91
274	37,167	**36,506**	405	5.2	92	96,928	47	14	92	40,840	45	4	92
283	38,735	**38,432**	414	5.2	95	102,258	47	14	95	42,468	45	4	95
407	67,374	**66,931**	581	5	136	173,566	48	14	136	75,581	46	4	136
408	67,582	**67,137**	583	5.1	136	173,876	48	14	137	75,658	46	4	136
409	67,753	**67,284**	585	5.2	137	173,974	48	14	137	76,219	46	4	137
571	112,569	**111,621**	870	5.2	191	291,271	51	14	191	126,061	49	4	191

In *A*, the values of $n = 11, 12, 13, 15, 16, 17, 18, 19, 20$ are from [6] and the other values are from [3]. The algorithm names are explained in Sect. 5. The improvements are emphasized using bold fonts the size is a multiple of three, say $3n$, then the sizes of the polynomials after splitting are n, $n+1$, and $n-1$; if the size is $3n-2$, then the new sizes are n and $n-1$. For example, for $3n-2=67$, the size of the new polynomial is 23 given in Table 2 and the other sizes are then both 22.

6 Conclusion

This paper has presented improvements in the bounds reported in [3,6] for binary polynomial multiplication through two new proposed algorithms along with the optimization and modification of previous algorithms. The use of the new 3-way and 5-way split algorithms together with the modification of Bernstein's 3-way split algorithm produces improved results. These results for values of n that are of interest for cryptographic applications are presented in the appendix. The latter also presents the algorithms for $n=9$ and $n=15$. Finally, it should be noted that the results in this paper can be further improved by eliminating common operations that appeared in the algorithms.

Acknowledgments The authors would like to thank undergraduate research assistant Ryan Young, who wrote a C-code for them to automate the generation of a part of the data included in Table 2 in Appendix A. The authors would also like to thank Dr. Rene Peralta for commenting on the explicit formulas presented in the paper. This work was supported in part by an NSERC grant awarded to Dr. M. Anwar Hasan. Part of this paper was written while Dr. Murat Cenk was a postdoctoral fellow in the Department of Electrical and Computer Engineering at the University of Waterloo. Dr. Murat Cenk was partially supported by TUBITAK under Grant No. BIDEB—114C052.

Appendix A: New bounds for multiplication over $\mathbb{F}_2$

We give the new bounds for certain values of n that are of interest for cryptographic applications. Note that the improvements can be further enhanced by obtaining the explicit algorithm and eliminating common operations as in [2,3]. The results are shown in Table 2.

Appendix B: Algorithms for $n = 9$ and $n = 15$

For $n=9$, $A=\sum_{i=0}^{8} b[i]X^i$, $B=\sum_{i=0}^{8} b[i]X^i$ and $C=AB=\sum_{i=0}^{16} c[i]X^i$. The coefficients of C are computed using the following algorithm:

Algorithm for $n = 9$

$t1 = a0 * b0$	$t22 = t20 + t21$	$t43 = b3 + b6$	$t64 = b2 + b5$	$t85 = t78 * t82$	$t106 = t26 + t30$	$c0 = t1$
$t2 = a0 * b1$	$t23 = a4 * b5$	$t44 = b4 + b7$	$t65 = t59 * t62$	$t86 = t79 * t81$	$t107 = t99 + t105$	$c1 = t4$
$t3 = a1 * b0$	$t24 = a5 * b4$	$t45 = b5 + b8$	$t66 = t59 * t63$	$t87 = t85 + t86$	$t108 = t100 + t106$	$c2 = t9$
$t4 = t2 + t3$	$t25 = t23 + t24$	$t46 = t40 * t43$	$t67 = t60 * t62$	$t88 = t78 * t83$	$t109 = t101 + t35$	$c3 = t102$
$t5 = a0 * b2$	$t26 = a5 * b5$	$t47 = t40 * t44$	$t68 = t66 + t67$	$t89 = t79 * t82$	$t110 = t76 + t84$	$c4 = t103$
$t6 = a1 * b1$	$t27 = a6 * b6$	$t48 = t41 * t43$	$t69 = t59 * t64$	$t90 = t80 * t81$	$t111 = t77 + t87$	$c5 = t104$
$t7 = a2 * b0$	$t28 = a6 * b7$	$t49 = t47 + t48$	$t70 = t60 * t63$	$t91 = t88 + t89$	$t112 = t107 + t110$	$c6 = t112$
$t8 = t5 + t6$	$t29 = a7 * b6$	$t50 = t40 * t45$	$t71 = t61 * t62$	$t92 = t90 + t91$	$t113 = t108 + t111$	$c7 = t113$
$t9 = t7 + t8$	$t30 = t28 + t29$	$t51 = t41 * t44$	$t72 = t69 + t70$	$t93 = t79 * t83$	$t114 = t109 + t92$	$c8 = t114$
$t10 = a1 * b2$	$t31 = a6 * b8$	$t52 = t42 * t43$	$t73 = t71 + t72$	$t94 = t80 * t82$	$t115 = t105 + t38$	$c9 = t123$
$t11 = a2 * b1$	$t32 = a7 * b7$	$t53 = t50 + t51$	$t74 = t60 * t64$	$t95 = t93 + t94$	$t116 = t106 + t39$	$c10 = t124$
$t12 = t10 + t11$	$t33 = a8 * b6$	$t54 = t52 + t53$	$t75 = t61 * t63$	$t96 = t80 * t83$	$t117 = t115 + t97$	$c11 = t122$
$t13 = a2 * b2$	$t34 = t31 + t32$	$t55 = t41 * t45$	$t76 = t74 + t75$	$t97 = t12 + t14$	$t118 = t116 + t98$	$c12 = t125$
$t14 = a3 * b3$	$t35 = t33 + t34$	$t56 = t42 * t44$	$t77 = t61 * t64$	$t98 = t13 + t17$	$t119 = t35 + t22$	$c13 = t126$
$t15 = a3 * b4$	$t36 = a7 * b8$	$t57 = t55 + t56$	$t78 = a0 + a6$	$t99 = t97 + t1$	$t120 = t46 + t117$	$c14 = t35$
$t16 = a4 * b3$	$t37 = a8 * b7$	$t58 = t42 * t45$	$t79 = a1 + a7$	$t100 = t98 + t4$	$t121 = t49 + t118$	$c15 = t38$
$t17 = t15 + t16$	$t38 = t36 + t37$	$t59 = a0 + a3$	$t80 = a2 + a8$	$t101 = t22 + t9$	$t122 = t54 + t119$	$c16 = t39$
$t18 = a3 * b5$	$t39 = a8 * b8$	$t60 = a1 + a4$	$t81 = b0 + b6$	$t102 = t99 + t65$	$t123 = t95 + t120$	
$t19 = a4 * b4$	$t40 = a3 + a6$	$t61 = a2 + a5$	$t82 = b1 + b7$	$t103 = t100 + t68$	$t124 = t96 + t121$	
$t20 = a5 * b3$	$t41 = a4 + a7$	$t62 = b0 + b3$	$t83 = b2 + b8$	$t104 = t101 + t73$	$t125 = t57 + t115$	
$t21 = t18 + t19$	$t42 = a5 + a8$	$t63 = b1 + b4$	$t84 = t78 * t81$	$t105 = t25 + t27$	$t126 = t58 + t116$	

For $n = 15$, $A = \sum_{i=0}^{14} a[i]X^i$, $B = \sum_{i=0}^{14} a[i]X^i$ and $C = AB = \sum_{i=0}^{28} c[i]X^i$. The coefficients of C are computed using the following algorithm:

Algorithm for $n = 15$

t1 = a[0] * b[0]	t59 = a[14] * b[12]	t117 = t114 + t115	t175 = t174 + t173	t233 = t230 + t220	t291 = t276 + t288
t2 = a[0] * b[1]	t60 = t57 + t58	t118 = t117 + t116	t176 = t162 * t166	t234 = t231 + t221	t292 = t277 + t289
t3 = a[1] * b[0]	t61 = t60 + t59	t119 = t105 * t109	t177 = t163 * t165	t235 = t232 + t222	t293 = t233 + t265
t4 = t2 + t3	t62 = a[13] * b[14]	t120 = t106 * t108	t178 = t176 + t177	t236 = t226 + t218	t294 = t234 + t266
t5 = a[0] * b[2]	t63 = a[14] * b[13]	t121 = t119 + t120	t179 = t163 * t166	t237 = t227 + t219	t295 = t235 + t61
t6 = a[1] * b[1]	t64 = t62 + t63	t122 = t106 * t109	t180 = t123 + t66	t238 = t35 + t9	t296 = t284 + t293
t7 = a[2] * b[0]	t65 = a[14] * b[14]	t123 = a[12] + a[9]	t181 = t124 + t67	t239 = t40 + t38	t297 = t285 + t294
t8 = t5 + t6	t66 = a[3] + a[0]	t124 = a[13] + a[10]	t182 = t125 + t68	t240 = t43 + t39	t298 = t286 + t295
t9 = t8 + t7	t67 = a[4] + a[1]	t125 = a[14] + a[11]	t183 = t126 + t69	t241 = t239 + t236	t299 = t178 + t186
t10 = a[1] * b[2]	t68 = a[5] + a[2]	t126 = b[12] + b[9]	t184 = t127 + t70	t242 = t240 + t237	t300 = t179 + t189
t11 = a[2] * b[1]	t69 = b[3] + b[0]	t127 = b[13] + b[10]	t185 = t128 + t71	t243 = t48 + t238	t301 = t296 + t299
t12 = t10 + t11	t70 = b[4] + b[1]	t128 = b[14] + b[11]	t186 = t180 * t183	t244 = t53 + t241	t302 = t297 + t300
t13 = a[2] * b[2]	t71 = b[5] + b[2]	t129 = t123 * t126	t187 = t180 * t184	t245 = t56 + t242	t303 = t298 + t194
t14 = a[3] * b[3]	t72 = t66 * t69	t130 = t123 * t127	t188 = t181 * t183	t246 = t61 + t243	t304 = t1 + t244
t15 = a[3] * b[4]	t73 = t66 * t70	t131 = t124 * t126	t189 = t187 + t188	t247 = t110 + t102	t305 = t4 + t245
t16 = a[4] * b[3]	t74 = t67 * t69	t132 = t130 + t131	t190 = t180 * t185	t248 = t113 + t103	t306 = t9 + t246
t17 = t15 + t16	t75 = t73 + t74	t133 = t123 * t128	t191 = t181 * t184	t249 = t247 + t244	t307 = t64 + t304
t18 = a[3] * b[5]	t76 = t66 * t71	t134 = t124 * t127	t192 = t182 * t183	t250 = t248 + t245	t308 = t65 + t305
