최적확장체에서 정의되는 타원곡선 상에서 효율적인 스칼라 곱셈 알고리즘

정병천[1] 이재원[1] 홍성민[2] 김환준[3] 김영수[4] 황인호[4] 윤현수[1]
[1]한국과학기술원 전자전산학과 전산학전공, [2]트러스컴, [3]이니텍, [4]국가보안기술연구소
[1]{bcchung, jaewon, hyoon}@calab.kaist.ac.kr, [2]smhong@truscom.com,
[3]hwanjoon@initech.com, [4]{ysk, ihhwang}@etri.re.kr

An Improved Scalar Multiplication on Elliptic Curves over Optimal Extension Fields

Byung-Chun Chung[1] Jaewon Lee[1] Seong-Min Hong[2] Hwan-Joon Kim[3]
Youngsoo Kim[4] Inho Hwang[4] Hyunsoo Yoon[1]
[1]Dept. of EECS, Division of Computer Science, KAIST, [2]truscom Co.,Ltd,
[3]INITECH Co.,Ltd, [4]National Security Research Institute

요 약

본 논문에서는 최적확장체(Optimal Extension Field; OEF)에서 정의되는 타원곡선 상에서 효율적인 스칼라 곱셈 알고리즘을 제안한다. 이 스칼라 곱셈 알고리즘은 프로비니어스 사상(Frobenius map)을 이용하여 스칼라 값을 Horner의 방법으로 base-ϕ 전개하고, 이 전개된 수식을 일괄처리 기법(batch-processing technique)을 사용하여 연산한다. 이 알고리즘을 적용할 경우, Kobayashi 등이 제안한 스칼라 곱셈 알고리즘보다 40% 정도의 성능향상을 보인다.

1. 서론

타원곡선 암호시스템은 Koblitz와 Miller가 각각 독립적으로 제안하였다[1,2]. 유한체 상에서 정의되는 타원곡선에서의 이산대수 문제는 타원곡선과 동일한 크기를 갖는 유한체에서의 이산대수 문제보다 더 어렵다고 알려져 있다. 따라서, 타원곡선 암호시스템은 ElGamal-형태의 암호시스템보다 작은 키 크기를 요구하기 때문에 고속의 소형 암호시스템 구현이 용이하다.

타원곡선 암호시스템에서 핵심 연산은 타원곡선 위의 한 점 P에 대한 스칼라 곱셈($k \cdot P$) 연산이다. 스칼라 곱셈 연산은 반복적인 덧셈과 두 배 연산으로 이루어지기 때문에, 암호시스템의 성능은 이러한 반복 회수에 비례하게 된다. 이러한 반복 회수를 줄이기 위한 방법으로 덧셈/뺄셈 사슬이나 프로비니어스 사상 등이 사용될 수 있는데, 후자의 경우가 더 효율적인 방법으로 알려져 있다[7].

프로비니어스 사상을 이용한 base-ϕ 전개 방식은 Koblitz에 의해 처음 제안되었다[4]. 그 이후 많은 연구가 이루어졌으며, 최근에는 Kobayashi 등이 GF(p)에서 정의되는 타원곡선 위의 GF(p^m)-rational 점에 적용할 수 있는 base-ϕ 전개 방식을 제안하였다[5]. 여기서, p는 32비트 또는 64비트 정도 되는 매우 큰 소수이다.

본 논문에서는 Kobayashi 등과 같이 최적확장체(OEF)[3]에서 정의되는 타원곡선에서 base-ϕ 전개 방식을 적용한 효율적인 스칼라 곱셈 알고리즘을 제안하고자 한다. OEF와 같이 특성값(characteristic)이 큰 유한체 상에서 정의되는 타원곡선에서는 base-ϕ 전개식을 연산하는 과정에서 일괄처리 기법(batch processing technique)[6]을 적용할 수 있다. 이 일괄처리 기법을 최적화하여 적용할 경우, Kobayashi 방법보다 40% 정도의 속도향상이 가능하다.

본 논문의 구성은 다음과 같다. 2절에서는 OEF에서 정의되는 타원곡선과 Kobayashi의 스칼라 곱셈 알고리즘에 대해서 살펴보고, 3절에서는 base-ϕ 전개 방식과 일괄처리 기법을 이용한 효율적인 스칼라 곱셈 알고리즘을 제안한다. 4절에서는 제안한 알고리즘 분석 및 성능 비교를 하고, 5절에서는 결론을 맺는다.

