



(19)대한민국특허청(KR)
(12) 공개특허공보(A)

(51) 。 Int. Cl. (11) 공개번호 10-2007-0056743
G06F 7/496 (2006.01) (43) 공개일자 2007년06월04일

(21) 출원번호 10-2005-0115773
(22) 출원일자 2005년11월30일
심사청구일자 2005년11월30일

(71) 출원인 고려대학교 산학협력단
서울 성북구 안암동5가1 고려대학교 내

(72) 발명자 임종인
서울 종로구 내수동 95 삼성파크텔리스 101-901
김창한
서울 서초구 반포동 20-9 주공아파트 328동 402호
박영호
서울 광진구 광장동 576 현대9차아파트 902동 1203호
장남수
서울 마포구 성산2동 200-84
김태현
서울 강서구 화곡2동 846-19
임대성
경기 안산시 상록구 이동 530-14 301호
지성연
경기 수원시 영통구 매탄2동 111-97
김성경
서울 성북구 안암동5가 11-5번지 리베르타 302호
윤택영
경기 성남시 중원구 성남동 3020번지

(74) 대리인 현종철

전체 청구항 수 : 총 4 항

(54) 불필요한 연산을 줄이도록 하는 카라슈바 곱셈 방법

(57) 요약

본 발명은 카라슈바 곱셈기에 관한 것으로, 특히 기존의 이진체에서 정의된 타원곡선의 카라슈바 병렬 곱셈기가 가지는 불필요한 연산을 제거함으로써 공간 복잡도와 시간 복잡도를 줄이기 위해, 기존의 카라슈바 알고리즘에 NRKOA(Non-Redundant Karatsuba-Ofman Algorithm) 알고리즘을 결합함으로써, 불필요한 연산을 줄이도록 하는 카라슈바 곱셈 방법을 제공하는데 그 목적이 있다. 따라서 공간 복잡도에서 효율성을 가지는 효과가 있으며, 이로 인해 낮은 공간 복잡도가 요구되는 스마트 카드, 모바일 디바이스 등에 효율적으로 적용될 수 있는 효과가 있다.

대표도

도 3

특허청구의 범위

청구항 1.

불필요한 연산을 줄이도록 하는 카라슈바 곱셈 방법에 있어서,

입력되는 값의 차수를 확인하는 제 1 단계;

상기 제 1 단계에서 차수가 7 이상이면, 입력값($c(x), a(x), b(x), n$) 및 연산을 위한 계수에 관한 초기화를 수행하는 제 2 단계;

상기 제 2 단계의 초기화가 완료되면, 상기 입력값에 대한 사전 덧셈 연산을 수행하여 상기 각각의 입력값에 대한 합을 구하는 제 3 단계;

상기 입력값에 대한 합을 구한 후, NRHKOA(Non-Redundant Karatsuba-Ofman Algorithm) 연산을 수행하여

$c(x)_{2^{m_1}}^{2n-2}, a(x)_{2^{m_1}}^{n-1}, b(x)_{2^{m_1}}^{n-1}, m_2$ 의 값을 구하는 제4 단계; 및

상기 제 1 단계 내지 제 4 단계를 통해 얻어진 $c(x)_{2^{m_1}}^{2n-2}, a(x)_{2^{m_1}}^{n-1}, b(x)_{2^{m_1}}^{n-1}, m_2$ 을 입력값으로 대치하여 상기 제 2 단계로 진행하여 상기 제 2 단계 이하의 과정을 재귀반복함으로써 출력을 구하는 제 5 단계를 포함하는 불필요한 연산을 줄이도록 하는 카라슈바 곱셈 방법.

청구항 2.

제 1 항에 있어서,

상기 제 5단계의 출력은 <수학식 13>과 같음을 특징으로 하는 불필요한 연산을 줄이도록 하는 카라슈바 곱셈 방법.

$$c(x)_{2^{m_1}}^{2^{3m_1}-2} = (t(x)_0^{2^{2m_1}-2} + c(x)_0^{2^{2m_1}-2} + c(x)_{2^{m_1}}^{2n-2})x^{2^{m_1}}$$

청구항 3.

제 1 항 또는 제 2 항에 있어서,

상기 제 3 단계의 입력값에 대한 합은, <수학식 14>와 <수학식 15>를 통해 이루어지는 것을 특징으로 하는 불필요한 연산을 줄이도록 하는 카라슈바 곱셈 방법.

$$ADD_{a(x)_i} = a_i + a_{2^{m_1+1}}, ADD_{b(x)_i} = b_i + b_{2^{m_1+1}},$$

$$ADD_{a(x)_i} = a_i, ADD_{b(x)_i} = b_i,$$

여기서, i 는 0부터 m_2-1 까지의 값으로 변화됨.

청구항 4.

제 3 항에 있어서,

상기 제 4 단계의 NRHKOA 연산은, <표 7>을 통해 그 결과를 제공하는 것을 특징으로 하는 불필요한 연산을 줄이도록 하는 카라슈바 곱셈 방법.