t19 = a[4] * b[4]	t77 = t67 * t70	t135 = t125 * t126	t193 = t190 + t191	t251 = t118 + t246	t309 = t247 + t307
t20 = a[5] * b[3]	t78 = t68 * t69	t136 = t133 + t134	t194 = t193 + t192	t252 = t186 + t148	t310 = t248 + t308
t21 = t18 + t19	t79 = t76 + t77	t137 = t136 + t135	t195 = t181 * t185	t253 = t189 + t151	t311 = t118 + t306
t22 = t21 + t20	t80 = t79 + t78	t138 = t124 * t128	t196 = t182 * t184	t254 = t194 + t156	t312 = t178 + t309
t23 = a[4] * b[5]	t81 = t67 * t71	t139 = t125 * t127	t197 = t195 + t196	t255 = t252 + t205	t313 = t179 + t310
t24 = a[5] * b[4]	t82 = t68 * t70	t140 = t125 * t128	t198 = t182 * t185	t256 = t253 + t208	t314 = t197 + t312
t25 = t23 + t24	t83 = t81 + t82	t141 = t138 + t139	t199 = t180 + a[6]	t257 = t254 + t213	t315 = t198 + t313
t26 = a[5] * b[5]	t84 = t68 * t71	t142 = a[9] + t85	t200 = t181 + a[7]	t258 = t249 + t255	t316 = t216 + t314
t27 = a[6] * b[6]	t85 = a[6] + a[0]	t143 = a[10] + t86	t201 = t182 + a[8]	t259 = t250 + t256	t317 = t217 + t315
t28 = a[6] * b[7]	t86 = a[7] + a[1]	t144 = a[11] + t87	t202 = t183 + b[6]	t260 = t251 + t257	c0 = t1
t29 = a[7] * b[6]	t87 = a[8] + a[2]	t145 = b[9] + t88	t203 = t184 + b[7]	t261 = t53 + t51	c1 = t4
t30 = t28 + t29	t88 = b[6] + b[0]	t146 = b[10] + t89	t204 = t185 + b[8]	t262 = t56 + t52	c2 = t9
t31 = a[6] * b[8]	t89 = b[7] + b[1]	t147 = b[11] + t90	t205 = t199 * t202	t263 = t261 + t64	c3 = t223
t32 = a[7] * b[7]	t90 = b[8] + b[2]	t148 = t142 * t145	t206 = t199 * t203	t264 = t262 + t65	c4 = t224
t33 = a[8] * b[6]	t91 = t85 * t88	t149 = t142 * t146	t207 = t200 * t202	t265 = t263 + t141	c5 = t225
t34 = t31 + t32	t92 = t85 * t89	t150 = t143 * t145	t208 = t206 + t207	t266 = t264 + t140	c6 = t233
t35 = t34 + t33	t93 = t86 * t88	t151 = t149 + t150	t209 = t199 * t204	t267 = t263 + t239	c7 = t234
t36 = a[7] * b[8]	t94 = t92 + t93	t152 = t142 * t147	t210 = t200 * t203	t268 = t264 + t240	c8 = t235
t37 = a[8] * b[7]	t95 = t85 * t90	t153 = t143 * t146	t211 = t201 * t202	t269 = t61 + t48	c9 = t258
t38 = t36 + t37	t96 = t86 * t89	t154 = t144 * t145	t212 = t209 + t210	t270 = t121 + t129	c10 = t259
