Radix-4 Modified Booth's 알고리즘을 응용한 타원곡선 스칼라 곱셈

문상국*

Elliptic Curve Scalar Point Multiplication Using Radix-4 Modified Booth's Algorithm

Sangook Moon*

요 약

타원곡선 암호시스템에서의 가장 큰 뼈대가 되는 연산은 스칼라 곱셈 연산이다. 이러한 타원 곡선 유한체 내에서 유한체 곱셈과 유한체 나눗셈보다 한 계층 상위의 개념에서 수행되는 스칼라 곱셈의 구현은 주로 두배점-덧셈(double-and-add)이라는 방식이 많이 쓰였고 [1], 최근에는 NAF(Non Adjacent Format) [2] 알고리즘이 제안되었다. 본 논문에서는 radix-4 Booth's 알고리즘을 응용하여 기존 방식보다 한 단계 더 효율적인 스칼라 곱셈 알고리즘을 제안하였다. 기존의 double-and-add 알고리즘으로 처리하였던 스칼라 곱셈 방식을 개선한 새로운 네배점-덧셈(quad-and-add) 알고리즘을 유도한 다음, 이를 사용하기 위하여 새로운 네배점(point quadruple; quad( )) 연산을 유도하고 증명하였다. 유도한 수식들은 C 프로그램과 HDL을 사용하여 실제 계산에 응용하여 증명하였다. 제안된 타원곡선 스칼라 곱셈 방식은 타원곡선 암호시스템 응용 분야의 효율적이고 빠른 연산을 처리하는데 적용할 수 있다.

ABSTRACT

The main back-bone operation in elliptic curve cryptosystems is scalar point multiplication. The most frequently used method implementing the scalar point multiplication, which is performed in the upper level of GF multiplication and GF division, has been the double-and-add algorithm, which is recently challenged by NAF(Non-Adjacent Format) algorithm. In this paper, we propose a more efficient and novel scalar multiplication method than existing double-and-add by applying redundant recoding which originates from radix-4 Booth's algorithm. After deriving the novel quad-and-add algorithm, we created a new operation, named point quadruple, and verified with real application calculation to utilize it. Derived numerical expressions were verified using both C programs and HDL (Hardware Description Language) in real applications. Proposed method of elliptic curve scalar point multiplication can be utilized in many elliptic curve security applications for handling efficient and fast calculations.

키워드

타원곡선, 유한체, 스칼라 곱셈, GF(2^m), HDL

Ⅰ. 서 론

Diffie-Hellman 키 교환 방식에 바탕을 둔 EC Elgamal 암호 시스템이나 ECDSA와 같은 타원 곡선 암호 어플리케이션들을 적용할 때 가장 시간이

*목원대학교 정보전자영상공학부

접수일자 : 2004. 07. 07

많이 걸리면서 대부분을 차지하는 동작이 다음 식과 같이 점 덧셈 연산을 반복하여 수행하는 스칼라 곱셈이다. 여기서 k는 $GF(2^m)$ 상의 임의의 정수이고 P는 $GF(2^m)$ 상에서 정의된 타원 곡선 위의 임의의 점이다.

$$kP = P + P + \ldots + P \quad (1)$$

일반적인 연산 방식에서 스칼라 곱셈을 한 번 수행 할 때는 여러 번의 점 덧셈 연산과 (더하고자 하는 두 점이 다를 때) 두배점 연산이 (더하고자 하는 두 점이 같을 때) 필요하고, 무엇보다도 식 (1)을 빠르게 처리할 수 있는 방법이 중요하다. 본 논문에서는 식 (1)의 스칼라 곱셈을 구현하는 데 대한 기존의 방법들을 소개하고, 본 논문에서 radix-4 Booth's 알고리즘을 적용하여 새롭게 제안하는 스칼라 곱셈 알고리즘과 비교하고 그 성능을 검증하고 평가한다.

