논문 2006-43SD-3-5

불필요한 연산이 없는 카라슈바 알고리즘과 하드웨어 구조

(An Efficient Architecture for Modified Karatsuba-Ofman Algorithm)

장 남 수*, 김 창 한**

(Nam Su Chang and Chang Han Kim)

요 약

Divide-and-Conquer.방법은 병렬 곱셈기의 구성에 잘 적용되며 가장 대표적으로 카라슈바 방법이 있다. Leone은 최적 반복 회수를 카라슈바 알고리즘에 적용하였으며 Ernst는 다중 분할 카라슈바 방법을 제안하였다. 본 논문에서는 카라슈바 알고리즘에서 불필요한 연산이 제거된 불필요한 연산이 없는 카라슈바 알고리즘과 효율적인 하드웨어 구조를 제안한다. 본 논문에서 제안하는 알고리즘은 기존의 카라슈바 알고리즘에 비교하여 같은 시간 복잡도를 가지나 공간 복잡도를 효율적으로 감소시킨다. 특히 확장체의 차수 n이 홀수 및 소수일 때 더 효율적이며 최대 43%까지 공간 복잡도를 줄일 수 있다.

Abstract

In this paper we propose the Modified Karatsuba-Ofman algorithm for polynomial multiplication to polynomials of arbitrary degree. Leone proposed optimal stop condition for iteration of Karatsuba-Ofman algorithm(KO). In this paper, we propose a Non-Redundant Karatsuba-Ofman algorithm (NRKOA) with removing redundancy operations, and design a parallel hardware architecture based on the proposed algorithm. Comparing with existing related Karatsuba architectures with the same time complexity, the proposed architecture reduces the area complexity. Furthermore, the space complexity of the proposed multiplier is reduced by 43% in the best case.

Keywords : Polynomial Multiplication, Karatsuba-Ofman Algorithm, Non-Redundant Karatsuba-Ofman Algorithm, Hardware Architecture

I. 서 론

타원 곡선 암호시스템(ECC)을 제외한 대부분의 공개키 암호시스템은 사이즈가 큰 키를 갖는다[5]. 이러한 성질은 스마트카드, 무선 통신과 같은 전력과 대역폭이 제한된 응용부분에서 비실용적이다. 또한 공개키 암호시스템을 구성하는 유한체 연산의 효율성은 암호시스템 설계에 중요한 부분을 차지한다.

* 학생회원, 고려대학교 정보보호대학원
(Center for Information and Security Technologies (CIST), Korea Univ.)
** 정회원, 세명대학교 정보보호학과
(Dept. of Information and Security, Semyumg Univ.)
※ 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 육성·지원사업의 연구결과로 수행되었음
접수일자: 2005년10월17일, 수정완료일: 2006년3월7일

두 다항식의 효율적인 곱셈 연산은 시그널 프로세싱, 암호학 및 부호 이론의 중요한 부분이다. 본 논문에서는 이진체에서의 곱셈 연산과 이진체에서 정의된 타원곡선에서의 효율성에 대하여 기술한다.

낮은 공간 복잡도를 요구하는 어플리케이션에서 공간 복잡도를 감소시키는 것은 디자인 측면에서 매우 중요한 관점이다. Leone은 [6]에서 낮은 공간 복잡도를 가지는 병렬 곱셈기를 제안하였다. 이는 기존의 전형적인 병렬 곱셈기[3,7,8]와 다르게 시간과 공간 복잡도의 trade-off를 이용하여 보다 낮은 공간복잡도를 제공한다. 또한 Ernst는 [4]에서 divide-and-conquer 방법인 카라슈바 방법을 적용한 일반적이고 확장 가능한 유한체 코프로세서를 제안하였다. 또한 [1,2]에서는 정해진 확장제 차수에서 카라슈바 방법을 효율적으로 설계할 수 있는 방법을 제안하였다.

기존의 이진체에서 정의된 타원곡선의 카라슈바 병

렬 곱셈기[1,2,3,4,6,7,8]는 불필요한 연산을 포함하고 있다. 이는 이진체의 위수가 소수이기 때문이다. 본 논문에서는 다항식 기저위에서 정의된 불필요한 연산이 없는 카라슈바 알고리즘을 제안하고 낮은 공간 복잡도를 제공하는 병렬 하드웨어 구조를 제안한다. 제안하는 하드웨어 곱셈기는 기존의 곱셈기와 같은 시간 복잡도를 가지는 반면에 더 낮은 공간복잡도를 제공하며 최대 43%정도 공간복잡도를 줄인다.