t39 = a[8] * b[8]	t97 = t87 * t88	t155 = t152 + t153	t213 = t212 + t211	t271 = t122 + t132	c11 = t260
t40 = a[9] * b[9]	t98 = t95 + t96	t156 = t155 + t154	t214 = t200 * t204	t272 = t267 + t270	c12 = t290
t41 = a[9] * b[10]	t99 = t98 + t97	t157 = t143 * t147	t215 = t201 * t203	t273 = t268 + t271	c13 = 291
t42 = a[10] * b[9]	t100 = t86 * t90	t158 = t144 * t146	t216 = t214 + t215	t274 = t269 + t137	c14 = t292
t43 = t41 + t42	t101 = t87 * t89	t159 = t157 + t158	t217 = t201 * t204	t275 = t272 + t223	c15 = t301
t44 = a[9] * b[11]	t102 = t100 + t101	t160 = t144 * t147	t218 = t12 + t1	t276 = t273 + t224	c16 = t302
t45 = a[10] * b[10]	t103 = t87 * t90	t161 = t104 + a[3]	t219 = t13 + t4	t277 = t274 + t225	c17 = t303
t46 = a[11] * b[9]	t104 = a[12] + a[6]	t162 = t105 + a[4]	t220 = t14 + t218	t278 = t159 + t167	c18 = t316
t47 = t44 + t45	t105 = a[13] + a[7]	t163 = t106 + a[5]	t221 = t17 + t219	t279 = t160 + t170	c19 = t317
t48 = t47 + t46	t106 = a[14] + a[8]	t164 = t107 + b[3]	t222 = t22 + t9	t280 = t205 + t216	c20 = t311
t49 = a[10] * b[11]	t107 = b[12] + b[6]	t165 = t108 + b[4]	t223 = t72 + t220	t281 = t208 + t217	c21 = t272
t50 = a[11] * b[10]	t108 = b[13] + b[7]	t166 = t109 + b[5]	t224 = t75 + t221	t282 = t148 + t197	c22 = t273
t51 = t49 + t50	t109 = b[14] + b[8]	t167 = t161 * t164	t225 = t80 + t222	t283 = t151 + t198	c23 = t274
t52 = a[11] * b[11]	t110 = t104 * t107	t168 = t161 * t165	t226 = t27 + t25	t284 = t278 + t280	c24 = t265
t53 = a[12] * b[12]	t111 = t104 * t108	t169 = t162 * t164	t227 = t30 + t26	t285 = t279 + t281	c25 = t266
t54 = a[12] * b[13]	t112 = t105 * t107	t170 = t168 + t169	t228 = t91 + t83	t286 = t175 + t213	c26 = t61
t55 = a[13] * b[12]	t113 = t111 + t112	t171 = t161 * t166	t229 = t94 + t84	t287 = t282 + t284	c27 = t64
t56 = t54 + t55	t114 = t104 * t109	t172 = t162 * t165	t230 = t228 + t226	t288 = t283 + t285	c28 = t65
t57 = a[12] * b[14]	t115 = t105 * t108	t173 = t163 * t164	t231 = t229 + t227	t289 = t156 + t286	
t58 = a[13] * b[13]	t116 = t106 * t107	t174 = t171 + t172	t232 = t99 + t35	t290 = t275 + t287	