II. 스칼라 곱셈

II.1 기존 방식

타원 곡선 암호 시스템에서 스칼라 곱셈을 구현할 때 가장 많이 사용되면서 기본적인 방법이 double-and-add 알고리즘을 사용한 방식이다. 이 방식은 RSA 암호 시스템에서 정수의 법 연산으로 지수 연산을 구현할 때 사용하는 square-and-multiply 방식과 유사하다고 할 수 있는데, $k = \sum_{i=0}^{m-1} b_i 2^i$ $(b_i \in 0,1)$ 일 때 알고리즘은 아래와 같다. 알고리즘 내에서 점 덧셈 연산은 *add*(), 두배점 연산은 *double*() 이라고 표기하기로 한다.

Double-and-add algorithm for computing kP
kP :
$k = \sum_{i=0}^{m-1} b_i 2^i$ $(b_i \in 0,1)$
$P := P(x_1, y_1)$
$Q := P$
for i from m-1 downto 0 do
$Q := double(Q)$
if $b_i = 1$ then
$Q := add(P, Q)$
end ($Q = kP$)

이 알고리즘에서 중요한 것은 최소 $m-1$번의 두배점 연산에 더하여 k를 이진수로 표현했을 때의 Hamming weight [3] 만큼의 점 덧셈 연산이 필요하다는 것이다. 이 방식을 개선하기 위해, 몇 가지 방식이 제안되었다. 그 중 하나가 Non-Adjacent -Format (NAF)을 이용한 방식인데, 알고리즘은 다음과 같다.

Binary NAF method for computing kP
$\mathrm{NAF}(k) = \sum_{i=0}^{t-1} k_i \cdot 2^i$
kP :
$Q = 0$
for i from $t-1$ downto 0 do
$Q = 2Q$
if $k_i = 1$ then $Q = Q + P$
if $k_i = -1$ then $Q = Q - P$
end ($Q = kP$)

위 방식은 kP를 계산함에 있어 k의 이진 표현에 대한 redundancy [4]를 사용한다. 하지만 k를 NAF 형태로 미리 바꾸어야 한다는 단점이 있다. 본 논문에서는 kP 계산을 효율적으로 구현하기 위해 이진 곱셈기에 많이 사용되는 Booth's 알고리즘 [5]을 적용하여 기존에 소개된 kP 계산 알고리즘들보다 한 단계 더 발전된 스칼라 곱셈 연산 알고리즘을 제안한다.

II.2 Quadruple-and-add 알고리즘

Booth's 알고리즘에서는 반복적으로 이어지는 수열을 효과적으로 이용하기 위해 다음 식 (2)와 같이 이진 수열을 변경하고, 결과는 signed-digit 형태로 표현될 수 있다 [6].

$$\begin{aligned} &\ldots 0\{11\ldots 11\}0\ldots \\ &= \ldots 1\,\{0\ldots 00\}0\ldots - \ldots 0\,\{00\ldots 01\}0\ldots \\ &= \ldots 1\{00\ldots 0\bar{1}\}0\ldots \end{aligned} \quad (2)$$

이 형태는 곱셈 알고리즘에 적용될 때 연속되는 '1'의 행렬을 연속되는 '0'의 행렬로 바꿈으로써 캐리(carry) 전달을 수반하는 이진 덧셈을 간단한 쉬프트(shift) 연산으로 대치한다. 마찬가지로, double-and-add 알고리즘에는 긴 '1'의 행렬이 있으면 점 덧셈 연산을 따로 해 주어야 하고 긴 '0'의 행렬이 있으면 두배점 연산만을 그 동안 수행하게 된다. Radix-4 modified Booth's algorithm은 한 번에 오퍼랜드 (operand)의 비트열의 세 비트를 참고하면서 한 번에 두 비트씩 연산을 처리하여 부분곱

의 계산량을 반으로 줄인다 [7]. 표 1에 radix-4 modified Booth's algorithm에서 사용되는 연산과 그 의미를 요약하였다. Booth's 알고리즘을 적용시키기 위해서는 일단 정수 k를 2의 보수 형식으로 표현해야 하기 때문에 k의 2진 비트 열의 최 상위 비트에 0을 붙여주어야 한다.