[표 7]

$NRHKOA(c(x), a_1(x), b_1(x), a_2(x), b_2(x), m_1, m_2)$
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. $NRHKOA(u(x), ADD_{a_1(x)}, ADD_{b_1(x)}, v(x), ADD_{a_2(x)}, ADD_{b_2(x)}, m_1/2, g).$
8. if $g = 0$ then
$KOA(c(x)^{2m_1-2}, a_1(x)^{m_1-1}, b_1(x)^{m_1-1}, m_1/2)$
9. $d(x)^{2m_1-2} = c(x)^{2m_1-2}$
10. else
$NRHKOA(c(x)^{2m_1-2}, a_1(x)^{m_1-1}, b_1(x)^{m_1-1}, d(x)^{2m_1-2}, a_2(x)^{m_1-1}, b_2(x)^{m_1-1}, m_1/2, g).$
11. $c(x)^{3m_1/2-2} = u(x) + c(x)_0^{m_1-2} + c(x)^{2m_1-2}$
12. $d(x)^{3m_1/2-2} = v(x) + d(x)_0^{m_1-2} + d(x)^{2m_1-2}$
13. Return($c(x), d(x)$).

명세서

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

본 발명은 공개키 암호화를 위한 빠른 유한체 연산을 제공하는 카라슈바(Karatsuba) 곱셈기에 관한 것으로, 특히 기존의 이진체에서 정의된 타원곡선의 카라슈바 병렬 곱셈기가 가지는 불필요한 연산을 제거함으로써 공간 복잡도와 시간 복잡도를 줄이는 방법에 관한 것이다.

공개키 암호시스템의 안전성은 단방향 함수(one-way function)에 근거를 두고 있는데, 단방향 함수란 한쪽 방향으로의 계산은 쉬운 반면, 반대 방향의 계산은 매우 힘든 함수를 말한다. 예를 들어, RSA(Rivest, Shamir, Adleman) 암호법의 기반이 되는 단방향 함수는 정수의 소인수분해 문제로서, 매우 큰 두 소수 p 와 q 의 곱 $n = pq$ 이 주어졌을 때 이를 다시 p 와 q 로 소인수 분해하기 힘들다는 사실을 가정하고 있다.

기존의 공개키 암호시스템에서 사용하던 단방향 함수는 여러 가지 공격방법이 연구되어 개인키를 모르고도 암호문을 해독할 수 있게 되었다. 이러한 공격을 막기 위한 대책은 아직까지는 키의 길이를 증가시키는 방법이 유일하며, 현재는 1024 비트(bit) 정도가 되어야 안전한 것으로 알려져 있다. 그러나 키의 길이를 계속해서 증가시키면 그만큼 키를 관리하는데 어려움이 커지고 암호화나 복호화에 걸리는 계산 시간이 늘어나는 단점이 있다. 이로 인해 공개키 암호시스템의 개념적 장점에도 불구하고 다양한 용도에 활용되지 못하고 있다. 특히 스마트카드(smart card), 이동 단말 등과 같이 기억 용량과 계산 능력이 부족한 환경에서는 공개키 암호시스템을 사용하기 힘들다.

이상과 같은 문제점을 극복하기 위해서 다양한 단방향 함수들이 제안되어, 새로운 공개키 암호시스템을 만들기 위한 연구가 많이 진행되어 왔지만, 대부분 안전성이나 실용성을 인정받지 못했다. 이 중 타원 곡선상에서의 이산 대수 문제를 기반으로 하는 타원곡선 암호시스템은 높은 보안성과 다양한 특성으로 인해서 최근까지 많은 연구의 대상이 되었다.

이산 대수 문제를 기반으로 하는 타원 곡선 암호시스템(Elliptic Curve Cryptosystem: ECC)에서 암호화를 처리하기 위해서는, 이산 대수를 위한 유한체 연산을 수행하게 되는데, 공개키 암호시스템을 구성하는 경우 이러한 유한체 연산의 성능은 암호 시스템 전체의 성능에 큰 영향을 미치게 된다.

이와 같은 유한체 연산은 두 다항식의 곱셈 연산으로 표현되는데, 이러한 유한체 연산의 효율적인 곱셈 연산은 시그널 프로세싱, 암호학 및 부호 이론의 중요한 부분이 된다.

일반적으로 낮은 공간 복잡도를 요구하는 어플리케이션에서 공간 복잡도를 감소시키는 것은 시스템을 디자인하는 측면에서 매우 중요한 관점이다. 이에 따라, "Leone"은 낮은 공간 복잡도를 가지는 병렬 곱셈기를 제안하였다. 이는 기존의 전형적인 병렬 곱셈기와 다르게 시간 복잡도와 공간 복잡도의 절충(trade-off)을 통하여, 상대적으로 낮은 공간복잡도를 제공할 수 있게 된다. 한편, "Ernst"는 분할 해결법(divide-and-conquer)인 카라슈바 방법을 적용한 일반적이고 확장 가능한 유한체 연산 장치를 제안하였다.