2. 관련연구

[5]에서 Kobayashi 등은 특정 타원곡선에 적용 가능한 스칼라 곱셈 알고리즘을 제시하였다. 이 타원곡선은 GF(p), $p>3$ 상에서 정의되는데, 다음과 같은 형태를 갖는다.

$$E: y^2 = x^3 + ax + b,$$

여기서 $a, b \in \mathrm{GF}(p)$ 이다.

$P=(x, y)$를 타원곡선 E위의 GF(p^m)-rational 점이라고 하자. 그러면 프로비니어스 사상 ϕ는 다음과 같이 정의된다.

$$\phi: (x, y) \rightarrow (x^p, y^p).$$

이 프로비니어스 사상은 E의 자기준동형(endomorphism) 사상이 된다. 그리고, 다음의 방정식을 만족한다.

$$\phi^2 - t\phi + p = 0, \quad -2\sqrt{p} \le t \le 2\sqrt{p}$$

원소를 다항식 기저로 표현할 경우, 이 프로비니어스 사상을 계산하는 데는 $2(m-1)$번의 GF(p)상의 곱셈을 필요로 한다[5]. 따라서, 프로비니어스 사상 연산은 빠르게 수행 가능한 연산임을 알 수 있다.

타원곡선에서의 스칼라 곱셈은 정수 k와 타원곡선 위의 한 점 P가 주어졌을 때, kP를 구하는 연산을 말한다. 여기서, 스칼라 k를 다음과 같이 프로비니어스 사상의 다항식으로 전개할 수 있다[5].

$$k = \sum_{i=0}^{l} u_i \phi^i, \quad \text{여기서 } -\frac{p}{2} \le u_i \le \frac{p}{2}$$

이 식에서 l은 대략 $2m+3$ 정도 된다.

이러한 k의 $2m+3$ 길이의 ϕ -adic 표현은 다음 정리1에 의하여 m 길이의 ϕ -adic 표현으로 나타낼 수 있다.

정리1 프로비니어스 사상 ϕ 의 m 반복은 항등 사상(identity map)이 된다. 즉,

$$\phi^m \equiv 1 \in End(E(\mathrm{GF}(p^m)))$$

따라서, ϕ -adic 표현은 다음과 같이 축소된다.

$$\sum_{i=0}^{2m+3} u_i \phi^i = \sum_{i=0}^{m-1} (u_i + u_{i+m} + u_{i+2m}) \phi^i$$
$$= \sum_{i=0}^{m-1} d_i \phi^i \text{ 여기서 } -\frac{3p}{2} \le d_i \le \frac{3p}{2}.$$

Kobayashi 등은 이 전개식을 연산하는 데 있어서, 반복적인 프로비니어스 사상을 연산한 후 테이블에 저장하고, 후에 이 테이블을 참조하는 방법을 제시하였다. 이를 알고리즘 1에 나타내었다. 확장 차수(extension degree) m = 7인 경우, 이 알고리즘을 적용하면 스칼라 곱셈을 $4.5\lceil \log_2 p \rceil$정도의 타원곡선 덧셈/두 배 연산으로 수행할 수 있다.

Algorithm 1 Base-ϕ Scalar Multiplication Procedure

INPUT: $k = \sum_{i=0}^{m-1} d_i \phi^i P \in E(\mathrm{GF}(p^m))$
OUTPUT: $Q = kP$
  for i = 0 **to** m - 1 **do**
    $P_i \leftarrow \phi^i P$ // *Store into the Table*
  $Q \leftarrow O$ // *여기서 O는 Point at Infinity*
  $j \leftarrow \lceil \log_2 p \rceil + 1$
  while $j \ge 0$ **do**
    $Q \leftarrow 2Q$
    for i = 0 **to** m - 1 **do**
      if d_{ij} = 1 **then** // d_{ij}는 d_i의 j번째 비트
        $Q \leftarrow Q + P_i$
    $j \leftarrow j - 1$

3. 제안 알고리즘

본 절에서는 효율적인 스칼라 곱셈 알고리즘을 제안한다. 기본 아이디어는 base-ϕ 전개식을 연산하는 데 있어서, Kobayashi와는 다른 계산 순서와 일괄처리 기법을 사용하는 것이다. 여기서 계산 순서는 Horner의 방법을 사용하는데 스칼라 곱셈 kP는 다음과 같이 전개된다.

$$kP = \sum_{i=0}^{m-1} d_i \phi^i P \quad (1)$$
$$= \phi(\cdots \phi(\phi(d_{m-1}P) + d_{m-2}P) \cdots + d_1 P) + d_0 P, \quad (2)$$
여기서 $-\frac{3p}{2} \le d_i \le \frac{3p}{2}$.

