# Generalizations of $n$-Term Karatsuba Like Formulae in $GF(2^n)$ with NAYK Algorithm

Muhamad Nursalman, *Member*, *IAENG*, Arif Sasongko, Yusuf Kurniawan, Kuspriyanto

***Abstract*—There have been many researches that were developed to reduce multiplication operations in polynomial multiplier in $GF(2^n)$. This is important because of its connection with the efficient implementation in restricted devices, such as in elliptic curve cryptography. Two of them are the researches conducted by Paar for $n$=4 and Montgomery for $n$=5,6,7. Their researches are better than previous researches, but they do not explain their methods, how they can find the multipliers. This is the knowledge gap and in this research sought to find a method is better than what they have done in finding the multipliers. The first step is to develop a formula that is better than Generalizations of The Karatsuba Algorithm, after then develop an exhaustive search algorithm for all possible existing products. Then, combine both of these formulas and algorithm, which we refer to as the NAYK algorithm. This algorithm can explain how to reduce the multiplications significantly, that is by identifying some multiplications or products included in the solution and some products which is not included in the solution. So the rest of products is reduced significantly. Then we use the products have been identified are included in the solution, and with reference to the upper bound of the function of O($n$), then the lack of products is added from a combination of existing residual products. This causes the search space becomes much smaller significanly than the Montgomery algorithm. Further, the NAYK algorithm allows to search multiplier for $n$>7. NAYK algorithm is suitable for use in composite field because it can improve efficiency significantly.**

***Index Terms*— Exhaustive Search Algorithm, Improving of Generalizations of The Karatsuba Algoritm, NAYK Algorithm, Generalizations of $n$-Term Karatsuba Like Formulae, polynomial multiplication in $GF(2^n)$, Composite Field**

## I. Introduction

THE efficiency of multiplying two polynomials is a crucial matter in fields such as cryptography and signal processing. The high resource requirements polynomial multiplication in the Galois Field $GF(2^n)$ leads to the high number of point multiplications up to the curve level in Elliptic Curve Cryptography (ECC) [13-14][21] resulting in very large processing times. This can not be separated from the process of polynomial multiplication in the arithmetic level in $GF(2^n)$, which consumes a lot of resources and causes the process to not only be time consuming, but also require a large area, especially if a large bit size is used $[3-5]$.

Manuscript submitted June 8, 2016, 4th revised June 22, 2017.

Muhamad Nursalman, Arif Sasongko, Yusuf Kurniawan and Kuspriyanto are with the School of Electrical Engineering and Informatics, Institut Teknologi Bandung, Jl. Ganesha 10, Bandung 40132, Indonesia (corresponding author, phone: +6285601939046; e-mail: mnursalman@upi.edu or mnursalman@students.itb.ac.id, e-mail: asasongko,yusufk,kuspriyanto@stei.itb.ac.id).

This large number of multiplications cause difficulties in the hardware implementation phase, especially if the algorithm is implemented in restricted devices [15-17][20]. To solve this problem, the number of multiplications performed in polynomial multiplication in $GF(2^n)$ must be reduced. However, reducing the number of multiplications is not an easy task, in the research conducted by Paar in which a formula for the multiplication in the multiplier in $GF(2^n)$ for $n = 4$ was developed [6-7], the process of reducing the multiplications was not described in detail. Similarly, Montgomery used so-called division-free formulas in finding multipliers in $GF(2^n)$ for $n = 5, 6$, and 7, [18] and thus no detailed description of the algorithm used was given.

The main problem in this research is how to reduce the number of multiplication processes in the multiplier in $GF(2^n)$ using a more uncomplicated method or algorithm compared to that used in in the two previous research carried out [6-7][18], even with limited computer resources. To solve this problem, we developed an alternative formula that is more efficient than Generalizations of The Karatsuba Algorithm (GKA) in reducing the number of multiplications in the multiplier in $GF(2^n)$ [16], and also developed an exhaustive search algorithm to find the solution to the equations of mathematics that appear in polynomial multiplications in $GF(2^n)$.

## II. Polynomial Multipliers in Galois Field

### A. Simple and General Karatsuba Multiplier

Starting from ordinary polynomial multiplication as follows [9].

$$\begin{aligned} C(x) &= A(x)B(x) \bmod f(x) \\ &= \left(\sum_{i=0}^{n-1} a_i x^i\right)\left(\sum_{j=0}^{n-1} b_j x^j\right) \bmod f(x) \\ &= \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\left(a_i b_j\right) x^{i+j} \bmod f(x) \end{aligned} \quad (1)$$

where

$$\begin{aligned} A(x) &= a_{n-1}x^{n-1} + \ldots + a_1 x + a_0 \\ &= \sum_{i=0}^{n-1} a_i x^i \end{aligned}$$

and

$$\begin{aligned} B(x) &= b_{n-1}x^{n-1} + \ldots + b_1 x + b_0 \\ &= \sum_{j=0}^{n-1} b_j x^j \end{aligned}$$

Then Karatsuba divide A and B in (1) into two parts as follows.

$$\begin{aligned} A &= a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1 x + a_0 \\ &= x^{n/2}\left[a_{n-1}x^{n/2-1} + \ldots + a_{n/2}\right] + \left[a_{n/2-1}x^{n/2-1} + \ldots + a_0\right] \\ &= x^{n/2}A_H + A_L \end{aligned}$$

$$\begin{aligned} B &= b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \ldots + b_1 x + b_0 \\ &= x^{n/2}\left[b_{n-1}x^{n/2-1} + \ldots + b_{n/2}\right] + \left[b_{n/2-1}x^{n/2-1} + \ldots + b_0\right] \\ &= x^{n/2}B_H + B_L \end{aligned}$$

Then

$$\begin{aligned} AB &= \left(x^{n/2}A_H + A_L\right)\left(x^{n/2}B_H + B_L\right) \bmod f(x) \\ &= \left[x^n A_H B_H + x^{n/2}\left(A_H B_L + A_L B_H\right) + A_L B_L\right] \\ &\quad \bmod f(x) \end{aligned} \tag{2}$$

Then Karatsuba manipulate the equation (2) in the brackets as follows.

$$\begin{aligned} &A_H B_L + A_L B_H \\ &= A_H B_L + A_L B_H + \left(A_H B_H - A_H B_H\right) + \left(A_L B_L - A_L B_L\right) \\ &= \left(A_H B_H + A_H B_L + A_L B_H + A_L B_L\right) - A_H B_H - A_L B_L \\ &= \left(A_H + A_L\right)\left(B_H + B_L\right) - A_H B_H - A_L B_L \end{aligned} \tag{3}$$

Then (3) substitute into (2) as follows.

$$\begin{aligned} AB = &\left[x^n\left(A_H B_H\right) + x^{n/2}\left(\left(A_H + A_L\right)\left(B_H + B_L\right)\right.\right. \\ &\left.\left. - A_H B_H - A_L B_L\right) + A_L B_L\right] \bmod f(x) \end{aligned} \tag{4}$$

Previously (2), there are four products that must be processed,

$A_H B_H$, $A_H B_L$, $A_L B_H$, and $A_L B_L$.

But, after using a simple algorithm Karatsuba above, the products are reduced by one, so only three products are processed in (4) [2][10][17], these are

$A_H B_H$, $\left(A_H + A_L\right)\left(B_H + B_L\right)$ and $A_L B_L$.

If we divide A and B to be more than two products and we use the Karatsuba multiplication properties, then the method is called as a General Karatsuba Multiplier [3-5].

### B. Generalizations of The Karatsuba Algorithm (GKA)

Then if we divide A and B into the smallest form, where an product has only two coefficients (if n is odd, then there is one product contains only a single coefficient), then this method of multiplication between A and B is called as generalizations of The Karatsuba Algorithm [1][22]. Then the form will be as follows.

*Generalizations of The Karatsuba Algorithm*

Take any polynomial of degree ($n$-1) with $n$ coefficient

$$A(x) = \sum_{i=0}^{n-1} a_i x^i,\ B(x) = \sum_{i=0}^{n-1} b_i x^i$$

$\forall i = 0, 1, \ldots, n-1$ compute

$D_i = a_i b_i$,

and $\forall i = 1, 2, \ldots, 2n-3$

and $\forall j, k \ni j + k = i,\ 0 \le j < k$ compute

$D_{j,k} = (a_j + a_k)(b_j + b_k)$,

Then $C'(x) = A(x)B(x) = \sum_{i=0}^{2n-2} c'_i x^i$

can be computed as follows

$c'_0 = D_0$

$c'_{2n-2} = D_{n-1}$

$$c'_i = \sum_{\substack{j+k=i \\ 0 \le j < k \le n-1}} D_{j,k} - \sum_{\substack{j+k=i \\ 0 \le j < k \le n-1}} \left(D_j + D_k\right), \tag{5}$$

where $i \in \{1,3,5,\ldots,2n-3\}$

$$c'_i = \sum_{\substack{j+k=i \\ 0 \le j < k \le n-1}} D_{j,k} - \sum_{\substack{j+k=i \\ 0 \le j < k \le n-1}} \left(D_j + D_k\right) + D_{i/2},$$

where $i \in \{2, 4, 6, \ldots, 2n-4\}$

### C. Polynomial Multiplier for Galois Field $GF((2^n)^4)$

Furthermore, for $n$ = 4 in the equation GKA, Paar modifies it as follows [6-7][19][24].

Suppose

$A(x) = a_3 x^3 + a_2 x^2 + a_1 x + a_0;\ a_i \in GF(2^n);\ A \in GF((2^n)^4)$

and

$B(x) = b_3 x^3 + b_2 x^2 + b_1 x + b_0;\ b_i \in GF(2^n);\ B \in GF((2^n)^4)$

and

$C(x) = A(x) \times B(x) \bmod P(x)$

Then, Paar manipulate it into

$c'_0 = d_0$

$c'_1 = d_0 + d_1 + d_2$

$c'_2 = d_0 + d_2 + d_3 + d_6$

$c'_3 = d_0 + d_1 + d_2 + d_3 + d_4 + d_5 + d_6 + d_7 + d_8$

$c'_4 = d_2 + d_5 + d_6 + d_8$

$c'_5 = d_6 + d_7 + d_8$

$c'_6 = d_8$

where

$$d_0 = a_0 b_0$$
$$d_1 = (a_0 + a_1)(b_0 + b_1)$$
$$d_2 = a_1 b_1$$
$$d_3 = (a_0 + a_2)(b_0 + b_2)$$
$$d_4 = (a_0 + a_1 + a_2 + a_3)(b_0 + b_1 + b_2 + b_3)$$
$$d_5 = (a_1 + a_3)(b_1 + b_3)$$
$$d_6 = a_2 b_2$$
$$d_7 = (a_2 + a_3)(b_2 + b_3)$$
$$d_8 = a_3 b_3$$

If the same multiplication count one, then the number of multiplication operations is nine, where previously the number was ten, it means the number of multiplication process is reduced by one. In this research did not examine the modulo process, but more toward reducing the number of arithmetic multiplication process in polynomial multiplication between A and B. For modulo process itself has been researched and discussed in [11-12], where the methods use the results of the multiplier which already exists [7][18] to construct a multiplier for a value of $n$, which is greater than 7, and then look for modulo function which is suitable for the constructors of a multiplier.

### D. 5,6,7-Term Karatsuba Like Formulae

After Paar has developed multiplication for $n$ = 4, then Montgomery developed multiplication for $n$ = 5, 6, and 7, by utilizing and manipulating properties of Karatsuba Multiplications as follows [18].