이하에서는 다음과 같은 기호를 사용한다. $a(x)$ 를 $GF(2^n)$ 의 원소라 가정하면 $(j-i)$ 차 다항식 $a(x)^j_i$ 는 $a(x)^j_i = a_i + a_{i+1}x + \dots + a_{i+j-1}x^{j-1}$ 이며 $(j-i)$ 가 음수인 경우 $a(x)^j_i$ 는 "0" 이다.

일반적인 유한체 곱셈 방법은 다음과 같다.

n 차 기약 다항식 $f(x)$ 에 의하여 생성된 유한체 $GF(2^n)$ 의 원소간의 곱셈에 관해 설명한다. 다항식 기저에 의해 $GF(2^n)$ 의 원소 $a(x)$ 와 $b(x)$ 는

$$a(x) = a_0 + a_1x + \dots + a_{n-2}x^{n-2} + a_{n-1}x^{n-1},$$

$$b(x) = b_0 + b_1x + \dots + b_{n-2}x^{n-2} + b_{n-1}x^{n-1}$$

와 같이 표현되며 이때 $a_i, b_i \in GF(2)$ 이다.

<수학식 1>과 같은 이진체 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 $c(x) = c_0 + c_1x + \dots + c_{2n-3}x^{2n-3} + c_{2n-2}x^{2n-2}$, $c_i \in GF(2)$ 를 n 차 기약 다항식 $f(x)$ 로 나눈 나머지를 구하는 과정인 모듈로 축소(Modulo Reduction)로 나뉜다.

이러한 연산을 위한 병렬 곱셈기는 AND 게이트와 XOR 게이트들로 구성되는데, AND, XOR, 전체 게이트의 수를 각각 #AND, #XOR, #TOT라 하고 전체 시간 지연을 T_{TOT} 라 할 때 일반적인 곱셈의 복잡도는 다음 <수학식 2>와 같다.

$$\begin{aligned}\# \text{AND} &= n^2, \\ \# \text{XOR} &= (n-1)^2, \\ \# \text{TOT} &= 2n^2 - 2n + 1, \\ T_{TOT} &= T_A + \lceil \log_2 n \rceil T_X.\end{aligned}$$

이러한 기본적인 유한체 연산에 대한 지식을 통해, 기존의 이진체에서 정의된 타원곡선의 카라슈바 곱셈 과정을 살펴보면, 다음<표 1>과 같다.

[표 1]

$KOA(c(x), a(x), b(x), n)$
INPUT : $a(x), b(x) \in GF(2^n), n = 2^k$
OUTPUT : $c(x) = a(x) \cdot b(x)$
1. if $n \leq 4$ 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. $KOA(c(x)_0^{2n-2}, a(x)_{n/2}^{n-1}, b(x)_{n/2}^{n-1}, n/2)$.
7. $c(x)_{n/2}^{3n/2-2} = u(x) + c(x)_0^{n-2} + c(x)_{n/2}^{2n-2}$.
8.. Return($c(x)$).

$GF(2^n)$ 에서 기본적인 KO(Karatsuba-Ofman) 방법은 유한체의 원소를 2중으로 분할하여 분할 해결법(divide-and-conquer)을 적용한다. 따라서 $n/2$ 이 짝수라면 KO는 반복 적용될 수 있다. 즉, $n-1$ 차의 다항식에 카라슈바 방법을 직접 적용하면 $\log_2 n$ 번 반복 수행이 가능하다.

<표 1>의 동작을 좀 더 살펴보면, 입력값에 대한 초기화를 1, 2과정을 통해 진행하며, 분할 과정을 3 과정을 통해 진행한다. 그리고 4 과정을 통해 $n/2$ 비트 다항식 곱셈기의 입력값의 연산을 수행하며, 제 5 과정을 통해 제 4 과정의 입력값 연산에 따른 나머지 값에 대한 연산을 수행한다. 그리고 제 6 과정을 통해 분할된 나머지 $n/2$ 비트에 대한 연산을 수행한다. 제 4 과정 내지 제 6 과정의 결과를 이용하여 제 7 과정을 수행함으로써, $n/2$ 비트 다항식 곱셈기의 출력을 얻을 수 있게 된다.

특히, <표 1>의 카라슈바 알고리즘은 복잡도를 고려할 때 XOR 연산이 두 과정으로 나누어지게 된다.

첫째는 제4 과정에서의 $n/2$ 비트 다항식 곱셈기의 입력 값의 연산이고, 두 번째는 제 7 과정에서의 $n/2$ 비트 다항식 곱셈기의 출력 값의 연산이다.

만약 카라슈바 알고리즘을 1번 반복 수행한다면 제 4 과정에서 $2(n/2)$ 개의 게이트가 필요하고 제7 과정에서 $(3n-4)$ 개의 게이트가 필요하다.