식(1)을 따르게 되면, 2절에서 언급한 바와 같이 먼저 P에 반복적인 프로비니어스 사상($\phi^i P$)을 적용한 후에, 스칼라 곱셈($d_i P$)을 수행해야 한다. 그러나 식(2)을 따르게 되면, 우선 스칼라 곱셈($d_i P$)을 먼저 계산하고, 그 다음에 프로비니어스 사상을 적용한 다음, 마지막으로 최종결과를 위해 m-1번의 덧셈을 추가로 수행해야 한다.

제안하고자 하는 알고리즘은 식(2)의 계산과정에서 스칼라 곱셈 $d_i P$을 하나씩 독립적으로 처리하지 않고, 몇 개씩 하나로 묶어 일괄적으로 처리함으로써 효율적인 계산을 가능하게 한다. 집합 개념을 이용한 다음의 정리2를 보면 기본 개념을 쉽게 이해할 수 있다.

정리2 랜덤하게 선택된 $a = \sum_i a_i 2^i$ $b = \sum_i b_i 2^i$가 주어졌을때, a와 b의 교집합을 $c = a \bigcap b = \sum_i a_i b_i 2^i$라 하면, 다음의 부등식을 만족한다.

$$w_H(a-c) + w_H(b-c) + w_H(c) \le w_H(a) + w_H(b)$$

여기서 w_H는 해밍중(Hamming weight)을 나타내고, $-$는 차집합을 나타낸다.

즉, 정리2에서 교집합을 사용할 경우 전체 해밍중이 감소함을 알 수 있다. 이는 n개의 경우로 확장 가능하다. n개의 경우에는 교집합 부분을 어떻게 구성하느냐에 따라 연산 회수의 차이가 발생하므로 효율적인 교집합 생성 방법을 필요로 한다.

본 논문에서는 이러한 효율적인 교집합 생성 알고리즘을 제시하여 스칼라 곱셈 연산의 속도를 개선하고자 한다. 이 알고리즘은 크게 두 단계로 이루어지는 데, 우선은 주어진 스칼라 d_i 들을 서로 소(disjoint)인 부분집합으로 나누고(partitioning), 그 다음에 이 부분집합들을 결합하여 원래의 각각의 집합을 복원하는 과정(combining)을 거친다. 분할 알고리즘(partitioning algorithm)과 복원 알고리즘(combining algorithm)은 각각 알고리즘 2와 3과 같다.

알고리즘 2에서 연산자 $\bigcap$, $\bigcup$, $-$는 각각 다음과 같이 구현이 가능하다.

$$a \bigcap b := a \& b \quad a \bigcup b := a | b \quad a - b := a \hat{} (a \& b)$$

여기서 &, |, ^는 비트 연산자이다.

알고리즘 2와 3에서 m은 7이하의 아주 작은 값이므로, 타원곡선 연산 외의 부가적인 단정도 연산 시간은 무시할 수 있다. 특히, 알고리즘 3은 연결리스트(linked list) 자료구조를 사용하면 효과적으로 구현 가능하다.

알고리즘 동작의 이해를 돕기 위해 간단한 예를 들어보자. m은 3이고, 각 스칼라 값은 8비트로 가정한다. 계산하고자 하는 aP, bP, cP는 다음과 같은 a, b, c값을 갖는다.

$$a = 01010011_2, \; b = 11000110_2, \; c = 10100110_2$$

$$a_{001} = a - (b \bigcup c) = 00010001_2$$
$$a_{010} = b - (a \bigcup c) = 00000000_2$$
$$a_{011} = (a \bigcap b) - c = 01000000_2$$
$$a_{100} = c - (a \bigcup b) = 00100000_2$$
$$a_{101} = (a \bigcap c) - b = 00000000_2$$
$$a_{110} = (b \bigcap c) - a = 10000100_2$$
$$a_{111} = a \bigcap b \bigcap c = 00000010_2$$

Algorithm 2 Partitioning Algorithm

INPUT: $k = \sum_{i=0}^{m-1} d_i \phi^i m$

OUTPUT: a_i, $1 \le i \le 2^m - 1$

for $i = (e_{m-1} \cdots e_1 e_0)_2$ from 1 to (2^m-1) do

$a_i \leftarrow \bigcap_{e_j=1} d_j - \bigcup_{e_j=0} d_j$

Algorithm 3 Combining Algorithm

INPUT: a_i $(1 \le i \le 2^m-1)$ $P \in E(\mathrm{GF}(p^m))$

OUTPUT: d_iP $(0 \le i \le m-1)$

Precompute 2^iP, $1 \le i \le \lceil \log_2 p \rceil$

$P_i \leftarrow \mathrm{ScalarMult}(a_i, P)$, $1 \le i \le 2^m-1$

Initialize $d_iP \leftarrow 0$, $0 \le i \le m-1$

for $i = 0$ to $m-1$ do

for $j = (e_{m-1} \cdots e_1 e_0)_2$ from 1 to (2^m-1) do

if $e_i = 1$ then

$d_iP \leftarrow d_iP + P_j$

if $j \neq (e_{m-1} \cdots e_{i+1} 0 \cdots 0)_2$ then

$P_{(e_{m-1} \cdots e_{i+1} 0 \cdots 0)_2} \leftarrow P_{(e_{m-1} \cdots e_{i+1} 0 \cdots 0)_2} + P_j$