본 논문의 구성은 다음과 같다. Ⅱ절에서는 이진체에서의 다항식 곱셈 방법에 대하여 기술하며 Ⅲ절에서는 불필요한 연산이 없는 카라슈바 알고리즘과 병렬처리 곱셈기를 제안한다. Ⅳ절에서는 제안하는 병렬곱셈기의 복잡도와 기존의 결과와의 비교결과를 보이며 Ⅴ절에서 결론을 내린다.

Ⅱ. 이진체 위의 다항식 곱셈

1. 기호

본 논문에서는 다음과 같은 기호를 사용한다. $a(x)$를 $GF(2^n)$의 원소라 가정하면 $(j-i)$차 다항식 $a(x)_i^j$는

$$a(x)_i^j = a_i + a_{i+1}x + \cdots + a_j x^{j-i},$$

이며 $(j-i)$가 음수인 경우 $a(x)_i^j = 0$이다.

2. 일반적인 곱셈 방법(SchoolBook : SB)

n차 기약다항식 $f(x)$에 의하여 생성된 유한체 $GF(2^n)$의 원소간의 곱셈에 관하여 살펴보자. 다항식 기저에 의해 $GF(2^n)$의 원소 $a(x)$와 $b(x)$는

$$a(x) = a_0 + a_1 x + \cdots + a_{n-2}x^{n-2} + a_{n-1}x^{n-1},$$
$$b(x) = b_0 + b_1 x + \cdots + b_{n-2}x^{n-2} + b_{n-1}x^{n-1}$$

와 같이 표현되며 이때 $a_i,\ b_i \in GF(2)$이다. 이진체 위의 곱셈은 이진체의 두 원소 $a(x)$와 $b(x)$를 $n-1$차 다항식으로 보고 곱하는 다항식 곱셈 과정과 곱셈의 결과인 $2n-2$차 다항식

$$c(x) = c_0 + c_1 x + \cdots + c_{2n-3}x^{2n-3} + c_{2n-2}x^{2n-2},\quad c_i \in GF(2)$$

를 n차 기약 다항식 $f(x)$로 나눈 나머지를 구하는 과정인 모듈로 축소(Modulo Reduction)로 나뉜다. 본 논문에서는 효율적인 다항식 곱셈 방법에 대해서만 기술한다. 병렬 곱셈기에서 AND, XOR, 전체 게이트의 수를 #AND, #XOR, #TOT라 하고 전체 시간 지연을 T_{TOT}라 할 때 일반적인 곱셈의 복잡도는 다음과 같다.

$$\#AND = n^2,$$

$$\#XOR = (n-1)^2,$$

$$\#TOT = 2n^2 - 2n + 1,$$

$$T_{TOT} = T_A + \lceil \log_2 n \rceil T_X.$$

3. 카라슈바 곱셈 방법(Karatsuba-Ofman : KOA)

$GF(2^n)$에서 기본적인 KOA방법은 유한체의 원소를 2중-분할하여 divide-and-conquer방법을 적용한다. 따라서 만약 $n/2$이 짝수라면 KOA는 반복 적용될 수 있다. 즉, $n-1$차의 다항식에 카라슈바 방법을 직접 적용하면 $\log_2 n$번 반복 수행이 가능하다. 복잡도를 고려할 때 XOR 연산은 두 과정으로 나누어진다. 첫째는 $n/2$ 비트 다항식 곱셈기의 입력 값의 연산이고, 두 번째는 $n/2$ 비트 다항식 곱셈기의 출력 값의 연산이다. 만약 카라슈바 방법을 1번 반복 수행한다면 알고리즘 1의 setp4에서 $2(n/2)$개의 XOR 게이트가 필요하고 step7에서 $3n-4$개의 XOR 게이트가 필요하다.

Algorithm 1. 카라슈바 알고리즘

$KOA(c(x), a(x), b(x), n)$

INPUT : $a(x), b(x) \in GF(2^n), n = 2^k$

OUTPUT : $c(x) = a(x) \cdot b(x)$

1. If $n \le 4$ then Mul(c(x),a(x),b(x),n) .
   Return(c(x)).
2. Set c(x)=0.
3. for i from 0 to n/2-1 do
   ADD $_{a(x)_i} = a_i + a_{n/2+i}$, ADD $_{b(x)_i} = b_i + b_{n/2+i}$.
4. KOA$(c(x), a(x)_0^{n/2-1}, b(x)_0^{n/2-1}, n/2)$.
5. KOA$(u(x)$, ADD$_{a(x)}$, ADD$_{b(x)}, n/2)$.
6. KOA$(c(x)_n^{2n-2}, a(x)_{n/2}^{n-1}, b(x)_{n/2}^{n-1}, n/2)$.
7. $c(x)_{n/2}^{3n/2-2} += u(x) + c(x)_0^{n-2} + c(x)_n^{2n-2}$.
8. Return(c(x)).