표 2는 Booth's 알고리즘을 적용시키면서 기존의 double-and-add 방식과의 차이점을 비교한 것이다. 먼저 기존의 double-and-add 방식은 m 비트로 이루어진 k의 Hamming weight 값에 맞추어 연산 단계의 수가 결정된다. 임의의 정수 k의 비트열에는 '1'이 몇 번 나올지 알 수가 없기 때문에 이를 확률적으로 계산하기로 한다. 첫 번째 $+P$가 기본적으로 선택되고 이후 $m-1$ 번의 *double*() 연산이 소요되는데 m 비트 중 '1'이 있을 확률은 0.5이므로 $m/2$ 번의 *add*() 연산이 소요된다고 예상할 수 있다. 여기서 Booth's 알고리즘을 적용하면 최상위 비트 자리의 0으로 인해 $\lceil \frac{m+1}{2} \rceil$ 번의 연산 단계가 필요하다. 그림 1에서 스칼라 곱셈을 수행할 때의 기존 방법과 개선된 방법의 연산 단계의 차이를 도식적으로 설명하였다. 첫 번째 연산 대상이 $+P$ 이거나 $+2P$ 이기 때문에 첫 번째 연산 회수를 제외한 나머지에 해당하는 연산 회수마다 *double*() 연산이 두 번씩 필요하고 처음 $2P$를 위한 계산이 소요되기 때문에 결과적으로 $1+2\cdot(\lceil \frac{m}{2} \rceil - 1) = 2\cdot\lceil \frac{m}{2} \rceil - 1$ 번의 *double*() 연산이 필요하게 된다.

표 1. Radix-4 modified Booth's algorithm
Table 1. Radix-4 modified Booth's algorithm

$k_i\ k_{i-1}$	k_{i-2}	$k'_i\ k'_{i-1}$	동작	설명
0 0	0	0 0	+0	0의 행렬
0 1	0	0 1	+P	한 번의 1
1 0	0	1 0	-2P	1 행렬의 시작
1 1	0	0 1	-P	1 행렬의 시작
0 0	1	0 1	+P	1 행렬의 끝
0 1	1	1 0	+2P	1 행렬의 끝
1 0	1	0 1	-P	한 번의 0
1 1	1	0 0	+0	1의 행렬

면적 면에서는 연산 단계마다 P 또는 $2P$에 해당하는 점 덧셈 연산을 해 주어야 하기 때문에 P를 저장하는 장소 이외에 $2P$를 저장하는 공간이 따로 필요하다. 연산 단계마다 *add*() 연산의 회수는 표 1을 참고하면 모두 8 가지 $k_ik_{i-1}(k_{i-2})$의 경우에 확률적으로 $\pm P$ 혹은 $\pm 2P$를 연산할 경우가 6번임을 알 수가 있고 연산 단계가 $m/2$ 번이므로 $6/8\cdot m/2 = (3/8)m$ 번 점 덧셈 연산을 해야 한다는 것을 알 수가 있다. 여기서, 또 하나의 개선점을 생각해보면, 네 배가 되는 점을 구하기 위해 한 연산 단계에 두 번씩 *double*() 연산을 수행하기보다는 네배점 연산 (point quadruple, *quad*())을 유도하여 최적화함으로써 다시 한번 알고리즘을 개선시킬 수 있다. 이는 아래 표 3에서 다시 비교하겠지만, *add*() 연산 측면에서 본다면 약 12.5% 만큼의 연산 회수를 줄일 수 있는 성능 향상을 보이는 것이다. 다음은 두배점 연산을 이용하여 최적화시킨 네배점 연산 공식의 유도 결과이다.