Suppose

$$(a_0 + a_1 X)(b_0 + b_1 X) = a_0 b_0 + (a_0 b_1 + a_1 b_0) X + a_1 b_1 X^2 \quad (6)$$

Note that

$$a_0 b_1 + a_1 b_0 = (a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1$$

Then the equation (6) can be written as follows

$$(a_0 + a_1 X)(b_0 + b_1 X)$$
$$= a_0 b_0 + ((a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1) X + a_1 b_1 X^2$$
$$= a_0 b_0 (1 - X)$$
$$+ (a_0 + a_1)(b_0 + b_1) X$$
$$+ a_1 b_1 (X^2 - X)$$

Then for $n$ = 3, the formula is

$$(a_0 + a_1 X + a_2 X^2)(b_0 + b_1 X + b_2 X^2)$$
$$= a_0 b_0 (K + 1 - X - X^2)$$
$$+ a_1 b_1 (K - X + X^2 - X^3)$$
$$+ a_2 b_2 (K - X^2 - X^3 + X^4)$$
$$+ (a_0 + a_1)(b_0 + b_1)(-K + X)$$
$$+ (a_0 + a_2)(b_0 + b_2)(-K + X^2)$$
$$+ (a_1 + a_2)(b_1 + b_2)(-K + X^3)$$
$$+ (a_0 + a_1 + a_2)(b_0 + b_1 + b_2) K$$

If $K = X^2$, then the number of multiplication changed from nine to six, this is more efficient. Then, Montgomery utilize this pattern to form the multipliers for $n$ = 5, 6, and 7, where the complexity of those formulas can be found in [18]. The multipliers are better than the multipliers that have been developed previously. But even so, it was very difficult to obtain for each multiplier, especially for $n$ = 7, even Montgomery himself called it the divison-free formulas.

### E. Knowledge Gap

In research conducted by Paar and Montgomery, the process of developing formulas and algorithms to obtain multipliers was not described. In polynomial multiplication in the Galois Field $GF(2^n)$, for $n$ = 7, Montgomery stated that obtaining the multipliers requires a resource-intensive calculation process, while computer resources limited. Therefore, in this research developed an improved method and algorithm for easy determination of multipliers in a polynomial multiplication in $GF(2^n)$.

## III. Research Methodology

Figure 1 illustrates previous researches carried out by several researchers regarding GKA, as well as how the current study relates to those researches.

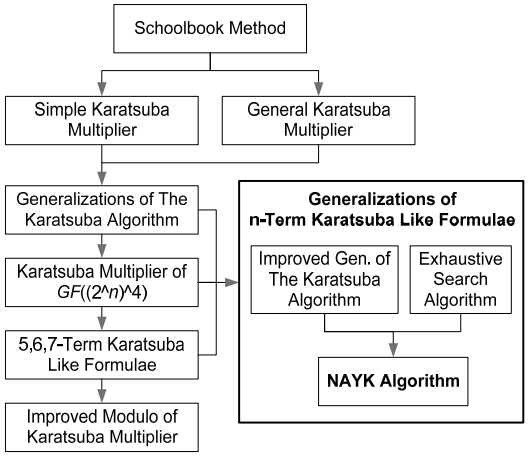


Fig. 1. Position and focus of the research of the Generalizations of n-Term Karatsuba Like Formulae with NAYK algorithm in Multiplier Researches Roadmap

There are three main steps to solve the problems mentioned above, the development of an improved formula compared to GKA, the development of an exhaustive search algorithm, and the combination of both methods,

subsequently named the Nursalman, Arif, Yusuf and Kuspriyanto (NAYK) algorithm. Previous researches on improving the modulo functions of karatsuba multiplications have been carried out by [11][12], including the determination of the best modulo functions that result in a reduced number of multiplications in polynomial multiplications in $GF(2^n)$. In this study, we did not focus on these modulo functions, but they can be integrated in further research for developing further improved algorithms for polynomial multiplications in $GF(2^n)$ for $n$=8, and for larger values of $n$.

### A. The Development of a Better Formula Than GKA

An improved formula was obtained by comparing the original equation $C'$ produced by GKA with the equation $C'$ produced by the Paar [7] and Montgomery [18] Multipliers (Figure 2). The Improved GKA formula combines at least two products into one new product, thus reducing the number of multiplications in the process.

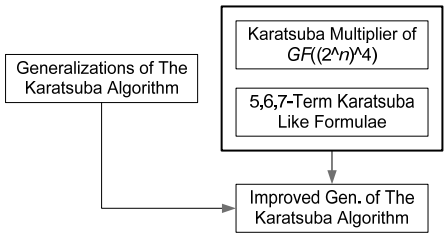


Fig. 2. Improving of the Generalizations of the Karatsuba Algorithm

In its application, the Improved GKA reduced the complexity of the polynomial multiplication process better than the original GKA. However, there are constraints in the algorithm for combining products for large values of $n$, resulting in difficulties for the algorithm to find a combination of combined products that are solutions to equation $C'$.

Therefore, we developed an exhaustive search algorithm that can find the combination of combined products that are solutions to equation $C'$.

### B. Exhaustive Search Algorithm

The algorithm developed is illustrated in Figure 3, and have the following steps:

1. Creating a matrix $C'$ with GKA formula
2. Creating a matrix $D$ for all combinations of existing values, 0,1,2, ..., $n$
3. Searching for all combinations of all rows in the matrix $D$
4. Summing up for each combination of line matrix $D$
5. Looking for a solution to the equation $C'$ from the sum of each combination

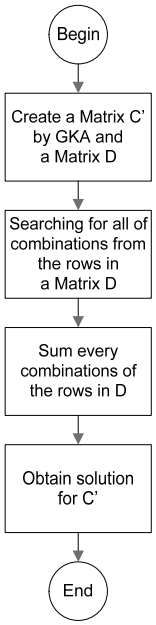


Fig. 3. The steps of the Exhaustive Search Algorithm

The algorithm was implemented in MATLAB and was then tested for a value of $n$. The results show that this algorithm can only run for $n$ = 4. For values of $n$ larger than 4, the amount of data processed was very large, requiring a very large amount of computer memory, and a very long processing time to find the solution.

### C. Nursalman, Arif and Yusuf Algorithm (NAYK Algorithm)

The Improved GKA algorithm was able to solve polynomial multiplications in $GF(2^n)$ for values of $n$ up to 4 with much less complexity compared to the original GKA, but failed to sufficiently reduce the complexity (determine all combinations of combined products that are solutions to equation $C'$) for larger values of $n$. In addition, the exhaustive search algorithm succeeded in efficiently finding combinations of combined products that are solutions to equation $C'$, but, similarly, only for values of $n$ of up to 4. To overcome the limitations of both algorithms, we developed a combination of the two, subsequently named the Nursalman, Arif, Yusuf, and Kuspriyanto (NAYK) Algorithm. The Improved GKA serves to identify which combinations of products are solutions to equation $C'$ and which are not. This reduces the number of rows in matrix $D$, which in turn reduces the number of lines from the many combinations of rows of matrix $D$. Through proper identification, only a small portion of the combinations of rows of matrix $D$ needs to be calculated. Thus, this method provides a reduction in the amount of data to be processed, and also reduces the computation time. Figure 4 illustrates the procedure of the NAYK algorithm.

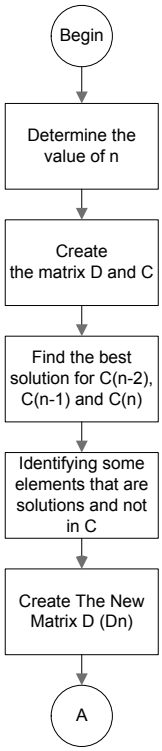


Fig. 4.a. The NAYK algorithm to find the new matrix *D*

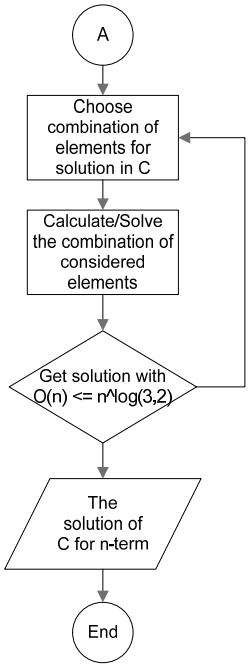


Fig. 4.b. The NAYK algorithm to find the products combination for solution equation of *C′*.

## IV. Results and Discussion

### A. The Development of a Better Formula Than GKA in $GF(2^n)$

In the Improved GKA formula, two or more products can be combined which consequently reduces the number of multiplications in the polynomial multiplication process. The stages of the development of the formula are as follows:

*1. Combining Products pattern in polynomial multiplication in $GF(2^n)$*

As has been described above, GKA has the form similar to equation (5). For $n$=4,

$$c_3' = D_{0,3} + D_{1,2} - \left(D_0 + D_3\right) - \left(D_1 + D_2\right)$$

The equation can be transformed into, the following:

$$\begin{aligned} c_3' &= D_{0,1,2,3} - D_{0,1} - D_{0,2} - D_{1,3} - D_{2,3} \\ &\quad - \left(D_0 + D_3\right) - \left(D_1 + D_2\right) \end{aligned} \tag{7}$$

which means that

$$D_{0,3} + D_{1,2} = D_{0,1,2,3} - D_{0,1} - D_{0,2} - D_{1,3} - D_{2,3},$$

where $D_{0,1}$, $D_{0,2}$, $D_{1,3}$, and $D_{2,3}$ are existing products. Note that in equation (7), there are two products missing, ($D_{0,3}$ and $D_{1,2}$), and a new product appears, $D(0,1,2,3)$. This means that the number of multiplications is reduced by one compared to the previous equation.

Note the first product on the right side of the above equation (5),

$$\sum_{\substack{j+k=i \\ 0\le j<k\le n-1}} D_{j,k}, \text{ for odd or even values of } n$$

By understanding the patterns of the above equation, it can be written in four forms as follows:

(1) For $1 < i \le n-1$ and $i \in$ odd number , then

$$D_{0,i} + D_{1,i-1} + \ldots + D_{(i-1)/2,(i+1)/2} = D_{0,1,2,\ldots,i} - \sum_{\substack{j+k\ne i \\ 0\le j<k\le i}} D_{j,k}$$

where $D_{0,1,2,\ldots,i} = (a_0 + a_1 + \ldots + a_i)(b_0 + b_1 + \ldots + b_i)$

(2) For $1 < i \le n-1$ and $i \in$ even number , then

$$\begin{aligned} &D_{0,i} + D_{1,i-1} + \ldots + D_{i/2-1,i/2+1} \\ &= D_{0,1,\ldots,i/2-1,i/2+1,\ldots,i} - \sum_{\substack{j+k\ne i \\ 0\le j<k\le i}} D_{j,k} - \sum_{s=0}^{i} D_s \end{aligned}$$

where

$$\begin{aligned} &D_{0,1,\ldots,i/2-1,i/2+1,\ldots,i} \\ &= (a_0 + a_1 + \ldots + a_{i/2-1} + a_{i/2+1} + \ldots + a_i) \\ &\times (b_0 + b_1 + \ldots + b_{i/2-1} + b_{i/2+1} + \ldots + b_i) \end{aligned}$$

(3) For $n-1 < i \le 2n-1$ and $i \in$ odd number , then

$$\begin{aligned} &D_{i-(n-1),n-1} + D_{i-(n-2),n-2} + \ldots + D_{(i-1)/2,(i+1)/2} \\ &= D_{i-(n-1),i-(n-2),\ldots,(i-1)/2,(i+1)/2,\ldots,n-1} - \sum_{\substack{j+k\ne i \\ i-(n-1)\le j<k\le n-1}} D_{j,k} \end{aligned}$$

where

$$\begin{aligned} &D_{i-(n-1),i-(n-2),\ldots,(i-1)/2,(i+1)/2,\ldots,n-1} \\ &= (a_{i-(n-1)} + \ldots + a_{n-1})(b_{i-(n-1)} + \ldots + b_{n-1}) \end{aligned}$$