[6]에서 KOA의 최적 반복회수를 $n/2^k = 4$로 제안하였다. 따라서 KOA가 m번의 반복 연산을 수행하며 최

하위 연산기를 SB 곱셈기를 사용한다고 가정할 때 전체 연산량은 다음과 같다.

$$\#\mathrm{AND} = (3/4)^m \cdot n^2,$$

$$\#\mathrm{XOR} = (n^2/4^m + 6n/2^m - 1)3^m - (8n-2),$$

$$\mathrm{TOT} = (2n^2/4^m + 6n/2^m - 1)3^m - (8n-2),$$

$$T_{TOT} = (2 \cdot m + \lceil \log_2 n \rceil) T_X + T_A.$$

위의 결과에 의하여 KOA 방법을 이용한 병렬 처리 곱셈기는 SB 방법을 이용한 병렬 처리 곱셈기보다 공간 복잡도가 낮다.

III. 제안하는 향상된 카라슈바 알고리즘

본 절에서는 이진체위의 불필요한 연산이 없는 카라슈바 알고리즘을 제안한다. 제안하는 알고리즘은 기존의 알고리즘보다 낮은 공간복잡도를 가진다. 제안하는 알고리즘은 새로운 2개의 추가 알고리즘인 불필요한 연산이 없는 부분 카라슈바 알고리즘과 4비트 곱셈기를 사용한다.

1. 제안하는 알고리즘의 아이디어

본 소절에서는 제안하는 알고리즘의 기본 아이디어를 기술한다. $GF(2^n)$에서 $|n|$을 n의 비트 길이라고 하자. a(x)와 b(x)를 n-1차 다항식이라 하고 $n = 2^m + t$라 하면 다항식 a(x)와 b(x)는 다음과 같은 두 부분으로 구분된다.

$$a(x) = A_1 + A_2 x^{2^m},\ b(x) = B_1 + B_2 x^{2^m},$$
$$where\ m = \begin{cases} |n|-1 & \text{if } t=0, \\ n & \text{if } t \neq 0. \end{cases}$$

따라서 A_1과 B_1은 a(x)와 b(x)의 하위 부분이며 A_2과 B_2는 상위 부분이다. 위의 식을 이용하여 c(x)를 전개하면 다음과 같다.

$$\begin{aligned} c(x) &= a(x) \cdot b(x) \\ &= (A_1 + A_2 x^{2^m}) \cdot (B_1 + B_2 x^{2^m}) \\ &= ((A_1 + A_2) \cdot (B_1 + B_2) + A_1 B_1 + A_2 B_2) x^{2^m} \\ &\quad + A_1 B_1 + A_2 B_2 x^{2 \cdot 2^m} \end{aligned} \quad (1)$$

KOA는 위의 식과 같이 31번의 다항식 곱셈으로 c(x)를 계산한다. 만약 $t \neq 0$이면 $(A_1 + A_2)(B_1 + B_2)$과 $A_1 \cdot B_1$은 2^m-비트 다항식 곱셈이고 $A_2 B_2$는 t-비트 다항식 곱셈이다[8]. 이 경우 $A_2 B_2$는 t-비트 다항식 곱셈이므로 $(A_1 + A_2)(B_1 + B_2)$은 불필요한 연산을 포함하게 된다.

예를 들어 a(x)와 b(x)가 $GF(2^3)$의 원소라고 하면 식 (1)에서 $(A_1 + A_2)(B_1 + B_2)$은 다음과 같다.

$$\begin{aligned} &(A_1 + A_2) \cdot (B_1 + B_2) \\ &\quad = \{(a_0 + a_2) + a_1 x\} \cdot \{(b_0 + b_2) + b_1 x\} \\ &\quad = (a_0 + a_2)(b_0 + b_2) + a_1 b_1 x^2 \\ &\qquad + \{(a_0 + a_1 + a_2)(b_0 + b_1 + b_2) \\ &\qquad\quad + (a_0 + a_2)(b_0 + b_2) + a_1 b_1\} x \end{aligned} \quad (2)$$

식 (1)의 $A_1 B_1$에서 $a_1 b_1$은 이미 계산되었으므로 식 (2)의 $a_1 b_1$은 불필요한 연산이다. 따라서 제안하는 알고리즘은 이와 같은 불필요한 연산을 사용하지 않는다. 즉, 불필요한 연산이 없는 부분 카라슈바 알고리즘을 이용하여 식 (1)의 $(A_1 + A_2)(B_1 + B_2)$에서 발생하는 모든 불필요한 연산을 계산하지 않는다.