표 2. 스칼라 곱셈별 타원 곡선 연산 수 비교
Table 2. Comparison of the number of EC operations among scalar multiplication methods

	steps	*add*()	*double*()	*neg*()	*quad*()
*	m	$\frac{1}{2}m$	$m-1$	0	0
**	$\lceil \frac{m+1}{2} \rceil$	$\frac{3}{8}m$	$2\lceil \frac{m}{2} \rceil - 1$	0	0
***	$\lceil \frac{m+1}{2} \rceil$	$\frac{3}{8}m$	1	2	$\lceil \frac{m}{2} \rceil$

* : double-and-add
** : Adopting Booth's recoding
*** : Booth's recoding with point quadruple

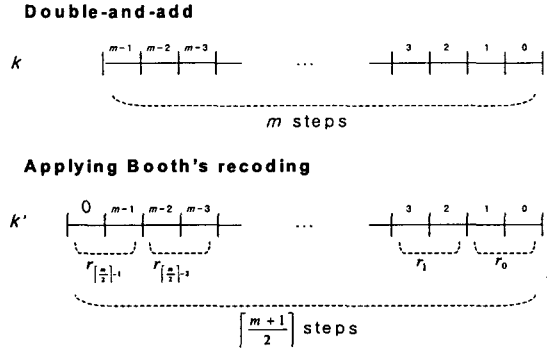

그림 1. 스칼라 곱셈 방법의 연산 단계 수 비교
Figure 1. Comparison of # of steps between the two algorithms

[$GF(2^m)$ 상의 네배점 연산 알고리즘]

◎ $P(x_1, y_1) = Q(x_1, y_1)$는 타원 곡선 위의 같은 한 점이다.
◎만일 $x_1 = 0$이면, 결과 값 $4P$는 O이다.
◎만일 $x_1 \neq 0$이면, 결과 값은 $4P(x_1, y_1) = R(x_3, y_3)$이고, x_3, y_3의 값은 다음과 같다.

$$x_3 = \lambda'^2 + \lambda' + a,$$
$$y_3 = x_1^2 + (\lambda' + 1)x_3,$$
$$\lambda' = x_2 + \lambda + 1 + \frac{x_1^2}{x_2},$$
$$x_2 = \lambda^2 + \lambda + a,$$
$$\lambda = \left(x_1 + \frac{y_1}{x_1}\right). \quad (3)$$

두배점 연산 알고리즘을 사용한 방법과 네배점 연산 알고리즘을 유도하여 적용한 방법에서 유한체 연산 수의 차이를 표 3에 정리하였다. 타원 곡선 위의 임의의 점의 네배점에 해당하는 점을 구하는 연산을 수행 할 때 유한체 곱셈이 한 번 줄어듦으로써 전체적인 스칼라 곱셈 시에 상당한 이득을 얻을 수 있다. 유한체 덧셈은 구조의 성능에 거의 영향을 미치지 않기 때문에 얼마 간의 수적 차이는 의미가 없다고 볼 수 있다. 이를 포함시켜 최종적으로 Booth's 알고리즘을 적용하여 double-and-add 알고리즘을 개선시킨 새로운 알고리즘은 다음과 같다. 알고리즘 내에서 점 역원 연산은 *neg*()로 표현하고 네배점 연산은 *quad*()로 표기한다.

표 3. 두배점 연산 사용시와 네배점 연산 사용시의 연산 회수 비교
Table 3. Comparison of the number of operations between with point double and with point quadruple

	multiplications	divisions	squarings	additions
2x *point double*	2 x 1 = 2	2 x 1 = 2	2 x 2 = 4	2 x 4 = 8
point quadruple	1	2	4	10

Suggested algorithm using radix-4 redundancy kP :

$k = \sum_{i=0}^{\lceil \frac{m}{2} \rceil - 1} r_i 4^i$ (r_i는 Booth recoding 값)