(4) For $n-1 < i \le 2n-1$ and $i \in$ even number , then

$$D_{i-(n-1),n-1} + D_{i-(n-2),n-2} + \ldots + D_{i/2-1,i/2+1}$$

$$= D_{i-(n-1),\ldots,n-1} - \sum_{\substack{j+k\neq i \\ i-(n-1)\le j<k\le i/2+1}} D_{j,k} - \sum_{s=i-(n-1)}^{n-1} D_s$$

where

$$D_{i-(n-1),i-(n-2),\ldots,i/2-1,i/2+1}$$

$$= (a_{i-(n-1)} + \ldots + a_{n-1})(b_{i-(n-1)} + \ldots + b_{n-1}).$$

*2. The Generalizations of The Combining Products Formula in GF($2^n$)*

The results of the four equations above we can make it in general form as follows.

if $i \in$ odd number , then

$$\sum_{\substack{k-j=1 \\ 0\le j<k\le i}} D_{p_j,p_k} = D_{p_{(0)},p_{(1)},p_{(2)},p_{(3)},\ldots,p_{(i-1)},p_{(i)}} - \sum_{\substack{k-j\neq 1 \\ 0\le j<k\le i}} D_{p_j,p_k},$$

where $p_{(0)} < p_{(1)} < \ldots < p_{(i)}; p_j < p_k,$

and $p_j, p_k \in \mathbb{N}^+ \ \forall\ 0 \le j < k \le i,$ (8)

and if $i \in$ even number , then

$$\sum_{\substack{k-j=1 \\ 0\le j<k\le i}} D_{p_j,p_k} = D_{p_{(0)},p_{(1)},p_{(2)},p_{(3)},\ldots,p_{(i-1)},p_{(i)}} - \sum_{\substack{k-j\neq 1 \\ 0\le j<k\le i}} D_{p_j,p_k} - \sum_{s=0}^{i} D_{p(s)}, \quad (10)$$

(9)

where $p_{(0)} < p_{(1)} < \ldots < p_{(i)}; p_j < p_k,$

and $p_j, p_k \in \mathbb{N}^+ \ \forall\ 0 \le j < k \le i.$

We obtain the Improved GKA formula by substituting equation (8) or (9) into equation (5). From this formula, we find that at least two products can be combined into a new product, meaning that two products disappear and one new product appears. Therefore, this formula reduces the number of multiplications by at least one. However, another algorithm that can combine existing products in equation C is still needed to further reduce the number of multiplications.

*3. Combining Products in The Equation C'(n-2), C'(n-1), and C'(n)*

To easily identify several products into a solution or not a solution, then it requires an algorithm combining products in the equation $C'(n\text{-}2)$, $C'(n\text{-}1)$, and $C'(n)$. This is because all three equations have the most abundant products when compared to other equations. Therefore, it is easy to be completed first and give a lot of information for the purpose of the identification process. Here are the steps.

*3.1. The Process of Combining Products of The Equation C'*

For $i = n\text{-}2$, $n = 5$, then

$$c'(i) = D(0,\ldots,n\text{-}1) + D(p_1(1), p_1(2), p_1(3), p_1(4)) + D(p_2(1), p_2(2), p_2(3), p_2(4)) + \left( \sum_{\substack{j+k\neq i \\ 0\le j<k\le n-1}} D(j,k) + \sum_{1\le j<k\le 4} D(p_1(j), p_1(k)) + \sum_{1\le j<k\le 4} D(p_2(j), p_2(k)) + D(n-1) \right) \quad (10)$$

Where

| | (1) | (2) | (3) | (4) |
|---|---|---|---|---|
| $P_1$ | $(i\text{-}1)/2\text{-}1$ | $(i+1)/2$ | $(i+1)/2+1$ | $(i+1)/2+2$ |
| $P_2$ | | $(i\text{-}1)/2$ | | |

For $i = n\text{-}1$, $n = 5$, then

$$c'(i) = D(0,\ldots,n\text{-}1) + D(p_1(1), p_1(2), p_1(3), p_1(4)) + D(p_2(1), p_2(2), p_2(3), p_2(4)) + \left( \sum_{\substack{j+k\neq i \\ 0\le j<k\le n-1}} D(j,k) + \sum_{1\le j<k\le 4} D(p_1(j), p_1(k)) + \sum_{1\le j<k\le 4} D(p_2(j), p_2(k)) \right) \quad (11)$$

Where

| | (1) | (2) | (3) | (4) |
|---|---|---|---|---|
| $P_1$ | $i/2\text{-}2$ | $i/2$ | $i/2+1$ | $i/2+2$ |
| $P_2$ | | $i/2\text{-}1$ | $i/2$ | |

For $i = n$, $n = 5$, then

$$c'(i) = D(0,\ldots,n\text{-}1) + D(p_1(1), p_1(2), p_1(3), p_1(4)) + D(p_2(1), p_2(2), p_2(3), p_2(4)) + \left( \sum_{\substack{j+k\neq i \\ 0\le j<k\le n-1}} D(j,k) + \sum_{1\le j<k\le 4} D(p_1(j), p_1(k)) + \sum_{1\le j<k\le 4} D(p_2(j), p_2(k)) + D(0) \right) \quad (12)$$

Where

| | (1) | (2) | (3) | (4) |
|---|---|---|---|---|
| $P_1$ | $(i\text{-}1)/2\text{-}2$ | $(i\text{-}1)/2\text{-}1$ | $(i\text{-}1)/2$ | $(i+1)/2+1$ |
| $P_2$ | | | $(i+1)/2$ | |

For $i = n\text{-}2$, $n > 5$ and $n$ is an odd number, then

$$c'(i) = D(0,\ldots,n\text{-}1) + D(p_1(1), p_1(2), p_1(3), p_1(4), p_1(5)) + D(p_2(1), p_2(2), p_2(3), p_2(4)) + \left( \sum_{\substack{j+k\neq i \\ 0\le j<k\le n-1}} D(j,k) + \sum_{1\le j<k\le 5} D(p_1(j), p_1(k)) + \sum_{s=1}^{5} D(p_1(s)) + \sum_{1\le j<k\le 4} D(p_2(j), p_2(k)) + D(n-1) \right)$$

Where

| | (1) | (2) | (3) | (4) | (5) |
|---|---|---|---|---|---|
| $P_1$ | ($i$-1)/2-2 | ($i$-1)/2-1 | ($i$+1)/2 | ($i$+1)/2+1 | ($i$+1)/2+2 |
| $P_2$ | ($i$-1)/2-1 | ($i$-1)/2 | ($i$+1)/2+1 | ($i$+1)/2+2 | - |

For $i = n$-1, $n > 5$ and $n$ is an odd number, then

$$c'(i) = D(0,\ldots,n\text{-}1) + D(p_1(1), p_1(2),\ldots, p_1(5)) + D(p_2(1), p_2(2),\ldots, p_2(5)) + \left( \sum_{\substack{j+k\neq i \\ 0\le j<k\le n-1}} D(j,k) + \sum_{1\le j<k\le 5} D(p_1(j), p_1(k)) + \sum_{s=1}^{5} D(p_1(s)) + \sum_{1\le j<k\le 5} D(p_2(j), p_2(k)) + \sum_{s=1}^{5} D(p_2(s)) \right)$$

Where

| | (1) | (2) | (3) | (4) | (5) |
|---|---|---|---|---|---|
| $P_1$ | $i$/2-3 | $i$/2-2 | $i$/2 | $i$/2+1 | $i$/2+2 |
| $P_2$ | $i$/2-2 | $i$/2-1 | $i$/2 | $i$/2+2 | $i$ |

For $i = n$, $n > 5$ and $n$ is an odd number, then

$$c'(i) = D(0,\ldots,n\text{-}1) + D(p_1(1), p_1(2), p_1(3), p_1(4), p_1(5)) + D(p_2(1), p_2(2), p_2(3), p_2(4)) + \left( \sum_{\substack{j+k\neq i \\ 0\le j<k\le n-1}} D(j,k) + \sum_{1\le j<k\le 5} D(p_1(j), p_1(k)) + \sum_{s=1}^{5} D(p_1(s)) + \sum_{1\le j<k\le 4} D(p_2(j), p_2(k)) + D(0) \right)$$

Where

| | (1) | (2) | (3) | (4) | (5) |
|---|---|---|---|---|---|
| $P_1$ | ($i$-1)/2-2 | ($i$-1)/2-1 | ($i$-1)/2 | ($i$+1)/2+1 | i-1 |
| $P_2$ | | | ($i$+1)/2 | | - |

*3.2. Completion of all other products*

Then, do the process of combining products for two coefficients (products of the equation in parentheses), use the combining formula (8) or (9) until the number of multiplications can not be further reduced.

For even values of $n$ the same steps can be used to make the formula imitate the pattern in equations (10), (11), and (12).

Then, to solve equation $c'(i)$ for $i$ other than $n$-2, $n$-1 and $n$, the following search algorithm is used.

B. *Exhaustive Search Algorithm for Polynomial Multiplier in GF($2^n$)*

This algorithm was developed to supplement the drawbacks of the improved GKA formula, namely its inability to compute polynomial multiplications in $GF(2^n)$ for large values of $n$. The exhaustive search algorithm is intended to find solutions to equation $C'$(i) for $i$ other than $n$-2, $n$-1, and $n$.

The following are the stages of development of the exhaustive search algorithm:

*1. Creating The Matrix C' with the GKA Formula*

The first step is to create matrix $C'$, in which each row is $c'(i)$, $i = 0, 1, ..., 2n$-2
and the columns are

$$a_0b_0, a_0b_1, ..., a_0b_{n-1},$$
$$a_1b_0, a_1b_1, ..., a_1b_{n-1},$$
...
$$a_{n-1}b_0, a_{n-1}b_1, ..., a_{n-1}b_{n-1},$$

Then, the matrix is filled with a value of 1 (one) in any positions in the matrix that corresponds to the value of $c'(i)$.

For example, for $n = 3$, then

$$c'(0) = a_0b_0$$
$$c'(1) = a_0b_1 + a_1b_0$$
$$c'(2) = a_0b_2 + a_1b_1 + a_2b_0$$
$$c'(3) = a_1b_2 + a_2b_1$$
$$c'(4) = a_2b_2$$

The matrix $C'$ is shown in Table I.

TABLE I
THE MATRIX $C'$ FOR N = 3

| | $a_0b_0$ | $a_0b_1$ | $a_0b_2$ | $a_1b_0$ | $a_1b_1$ | $a_1b_2$ | $a_2b_0$ | $a_2b_1$ | $a_2b_2$ |
|---|---|---|---|---|---|---|---|---|---|
| $c'_0$ | 1 | | | | | | | | |
| $c'_1$ | | 1 | | 1 | | | | | |
| $c'_2$ | | | 1 | | 1 | | 1 | | |
| $c'_3$ | | | | | | 1 | | 1 | |
| $c'_4$ | | | | | | | | | 1 |

*2. Creating a matrix D with The Combining Products Formula for all Combinations*

A matrix $D$ is then created using the improved GKA formula, where the rows contain all the possible combinations of existing indexes and the columns are identical to those of matrix $C'$.

The number of possible combinations for a given value of $n$ is $\Sigma(n!/((n-i)!i!)$ or $2^n$-1.

For values of $n$, then there will be an $n$ index ranging from 0 to $n$-1. Thus, the combinations are:

0, …, $n$-1 (one index),
01, …, 0($n$-1) (two indexes),
…
0123…($n$-1) ($n$ indexes).

