The Efficient Multiplier GF(2^(8)) is Formed by The NAYK Algorithm

Muhamad Nursalman[1], Arif Sasongko[2]

{mnursalman@upi.edu[1], asasongko@stei.itb.ac.id[2]}

Department of Computer Science Education, FPMIPA, Universitas Pendidikan Indonesia[1]

School of Electronics and informatics Engineering, Institut Teknologi Bandung[2]

Abstract. The efficient multiplier in Finite Field is needed in its implementation in the cryptography field. The NAIK algorithm provides fast steps and efficient solutions in forming the desired multiplier. The formation of an efficient multiplier GF(2^8) will be formed with the NAYK algorithm without being constructed from the smallest values, but directly from the value 8 itself. In comparison the NAYK algorithm provides a more efficient solution.

Keywords: Efficient multiplier, Finite field, GF(2^8), NAYK algorithm, Generalization of karatsuba algorithm

1 Introduction

This research developed two methods, these are the methods to reduce of sum of multiplication and to find solution for all formula by exhaustive search. Both the development of formula and algorithm that if executed one by one in the process of reducing the multiplication does not give better results, due to the constraints of the selection of products to be combined quite difficult [4], especially for large n, moreover the memory size is bounded and processing time is very long. But if we combine both and by utilizing the properties of algebra which appeared in the multiplication of polynomials in $GF(2^n)$, then both the development of formula and algorithm give the process much better than the research that has been done [3,5-9,]. In addition, when compared with the previous researches [3,5,9], the combination of both formula and algorithm provides a process much easier.

2 Related Research and Theory

2. 1 Algorithm of Karatsuba Improved by Christof Paar

The formula in polynomial with degree (n-1) [1,2], then calculate

MSCEIS 2019, October 12, Bandung, Indonesia

DOI 10.4108/eai.12-10-2019.2296349

$$D(j,k) = (a(j) + a(k))(b(j) + b(k))$$

and

$$D(i) = a(i)b(i)$$

Hence the polynomial formula can be calculated like below

$$c'(i) = \sum_{j,k} D(j,k) - \sum_{j,k} \left(D\left(\frac{i}{2}\right) + D(k) + D(j) \right),$$

for *i* is even,

$$c'(i) = \sum_{j,k} D(j,k) - \sum_{j,k} (D(k) + D(j)),$$

for *i* is odd, and

$$c\,'(0) = D(0)$$

$$c'(2n-2) = D(n-1)$$

Moreover, multiplier for Finite Field GF((2^n)^4) for equation n = 4,

Paar changed it like in [2,7,10,11].

Take

$$A(x) = a(0)x^0 + \cdots + a(3)x^3;\ A \in GF((2^n)^4);\ a_i \in GF(2^n)$$

and

$$B(x) = b(0)x^0 + \cdots + b(3)x^3;\ B \in GF((2^n)^4);\ b_i \in GF(2^n)$$

and

$$C = AB\ mod\ P$$

Moreover, Paar modified it into

$$\begin{aligned} c'(3) = {} & D(0) + D(1) \\ & + D(2) + D(3) \\ & + D(0,1) + D(0,2) \\ & + D(1,3) + D(2,3) \\ & + (a_0 + a_1 + a_2 + a_3)(b_0 + b_1 + b_2 + b_3) \end{aligned}$$

where

$$D(0) = a(0)b(0)$$
$$D(1) = a(1)b(1)$$
$$D(2) = a(2)b(2)$$
$$D(3) = a(3)b(3)$$

$$D(0,1) = (a(0) + a(1))(b(0) + b(1))$$

$$D(0,2) = (a(0) + a(2))(b(0) + b(2))$$

$$D(1,3) = (a(1) + a(3))(b(1) + b(3))$$

$$D(2,3) = (a(2) + a(3))(b(2) + b(3))$$

$$D(2,3) = (a(2) + a(3))(b(2) + b(3))$$

Now the number of multiplications has changed from ten to nine. But Paar did not explain the process of the reduction. But several researches [12,13] has discussed of modulo process. The results use to build for another multiplication for n bigger than 7 [3,7].

2.2 Montgomery Multiplier

Multiplier for n = 5, 6 and 7 has developed by Montgomery. For example, this is multiplier for n = 6 [3].