$P = P(x_1, y_1)$
$2P = double(P)$
$Q = \{0P, +P, +2P, -P, -2P\}$ 중 하나
for i from $\lceil \frac{m}{2} \rceil - 1$ downto 0 do
$Q = quad(Q)$
if $(r_i == +P)$ then
$Q = add(P, Q)$
if $(r_i == +2P)$ then
$Q = add(2P, Q)$
if $(r_i == -P)$ then
$tempP = neg(P)$
$Q = add(tempP, Q)$
if $(r_i == -2P)$ then
$temp2P = neg(2P)$
$Q = add(temp2P, Q)$
end ($Q = kP$)

double-and-add 방식에서는 m 단계를 사용하지만 제안된 알고리즘을 사용하여 스칼라 곱셈을 연산하고자 하는 경우에는 그 단계의 수가 $\lceil \frac{m+1}{2} \rceil$ 개로 줄게 된다. 또한, 점 Q를 리코딩하여 각각의 리코딩 값에 따라서 단계별로 $\pm P$, $\pm 2P$ 연산을 적용하거나 혹은 *quad*() 연산만을 실행하면 결과 값을 얻을 수 있다. 제안한 스칼라 곱셈 알고리즘은 기존 double-and-add 알고리즘과 비교 할 때 m 비트 쉬프터 (shifter)와 3 비트 radix-4 Booth recoding 회로의 추가만으로 구현된다. 추가되는 회로는 그림 3과 같다.

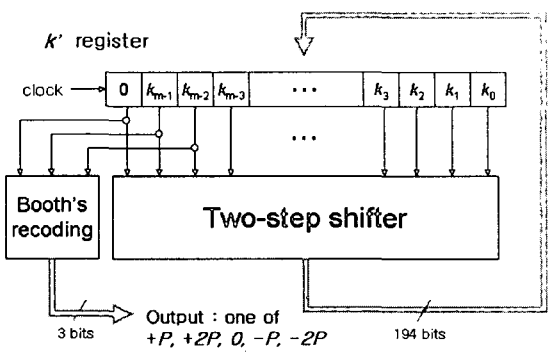

그림 2. Quad-and-add 회로 구현에 필요한 추가 회로
Figure 2. Additional hardware logic required for implementing Quad-and-add

III. 제안한 스칼라 곱셈 알고리즘 성능 평가

제안된 quad-and-add 알고리즘에서 유한체 곱셈기는 각각 [8]에서 제안된 곱셈기와 나눗셈기를 사용한다고 가정하고 성능을 평가하였다. 제안된 quad-and-add 알고리즘은 한 번 스칼라 곱셈 시 $(3/8)m$ 번의 *add*(), $2P$를 계산하기 위한 한 번의 *double*(), 그리고 $\lceil \frac{m}{2} \rceil - 1$ 번의 *quad*() 연산이 필요하다. 또한 이를 유한체 연산으로 바꾸어 계산하면 표 4에 보이듯이 $(7/8)m+1$ 번의 유한체 곱셈, $(11/8)m+1$ 번의 유한체 나눗셈, $(19/8)m+2$ 번의 유한체 제곱 연산이 필요하고, $8m+6$ 번의 유한체 덧셈 연산이 필요하다. 이렇게 보았을 때, 제안된 스칼라 곱셈 알고리즘은 유한체 곱셈 연산 부분에서 약 42%의 개선율을 보이고 유한체 나눗셈 부분에서 약 9%, 유한체 제곱 연산 부분에서 약 5%의 개선율을 보인다. 유한체 곱셈과 제곱이 같은 자원을 공유하여 사용한다고 생각하면 두 연산을 같이 고려하여 약 19% 정도 유한체 곱셈 측면에서 이득이 있다는 것을 의미한다.

표 4. 유한체 연산 개수의 비교
Table 4. Comparison of the number of GF operations

	double-and-add	quad-and-add	Reduction
multiplication	$\frac{3}{2}m$	$\frac{7}{8}m+1$	$\simeq 0.58$
division	$\frac{3}{2}m$	$\frac{11}{8}m+1$	$\simeq 0.91$
square	$\frac{5}{2}m$	$\frac{19}{8}m+2$	$\simeq 0.95$
addition	$8m$	$8m+6$	$\simeq 1$

설계에 걸리는 시간을 단축하고 오류를 쉽게 수정할 수 있도록 RTL 수준에서 HDL을 사용하여 하드웨어를 기술하고 합성을 수행하였다. 사용된 표준 셀 (standard cell) 라이브러리는 삼성전자의 0.35um 공정의 std90이다. 프로토타입으로 구현한 193비트 타원곡선암호용프로세서로 한 번의 스칼라 곱셈을 구현했을 경우 표 5와 같은 결과를 보였다. 소프트웨어 측면에서만도 약 30%의 성능향상을 보였고, 하드웨어상에서는 기존 구조보다 많게는 약 7배의 성능 향상을 보였다.