"Leone"의 논문에서는 KOA의 최적 반복회수를 $n/2^k=4$ 로 제안하였다. 따라서 KOA가 m 번의 반복 연산을 수행하며 최하위 연산기를 SB 곱셈기를 사용한다고 가정할 때 전체 연산량은 다음의 <수학식 3>과 <수학식 4>와 같다.

$$\begin{aligned}\#AND &= (3/4)^m \cdot n^2, \\ \#XOR &= (n^2/4^m + 6n/2^m - 1)3^m - (8n - 2), \\ TOT &= (2n^2/4^m + 6n/2^m - 1)3^m - (8n - 2),\end{aligned}$$

$$T_{TOT} = 3 \cdot \lceil \log_2 2^m \rceil + \lceil \log_2 (n/2^m) \rceil T_X + T_A = (2 \cdot m + \lceil \log_2 n \rceil) T_X + T_A.$$

위의 <수학식 3>과 <수학식 4>의 결과에 따르면, KO 방법을 이용한 병렬 처리 곱셈기는 SB 방법을 이용한 병렬 처리 곱셈기 보다 공간 복잡도가 낮게 된다.

그러나, 이와 같은 기존의 이진체에서 정의된 타원곡선의 카라슈바 곱셈 방법은 불필요한 연산을 포함하고 있다. 이는 이진체의 위수가 소수이기 때문이다. 따라서, 이와 같은 불필요한 연산을 줄이도록 하는 새로운 카라슈바 곱셈 방법에 대한 연구가 필요하다.

발명이 이루고자 하는 기술적 과제

상기와 같은 문제점을 해결하기 위한 본 발명은, 공간 복잡도와 시간 복잡도를 줄임으로써 불필요한 연산을 줄이는 암호화가 가능한 카라슈바 곱셈 방법을 제공하는데 그 목적이 있다.

발명의 구성

상기의 목적을 달성하기 위한 본 발명은, 불필요한 연산을 줄이도록 하는 카라슈바 곱셈 방법에 있어서, 입력되는 값의 차수를 확인하는 제 1 단계; 상기 제 1 단계에서 차수가 7 이상이면, 입력값($c(x), a(x), b(x), n$) 및 연산을 위한 계수에 관한 초기화를 수행하는 제 2 단계; 상기 제 2 단계의 초기화가 완료되면, 상기 입력값에 대한 사전 덧셈 연산을 수행하여 상기 각각의 입력값에 대한 합을 구하는 제 3 단계; 상기 입력값에 대한 합을 구한 후, NRHKO(Non-Redundant

Karatsuba-Ofman Algorithm) 연산을 수행하여 $c(x)_{2^{m_1}}^{2n-2}, a(x)_{2^{m_1}}^{n-1}, b(x)_{2^{m_1}}^{n-1}, m_2$ 의 값을 구하는 제4 단계; 및

상기 제 1 단계 내지 제 4 단계를 통해 얻어진 $c(x)_{2^{m_1}}^{2n-2}, a(x)_{2^{m_1}}^{n-1}, b(x)_{2^{m_1}}^{n-1}, m_2$ 을 입력값으로 대치하여 상기 제 2 단계로 진행하여 재귀반복함으로써 출력을 구하는 제 5 단계를 포함한다.

이하, 첨부된 도면을 참조하여 본 발명에 따른 바람직한 실시예를 상세히 설명한다. 또한, 본 발명을 설명함에 있어서, 관련된 공지기능 혹은 구성에 대한 구체적인 설명이 본 발명의 요지를 불필요하게 흐릴 수 있다고 판단되는 경우 그 상세한 설명은 생략한다.

종래의 카라슈바 알고리즘과는 상이한 본 발명에서 제안하는 알고리즘을 살펴보면 다음과 같다.

GF(2^n)에서 $|n|$ 을 n 의 비트 길이라고 하자. $a(x)$ 와 $b(x)$ 를 $n-1$ 차 다항식이라 하고, $n=2^m+t$ 라 하면 다항식 $a(x)$ 와 $b(x)$ 는 다음의 <수학식 5>와 같은 두 부분으로 구분된다.