For $n = 3$, the combination is represented in binary code as in Table II:

TABLE II
REPRESENTING OF THE COMBINATION IN BINARY CODE FOR N = 3

| row | 0 | 1 | 2 | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | | - |
| 1 | 0 | 0 | 1 | = | D(2) |
| 2 | 0 | 1 | 0 | = | D(1) |
| 3 | 0 | 1 | 1 | = | D(1,2) |
| 4 | 1 | 0 | 0 | = | D(0) |
| 5 | 1 | 0 | 1 | = | D(0,2) |
| 6 | 1 | 1 | 0 | = | D(0,1) |
| 7 | 1 | 1 | 1 | = | D(0,1,2) |

Matrix $D$ for $n = 3$ will then have the following form (Table III):

TABLE III
THE MATRIX D FOR N = 3

| | $a_0b_0$ | $a_0b_1$ | $a_0b_2$ | $a_1b_0$ | $a_1b_1$ | $a_1b_2$ | $a_2b_0$ | $a_2b_1$ | $a_2b_2$ |
|---|---|---|---|---|---|---|---|---|---|
| 001 | | | | | | | | | 1 |
| 010 | | | | | 1 | | | | |
| 011 | | | | | 1 | 1 | | 1 | 1 |
| 100 | 1 | | | | | | | | |
| 101 | 1 | | 1 | | | | 1 | | 1 |
| 110 | 1 | 1 | | 1 | 1 | | | | |
| 111 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

*3. Combining all rows in matrix D*

To combine all rows of the matrix $D$, then the process is the same as the combined index for the matrix $D$. Using the tables III, then we can use the column line to combine it. The number of lines is $2^{(2^n-1)}-1$.

*4. Summing up for each combination of rows in matrix D*

For each a combination result of the above, then each index on the combination represented as in table II and III. Thereafter, for each the combination is summed. Then any combination thereof will be represented as a table III, and can be searched from any such combination that satisfies the equation $C'$.

*5. Searching for a solution to equation C'*

The sums of each of the results of the above combinations were checked whether they satisfy equation $C'$. For example,

$$c'(2) = a_0b_2 + a_1b_1 + a_2b_0$$

will be equal to an XOR operation for rows 7/111/(012), 6/110/(01), and 3/011/(12), resulting in the following table:

| | $a_0b_0$ | $a_0b_1$ | $a_0b_2$ | $a_1b_0$ | $a_1b_1$ | $a_1b_2$ | $a_2b_0$ | $a_2b_1$ | $a_2b_2$ |
|---|---|---|---|---|---|---|---|---|---|
| 110 | 1 | 1 | | 1 | 1 | | | | |
| 011 | | | | | 1 | 1 | | 1 | 1 |
| 111 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| c'(2) = | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |

This can also be represented as the following:

$$c'(2) = a_0b_2 + a_1b_1 + a_2b_0 = D(0,2) + D(0) + D(1) + D(2)$$

This exhaustive search algorithm succeeded in solving equation C'($i$) for $i$ other than $n$-2, $n$-1, and $n$, by searching for combinations of products that satisfy the equations. However, for values of $n$ of more than 4, the algorithm fails to find the solutions in a reasonable amount of time and computer memory. The problem that arises is, then, to search for any combination set that satisfies equation $C'$ that provides the least number of multiplications.

*C. Nursalman, Arif, Yusuf and Kuspriyanto Algorithm (NAYK Algorithm)*

This algorithm combines the Improved GKA algorithm and the Exhaustive Search Algorithm in order to overcome the flaws of both algorithms. The algorithm includes the following steps:

1. Identify the products that are significantly involved in solutions for $C'$.
2. Identify the products are not involved in solutions for $C'$.
3. Reduce the number of rows of the combinations in matrix $D$.
4. Utilize the properties that arise from polynomial multiplication in $GF(2^n)$ such as symmetry.

By significantly reducing the number of combinations in matrix $D$, the algorithm is much simpler and can be processed much easier.

As an example, the following are details of the steps for $n$=5.

*1. Determine the value of n, matrix C', and matrix D*

Let $n = 5$

The Exhaustive Search Algorithm is used to calculate matrix $C'$ and $D$. The matrix $D$ consists of 31 rows.

*2. Find a solution for c'(n-2), c'(n-1) and c'(n)*

To find the solutions to equation $C'(i)$ for $i$ values of $n$-2, $n$-1, and $n$, the Improved GKA formula was used by substituting equation (8) or (9) into (5). For $n = 5$ look at (10), (11) and (12). The following formulas are then obtained:

$$\begin{aligned} c'(3) &= D(0,1,2,3,4) + D(0,2,3,4) + D(0,1,3,4) \\ &\quad + D(3,4) + D(0,4) + D(4) \\ c'(4) &= D(0,1,2,3,4) + D(0,2,3,4) + D(0,1,2,4) \\ &\quad + D(0,2) + D(2,4) \\ c'(5) &= D(0,1,2,3,4) + D(0,1,2,4) + D(0,1,3,4) \\ &\quad + D(0,1) + D(0,4) + D(0) \end{aligned} \tag{13}$$

*3. Identification of products that are solutions to equation C'*

Note that in order to identify the products involved in the equation in $C'$ is seeing the results of all three equations in the middle of the above (13), then we will get the products as follows.

$D(0,1,2,3,4)$, $D(0,2,3,4)$, $D(0,1,2,4)$, $D(0,1,3,4)$, $D(0,1)$, $D(3,4)$, $D(0,4)$, $D(1,3)$, $D(0)$, and $D(4)$.

Note, the form of $D(0,1)$ in the equation $c'(1)$ and $D(3,4)$ in equation $c'(7)$ the multiplication are already difficult to be reduced again, the formula is in (5) with $n = 5$. therefore, both the equations have a fixed form. So consequently the equation of $c'(0)$ and $c'(8)$ have a fixed form too. Then the products included in the solution in $C'$ will increase, those are

$D(1)$ and $D(3)$.

Thus, the number had grown to 11. The upper bound of the function $O(5)$ of Karatsuba multiplier is 12.8 or rounded to 13. This means that we have at most two products to be added to the solution. If more than that, it means that the

algorithm is not better than the Montgomery algorithm. for $D(0,2)$ and $D(2,4)$ are not added in the solution because it would involve $D(2)$ as well, so the number of the solution is greater than 13.

*4. Identify the products that are not solutions to equation C′*

In order to significantly reduce the number of rows in matrix $D$ the products that are not solutions to equation $C'$ must be identified. These products will not appear again in the new $C'$ equation.

Note that the new equations of $c'(3)$ and $c'(5)$ have a fixed form, in which the number of multiplications cannot be further reduced. Therefore, the products that do not appear in these new equations are not included in the solution. These products are

$D(0,2)$, $D(0,3)$, $D(1,2)$, $D(1,4)$, $D(2,3)$, and $D(2,4)$.

In addition, note that equation $c'(4)$ contains the $D(1,3)$ product, which does not appear in the new equations of $c'(3)$, $c'(4)$, and $c'(5)$. Thus, this product is also not included in the solution to $C'$, increasing the number of products that are not included to 7.

*5. Create a new matrix D*

The rows of matrix $D$ are reduced with both products that are included in the solution and those that are not.

There are 31 rows in matrix $D$, meaning that the number of rows of the new matrix $D$ is 31-11-7 = 13 (Table IV).

*6. Choose the combination of products for the solution to C′.*

To find the solution of the equations of $c'(2)$ and $c'(6)$, we only choose two products from 13 products outside products that have been identified are included and not included in the solution. In other words, we are looking for a combination of two products from 13 products, then for each combination were combined with 11 products, which are included in the solution. Then each combination will consist of 13 products. Therefore, the search space of the solution becomes 78, which is much smaller compared to the algorithm developed by Montgomery, in which the search space is $2.1 \cdot 10^8$.

TABLE IV
THE NEW MATRIX D FOR N = 5

| No | Decimal | Binary | The New Matrix D |
|---|---|---|---|
| 1 | 4 | 00100 | 2 |
| 2 | 7 | 00111 | 234 |
| 3 | 11 | 01011 | 134 |
| 4 | 13 | 01101 | 124 |
| 5 | 14 | 01110 | 123 |
| 6 | 15 | 01111 | 1234 |
| 7 | 19 | 10011 | 034 |
| 8 | 21 | 10101 | 024 |
| 9 | 22 | 10110 | 023 |
| 10 | 25 | 11001 | 014 |
| 11 | 26 | 11010 | 013 |
| 12 | 28 | 11100 | 012 |
| 13 | 30 | 11110 | 0123 |

*7. Calculate the combination of selected products for the solution in C′*

Each of the search space is calculated to find combinations that are included in the solution to equation $C'$. So we get on the line 38th is the combination solution, that is [13 22] or [01101 10110] or [124 023]. then the equation of $c'(2)$ and $c'(6)$ will have the following form.

$$c'(2) = D(0,1,2,4) + D(1,2,4) + D(0,1) + D(0,4) + D(4) + D(0)$$

$$c'(6) = D(0,2,3,4) + D(0,2,3) + D(3,4) + D(0,4) + D(0) + D(4)$$

Note that there are five different indexes of these solutions (0, 1, 2, 3, and 4), and that 2 is the middle value. Due to the symmetrical property of polynomial multiplication in $GF(2^n)$, if we reflect 0 and 1 to 2, then the results of the reflection are 4 and 3, respectively, and vice versa. If $D(0,1,2,4)$ is used in the solutions in $c'(2)$, then $D(0,2,3,4)$ can also be used in the solutions in $c'(6)$, and this can also be done for all other products. This means that we can easily obtain a solution to $c'(6)$ by obtaining the solution to $c'(2)$. So the results of $c'(6)$ is the result of reflection from $c'(2)$ on a symmetrical lines in $C'$, that is $c'(4)$. This means that to obtain a solution to equation $C'$, we simply need to find a solution to $c'(2)$ up to $c'(n\text{-}1)$, and by reflecting $c'(2)$ up to $c'(n\text{-}2)$, the solutions for $c'(n)$ up to $c'(2n\text{-}2)$ can be found. Other solutions to $c'(0)$ and $c'(1)$ need not be found due to their fixed form.

For $n$ = 5 or above, not all of the products of the new matrix $D$ in table IV need to be used. Note that if $D(2)$, $D(1,2,3)$ and $D(0,2,4)$ are reflected, then the results are identical, or symmetrical. Therefore, for 10 products, half is a mirror of the other half. This means that the search space can be decreased to 3 + 5 = 8 rows in the new matrix $D$. Then we simply seek 1 from 8 products to be combined with the 11 products of the solution that have been identified previously to find solutions to $c'(2)$ or $c'(6)$, further reducing the search space to 8 products. This is significantly lower than that found by Montgomery.