1. 불필요한 연산이 없는 카라슈바 알고리즘 (NRKOA)

Algorithm 2. 불필요한 연산이 없는 카라슈바 알고리즘

NRKOA(c(x), a(x), b(x), n)

INPUT : $a(x), b(x) \in GF(2^n), n = 2^k$

OUTPUT : $c(x) = a(x) \cdot b(x)$

1. If $n \leq 7$ then Mul(c(x),a(x),b(x),n) .
   Return(c(x)).
2. Set c(x)=0.
3. Set $m_1 = \lfloor \log_2(n-1) \rfloor$, $m_2 = n - 2^{m_1}$
4. for i from 0 to $m_2 - 1$ do
   ADD $_{a(x)_i} = a_i + a_{2^{m_1}+1}$, ADD $_{b(x)_i} = b_i + b_{2^{m_1}+1}$,
5. for i from 0 to $m_2 - 1$ do
   ADD $_{a(x)_i} = a_i$, ADD $_{b(x)_i} = b_i$.
6. NRHKOA$(c(x), a(x)_0^{2^{m_1}-1}, b(x)_0^{2^{m_1}-1}$,
   $t(x), ADD(a(x)), b(x), 2^{m_1}, m_2)$.
7. NRKOA$(c(x)_{2^{m_1}}^{2n-2}, a(x)_{2^{m_1}}^{n-1}, b(x)_{2^{m_1}}^{n-1}, m_2)$.
8. $c(x)_{2^{m_1}}^{2^{3m_1}-2} += (t(x)_0^{2^{2m_1}-2} + c(x)_0^{2^{2m_1}-2}$
   $+ c(x)_{2^{2m_1}}^{2n-2}) x^{2^{m_1}}$
9. Return(c(x)).

본 소절에서는 불필요한 연산이 없는 카라슈바 알고리즘을 제안한다. a(x)와 b(x)를 n-1차 다항식이라 하고 $n = 2^{m_1} + m_2$ 라 하면 다항식 c(x)=a(x)b(x)는 식

(3)과 같다.

$$
\begin{aligned}
c(x) &= a(x)\ b(x) \\
&= (a(x)_0^{2^{m_1-1}} + a(x)_{2^{m_1}}^{n-1} x^{2^{m_1}})(b(x)_0^{2^{m_1-1}} + b(x)_{2^{m_1}}^{n-1} x^{2^{m_1}}) \\
&= \frac{a(x)_0^{2^{m_1-1}} \cdot b(x)_0^{2^{m_1-1}}}{} + \frac{a(x)_{2^{m_1}}^{n-1} \cdot b(x)_{2^{m_1}}^{n-1} x^{2^{m_1+1}}}{} \\
&\quad + \Big[\frac{\{a(x)_0^{2^{m_1-1}} + a(x)_{2^{m_1}}^{n-1}\}\{b(x)_0^{2^{m_1-1}} + b(x)_{2^{m_1}}^{n-1}\}}{} \\
&\quad + a(x)_0^{2^{m_1-1}} \cdot b(x)_0^{2^{m_1-1}} + a(x)_{2^{m_1}}^{n-1} \cdot b(x)_{2^{m_1}}^{n-1} \Big] x^{2^{m_1}}
\end{aligned}
\tag{3}
$$

우리는 알고리즘 1의 step 5-6을 NRHKOA를 사용하여 계산한다. $a(x)_0^{2^{m_1-1}} \cdot b(x)_0^{2^{m_1-1}}$에서 사용된 연산을 이용하여 불필요한 연산을 수행하지 않기 때문에 NRHKOA는 식(3)의 $\{a(x)_0^{2^{m_1-1}} + a(x)_{2^{m_1}}^{n-1}\}\{b(x)_0^{2^{m_1-1}} + b(x)_{2^{m_1}}^{n-1}\}$을 효율적으로 계산한다.

Algorithm 3. 불필요한 연산이 없는 부분 카라슈바 알고리즘

NRHKOA$(c(x), a_1(x), b_1(x), a_2(x), b_2(x), m_1, m_2)$

INPUT : $a_1(x), b_1(x), a_2(x), b_2(x), m_1, m_2$.