Write that

$$a_0b_1 + a_1b_0 = (a_0 + a_1)(b_0 + b_1) - a_0b_0 - a_1b_1$$

and

$$(a_1X + a_0)(b_1X + b_0) = a_1b_1X^2 + (a_0b_1 + a_1b_0)X + a_0b_0$$

Hence it can be written as follows

$$\begin{aligned}(a_1X + a_0)(b_1X + b_0) &= a_1b_1X^2 + \big((a_0 + a_1)(b_0 + b_1) - a_0b_0 - a_1b_1\big)X + a_0b_0 \\ &= a_1b_1(X^2 - X) \\ &\quad +(a_0 + a_1)(b_0 + b_1)X \\ &\quad +a_0b_0(1 - X)\end{aligned}$$

Hence for n = 3, the equation is

$$
\begin{aligned}
&(a_2X^2 + a_1X + a_0)(b_2X^2 + b_1X + b_0) \\
&= a_0b_0(K + 1 - X - X^2) \\
&+a_1b_1(K - X + X^2 - X^3) \\
&+a_2b_2(K - X^2 - X^3 + X^4) \\
&+(a_0 + a_1)(b_0 + b_1)(-K + X) \\
&+(a_0 + a_2)(b_0 + b_2)(-K + X^2) \\
&+(a_1 + a_2)(b_1 + b_2)(-K + X^3) \\
&+(a_0 + a_1 + a_2)(b_0 + b_1 + b_2)K
\end{aligned}
$$

The multiplication number has changed from nine to six, where $K = X^2$, and it make multiplier more efficient. Moreover, the formula uses to construct for the multipliers for n = 5, 6, and 7, where the result of the big O function can be seen in [3]. The results are more efficient than the previous multiplier. But Montgomery called it the division-free formula, because it was very hard to construct for each multiplier, particularly for n = 7.

3 Method

3.1 Algorithm of NAYK

Algorithm of NAYK [14] integrate the methods of development of GKA and the exhaustive search algorithm above, where both are mutually supportive of each other. The development of GKA serves to identify some joint products into a solution and which would not be the solution in equation C. This method reduce the number of multiplication in D matrix, then automatically it will cut down of the rows of the D matrix. In addition to this algorithm, not all possible combinations that appeared to be calculated, but through proper identification, just only a few equation needs to be solved. Hence this algorithm can reduce amount of equation to be solved, and also make it more faster. Here is how the NAYK Algorithm works.

1. Determine the value of n
2. Create a C' matrix
3. Find the (best) solution for c'(n-2), c'(n-1), and c'(n)
4. Identify the elements that are the solution in C' from points 2 and 3 (this is called the C'_solution1 matrix)
5. Identify the elements that are not the solution in C' (this is called the C'_resolution matrix)

6. Create a D matrix
7. Create a new D matrix by means of
   D_new = D - (C'_solution1 + C'_solution)
8. Choose the combination of elements from the elements of the C'_solution1 matrix and the new D_ matrix to be the solution to the equation c'(i), i = 0, 1, 2, ..., n-1.
9. For each combination of elements selected then match with c'(i), for a particular i, with i = 0, 1, 2, ..., n-1.
10. Select the solution with O (n) ≤ n ^ log2 (3).

4 Result and Discussion

In this section, we will describe the results obtained on the steps in the above methods and discuss them one by one. Then the results will be compared with existing research.

4.1 Improving Formula better than Paar Algorithm in GF(2^n)

This will show how to construct the formula better than Paar algorithm.

$$
\begin{aligned}
c_0' &= a_0 b_0 \\
c_1' &= a_0 b_1 + a_1 b_0 \\
c_2' &= a_0 b_2 + a_1 b_1 + a_2 b_0 \\
c_3' &= a_0 b_3 + a_1 b_2 + a_2 b_1 + a_3 b_0 \\
&\vdots \\
c_{m-1}' &= a_0 b_{m-1} + a_1 b_{m-2} + \cdots + a_{m-2} b_1 + a_{m-1} b_0 \\
c_m' &= a_1 b_{m-1} + a_2 b_{m-2} + \cdots + a_{m-2} b_2 + a_{m-1} b_1 \\
&\vdots \\
c_{2m-4}' &= a_{m-3} b_{m-1} + a_{m-2} b_{m-2} + a_{m-1} b_{m-3} \\
c_{2m-3}' &= a_{m-2} b_{m-1} + a_{m-1} b_{m-2} \\
c_{2m-2}' &= a_{m-1} b_{m-1}
\end{aligned}
$$

or it can write as follows

$$
\begin{aligned}
c_0' &= a_0 b_0 \\
c_1' &= (a_0 b_1 + a_1 b_0) \\
c_2' &= (a_0 b_2 + a_2 b_0) + a_1 b_1 \\
c_3' &= (a_0 b_3 + a_3 b_0) + (a_1 b_2 + a_2 b_1) \\
&\vdots
\end{aligned}
$$