*8. Get the solution with $O(n) \le n^{\log(3,2)}$, where $O(n)$ rounded off by default.*

If not, then choose another combination of products for a solution $C'$, or repeat the steps above. Then, we get the solution for the equation of $C'$, for $n$ = 5, then the form is as follows (14).

$$
\begin{aligned}
c'(0) &= D(0)\\
c'(1) &= D(0,1) + D(0) + D(1)\\
c'(2) &= D(0,1,2,4) + D(1,2,4)\\
&\quad + D(0,1) + D(0,4) + D(4) + D(0)\\
c'(3) &= D(0,1,2,3,4) + D(0,2,3,4)\\
&\quad + D(0,1,3,4) + D(3,4) + D(0,4) + D(4)\\
c'(4) &= D(0,1,2,3,4) + D(0,2,3) + D(1,2,4)\\
&\quad + D(3,4) + D(0,1) + D(0) + D(1) + D(3) + D(4)\\
c'(5) &= D(0,1,2,3,4) + D(0,1,2,4)\\
&\quad + D(0,1,3,4) + D(0,1) + D(0,4) + D(0)\\
c'(6) &= D(0,2,3,4) + D(0,2,3)\\
&\quad + D(3,4) + D(0,4) + D(4) + D(0)\\
c'(7) &= D(3,4) + D(3) + D(4)\\
c'(8) &= D(4)
\end{aligned}
\qquad (14)
$$

From the above it can be observed that there are 13 multiplication operations, resulting in the same complexity as the Montgomery algorithm, and the upper bound Karatsuba algorithm $O(5) = 12.8$ with rounded. This means that solutions are acceptable. The question now is, whether 13 is the smallest bound to the number of multiplication operations in polynomial multiplication in $GF(2^5)$

*9. 13 is the lower bound for n = 5*

We saw earlier in the identification process, that the products are involved as many as 11, which still has not formed a full solution because it still requires two other products, This means that 11 is not a lower bound, and based on the results of the above that upper bound of the latest is 13 multiplication.

The question is, is it possible that 12 is the new lower bound? If 12 is the new lower bound, then the number of products of the longest being the solution for any $c'(i)$ is 10. Then the rest of the equation $c'(j)$ are 4, it means that one products is loaded by two equations of $c'(j)$, this is the minimum amount required to complete the two equations. So that the rest of the two products is loaded by 4 equations of $c'(j)$. Consider again that the products involved to be the solution. Note that the form of $c'(3)$ and $c'(5)$ are fixed, and the remaining are $c'(2)$ and $c'(6)$, which later those results will be substituted into $c'(4)$.

The products are involved as many as 11, it means staying one more, then the remaining one will be published by the two equations $c'(2)$ and $c'(6)$, where the equation of $c'(2)$ is the mirror of the equation of $c'(6)$. Then the combinations of $D$ is $C(14,1) = 14$, which is the matrix $D$ itself. Products of the matrix D which is a mirror for himself are 2, 13, 123, and 024, then one by one the product is combined with the 11 previous products, it was found that these products no one becomes a solution for the $c'(2)$ and $c'(6)$.

So based on the method developed above, we can conclude that the amount of 12 products is not a solution for $n = 5$, it means that the 13 products is an upper bound, but also a lower bound for the solution $n = 5$.

Furthermore, we can repeat the same steps above to get a solution for $n = 6$ and 7. Here are the form of multipliers for $n$=6 and 7, which differs from Montgomery. For $n = 6$, we get two different forms than the forms found by Montgomery but still have the same complexity, 17 multiplications. Here is the first form (15).

$$
\begin{aligned}
c'(0) &= D(0)\\
c'(1) &= D(0,1) + D(0) + D(1)\\
c'(2) &= D(0,1,2) + D(0,1) + D(1,2)\\
c'(3) &= D(0,1,2,3,4,5) + D(1,2,4,5) + D(0,1,3,4)\\
&\quad + D(0,1,2) + D(0,3,5) + D(0,1) + D(1,4)\\
&\quad + D(2,3) + D(3,4) + D(0) + D(5)\\
c'(4) &= D(0,1,2,3,4,5) + D(1,2,4,5) + D(0,1,2)\\
&\quad + D(0,3,5) + D(1,2) + D(2,3) + D(0)\\
&\quad + D(4) + D(5)\\
c'(5) &= D(0,1,2,3,4,5) + D(1,2,4,5) + D(0,1,3,4)\\
&\quad + D(0,1,2) + D(3,4,5) + D(0,1) + D(1,2)\\
&\quad + D(3,4) + D(4,5)\\
c'(6) &= D(0,1,2,3,4,5) + D(0,1,3,4) + D(0,2,5)\\
&\quad + D(3,4,5) + D(2,3) + D(3,4) + D(0)\\
&\quad + D(1) + D(5)\\
c'(7) &= D(0,1,2,3,4,5) + D(0,1,3,4) + D(1,2,4,5)\\
&\quad + D(0,2,5) + D(3,4,5) + D(1,2) + D(1,4)\\
&\quad + D(4,5) + D(2,3) + D(0) + D(5)\\
c'(8) &= D(3,4,5) + D(3,4) + D(4,5)\\
c'(9) &= D(4,5) + D(4) + D(5)\\
c'(10) &= D(5)
\end{aligned}
\qquad .....(15)
$$

Then for the second form, replace the equation $c'(3)$ and $c'(7)$ in (15) with the following equations.

$$
\begin{aligned}
c'(3) &= D(0,2,3,5) + D(0,2,5) + D(3,4,5)\\
&\quad + D(2,3) + D(3,4) + D(4,5) + D(1) + D(4)
\end{aligned}
$$

$$
\begin{aligned}
c'(7) &= D(0,2,3,5) + D(0,1,2) + D(0,3,5)\\
&\quad + D(0,1) + D(1,2) + D(2,3) + D(1) + D(4)
\end{aligned}
$$

In this second form of the equation $D(1,4)$ does not exist but appears equation $D(0,2,3,5)$. The complexity of the number of multiplications is the same but the number of add operations increased two products, which means that the first form solution is better than the second.

Then for $n = 7$ with NAYK algorithms can we write as follows (16).

$$
\begin{aligned}
c'(0) &= D(0)\\
c'(1) &= D(0,1) + D(0) + D(1)\\
c'(2) &= D(0,2) + D(0) + D(1) + D(2)\\
c'(3) &= D(1,2,3,5,6) + D(0,2,3,5,6)\\
&\quad + D(0,1,5,6) + D(0,1) + D(0,2)\\
&\quad + D(1,3) + D(5,6) + D(2) + D(3)\\
c'(4) &= D(0,4) + D(1,3) + D(0) + D(1)\\
&\quad + D(2) + D(3) + D(4)\\
c'(5) &= D(0,1,2,3,4,5,6) + D(0,1,3,4,6)\\
&\quad + D(1,2,4,5) + D(0,2) + D(2,6)\\
&\quad + D(3,5) + D(5,6) + D(0) + D(1)\\
&\quad + D(3) + D(4)\\
c'(6) &= D(0,1,2,3,4,5,6) + D(1,2,3,5,6)\\
&\quad + D(0,1,3,4,5) + D(0,2) + D(1,3)\\
&\quad + D(3,5) + D(4,6) + D(0) + D(2)\\
&\quad + D(4) + D(6)\\
c'(7) &= D(0,1,2,3,4,5,6) + D(0,2,3,5,6)\\
&\quad + D(1,2,4,5) + D(0,1) + D(0,4)\\
&\quad + D(1,3) + D(4,6) + D(2) + D(3)\\
&\quad + D(5) + D(6)\\
c'(8) &= D(2,6) + D(3,5) + D(2) + D(3)\\
&\quad + D(4) + D(5) + D(6)\\
c'(9) &= D(0,1,3,4,5) + D(0,1,3,4,6)\\
&\quad + D(0,1,5,6) + D(0,1) + D(3,5)\\
&\quad + D(4,6) + D(5,6) + D(3) + D(4)\\
c'(10) &= D(4,6) + D(4) + D(5) + D(6)\\
c'(11) &= D(5,6) + D(5) + D(6)\\
c'(12) &= D(6)
\end{aligned}
\qquad \text{.....(16)}
$$

The above solution difference with Montgomery, it lies in the equation $c'(3)$ and $c'(9)$. Both of these equations written by Montgomery to be as follows.

$$
\begin{aligned}
c'(3) &= D(0,1,3,4,5) + D(0,2,3,4,6) + D(0,2,3,5,6)\\
&\quad + D(1,2,4,5) + D(0,1) + D(1,3) + D(4,6)\\
&\quad + D(5,6) + D(2)
\end{aligned}
$$

and

$$
\begin{aligned}
c'(9) &= D(1,2,3,5,6) + D(0,2,3,4,6) + D(0,1,3,4,6)\\
&\quad + D(1,2,4,5) + D(0,1) + D(0,2) + D(3,5)\\
&\quad + D(5,6) + D(4)
\end{aligned}
$$

In the solution of the equation $C'$ with NAYK algorithm there is products of $D(0,1,5,6)$ and there is no product of $D(0,2,3,4,6)$. It is clear that the number of operations of addition in $D(0,1,5,6)$ is less than the $D(0,2,3,4,6)$, besides Montgomery write the equation $c'(3)$ and $c'(9)$ more long. This means that the solutions of the equation $C'$ with NAYK algorithm is better than Montgomery.

Note that if the index $i<n-1$ is reflected to its midpoint, $n-1$, then the result is $n-1+i$. Therefore, the results of reflection or partner of $D(i_1,i_2,...,i_m)$ is

$$D(n-1+i_{(1)}, n-1+i_{(2)}, ..., n-1+i_{(m)}),$$

and for $i>n-1$, then the result of the reflection is $n-1-i$, so that pairs of $D(i_1,i_2,...,i_m)$ is

$$D(n-1-i_{(1)}, n-1-i_{(2)}, ..., n-1-i_{(m)}).$$

While reflecting n-1 is himself. This is the reflection properties that appear in the equation in $C'$.

Consider the results of the search for a solution of the equation $C'$ for $n = 5$, we get that the search space can be reduced again from 78 to 8 by utilizing the properties of reflection that appears on the solution in $C'$, this is better than Montgomery did, where the search space is $2.1\cdot10^8$, although Montgomery then automatically selects the three following multiplication $a_0b_0$, $a_{n-1}b_{n-1}$, and $(a_0b_0+...+a_{n-1}b_{n-1})$, so the search space becomes $1.3\cdot10^7$. Utilization of the reflection properties as above to determine the search space, from 78 to 8, very risky if not described in more detail as was done by NAYK algorithm to search for a solution for $n = 5$. Because it could be in one equation $c'(i)$ there are two or more variations of products in pairs as part of the solution. So if various products in pairs are separated into two different groups, and we only took one of them to complete a $c'(i)$, then $c'(i)$ will not have a solution. For example, consider the equation $c'(3)$ for $n = 6$ and $c'(6)$ for $n = 7$ follows below,

$$
\begin{aligned}
c'(3) &= D(0,1,2,3,4,5) + D(1,2,4,5) + D(0,1,3,4)\\
&\quad + D(0,1,2) + D(0,3,5) + D(0,1) + D(1,4)\\
&\quad + D(2,3) + D(3,4) + D(0) + D(5)
\end{aligned}
$$

and

$$
\begin{aligned}
c'(6) &= D(0,1,2,3,4,5,6) + D(1,2,3,5,6)\\
&\quad + D(0,1,3,4,5) + D(0,2) + D(1,3)\\
&\quad + D(3,5) + D(4,6) + D(0) + D(2)\\
&\quad + D(4) + D(6)
\end{aligned}
$$

For $n = 6$, products of $D(1,2,4,5)$ is a pair of $D(0,1,3,4)$, and for $n = 7$, $D(1,2,3,5,6)$ is a pair of $D(0,1,3,4,5)$. If they are separated into different groups and if we only look for solutions to $c'(3)$ or $c'(6)$ with one of these groups, then both of these equations will not have a solution, either with an algorithm Montgomery nor with NAYK algorithm. Therefore, it is very risky if separated the two groups are mutually coupled to resolve $C'$. Unless already identified some significant products which became a solution and which would not be a solution and with the separation there is a solution for every $c'(i)$ sought as an example for $n = 5$ with NAYK algorithm above.

Here are some products that are identified as the solution and not the solution for $n = 6$.