OUTPUT : $c(x) = a_1(x) \cdot b_1(x), d(x) = a_2(x) \cdot b_2(x)$

1. If $m_1 = 4$ then
   4bitMul$(c(x), a_1(x), b_1(x), m_2, 1)$.
   4bitMul$(d(x), a_2(x), b_2(x), m_2, 0)$,
   Return(c(x),d(x)).
2. else
3. Set $f = \min(m_{1/2}, m_2), g = \max(m_2 - (m_{1/2}), 0)$
4. for i from 0 to $(m_1/2) - 1$ do
   $ADD_{a_1(x)_i} = a_{1,i} + a_{1,(m_1/2)+i}$,
   $ADD_{b_1(x)_i} = b_{1,i} + b_{1,(m_1/2)+i}$
5. for i from f to $(m_1/2) - 1$ do
   $ADD_{a_2(x)_i} = ADD_{a_1(x)_i}$,
   $ADD_{b_2(x)_i} = ADD_{b_1(x)_i}$
6. $NRHKOA(c(x), a_1(x)_0^{m_1/2-1}, b_1(x)_0^{m_1/2-1}, d(x), a_2(x)_0^{m_1/2-1}, b_2(x)_0^{m_1/2-1}, m_1/2, f)$.
7. $NRHKOA(u(x), ADD_{a_1(x)}, ADD_{b_1(x)}, v(x), ADD_{a_2(x)}, ADD_{b_2(x)}, m_1/2, f)$.
8. if $g = 0$ then
   $KOA(c(x)_{m_1}^{2m_1-2}, a_1(x)_{m_1/2}^{m_1-1}, b_1(x)_{m_1/2}^{m_1-1}, m_1/2)$
9. $d(x)_{m_1}^{2m_1-2} = c(x)_{m_1}^{2m_1-2}$
10. else
    $NRHKOA(c(x)_{m_1}^{2m_1-2}, a_1(x)_{m_1/2}^{m_1-1}, b_1(x)_{m_1/2}^{m_1-1}, d(x)_{m_1}^{2m_1-2}, a_2(x)_{m_1/2}^{m_1-1}, b_2(x)_{m_1/2}^{m_1-1}, m_1/2, g)$.
11. $c(x)_{m_1/2}^{3m_1/2-2} += u(x) + c(x)_0^{m_1-2} + c(x)_{m_1}^{2m_1-2}$
12. $d(x)_{m_1/2}^{3m_1/2-2} += v(x) + d(x)_0^{m_1-2} + d(x)_{m_1}^{2m_1-2}$
13. Return(c(x),d(x)).

차수를 $n = 2^{m_1} + 2^{m_1-1}$라고 가정하면 (3)에서 기존의 KOA는 $(a(x)_0^{2^{m_1-1}} + a(x)_{2^{m_1}}^{n-1})(b(x)_0^{2^{m_1-1}} + b(x)_{2^{m_1}}^{n-1})$와 $a(x)_0^{2^{m_1-1}} b(x)_0^{2^{m_1-1}}$를 같은 복잡도로 계산한다. 그러나 제안하는 알고리즘은 이진체의 차수 n의 정보를 이용하여 불필요한 연산을 수행하지 않는다.

알고리즘 2는 알고리즘 3을 보조 알고리즘으로 사용한다. [6]에서는 4차 이상에서 효율성이 있으나 제안하는 알고리즘은 일반적인 차수에 모두 적용 가능하므로

Algorithm 4. 4 비트 다항식 곱셈 알고리즘

4bitmul$(c(x), a(x), b(x), d(x), k, flag)$

INPUT : $a(x), b(x) \in GF(2^4), d(x), k, flag$

OUTPUT : $c(x) = a(x) \cdot b(x), d(x)$

1. If $flag = 1$ then
   $c(x) = a(x)_0^{k-1} \cdot b(x)_0^{k-1} + (a(x)_0^{k-1} \cdot b(x)_k^3 + a(x)_k^3 \cdot b(x)_0^{k-1}) x^k$.
2. $d(x) = a(x)_k^3 \cdot b(x)_k^3 x^{2k}$.
3. $c(x)_{2k}^6 += d(x)_{2k}^6$.
4. Else $c(x) = a(x)_0^{k-1} \cdot b(x)_0^{k-1} + (a(x)_0^{k-1} \cdot b(x)_k^3 + a(x)_k^3 \cdot b(x)_0^{k-1}) x^k$.
5. $c(x)_{2k}^6 += d(x)_{2k}^6$.
6. Return(c(x),d(x)).