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

따라서 A_1 과 B_1 은 $a(x)$ 와 $b(x)$ 의 하위 부분이며 A_2 과 B_2 는 상위 부분이다. 위의 <수학식 5>를 이용하여 $c(x)$ 를 전개하면 다음과 같다.

$$c(x) = a(x) \cdot b(x) = (A_1 + A_2 x^{2^n}) \cdot (B_1 + B_2 x^{2^n}) \\ = ((A_1 + A_2) \cdot (B_1 + B_2) + A_1 B_1 + A_2 B_2) x^{2^n} + A_1 B_1 + A_2 B_2 x^{2^{2n}}$$

KOA(Karatsuba-Ofman Algorithm)는 위의 식과 같이 31번의 다항식 곱셈으로 $c(x)$ 를 계산한다. 만약 t 가 0이 아니면 $(A_1 + A_2)(B_1 + B_2)$ 과 $A_1 B_1$ 은 2^m -비트 다항식 곱셈이고 $A_2 B_2$ 는 t -비트 다항식 곱셈이다. 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 $GF(2^n)$ 의 원소라고 하면 식 (1)에서 $(A_1 + A_2)(B_1 + B_2)$ 은 다음과 같다.

$$(A_1 + A_2) \cdot (B_1 + B_2) = (a_0 + a_2) + a_1 x \cdot (b_0 + b_2) + b_1 x \\ = (a_0 + a_2)(b_0 + b_2) + a_1 b_1 x^2 + (a_0 + a_1 + a_2)(b_0 + b_1 + b_2) + (a_0 + a_2)(b_0 + b_2) + a_1 b_1 x$$

<수학식 6>의 $A_1 B_1$ 에서 $a_1 b_1$ 은 이미 계산되었으므로 <수학식 7>의 $a_1 b_1$ 은 불필요한 연산이다. 따라서, 본 발명에서 제안하는 알고리즘은 이와 같은 불필요한 연산을 사용하지 않는 것이다. 즉, 불필요한 연산이 없는 부분 카라슈바 알고리즘을 이용하여 <수학식 6>의 $(A_1 + A_2)(B_1 + B_2)$ 에서 발생하는 모든 불필요한 연산을 계산하지 않는다.

본 발명의 실시예에서는 불필요한 연산이 없는 카라슈바 알고리즘을 제안하는데, 이를 위하여 $a(x)$ 와 $b(x)$ 를 $n-1$ 차 다항식이라 하고 $n=2^{m_1} + m_2$ 라 하면 다항식 $c(x)=a(x)b(x)$ 는 다음 <수학식 8>과 같다.

$$c(x) = a(x) \cdot b(x) = (a(x)_0^{2^{m_1}-1} + a(x)_{2^{m_1}}^{n-1} \cdot x^{2^{m_1}}) (b(x)_0^{2^{m_1}-1} + b(x)_{2^{m_1}}^{n-1} \cdot x^{2^{m_1}}) \\ = \frac{a(x)_0^{2^{m_1}-1} \cdot b(x)_0^{2^{m_1}-1} + a(x)_{2^{m_1}}^{n-1} \cdot b(x)_{2^{m_1}}^{n-1} \cdot x^{2^{2m_1}}}{+ [\frac{a(x)_0^{2^{m_1}-1} + a(x)_{2^{m_1}}^{n-1} \cdot b(x)_0^{2^{m_1}-1} + b(x)_{2^{m_1}}^{n-1} \cdot a(x)_0^{2^{m_1}-1} + a(x)_0^{2^{m_1}-1} \cdot b(x)_{2^{m_1}}^{n-1} + a(x)_{2^{m_1}}^{n-1} \cdot b(x)_0^{2^{m_1}-1} + a(x)_0^{2^{m_1}-1} \cdot b(x)_{2^{m_1}}^{n-1}] x^{2^{m_1}}}$$

이상과 같은 본 발명의 실시예에서는 <표 1>의 카라슈바 알고리즘의 제 5 과정과 제 6 과정을 NRHKOA 방법으로 대체하

여 연산함으로써 불필요한 연산을 수행하지 않도록 한다. 즉, $a(x)_0^{2^{m_1}-1} \cdot b(x)_0^{2^{m_1}-1}$ 에서 사용된 연산을 이용하여

불필요한 연산을 수행하지 않기 때문에 NRHKOA 방법은 <수학식 8>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 계산하게 된다.

차수를 $n=2^{m_1} + 2^{m_1-1}$ 라고 가정하면 <수학식 8>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. 그

러나 본 발명의 실시예에서 제안하는 불필요한 연산이 없는 카라슈바 알고리즘은 이진체의 차수 n 의 정보를 이용하여 불필요한 연산을 수행하지 않는다.

본 발명의 실시예에 따른 불필요한 연산이 없는 카라슈바 알고리즘은 <표 2>와 같다.

[표 2]

$NRKOA(c(x), a(x), b(x), n)$
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
$ADD_{a(x)} = a_i + a_{2^{m_1} + i}$, $ADD_{b(x)_i} = b_i + b_{2^{m_1} + i}$.
5. for i from 0 to $m_2 - 1$ do
$ADD_{a(x)} = a_i$, $ADD_{b(x)_i} = b_i$.
6. $NRHKO(A(c(x), a(x)_{2^{m_1}-1}^{2^{m_1}-1}, b(x)_{2^{m_1}-1}^{2^{m_1}-1}, t(x), ADD(a(x)), b(x), 2^{m_1}, m_2)$.
7. $NRKOA(c(x)_{2^{m_1}-2}^{2^{m_1}-2}, a(x)_{2^{m_1}-1}^{2^{m_1}-1}, b(x)_{2^{m_1}-1}^{2^{m_1}-1}, m_2)$.
8. $c(x)_{2^{m_1}-2}^{2^{m_1}-2} = (t(x)_{2^{m_1}-2}^{2^{m_1}-2} + c(x)_{2^{m_1}-2}^{2^{m_1}-2} + c(x)_{2^{m_1}-2}^{2^{m_1}-2})x^{2^{m_1}}$.
9. Return($c(x)$).