Some products for the solution:

$D(01,2,3,4,5)$, $D(0,1,3,4)$, $D(1,2,4,5)$, $D(0,1,2)$, $D(0,3,5)$, $D(3,4,5)$, $D(0,2,5)$, $D(0,1)$, $D(1,2)$, $D(1,4)$, $D(2,3)$, $D(3,4)$, $D(4,5)$, $D(0)$, $D(1)$, $D(4)$, and $D(5)$

Some products are not the solution:

$D(0,5)$, $D(0,2)$, $D(3,5)$, $D(2,5)$, $D(0,3)$, $D(2,4)$, $D(1,3)$, $D(1,5)$, $D(0,4)$, $D(2)$, and $D(3)$.

From 63 existing products, then reduced and the rest became 63-16-11 = 36. Because the function $O(6)$ gives the upper bound is 17, so we are now looking for 17-16 = 1 from 36, then the product is combined with the 16 products to be the solution for $C'$. This means there are as many as 36 possible search space for $n = 6$. But then consider, that only one product we were looking for which is contained by the two equations $c'(3)$ and $c'(7)$, this means that the products are symmetrical. Then we find the products that are symmetrical are not included in the two groups above, these products are $D(1,4)$, $D(1,2,3,4)$, $D(0,2,3,5)$ and $D(0,1,4,5)$. This means that we are not looking for one from 36 products, but we are looking for 1 from 4 symmetrical products. So the search space is reduced from 36 to 4.

Here are some products that are identified as the solution and not the solution for $n = 7$.

Some products for the solution:

$D(0,1,2,3,4,5,6)$, $D(0,1,3,4,5)$, $D(1,2,3,5,6)$, $D(0,1,3,4,6)$, $D(0,2,3,5,6)$, $D(1,2,4,5)$, $D(0,1)$, $D(0,2)$, $D(0,4)$, $D(1,3)$, $D(2,6)$, $D(3,5)$, $D(4,6)$, $D(5,6)$, $D(0)$, $D(1)$, $D(2)$, $D(3)$, $D(4)$, $D(5)$, and $D(6)$.

Some products are not the solution:

$D(0,3)$, $D(0,5)$, $D(0,6)$, $D(1,2)$, $D(1,4)$, $D(1,5)$, $D(1,6)$, $D(2,3)$, $D(2,4)$, $D(2,5)$, $D(3,4)$, $D(3,6)$, and $D(4,5)$.

From 127 products, was reduced to 127-21-13 = 93. But Consider that the upper bound is $O(7) = 22$, while some products that have been identified as a solution there are 21, which means that we are now looking for one from 93. But then consider, that only one product we were looking for which is contained by the two equations $c'(3)$ and $c'(9)$, this means that the products are symmetrical. Then we find the products that are symmetrical are not included in the two groups above, these products are

$D(2,3,4)$, $D(1,3,5)$, $D(0,3,6)$, $D(1,2,4,5)$, $D(0,2,4,6)$, $D(0,1,5,6)$, $D(1,2,3,4,5)$, $D(0,2,3,4,6)$, and $D(0,1,3,5,6)$.

This means that we are not looking for one from 93 products, but we are looking for one from 9 symmetrical products. So the search space is reduced from 93 to 9.

Here below, table V, is the comparison of the search space between Montgomery algorithm and NAYK algorithm to find the products for solution equation of $C'$.

TABLE V
THE COMPARISON OF THE SEARCH SPACE BETWEEN MONTGOMERY ALGORITHM AND NAYK ALGORITHM

| n | The Search Space | | |
|---|---|---|---|
| | Montgomery Algorithm [18] | NAYK Algorithm | |
| 5 | $1.3 \cdot 10^7$ | 78 | 8 |
| 6 | $1.7 \cdot 10^{13}$ | 36 | 4 |
| 7 | $1.1 \cdot 10^{22}$ | 93 | 9 |

In Table V, there are two values for the search space resulted from the NAYK algorithm. The right values are obtained by utilizing the properties of symmetry of polynomial multiplications in $GF(2^n)$, resulting in a much smaller search space. The NAYK algorithm can also provide a very small search space for values of $n$ of larger than 7, for which the Montgomery algorithm cannot. This is because the NAYK algorithm identifies which products are solutions and which are not, and utilizes these products for completing other equations of $c'(i)$, significantly reducing the search space. From these results, we can conclude that the NAYK algorithm is much simpler and less resource-consuming in finding a polynomial multiplier in $GF(2^n)$ than the algorithm developed by Montgomery.

To see how the NAYK algorithm can improve calculation efficiency, it was compared with two other methods, namely the School Book and the Simple Karatsuba methods for $n = 6$ and 7. We constructed a 169 bit multiplier in the composite field $GF((2^p)^q)$, where $p = q = 13$ bits, or $p \times q = 13 \times 13 = 169$ bits, while $p$ and $q$ themselves will be constructed from $n = 6$ and 7, that is $13 = 7 + 6$.

A comparison of the complexity function $O(n)$ for the three different methods is shown in Table VI and Figure 5. For the School Book method, $O(n) = n^2$, while for the Simple Karatsuba, $O(n) \leq 3n^2/4$. This is because by using a small number of bits, only one iteration is needed, so the efficiency is approximately $3n/4$ (upper limit). As for the NAYK method, the upper limit is the standard rounding of the value of $n^{\log(3,2)} \geq O(n)$.

The Karatsuba method for $p$ and $q$ was used for the first composite field, while for the second composite field, the Karatsuba method was used for $p$ and the NAYK method for $q$. To compare the algorithms, the implementation of the two methods were simulated in Quartus II:

TABLE VI
COMPARISON OF THE COMPLEXITY FUNCTION $O(N)$ FOR SMALL BITS

| n | O(n) for small bits | | |
|---|---|---|---|
| | School Book | Karatsuba | NAYK |
| 1 | 1 | 1 | 1 |
| 2 | 4 | 3 | 3 |
| 3 | 9 | 7 | 6 |
| 4 | 16 | 12 | 9 |
| 5 | 25 | 19 | 13 |
| 6 | 36 | 27 | 17 |
| 7 | 49 | 37 | 22 |

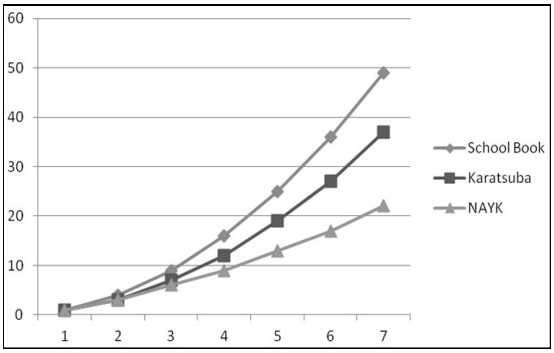


Fig. 5. Comparison of the Complexity Function $O(n)$ for Small Bits

Due to the low number of bits used (169 bits), only one iteration is needed for the Karatsuba algorithm. The following formula (17) was used:

$$\begin{aligned} AB &= \left[ x^{12}\left(A_H B_H\right) + x^6\left(\left(A_H + A_L\right)\left(B_H + B_L\right)\right.\right. \\ &\quad \left.\left. - A_H B_H - A_L B_L\right)\right] \bmod f(x) + A_L B_L \\ &= x^{12}\left(A_H B_H\right) \bmod f(x) \\ &\quad + x^6\left(\left(A_H + A_L\right)\left(B_H + B_L\right) - A_H B_H - A_L B_L\right) \bmod f(x) \\ &\quad + A_L B_L \end{aligned} \tag{17}$$

Each element is divided into part $H$ (high) and $L$ (low), where $H$ and $L$ each consists of 7 and 6 coefficients (7 + 6 = 13 bits). The modulo function used is as follows:

$$f(x) = x^{13} + x^4 + x^3 + x + 1 \tag{18}$$

From these functions, we obtain the following algorithm for the finite state machine:

$S1 : RM_0 \leftarrow A_H + A_L$

$S2 : RM_1 \leftarrow B_H + B_L$

$S3 : RM_0 \leftarrow RM_0 \times RM_1$

$S4 : RM_1 \leftarrow A_H \times B_H$

$S5 : RM_2 \leftarrow A_L \times B_L$

$S6 : RM_0 \leftarrow RM_0 + RM_1$

$S7 : RM_0 \leftarrow RM_0 + RM_2$

$S8 : RM_1 \leftarrow RM_1 \times x^6$

$S9 : RM_1 \leftarrow RM_1 \times x^6$

$S10 : RM_0 \leftarrow RM_0 \times x^6$

$S11 : RM_0 \leftarrow RM_0 + RM_1$

$S12 : Output \leftarrow RM_0 + RM_2$

From the algorithm, the following finite state machine is obtained (Figure 6):

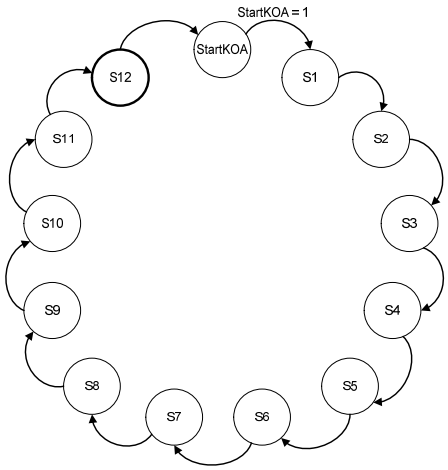


Fig. 6. Finite State Machine (FSM) for polynomial multiplier in $GF(2^{13})$

The implementation of equations (15) and (16) is illustrated in the following datapath circuit (Figure 7):

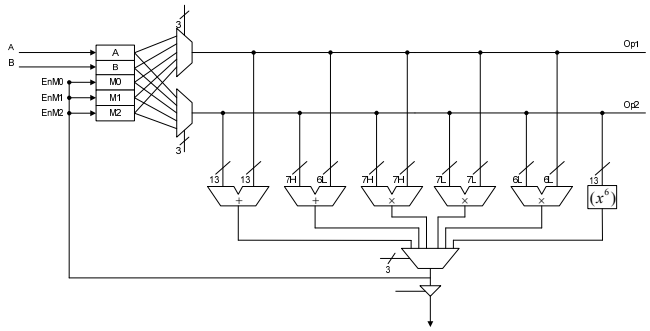


Fig. 7. Datapath for polynomial multiplier in $GF(2^{13})$

The combination of the datapath and finite state machine was simulated with VHDL in Quartus II. Below is a sample the FSM module that regulates how the system works:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY fsm IS PORT (
   Clock, Reset, Start: IN std_logic;
   OE, enM0, enM1, enM2: OUT std_logic;
   IE_op1, IE_op2, IE_op3: OUT std_logic_vector(2 downto 0));
END fsm;

ARCHITECTURE fsm_KOA13 OF fsm IS
   TYPE state_type IS (s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10,
s11, s12);
      signal next_state, current_state: state_type;

BEGIN
   process (Clock, Reset)
begin
   if Reset = '1' then
      current_state <= s0;
   elsif (rising_edge(Clock)) then
      current_state <= next_state;
   end if;
end process;

process (current_state)
begin
   next_state <= current_state;
   case current_state is
when s0 =>
        IF (Start = '1') THEN next_state <= s1;
        ELSE next_state <= s0;
        END IF;
      when s1 => next_state <= s2;
      when s2 => next_state <= s3;
      when s3 => next_state <= s4;
      when s4 => next_state <= s5;
      when s5 => next_state <= s6;
      when s6 => next_state <= s7;
      when s7 => next_state <= s8;
      when s8 => next_state <= s9;
      when s9 => next_state <= s10;
      when s10 => next_state <= s11;
      when others => next_state <= s12;
   end case;
end process;
```