If m-1 is odd, then

$$
\begin{aligned}
c'_{m-1} &= \left(a_0 b_{m-1} + a_{m-1} b_0\right) + \left(a_1 b_{m-2} + a_{m-2} b_1\right) + \ldots + \left(a_{m/2-1} b_{m/2} + a_{m/2} b_{m/2-1}\right) \\
c'_m &= \left(a_1 b_{m-1} + a_{m-1} b_1\right) + \left(a_2 b_{m-2} + a_{m-2} b_2\right) + \ldots + \left(a_{(m-2)/2} b_{(m+2)/2} + a_{(m+2)/2} b_{(m-2)/2}\right) \\
&\quad + a_{m/2} b_{m/2}
\end{aligned}
$$

If m-1 is even, then

$$
\begin{aligned}
c'_{m-1} &= \left(a_0 b_{m-1} + a_{m-1} b_0\right) + \left(a_1 b_{m-2} + a_{m-2} b_1\right) + \ldots + \left(a_{(m-3)/2} b_{(m+1)/2} + a_{(m+1)/2} b_{(m-3)/2}\right) \\
&\quad + a_{(m-1)/2} b_{(m-1)/2} \\
c'_m &= \left(a_1 b_{m-1} + a_{m-1} b_1\right) + \left(a_2 b_{m-2} + a_{m-2} b_2\right) + \ldots + \left(a_{(m-1)/2} b_{(m+1)/2} + a_{(m+1)/2} b_{(m-1)/2}\right) \\
&\vdots \\
c'_{2m-4} &= \left(a_{m-3} b_{m-1} + a_{m-1} b_{m-3}\right) + a_{m-2} b_{m-2} \\
c'_{2m-3} &= \left(a_{m-2} b_{m-1} + a_{m-1} b_{m-2}\right) \\
c'_{2m-2} &= a_{m-1} b_{m-1}
\end{aligned}
$$

Suppose

$$a_i b_i = D_i \text{ and } \left(a_i + a_j\right)\left(b_i + b_j\right) = D_{i,j},$$

then

$$
\begin{aligned}
a_i b_j + a_i b_j &= \left(a_i + a_j\right)\left(b_i + b_j\right) - a_i b_i - a_j b_j \\
&= D_{i,j} - D_i - D_j \\
&= D_{i,j} - \left(D_i + D_j\right)
\end{aligned}
$$

Then it can be written as follows

$$
\begin{aligned}
c'_0 &= D_0 \\
c'_1 &= D_{0,1} - D_0 - D_1 \\
c'_2 &= D_{0,2} + D_1 - \left(D_0 + D_2\right) \\
c'_3 &= D_{0,3} + D_{1,2} - \left(D_0 + D_3\right) - \left(D_1 + D_2\right) \\
&\vdots
\end{aligned}
$$

If $m-1 \in \{1,3,5,...\},$ then

$$
\begin{aligned}
c'_{m-1} &= D_{0,m-1} + D_{1,m-2} + \ldots + D_{m/2-1,m/2} - \left(D_0 + D_{m-1}\right) - \left(D_1 + D_{m-2}\right) \\
&\quad - \ldots - \left(D_{m/2-1} + D_{m/2}\right) \\
c'_m &= D_{1,m-1} + D_{2,m-2} + \ldots + D_{(m-2)/2,(m+2)/2} + D_{m/2} - \left(D_1 + D_{m-1}\right) - \left(D_2 + D_{m-2}\right) - \ldots \\
&\quad - \left(D_{(m-2)/2} + D_{(m+2)/2}\right) \\
&\quad \vdots
\end{aligned}
$$

If $m-1 \in \{2,4,6,...\},$ then

$$
\begin{aligned}
c'_{m-1} &= D_{0,m-1} + D_{1,m-2} + \ldots + D_{(m-3)/2,(m+1)/2} + D_{(m-1)/2} - \left(D_0 + D_{m-1}\right) - \left(D_1 + D_{m-2}\right) - \ldots \\
&- \left(D_{(m-3)/2} + D_{(m+1)/2}\right) \\
c'_m &= D_{1,m-1} + D_{2,m-2} + \ldots + D_{(m-1)/2,(m+1)/2} - \left(D_1 + D_{m-1}\right) - \left(D_2 + D_{m-2}\right) - \ldots \\
&- \left(D_{(m-1)/2} + D_{(m+1)/2}\right) \\
&\vdots \\
c'_{2m-4} &= D_{m-3,m-1} + D_{m-2} - \left(D_{m-3} + D_{m-1}\right) \\
c'_{2m-3} &= D_{m-2,m-1} - \left(D_{m-2} + D_{m-1}\right) \\
c'_{2m-2} &= D_{m-1}
\end{aligned}
$$