<표 2>의 동작을 좀 더 살펴보면, <표 1>에서의 입력값에 대한 초기화를 1, 2, 3 과정을 통해 진행하며, 4, 5 과정을 ADD 값을 특정한다.

좀 더 상세히는 "Leone"의 논문에서는 4차 이상에서 효율성이 있으나 본 발명에서 제안하는 알고리즘은 일반적인 차수에 모두 적용 가능하므로 7차 이상에서 일반적인 곱셈 방법보다 효율성을 가진다. 따라서 <표 2>의 제 1 과정에서 n 을 "7"과 비교한다. 즉, n 이 7보다 작은 경우는 본 발명에서 제안하는 방법이 적용되지 않으며, n 이 7보다 큰 경우에만 나머지 단계로의 진행이 실행된다.

그리고 제 4 과정과 제 5 과정은 사전 덧셈 연산을 수행하여 각각의 입력값에 대한 합을 구한 것이다. 그리고 제 6 과정을

통해 $NRHKO(A$ 연산을 수행하여 $c(x)_0^{2^{m_1+1}-2}$ 와 $t(x)$ 를 계산한다. 또한, 제 6 과정에서 m_2 는 다항식 $ADD_{a(x)}$ 와 $ADD_{b(x)}$ 가 불필요하지 않은 m_2 개의 계수를 가짐을 의미한다.

그리고 제 6 과정에서 $NRHKO(A$ 연산이 종료되면, 제 7 과정에서 $NRKOA$ 연산을 수행하고, 이를 제 8 과정의

$c(x)_{2^{m_1}-2}^{2^{3m_1}-2} = (t(x)_0^{2^{2m_1}-2} + c(x)_0^{2^{2m_1}-2} + c(x)_{2^{2m_1}}^{2^{2m_1}-2})x^{2^{m_1}}$ 을 통해 재귀반복함으로써 $c(x)$ 를 출력한다.

여기서, <표 2>의 불필요한 연산이 필요없는 카라슈바 알고리즘은 <표 3>의 부분 알고리즘을 보조 알고리즘으로 사용한다.

[표 3]

$NRHKOA(c(x), a_1(x), b_1(x), a_2(x), b_2(x), m_1, m_2)$
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. $NRHKOA(u(x), ADD_{a_1(x)}, ADD_{b_1(x)}, v(x), ADD_{a_2(x)}, ADD_{b_2(x)}, m_1/2, g).$
8. if $g = 0$ then
$KOA(c(x)^{2m_1-2}, a_1(x)_{m_1/2}^{m_1-1}, b_1(x)_{m_1/2}^{m_1-1}, m_1/2)$
9. $d(x)_{m_1/2}^{2m_1-2} = c(x)_{m_1/2}^{2m_1-2}$
10. else
$NRHKOA(c(x)^{2m_1-2}, a_1(x)_{m_1/2}^{m_1-1}, b_1(x)_{m_1/2}^{m_1-1}, d(x)_{m_1/2}^{2m_1-2}, a_2(x)_{m_1/2}^{m_1-1}, b_2(x)_{m_1/2}^{m_1-1}, m_1/2, g).$
11. $c(x)_{m_1/2}^{3m_1/2-2} = u(x) + c(x)_{m_1/2}^{m_1-2} + c(x)_{m_1/2}^{2m_1-2}$
12. $d(x)_{m_1/2}^{3m_1/2-2} = v(x) + d(x)_{m_1/2}^{m_1-2} + d(x)_{m_1/2}^{2m_1-2}$
13. Return($c(x), d(x)$).

<표 3>의 알고리즘은 <표 2>의 제 6 과정의 NRHKOA 연산에 관한 것이다.

즉, 제 1 과정에서 m_1 이 4임을 확인하여 4비트 곱셈 연산을 사용하거나, 이하의 과정을 진행한다. 여기서, 4비트 곱셈 연산은 후술하는 <표 4>를 통해 살펴보기로 한다.

<표 3>은 두가지 연산으로 구분된다.

첫 번째는 $c(x) = a_1(x) \cdot b_1(x)$ 의 계산이며 두 번째는 $d(x) = a_2(x) \cdot b_2(x)$ 의 계산이다. <표 3>에서 m_1 은 언제나 짝수이므로, $a_1(x) \cdot b_1(x)$ 와 $a_2(x) \cdot b_2(x)$ 는 NRHKOA에 의해 계산된다.