```
PROCESS(current_state)
BEGIN
CASE current_state IS
   WHEN s1 => IE_op1<="000"; IE_op2 <= "000"; IE_op3 <= "001"; OE
<= '0'; enM0 <= '1'; enM1 <= '0'; enM2 <= '0';
   WHEN s2 => IE_op1<="001"; IE_op2 <= "001"; IE_op3 <= "001"; OE
<= '0'; enM0 <= '0'; enM1 <= '1'; enM2 <= '0';
   WHEN s3 => IE_op1<="010"; IE_op2 <= "011"; IE_op3 <= "011"; OE
<= '0'; enM0 <= '1'; enM1 <= '0'; enM2 <= '0';
   WHEN s4 => IE_op1<="000"; IE_op2 <= "001"; IE_op3 <= "010"; OE
<= '0'; enM0 <= '0'; enM1 <= '1'; enM2 <= '0';
   WHEN s5 => IE_op1<="000"; IE_op2 <= "001"; IE_op3 <= "100"; OE
<= '0'; enM0 <= '0'; enM1 <= '0'; enM2 <= '1';
   WHEN s6 => IE_op1<="010"; IE_op2 <= "011"; IE_op3 <= "000"; OE
<= '0'; enM0 <= '1'; enM1 <= '0'; enM2 <= '0';
   WHEN s7 => IE_op1<="010"; IE_op2 <= "100"; IE_op3 <= "000"; OE
<= '0'; enM0 <= '1'; enM1 <= '0'; enM2 <= '0';
   WHEN s8 => IE_op2 <= "011"; IE_op3 <= "101"; OE <= '0'; enM0 <=
'0'; enM1 <= '1'; enM2 <= '0';
   WHEN s9 => IE_op2 <= "011"; IE_op3 <= "101"; OE <= '0'; enM0 <=
'0'; enM1 <= '1'; enM2 <= '0';
   WHEN s10 => IE_op2 <= "010"; IE_op3 <= "101"; OE <= '0'; enM0 <=
'1'; enM1 <= '0'; enM2 <= '0';
   WHEN s11 => IE_op1<="011"; IE_op2 <= "010";
IE_op3 <= "000"; OE <= '0'; enM0 <= '1'; enM1 <= '0'; enM2 <= '0';
   WHEN others => IE_op1<="010"; IE_op2 <= "100"; IE_op3 <= "000";
OE <= '1'; enM0 <= '0'; enM1 <= '0'; enM2 <= '0';
END CASE;
END PROCESS;
END fsm_KOA13;
```

The results of the simulation are shown in Figure 8.a and 8.b.

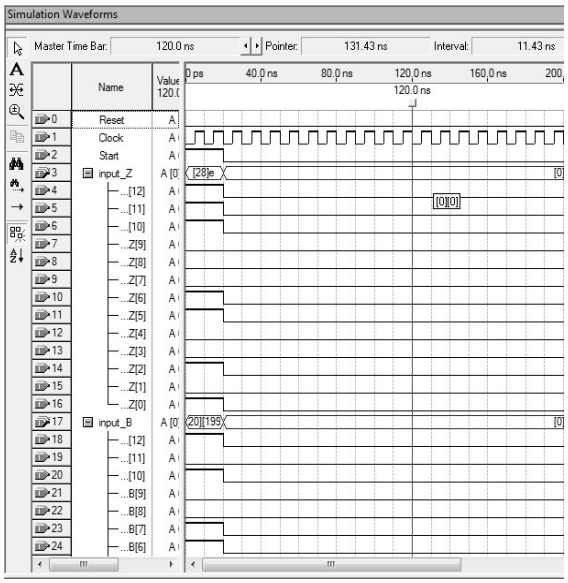


Fig. 8.a. Simulation result for polynomial multiplier in $GF(2^{13})$, part 1

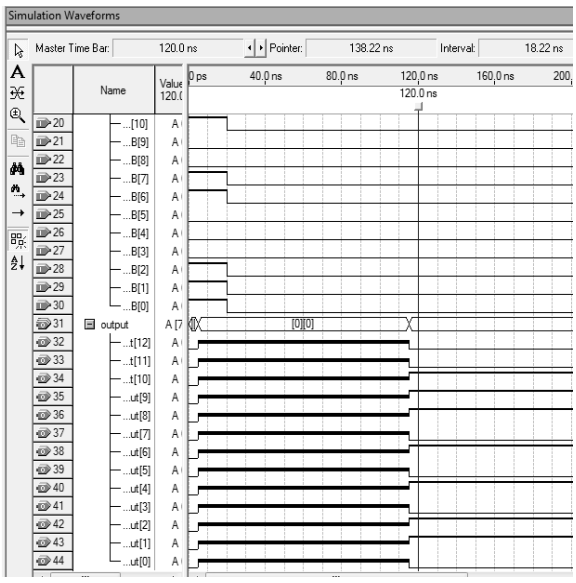


Fig. 8.b. Simulation result for polynomial multiplier in $GF(2^{13})$, part 2

In Quartus II, the complexity of the Simple Karatsuba algorithm is shown by a Combinational ALUTs (Adaptive Look Up Tables), in which a Combinational ALUTs consists of more than one logic gate. It can be seen that to run the programs of multiplier in $GF(2^{13})$ using this algorithm requires 117 combinational ALUTs (Figure 9).

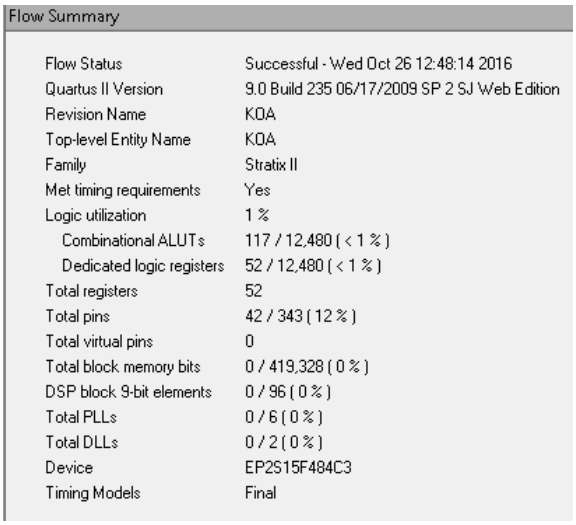


Fig. 9. Flow summary from Simulation result of polynomial multiplier in $GF(2^{13})$ for Karatsuba Method.

On the other hand, the complexity of the NAYK algorithm requires as many as 182 Combinational ALUTs. This value is greater than the complexity of the Simple Karatsuba method because XOR process increases significantly. But if the NAYK algorithm is implemented with a composite filed on the outside power, the XOR operation will be executed only once. While a large impact on efficiency is the number of multiplication process that has been generated by the algorithm NAYK, these multiplication process are then operated with power on the inside.

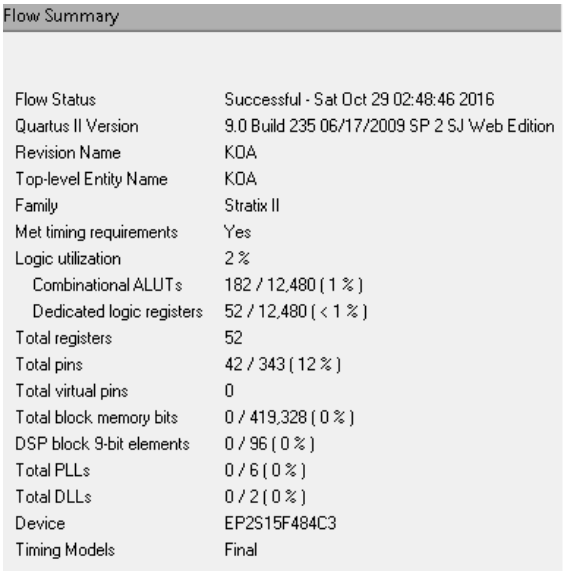


Fig. 10. Flow summary from Simulation result of polynomial multiplier in $GF(2^{13})$ for NAYK Method.

Because formula (17) uses the multipliers of $n$ = 7 and 6,

the number of multiplication processes generated by the NAYK algorithm is 22+(22-1)+17=60. Thus, the composite complexity of Karatsuba-NAYK is (117*60+182)=7,202.00 Combinational ALUTs.

Meanwhile, the number of multiplications generated by the Karatsuba algorithm is 49+(49-1)+36=133. Therefore, the composite complexity of Karatsuba-Karatsuba is 117*133+117=15,678.00 Combinational ALUTs. A comparison of the complexities of the composite methods is shown in Table VII.

TABLE VII
COMPARISON OF COSTS FOR THE COMPOSITE FIELD KARATSUBA-NAYK IN COMBINATIONAL ALUTS

| Composite Methods | O(13) | Combinational ALUTs | |
|---|---|---|---|
| | | $GF(2^{13})$ | $GF((2^{13})^{13})$ |
| Karatsuba-Karatsuba | 133 | 117 | 15,678.00 |
| Karatsuba-NAYK | 60 | 182 | 7,202.00 |

It can be clearly observed that the costs required for executing the composite Karatsuba-NAYK algorithm is as low as about 50% of the costs required for the composite Karatsuba-Karatsuba method. This is caused by the fact that the NAYK method significantly reduces the number of multiplications. Therefore, it has been proven that the NAYK method can improve the efficiency of polynomial multiplication in the Galois Field by reducing the complexity or cost (measured in Combinational ALUTs) of the calculations.

Therefore, it has been proven that the NAYK can improve the efficiency of the area and also the speed by reducing the complexity or cost of combinational ALUTs.

The results of this development, NAYK method, are well suited for the composite field, which is placed on the outside. Because the number of multiplication of this method is much less when compared with the results of the Simple Karatsuba algorithm. This causes the amount of calculation becomes much diminished significantly, and this will improve the efficiency of the area. Moreover, it provides an easy step in finding a polynomial multiplier in $GF(2^n)$.

Furthermore, with NAYK algorithm we can find and develop bigger bits and better of the polynomial multiplier for $n>7$ more quickly and produce a number of products are much smaller because the upper limit is a function $O(n)$ of Montgomery.

Moreover, with the implementation of NAYK algorithm results in composite field, implementation of ECC will be much better than before because at least the processing time will be two times faster than before. This is due to the process of arithmetic can be reduced by half, as described above. In fact we can apply the results in other processes such as inverse that uses a multiplication operation in the process, so that the ECC processing time will be much faster.

In addition, since the implementation of NAYK algorithm can reduce the number of arithmetic processes significantly, then the area of ECC can be reduced smaller than before.

Why this research is necessary, because the results will have a great influence on the efficiency of the area as well as time in ECC as a whole. Now, look at figure 11 that in ECC there are three levels of calculation.

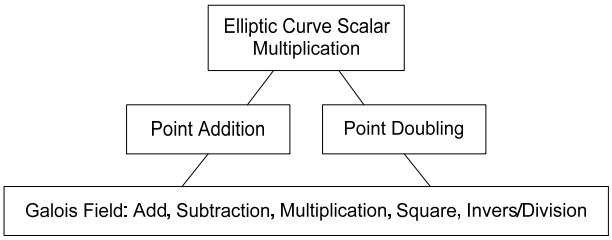


Fig. 11. Three levels of process in ECC

In the encryption process there is a public key generation process $P_b = k_b.B$ with a very large integer $k_b$, this is at the level of scalar multiplication in EC. For example if $k_b = 100$, then the doubling and addition rule of $P_b$ can be calculated as $P_b=100B=2(2(2(2B+B)))+B))$, then to do the calculation will need 2 addition and 6 doubling. Meanwhile, if the ECC is applied in affine coordinates, then at the level of $GF(2^n)$ the addition process require 1 inverse, 2 multiplication, and 1 square, and the doubling process require 1 inverse, 2 multiplication, and 2 square. Then it is known that the square process is also a multiplication process, so the calculation of $P_b$ will at least involve (2+1)2+(2+2)6=30 multiplication, while the inverse process is only done as much as 1*2+1*6=8, but the inverse process does not take up a lot of resources in $GF(2^n)$ because the process is simple unlike the complexity of the multiplication process [25], For comparison see figure 12.

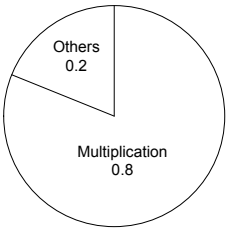


Fig. 12. Distribution area in ECC

So if a multiplication process at the arithmetic level of $GF(2^n)$ can cause enlargement of the area, then with the processes of as much as 30 multiplication will cause more enlargement of the area. Thus it is clear that the multiplication process in arithmetic $GF(2^n)$ gives a direct influence on the encryption process in ECC. Using the results from the NAYK Algorithm as the results in table VII states that the algorithm can reduce the efficiency of the complexity area of the product around 50%, so that the overall efficiency of the ECC process is about 40%, see figure 13.

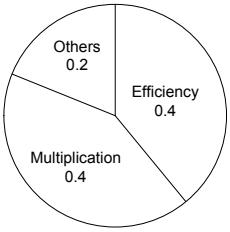


Fig. 13. Distribution and efficiency area in ECC after using NAYK Algorithm

To speed up the processing time it depends on how the architecture is designed from the start level in $GF(2^n)$ to the scalar multiplication process in EC. But with NAYK algorithm results, ECC architecture can be designed to be better and can speed up the processing time.

Further research on [26-28], the multiplication forms in $GF(2^n)$ can be improved in efficiency if using NAYK algorithm, because this algorithm is not only used in ECC but in other research field can be used as in error correction codes.