7차 이상에서 일반적인 곱셈 방법보다 효율성을 가진다. 따라서 NRKOA의 step1에서 n을 7과 비교한다. 만약 n이 7보다 작은 경우 일반적인 곱셈을 수행하여 값을 반환하고 그렇지 않은 경우 나머지 단계를 수행한다. step 4와 step 5에서는 사전 덧셈연산을 수행하며 step 6에서는 NRHKOA를 이용하여 $c(x)_0^{2^{m_1+1}-2}$와 $t(x)$를 계산한다. 또한 값 m_2는 다항식 $ADD_{a(x)}$와 $ADD_{b(x)}$가 m_2개의 불필요하지 않은 계수를 가짐을 의미한다. NRKOA는 step 8에서 재귀 반복된다.

알고리즘 3은 알고리즘 2의 step 6의 계산에 사용된다. step 1에서 $m_1 = 4$임을 확인하여 알고리즘 4를 사용하거나 또는 나머지 단계를 수행한다. 알고리즘 3은 두 가지의 연산으로 구분된다. 첫 번째는 $c(x) = a_1(x) \cdot b_i(x)$의 계산이며 두 번째는 $d(x) = a_2(x) \cdot b_2(x)$의 계산이다. 알

고리즘 3에서 m_1은 언제나 짝수이므로 $a_1(x)\cdot b_1(x)$와 $a_2(x)\cdot b_2(x)$은 NRHKOA에 의하여 계산된다. step 3에서 f는 $a_1(x)_0^{m_1/2-1}$, $b_1(x)_0^{m_1/2-1}$, ADD $_{a_2(x)}$와 ADD $_{b_2(x)}$가 f개의 불필요하지 않은 계수를 가짐을 의미하며, g는 $a_1(x)_{m_1/2}^{m_1-1}$와 $b_1(x)_{m_1/2}^{m_1-1}$가 g개의 불필요하지 않은 계수를 가짐을 의미한다. 즉,

$$a_2(x)_f^{m_1/2-1}=a_1(x)_f^{m_1/2-1},\quad b_2(x)_f^{m_1/2-1}=b_1(x)_f^{m_1/2-1}$$
$$ADD_{a_2(x)_f^{m_1/2-1}}=ADD_{a_1(x)_f^{m_1/2-1}},\quad ADD_{b_2(x)_f^{m_1/2-1}}=ADD_{b_1(x)_f^{m_1/2-1}},$$
$$a_2(x)_g^{m_1-1}=a_1(x)_g^{m_1-1},\qquad b_2(x)_g^{m_1-1}=b_1(x)_g^{m_1-1}$$

이다. step 8에서 g=0이며 알고리즘 3에서 $d(x)_{m_1}^{2m_1-2}$을 계산하지 않으며 그렇지 않은 경우 step 10의 $d(x)_{m_1}^{2m_1-2}=a_2(x)_{m_1/2}^{m_1-1}\cdot b_2(x)_{m_1/2}^{m_1-1}$에서 계산된다.

알고리즘 4는 4-비트 다항식 곱셈 알고리즘으로 만약 step 1에서 flag=1이면 a(x)와 b(x)는 4비트 다항식이다. 그렇지 않은 경우 a(x)와 b(x)는 k개의 필요한 계수를 가지며 알고리즘 4는 $d(x)=a(x)_k^3\cdot b(x)_k^3 x^{2k}$를 계산하지 않는다. 알고리즘 4의 자세한 구조는 그림 1과 같다.

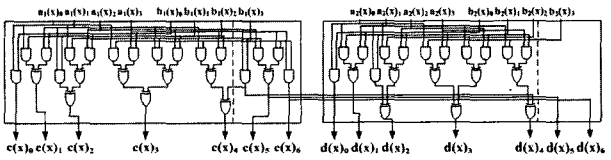

그림 1. 4비트 다항식 곱셈기의 구조(알고리즘 4)
Fig. 1. Architecture of proposed 4-bit multiplier.

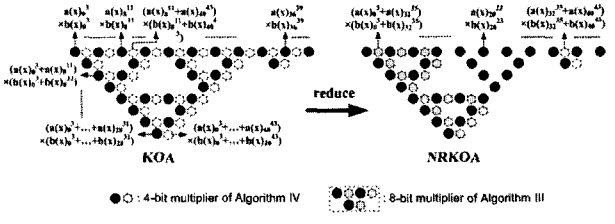

그림 2. GF(2^{44})에서의 NRKOA 곱셈기의 구조
Fig. 2. Architecture of the NRKOA multiplier over GF(2^{44}).