그리고 <표 3>의 제 3 과정에서 f는 $a_1(x)_0^{m_1/2-1}, b_1(x)_0^{m_1/2-1}, ADD_{a_2(x)}$ 와 $ADD_{b_2(x)}$ 라 f개의 불필

요하지 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, <수학식 9>로 표시할 수 있다.

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

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

$$a_2(x)_g^{m_1-1} = a_1(x)_g^{m_1-1}, \quad b_2(x)_g^{m_1-1} = b_1(x)_g^{m_1-1}$$

그리고, <표 3>의 제 8 과정에서 $g=0$ 이면 <표 3>의 알고리즘에서는 $d(x)_{m_1}^{2m_1-2}$ 을 계산하지 않으며, 그렇지 않은 경우에는 제 10 과정의 $d(x)_{m_1}^{2m_1-2} = a_2(x)_{m_1/2}^{m_1-1} \cdot b_2(x)_{m_1/2}^{m_1-1}$ 에서 계산된다.

<표 3>의 제 1 과정에서 m_1 이 4인 경우 동작하는 4비트 다항식의 곱셈 알고리즘은 <표 4>에서 설명한다.

[표 4]

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($c(x), d(x)$).	

<표 4>의 제 1 과정에서 $flag$ 가 1인 경우 $a(x)$ 와 $b(x)$ 는 4비트 다항식으로

$$c(x) = a(x)_0^{k-1} \cdot b(x)_0^{k-1} + (a(x)_0^{k-1} \cdot b(x)_k^3 + a(x)_k^3 \cdot b(x)_0^{k-1})x^k \text{ 와}$$

$$d(x) = a(x)_k^3 \cdot b(x)_k^3 x^{2k} \text{ 을 통해 } c(x)_{2k}^6 += d(x)_{2k}^6 \text{ 을 연산한다.}$$

한편, $flag$ 가 1이 아닌 경우에는 $a(x)$ 와 $b(x)$ 는 k 개의 필요한 계수를 가지며, $d(x) = a(x)_k^3 \cdot b(x)_k^3 x^{2k}$ 를 연산하지 않고 $c(x) = a(x)_0^{k-1} \cdot b(x)_0^{k-1} + (a(x)_0^{k-1} \cdot b(x)_k^3 + a(x)_k^3 \cdot b(x)_0^{k-1})x^k$ 와 $c(x)_{2k}^6 += d(x)_{2k}^6$ 만을 연산하여 출력한다.

<표 4>는 도 1을 통해 구현된다.

도 1 은 본 발명에 사용되는 4비트 다항식 곱셈기의 구조에 관한 일실시에 구성도이다.

도 1을 통해 <표 4>와 같이 4비트의 $a(x)$ 와 $b(x)$ 입력을 통해 $c(x)$ 와 $d(x)$ 출력을 얻음을 알 수 있다.

도 2는 본 발명에 사용되는 $GF(2^n)$ 다항식에서의 기존의 카라슈바 알고리즘(KOA)과 본 발명에 따른 카라슈바 알고리즘(NRKOA)의 구조에 관한 일실시에 비교도이다.

도 2는 $n=44$ 인 경우 KOA와 NRKOA의 연산과정을 나타내는 것이다. 도시된 바와 같이 KOA는 61개의 4-비트 곱셈기를 사용하지만 본 발명의 실시예에서 제안하는 NRKOA는 51개의 4-비트 곱셈기를 사용하기 때문에 그 연산량이 적어짐을 알 수 있다. 즉, NRKOA는 기존의 KOA보다 공간 복잡도에서 효율성을 가진다.

도 3 은 본 발명의 실시예에 있어서 <표 3>의 알고리즘을 통해 사용되는 8비트 곱셈기에 대한 일실시에 구성도이다.

도 3을 참조하면, 본 발명의 실시예에 있어서 <표 3>의 알고리즘을 통해 사용되는 8비트 곱셈기는, 입력을 받기 위한 2개의 4비트 곱셈기(31, 32), 입력값을 XOR 연산하는 4개의 XOR 연산기(301, 302, 303, 304), 입력값을 XOR 연산한 값들을 입력으로 한 4비트 곱셈기(33) 및 상기 3개의 4비트 연산기의 출력을 각각 XOR 연산하여 출력하는 6개의 XOR 연산기(305 내지 310)로 이루어진다.

이상의 본 발명의 실시예에서 제안하는 NRKOA 곱셈기의 효율성을 비교하면, <표 3>의 제 9 과정에서 $d(x)_{m_1}^{2m_1-2}$ 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 몇가지 기호를 정의한다.

#KOA(u) : u-비트 KOA 곱셈기의 복잡도

#NRKOA(u) : u-비트 NRKOA 곱셈기의 복잡도

#NRHKOA(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.

그리고 <표 3>에서 NRHKOA가 한번 반복되었을 때의 복잡도는 다음 <수학식 10>, <수학식 11>과 같다.