## V. Conclusion & Future Work

In this study, we have developed an improved Generalizations of the Karatsuba Algorithm (GKA) formula for use in polynomial multiplications in the Galois Field $GF(2^n)$. The aim was to improve the efficiency of the calculations by developing an algorithm in which the number of multiplications is significantly reduced compared to previous methods. The Improved GKA formula utilizes symmetry, a property that arises in the multiplication process. By solving only half of the equations in $C'$ (including for $c'(n\text{-}1)$), the solutions to the other half can be obtained using symmetry. However, for large values of $n$, the algorithm fails to execute with acceptable time and computer resources. In order to solve this problem, we developed an Exhaustive Search Algorithm to find the combinations of products that are solutions to equation $C'$, hence reducing the number of multiplications required. For large values of $n$, though, this algorithm requires large amounts of resources. To overcome the shortcomings of both algorithms, we developed the NAYK algorithm that combines both the Improved GKA and Exhaustive Search Algorithm. The NAYK algorithm has a much lower complexity compared to previous methods to solve polynomial multiplications in $GF(2^n)$.

In further research, we aim to develop improved algorithms for polynomial multiplications in $GF(2^n)$ for values of $n$ of larger than 7, and implement the algorithms in a composite field. The development of such algorithms is useful in a variety of applications, including Elliptic Curve Cryptography (ECC). By utilizing the NAYK algorithm, limited computer resources is no longer an obstacle to implementing ECC.

## References


[1] Andre Weimerskirch and Christof Paar, "*Generalizations of the Karatsuba Algorithm for Efficient Implementations*," Ruhr-Universitat Bochum, Germany, 2003

[2] Berk Sunar, "*A Generalized Method for Constructing Subquadratic Complexity GF*($2^k$) *Multipliers*," IEEE Transactions on Computers, Vol. 53, No. 9, September 2004

[3] Chester Rebeiro, Debdeep Mukhopadhyay, "*High Performance Elliptic Curve Crypto-Processor for FPGA Platforms*," Dept. of Computer Science and Engineering, IIT Kharagapur, 2008

[4] Chester Rebeiro, Debdeep Mukhopadhyay, "*High Speed Compact Elliptic Curve Cryptoprocessor for FPGA Platforms*," INDOCRYPT 2008: 376-388

[5] Chester Rebeiro, Sujoy Sinha Roy, Debdeep Mukhopadhyay, "*Pushing the Bound of High Speed GF(*$2^m$*) Elliptic Curve Scalar Multiplier on FPGAs*," CHES Springer, December 2012

[6] Cristof Paar, "*A New Architecture for a Parallel Finite Field Multiplier with Low Complexity Based on Composite Fields*," IEEE Transactions on Computers, July 1996

[7] Cristof Paar, Peter Fleischmann, Peter Roelse, "*Efficient Multiplier Architectures for Galois Fields GF (*$2^{4n}$*)*," IEEE Transactions on Computers, February 1998

[8] Daniel V. Bailey and Christof Paar, "*E cient Arithmetic in Finite Field Extensions with Application in Elliptic Curve Cryptography*," Journal of Cryptology, 2001

[9] Don Johnson, Alfred Menezes, Scott Vanstone, "*The Elliptic Curve Digital Signature Algorithm (ECDSA)*", Certicom, 2001

[10] Haining Fan, Jiaguang Sun, Ming Gu and Kwok-Yan Lam, "*Overlap-free Karatsuba-Ofman Polynomial Multiplication Algorithms*," IET Information security, vol. 4, no. 1, pp. 8-14, 2010

[11] Haining Fan and M. Anwar Hasan, Senior Member, IEEE, "*Comments on "Five, Six, and Seven-Term Karatsuba Like Formulae*"," IEEE Transactions on Computers, Vol. 56, No. 5, May 2007

[12] Ivan Oseledets, "*Improved n-Term Karatsuba Like Formulas in GF(2)*," IEEE Transactions on Computers, Vol. 60, No. 8, August 2011

[13] M. Ernst, M. Jung, F. Madlener, S. Huss, and R. Blumel, "*A Reconfigurable System on Chip Implementation for Elliptic Curve Cryptography over GF(*$2^n$*)*," Integrated Circuits and System Lab., Computer Science Department, Darmstadt University of Technology, Germany

[14] Misrolav Knezevic, "*Efficient Hardware Implementations of Cryptographic Primitives*," Disertation, Arenberg Doctoral School of Science, Engineering & Technology, Maret 2011

[15] M. Machhout, M. Zeghid, W. El hadj youssef, B. Bouallegue, A. Baganne, and R. Tourki, "*Efficient Large Numbers Karatsuba-Ofman Multiplier Designs for Embedded Systems*," World Academy of Science, Engineering and Technology 28, 2009

[16] Muhamad Nursalman, Arif Sasongko, Yusuf Kurniawan, Kuspriyanto, "*Improved Generalizations of The Karatsuba Algorithm in GF(*$2^n$*)*," IEEE International Conference on Advance Informatics: Concepts, Theory and Applications, August 2014, Bandung-Indonesia

[17] Muhamad Nursalman, Arif Sasongko, Yusuf Kurniawan, Sarwono Sutikno, "*Architecture Design and Implementation of KOA Multiplier for Small Bits in Galois Field*," IEEE International Conference on Electronics Technology and Industrial Development, October 2013, Bali-Indonesia

[18] Peter L. Montgomery, "*Five, Six, and Seven-Term Karatsuba Like Formulae*," IEEE Transactions on Computers, Vol. 54, No. 3, March 2005

[19] Sameh M. Shohdy, Ashraf B. El-Sisi, and Nabil Ismail, "*Hardware Implementation of Efficient Modi ed Karatsuba Multiplier Used in Elliptic Curves*," International Journal of Network Security, Vol.11, No.3, PP.155-162, Nov. 2010

[20] Sandeep S. Kumar, *"Elliptic Curve Cryptography for Constrained Devices"*, Verlag Dr. Muller, Saarbrucken, Germany, 2008

[21] Steffen Peter and Peter Langendorfer, "*An Ef cient Polynomial Multiplier in GF(*$2^m$*) and its Application to ECC Designs*," IHP GmbH, Frankfurt(Oder), Germany

[22] Sudhanshu Mishra, Manoranjan Pradhan, "*Synthesis Comparison of Karatsuba Multiplier using Polynomial Multiplication, Vedic Multiplier and Classical Multiplier*," International Journal of Computer Applications (0975 – 8887) Volume 41– No.9, March 2012

[23] Vinodh Gopal (Intel Corporation, USA), Satyajit Grover, Michael E. Kounavis, "*Fast Multiplication Techniques for Public Key Cryptography*," IEEE Symposium on Computers and Communications, 2008. ISCC 2008

[24] Y.A.Suryawanshi, Neha Trimbak Khadgi, "*Design Of Elliptic Curve Crypto Processor with Modified Karatsuba Multiplier and its Performance Analysis*," International Journal of Distributed and Parallel Systems (IJDPS) Vol.4, No.3, May 2013

[25] Darrel Hankerson, Alfred Menezes, Scott Vanstone, "*Guide to Elliptic Curve Cryptography*," Springer-Verlag, New York, Inc. 2004

[26] Xiaoqiang ZHANG, Ning WU, Gaizhen YAN, and Liling DONG, "*Hardware Implementation of Compact AES S-box*," IAENG International Journal of Computer Science, vol. 42, no.2, pp125-131, 2015

[27] Dindayal Mahto, Danish Ali Khan, and Dilip Kumar Yadav, "*Security Analysis of Elliptic Curve Cryptography and RSA*," Lecture Notes in Engineering and Computer Science: Proceedings of The World Congress on Engineering 2016, 29 June - 1 July, 2016, London, U.K., pp419-422

[28] Yaoping Liu, Ning Wu, Xiaoqiang Zhang, Liling Dong, and Lidong Lan, "*An Area Optimized Implementation of AES S-Box Based on Composite Field and Evolutionary Algorithm*," Lecture Notes in Engineering and Computer Science: Proceedings of The World Congress on Engineering and Computer Science 2015, 21-23 October, 2015, San Francisco, USA, pp33-37

**Muhamad Nursalman** received his Bachelor (Mathematics, 2002) and Master degree (Informatics, 2005) from Institut Teknologi Bandung, Indonesia. Now he is continuing his graduation in Doctoral degree at the School of Electrical Engineering and Informatics, at the same university. Since 2006, he has been with Department of Computer Science, Universitas Pendidikan Indonesia, where now he is an Assistant Professor. His research interests focus on Mathematics and Cryptography.

**Arif Sasongko** received his Bachelor and Master degree (Electrical Engineering) from Institut Teknologi Bandung, Indonesia, and he received his Ph.D degree from Universite Joseph Fourier, France, in 2005. Since 2006, he has been with the School of Electrical Engineering and Informatics, Institut Teknologi Bandung, Indonesia. Now, he is an Associate Professor with research interests focus on Cryptography, SoC, and Embedded Systems.

**Yusuf Kurniawan** received his Bachelor, Master and Doctoral degree from Institut Teknologi Bandung, Indonesia. Since 2008, he has been with the School of Electrical Engineering and Informatics, Institut Teknologi Bandung, Indonesia. Now, he is an Associate Professor with research interests focus on Telecommunication and Electrical Engineering, and Cryptography. He is known as the inventor of Bc1, Bc2, and Bc3. He has improved block cipher algorithm.

**Kuspriyanto** received his Bachelor degree (Electrical Engineering) from Institut Teknologi Bandung, Indonesia, and he received his DEA and Ph.D degree from Universite Des Sciences et Techniques du Languedoc, France, in 1981. Since 1976, he has been with the School of Electrical Engineering and Informatics, Institut Teknologi Bandung, Indonesia. Now, he is a Professor with research interests focus on the Automatic System, Computer Architecture, and Cryptography.