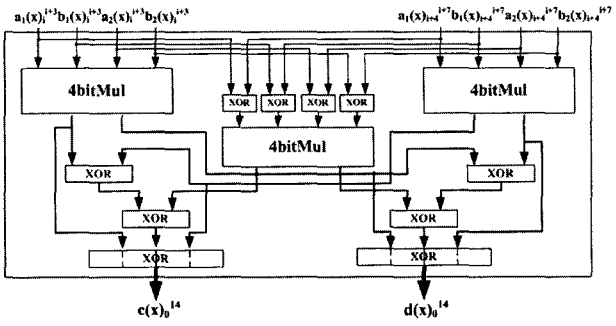

그림 3. 8 비트 다항식 곱셈기의 구조
Fig. 3. Architecture of 8-bit NRHKOA multiplier.

그림 2는 n=44인 경우 KOA와 NRKOA의 연산과정을 나타낸다. 그림 2에서 KOA는 61개의 4-비트 곱셈기를 사용하지만 제안하는 NRKOA는 51개의 4-비트 곱셈기를 사용한다. 따라서 NRKOA는 기존의 KOA보다 공간 복잡도에서 효율성을 가진다. 그림 3은 알고리즘 3에서 두 개의 곱셈 와 의 구조를 나타낸다.

Ⅳ. 효율성 비교

본 절에서는 NRKOA 곱셈기의 효율성을 비교한다. 알고리즘 3의 step 9에서 $d(x)_{m_1}^{2m_1-2}$는 $c(x)_{m_1}^{2m_1-2}$에 의하여 결정되므로 NRHKOA는 어떠한 계산도 수행하지 않는다. 이와 같은 특성을 이용하여 간단하게 효율성을 기술하면 다음과 같다. 복잡도의 기술을 위하여 다음과 같은 몇 가지 기호를 정의한다.

- #KOA(u) : u-비트 KOA 곱셈기의 복잡도
- #NRKOA(u) : u-비트 NRKOA 곱셈기의 복잡도
- #NRKOA(u) : u-비트 NRHKOA 곱셈기의 복잡도(v는 불필요하지 않은 계수의 수)

만약 $n=2^{k_1}+m_2=m_1+n_2$,이면 NRKOA의 복잡도는 $\#NRHKOA(m_1,m_2)+\#NRKOA(m_2)+6m_1+2n_1-4$이고 이때 $6m_1+2m_2-4$는 추가되는 XOR 연산이다. 알고리즘 3에서 NRHKOA가 한번 반복되었을 때의 복잡도는 다음과 같다.

$$\begin{aligned}\#NRKOA(n)&=\#NRKOA(m_2)+\#NRHKOA(m_1,m_2)+\delta_1,\\&=\#NRKOA(m_2)+\underline{2\cdot\#NRHKOA(m_1/2,f_1)},\\&\quad+\underline{\#KOA(m_1/2)+\delta_1+\delta_2}\end{aligned}$$

$$\begin{aligned}\#KOA(n)&=\#KOA(m_2)+2\cdot\#KOA(m_1)+\delta_1,\\&=\#KOA(m_2)+\underline{4\cdot\#KOA(m_1/2)}\\&\quad+\underline{2\#KOA(m_1/2)+\delta_1+\delta_2},\end{aligned}$$

이때 $g_1=0$이고 δ는 추가되는 XOR 연산이다. 만약 NRHKOA가 두 번 반복되며 $g_1=0, g_2=0$이면

$$\begin{aligned}\#NRKOA(n)&=\#KOA(n)-\#KOA(m_1/2)\\&\quad-2\cdot\#KOA(m_1/4).\end{aligned}$$

만약 $n=2^k+1$인 경우 기존의 KOA보다 대략 43% 정도의 공간복잡도가 감소하며 이때가 가장 좋은 효율성을 가진다. 또한 $n=2^3-1$인 경우 가장 낮은 효율성

표 1. GF(2^n)에서 NRKOA와 기존 방법의 효율성 비교.
Table 1. Comparing the complexity of parallel multiplier over GF(2^n) between SB, KOA, and NRKOA.