표 5. 스칼라 곱셈 수행 결과
Table 5. Scalar multiplication results

	스칼라 곱셈 수행시간		
	사이클수	20MHz환산 수행시간	환산면적
PIII933MHz, C code double-and-add	-	10.436 sec	-
PIII933MHz, C code quad-and-add	-	7.106 sec	-
GF(2^{167}) standard-basis implementation [9]	526,718	26.335 msec	20k gates
163b cryptoprocessor [10]	258,000	12.9 msec	24k gates
proposed 193b prototype	83,268	4.163 msec	59k gates

IV. 결 론

본 논문에서는 radix-4 Booth's 알고리즘을 적용한 새로운 스칼라 곱셈 방식을 제안하고, 성능을 평가하였다. 결과는 스마트 카드와 같은 소형 정보보호기기에 구현하기에 적합한 크기에 기존 구조보다 획기적으로 향상된 성능을 보여주었다.

만일 네배점 연산인 *quad*() 알고리즘을 보다 전문적이고 폭넓은 수학적 지식의 도움을 받아 좀 더 최적화시킬 수 있다면 제안된 Booth recoding을 적용하는 방법은 더욱더 효율적인 방법이 될 것이다. 또한, radix-4 방식에 이어서 차후 radix-8이나 그 이상으로 구현할 경우 보다 대폭적인 성능 향상이 기대되어 복잡한 타원곡선 연산을 수행하는 데 매우 효율적으로 응용될 수 있을 것이다.

참고문헌

[1] http://www.certicom.com
[2] D. Hankerson, J. L. Hernandez, and A. Menezes, "Software Implementation of

Elliptic Curve Cryptography over Binary Fields," Crypto95, 1995..
[3] http://www.its.bldrdoc.gov/fs-1037/dir-017/_2530. htm
[4] I. Koren, Computer Arithmetic Algorithms, Chapter 6, Prentice Hall International, pp. 99-106, 1993.
[5] A. D. Booth, "A Signed Binary Multiplication Algorithm," Quart. J. Mech. Appl.. Math., Vol. 4, Pt. 2, pp. 236-240, 1951.
[6] K. Hwang, Computer Arithmetic, John Wiley & Sons, 1979.
[7] L. P. Rubinfield. "A Proof of the Modified Booth's Algorithm for Multiplication," IEEE Transactions on Computers, Vol. C-24, No. 10, pp. 1014-1015, Oct. 1975.
[8] 문상국, "타원 곡선 암호용 프로세서를 위한 고속 VLSI 알고리즘의 연구와 구현," 연세대학교 대학원 박사학위논문집, 2002.
[9] G. Orlando, C. Paar, "A Super-Serial Galois Fields Multiplier for FPGAs and its Application to Public-Key Algorithms," Proceedings of 7th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 232-239, 1999.
[10] 최용제, 김호원, 김무섭, 박영수, "IC 카드를 위한 polynomial 기반의 타원 곡선 암호시스템 연산기 설계," 2001년도 대한전자공학회 하계종합학술대회 논문지 제 24권 제 1호, pp. 305-308, 2001.

저자소개

문상국(Sangook Moon)

1995년 연세대학교 전자공학사
1997년 연세대학교 전자공학석사
2002년 연세대학교 전자공학박사
2002.2~2004.2 하이닉스반도체 선임연구원
2004.3~현재 목원대학교 정보전자영상공학부 전임강사
※관심분야 : 정보보호 VLSI 설계, Data encryption, 유비쿼터스 컴퓨팅 보안