In another form it can be written as follows

For $m-1 \in \{1,3,5,\dots\}$, then

$$D_{p_0,p_1} + D_{p_2,p_3} + \cdots + D_{p_{m-2},p_{m-1}} = D_{p_{(0)},p_{(1)}\cdots,p_{(m-1)}} - \sum_{\substack{k-j\neq 1 \\ 0\leq j<k\leq m-1}} D_{p_j,p_k},$$

where

$$p_j < p_{j+1}; p_{(0)} < p_{(1)} < \cdots < p_{(m-1)}; p_j < p_k \text{ dan } p_j, p_k \in \mathbb{N}^+ \; \forall \; 0 \leq j < k \leq m-1$$

For $m-1 \in \{2,4,6,...\}$, then

$$D_{p_0,p_1} + D_{p_2,p_3} + \ldots + D_{p_{m-2},p_{m-1}} = D_{p_{(0)},p_{(1)}\cdots,p_{(m-1)}} - \sum_{\substack{k-j\neq 1 \\ 0\leq j<k\leq m-1}} D_{p_j,p_k} - \sum_{s=0}^{m-1} D_{p(s)},$$

where

$$p_j < p_{j+1}; p_{(0)} < p_{(1)} < \cdots < p_{(m-1)}; p_j < p_k \text{ dan } p_j, p_k \in \mathbb{N}^+ \; \forall \; 0 \leq j < k \leq m-1$$

4.2 Algorithm of Brute Force Search for Multiplier in Finite Field

To solve efficient multiplier from the above equation, this research use algorithm of Brute Force Search to find the best solution for the formula n = 8. It is easy to follow the all steps of this algorithm from previous researcher, which is explained in [14].

With combining of improving algorithm of Paar and brute force search, then it can result of multiplier in finite field GF(2^8) as follows.

c'(4) = D(013467)+D(0367)+D(1467)+D(01)+D(34)+D(67)

c'(5) = D(013467)+D(012456)+D(1347)+D(0367)+D(0245)+D(124)+D(04)+D(26)
+D(37)+D(56)

c'(6) = D(012456)+D(0245)+D(1246)+D(01)+D(56)

c'(7) = D(013467)+D(0346)+D(1347)+ D(01)+ D(25)+D(6,7)

for i = 1 to 3, the c'(i) is the same equation with the original, and for another c'(i), it can find easily with the theorem of reflection in the NAYK algorithm [14].

5 Conclusion

NAYK algorithm makes it possible to find a solution for the equation C with the number of multiplications slightly, where the upper bound is $O(n) \leq n^{\log(3,2)}$.

References

[1] Cristof Paar.: "A New Architecture for a Parallel Finite Field Multiplier with Low Complexity Based on Composite Fields". IEEE Transactions on Computers (1996)

[2] Cristof Paar, Peter Fleischmann, Peter Roelse.: "Efficient Multiplier Architectures for Galois Fields GF (24n)". IEEE Transactions on Computers (1998)

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

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

[5] Andre Weimerskirch and Christof Paar.: "Generalizations of the Karatsuba Algorithm for Efficient Implementations". Ruhr-Universitat Bochum, Germany (2003)

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

[7] Cristof Paar, Peter Fleischmann, Peter Roelse.: “Efficient Multiplier Architectures for Galois Fields GF (24n)”. IEEE Transactions on Computers (1998)

[8] Daniel V. Bailey and Christof Paar.: “Efficient Arithmetic in Finite Field Extensions with Application in Elliptic Curve Cryptography”. Journal of Cryptology (2001)

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

[10] Sameh M. Shohdy, Ashraf B. El-Sisi, and Nabil Ismail.: “Hardware Implementation of Efficient Modifled Karatsuba Multiplier Used in Elliptic Curves”. International Journal of Network Security. Vol.11, No.3, pp.155-162 2010)

[11] Ivan Oseledets.: “Improved n-Term Karatsuba Like Formulas in GF(2)”. IEEE Transactions on Computers. Vol. 60, No. 8 (2011)

[12] 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 (2007)

[13] Ivan Oseledets.: “Improved n-Term Karatsuba Like Formulas in GF(2)”. IEEE Transactions on Computers. Vol. 60, No. 8 (2011)

[14] Muhamad Nursalman, Arif Sasongko, Yusuf Kurniawan, Kuspriyanto.: “Generalizations of n-Term Karatsuba Like Formulae in GF(2n) with NAYK Algorithm”. IAENG International International Journal of Computer Science, 44:4, IJCS_44_4_02 (2017)