Multiplication Method		n=113	n=131	n=163
SB	#Gate	25,313	34,061	53,813
	Delay	$7T_X+T_A$	$8T_X+T_A$	$8T_X+T_A$
KOA	#Gate	11,648	25,078	27,785
	Delay	$17T_X+T_A$	$20T_X+T_A$	$20T_X+T_A$
NRKOA	#Gate	11,138	15,939	21,791
	Delay	$17T_X+T_A$	$20T_X+T_A$	$20T_X+T_A$
Reduce		4.38%	36.45%	21.58%

Multiplication Method		n=113	n=131	n=163
SB	#Gate	74,113	159,613	650,941
	Delay	$8T_X+T_A$	$9T_X+T_A$	$10T_X+T_A$
KOA	#Gate	33,381	78,028	237,848
	Delay	$20T_X+T_A$	$23T_X+T_A$	$26T_X+T_A$
NRKOA	#Gate	27,803	58,828	159,090
	Delay	$20T_X+T_A$	$23T_X+T_A$	$26T_X+T_A$
Reduce		16.72%	32.3%	33.12%

을 보이며 단지 1개의 게이트만 감소한다. 표 1은 기존 방법의 병렬 곱셈기와 제안하는 알고리즘으로 구성된 병렬 곱셈기의 효율성을 보인다. 표의 결과에 의하면 기존의 KOA와 같은 시간 복잡도를 가지지만 공간 복잡도는 감소하며 제안하는 곱셈기의 효율성은 n의 헤밍 웨이트에 의존한다.

V. 결 론

본 논문에서는 카라슈바 알고리즘에서 불필요한 연산을 제거한 불필요한 연산이 없는 카라슈바 알고리즘과 병렬 하드웨어 구조를 제안하였다. 제안하는 알고리즘은 기존의 KOA와 같은 시간 복잡도를 가지지만 공간 복잡도에서 효율성을 가진다. 제안하는 알고리즘을 적용한 경우 최대 43%까지 공간 복잡도를 줄일 수 있으며 ECC와 같이 소수 차수를 가지는 확장체에서 더욱 효율적이다. 이러한 특징은 낮은 공간 복잡도가 요구되는 스마트카드, 모바일 디바이스 등에 효율적으로 적용된다.

참 고 문 헌

[1] 장남수, 한동국, 정석원, 김창한, "유한체 GF(2^n)에서 낮은 공간 복잡도를 가지는 새로운 다중 분할 카라슈바 방법의 병렬처리 곱셈기", 대한전자공학회논문지(SC), 41. 1, pp.33-40, 2004.

[2] 장남수, 김창한, "확장체 GF(p^n)에서 효율적인 다항식 곱셈 방법", 대한전자공학회논문지(SD), 42. 5, pp.23-30, 2005.

[3] G. Drolet, "A New Representation of Elements of Finite Fields $GF(2^m)$ Yielding Small Complexity Arithmetic circuit}, IEEE Trans. on Computers, vol 47, 1998, 353-356.

[4] M. Ernst, M. Jung, F. Madlener, S. Huss, and R. Blümel, "A Reconfigurable System on Chip Implementation for Elliptic Curve Cryptography over $GF(2^n)$", In Work shop on Cryptographic Hardware and Embedded Systems (CHES'02), LNCS2523, (2002), 381-399.

[5] N. Koblitz, "Elliptic Curve Cryptosystems", Mathematics of Computation, vol. 48, 1987, 203-209

[6] M. Leone, "A New Low Complexity Parallel Multiplier for a Class of Finite Fields", In Workshop on Cryptographic Hardware and Embedded Systems (CHES'01), LNCS2162, (2001), 160-170.

[7] C. Paar, "A new architecture for a parallel finite fields multiplier with Low Complexity Based on Composite Fields", IEEE Trans. on Computers, vol45, no. 7, ,July 1996, 846-861.

[8] F. Rodriguez-Henriquez and C. K. Koc, "On fully parallel Karatsuba multipliers for GF(2^m)", Proceedings of the international Conference on Computer Science and Technology -CST 2003, Acta Press, Cancun, Mexico, May 2003, 405-410.

저 자 소 개

장 남 수(학생회원)
2002년 2월 서울시립대학교 수학과 학사.
2004년 8월 고려대학교 정보보호 대학원 석사.
2005년 2월~현재 고려대학교 정보보호대학원 박사과정.
<주관심분야 : 공개키 암호, 암호칩 설계 기술, 부채널 공격 방법론>

김 창 한(정회원)-교신저자
1985년 2월 고려대학교 수학과 학사.
1987년 2월 고려대학교 수학과 석사.
1992년 2월 고려대학교 수학과 박사.
1992년 3월 세명대학교 정보보호학과 교수
<주관심분야 : 정수론, 공개키 암호, 암호 프로토콜>