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

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

이때 g_1 은 0이고 δ 는 추가되는 XOR 연산이다. 만약 NRHKOA가 두 번 반복되며 g_1 과 g_2 가 모두 0이면, 그 복잡도는 <수학식 12>와 같다.

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

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 효율성을 보이며 단지 1개의 게이트만 감소한다. <표 5>와 <표 6>은 기존 방법의 병렬 곱셈기와 본 발명에서 제안하는 알고리즘으로 구성된 병렬 곱셈기의 효율성을 보인다. 표의 결과에 의하면 기존의 KOA와 같은 시간 복잡도를 가지지만 공간 복잡도는 감소하며 제안하는 곱셈기의 효율성은 n의 헤밍 위이트에 의존한다.

[표 5]

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%

[표 6]

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%

상술한 바와 같은 본 발명의 방법은 프로그램으로 구현되어 컴퓨터로 읽을 수 있는 형태로 기록매체(씨디롬, 램, 플로피 디스크, 하드 디스크, 광자기 디스크 등)에 저장될 수 있다.

이상에서 설명한 본 발명은, 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자에 있어 본 발명의 기술적 사상을 벗어나지 않는 범위 내에서 여러 가지 치환, 변형 및 변경이 가능하므로 전술한 실시예 및 첨부된 도면에 의해 한정되는 것이 아니다.

발명의 효과

본 발명에서는 카라슈바 알고리즘에서 불필요한 연산을 제거한 불필요한 연산이 없는 카라슈바 알고리즘과 병렬 하드웨어 구조를 제안하여 기존의 KOA(Karatsuba-Ofman Algorithm)와 같은 시간 복잡도를 가지지만 공간 복잡도에서 효율성을 가지는 효과가 있다.

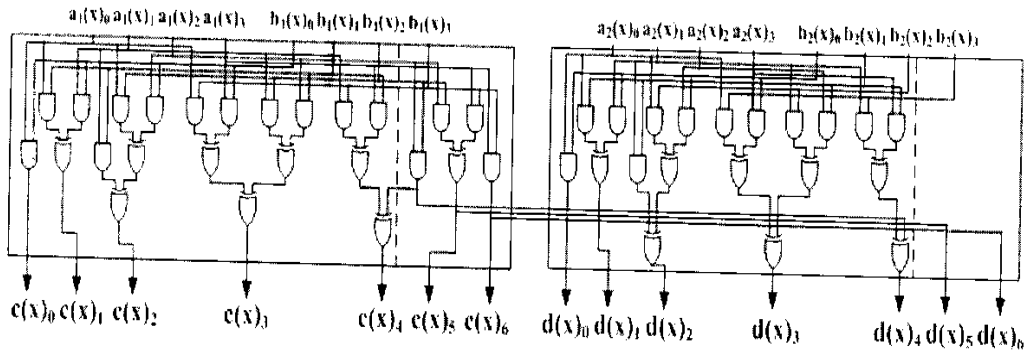
또한, 본 발명에 따른 알고리즘을 적용한 경우 최대 43%까지 공간 복잡도를 줄일 수 있으며 ECC(Elliptic Curve Cryptosystem)와 같이 소수 차수를 가지는 확장체에서 더욱 효율적이기 때문에, 낮은 공간 복잡도가 요구되는 스마트 카드, 모바일 디바이스 등에 효율적으로 적용될 수 있는 효과가 있다.

도면의 간단한 설명

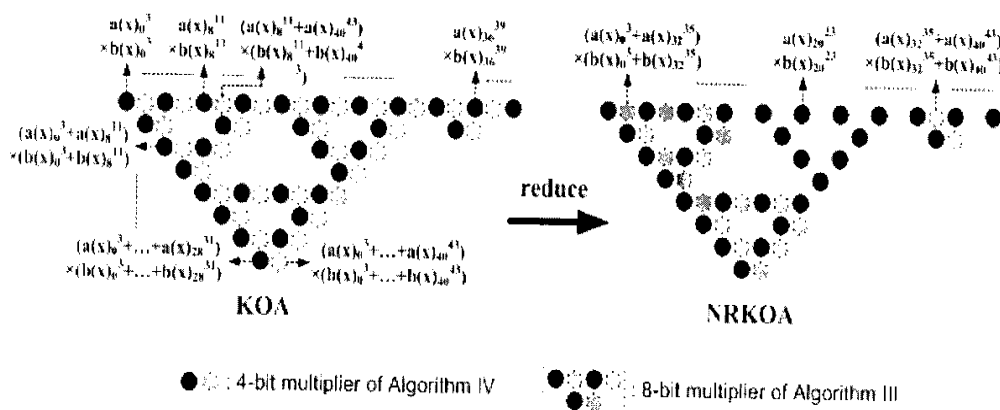
카라슈바 알고리즘을 설명한 도면

도면

도면1



도면2



도면3