References

1. Barbulescu, R., Detrey, J., Estibals, N., Zimmermann, P.: Finding optimal formulae for bilinear maps. In: WAIFI, pp. 168–186 (2012)
2. Bernstein, D.J.: Minimum number of bit operations for multiplication (2013). http://binary.cr.yp.to/m.html. Accessed 25 Jan 2013
3. Bernstein, D.J.: Batch binary edwards. In: Advances in Cryptology—CRYPTO 2009, LNCS, vol. 5677, pp. 317–336 (2009)
4. Bodrato, M.: Towards optimal toom-cook multiplication for univariate and multivariate polynomials in characteristic 2 and 0. In: WAIFI, pp. 116–133 (2007)
5. Bodrato, M., Zanoni, A.: Integer and polynomial multiplication: towards optimal toom-cook matrices. In: ISSAC, pp. 17–24 (2007)
6. Boyar, J., Dworkin, M., Fischer, M., Peralta, R., Visconti, A., Schiavo, C., Turan, M., Calik, C., Wood, C.: Past collaborators include: M. Bartock, B. Strackbein, C. Baker, J. Svensson, H. Gao, S. Zimmermann, and M. Bocchi. Circuit minimization work. A web page including explicit formulas for multiplication over the binary field by the Circuit Minimization Team at the Yale University (2013). http://www.cs.yale.edu/homes/peralta/CircuitStuff/CMT.html. Accessed 25 Nov 2013
7. Brent, R.P., Gaudry, P., Thomé, E., Zimmermann, P.: Faster multiplication in GF(2)[x]. In: ANTS, pp. 153–166 (2008)
8. Cenk, M., Koç, Ç.K., Özbudak, F.: Polynomial multiplication over finite fields using field extensions and interpolation. In: IEEE Symposium on Computer Arithmetic, pp. 84–91 (2009)
9. Cenk, M., Hasan, M.A., Negre, C.: Efficient subquadratic space complexity binary polynomial multipliers based on block recombination. IEEE Trans. Comput. **63**(9), 2273–2287 (2014)
10. Cenk, M., Negre, C., Hasan, M.A.: Improved three-way split formulas for binary polynomial multiplication. In: Selected Areas in Cryptography, pp. 384–398 (2011)
11. Cenk, M., Negre, C., Hasan, M.A.: Improved three-way split formulas for binary polynomial and toeplitz matrix vector products. IEEE Trans. Comput. **62**(7), 1345–1361 (2013)
12. Cenk, M., Özbudak, F.: Improved polynomial multiplication formulas over $\mathbb{F}_2$ using Chinese remainder theorem. IEEE Trans. Comput. **58**(4), 572–576 (2009)
13. Dyka, Z., Langendoerfer, P., Vater, F.: Combining multiplication methods with optimized processing sequence for polynomial multiplier in GF(2^k). In: WEWoRC, pp. 137–150 (2011)
14. Dyka, Z., Langendoerfer, P., Vater, F., Peter, S.: Towards strong security in embedded and pervasive systems: energy and area optimized serial polynomial multipliers in GF(2^k). In: NTMS, pp. 1–6 (2012)
15. Erdem, S.S., Koç, Ç.K.: A less recursive variant of karatsuba-ofman algorithm for multiplying operands of size a power of two. In: IEEE Symposium on Computer Arithmetic, pp. 28–35 (2003)
16. Erdem, S.S., Yanik, T., Koç, Ç.K.: Polynomial basis multiplication over GF(2^m). Acta Appl. Math. **93**(1–3), 33–55 (2006)
17. Fan, H., Hasan, M.A.: Comments on "five, six, and seven-term Karatsuba-like formulae". IEEE Trans. Comput. **56**(5), 716–717 (2007)
18. Fan, H., Sun, J., Gu, M., Lam, K.-Y.: Overlap-free Karatsuba–Ofman polynomial multiplication algorithms. Inf. Secur. IET **4**, 8–14 (2010)
19. Karatsuba, A.A., Ofman, Y.: Multiplication of multidigit numbers on automata. Sov. Phys. Dokl. **7**, 595–596 (1963)
20. Montgomery, P.L.: Five, six, and seven-term Karatsuba-like formulae. IEEE Trans. Comput. **54**(3), 362–369 (2005)
21. Negre, C.: Improved three-way split approach for binary polynomial multiplication based on optimized reconstruction. In: Technical Report hal-00788646, Team DALI/LIRMM, on Hyper Articles en Ligne (HAL) (2013)
22. Negre, C.: Efficient binary polynomial multiplication based on optimized Karatsuba reconstruction. J. Cryptogr. Eng. **4**(2), 91–106 (2014)
23. Chang, N.S., Kim, C.H., Park, Y.-H., Lim, J.: A non-redundant and efficient architecture for Karatsuba–Ofman algorithm. In: ISC, pp. 288–299 (2005)
24. Sunar, B.: A generalized method for constructing subquadratic complexity GF(2^k) multipliers. IEEE Trans. Comput. **53**, 1097–1105 (2004)
25. von zur Gathen, J., Shokrollahi, J.: Efficient fpga-based karatsuba multipliers for polynomials over F_2. In: Selected Areas in Cryptography, pp. 359–369 (2005)
26. Winograd, S.: Arithmetic Complexity of Computations. Society For Industrial and Applied Mathematics, Philadelphia (1980)
27. Zhou, G., Michalik, H.: Comments on "a new architecture for a parallel finite field multiplier with low complexity based on composite field". IEEE Trans. Comput. **59**(7), 1007–1008 (2010)
28. Zhou, G., Michalik, H., Hinsenkamp, L.: Complexity analysis and efficient implementations of bit parallel finite field multipliers based on karatsuba-ofman algorithm on fpgas. IEEE Trans. VLSI Syst. **18**(7), 1057–1066 (2010)