delete P_j

사전에 $2P, 2^2P, \cdots, 2^7P$를 계산하고,

그 다음에 $P_i = a_iP$, $1 \le i \le 2^3-1$을 계산한다.

$aP = P_{001} + P_{011} + O + P_{111}$

$P_{010} = P_{010} + P_{011}$

$P_{100} = P_{100} + O$

$P_{110} = P_{110} + P_{111}$

$bP = P_{010} + P_{110}$

$P_{100} = P_{100} + P_{110}$

$cP = P_{100}$

이 방법은 7번의 두 배 연산과 8번의 덧셈 연산(a_iP연산에서 2번, 나머지 연산에서 6번)이 필요한 반면, Kobayashi 방법은 7번의 두 배 연산과 11번의 덧셈연산이 필요하다.

4. 알고리즘 분석 및 성능 비교

타원곡선에서 스칼라 곱셈 연산은 덧셈과 두 배 연산의 반복으로 수행된다. 따라서, 이 절에서는 알고리즘의 성능을 덧셈과 두 배 연산의 반복 회수 관점에서 분석하고 비교한다.

표현의 간결성을 위해 $b_p = \lceil \log_2 p \rceil + 1$라 하자.

정리3 P는 $\mathrm{GF}(p)$에서 정의되는 타원곡선 위의 $\mathrm{GF}(p^m)$-rational 점이고 k는 $m(b_p-1)$비트 정수라 할 때, 스칼라 곱셈 kP는 $m-1$번의 프로베니어스 사상과 m번의 b_p비트 스칼라 곱셈, 그리고 $m-1$번의 타원곡선 덧셈 연산을 필요로 한다.

정리4 $\overline{w_H} > 2^{s-1}$을 만족하는 랜덤한 s개의 b_p비트 정수 $a_0, a_1, \ldots, a_s$가 주어졌을 때, 스칼라 곱셈 a_iP $(0 \le i \le s-1)$는 b_p-1번의 두 배 연산과 평균 $\frac{2^s-1}{2^{s-1}}\overline{w_H} + 2^s - 2s$번의 덧셈 연산이 필요하다. 여기서 $\overline{w_H}$는 a_i들의 평균 해밍중을 나타낸다.

정리4에서 a_i가 b_p비트이므로 알고리즘 2의 분할과정에서 0이 아닌 부분집합의 수는 최대 b_p개만 가능하다. 따라서 필요 이상의 a_i를 사용하면 오히려 이득이 줄어드는 효과를 가져오기 때문에 $\overline{w_H} > 2^{s-1}$의 조건이 필요하다.

Kobayashi가 사용한 파라미터 $p = 2^{31}-1$ $m = 7$인 경우, 제안 알고리즘은 $s = 4$일 때 가장 좋은 결과를 얻을 수 있다. 따라서 4개와 3개로 각각 나누어 처리할 경우 알고리즘 3에서는 66번의 덧셈 연산이 소요된다. 정리3에 의하여 스칼라 곱셈에 소요되는 총 연산은 103번의 덧셈/두 배 연산이다. 이는 Kobayashi의 143회에 비해 40%정도 향상된 수치이다. 또한 Kobayashi방법에 비해 부가적인 오버헤드는 그리 크지 않음을 알 수 있다. 비교 결과를 [표 1]에 정리하였다.

[표 1] 스칼라 곱셈 알고리즘 비교

Algorithm	Doubling	Addition	Total
Binary	216	108	324
Signed Binary	216	72	288
Kobayashi[5]	31	112	143
Proposed	**31**	**72**	**103**

5. 결론

본 논문에서는 OEF에서 정의되는 타원곡선에서 효율적인 스칼라 곱셈 알고리즘을 제안하였다. 이 알고리즘은 Kobayashi 등의 방법보다 대략 40% 정도의 속도 개선이 있었으며, 이진방법보다는 3배정도 빠름을 알 수 있었다. 이는 타원곡선 암호시스템이 무선환경과 같은 대역폭과 계산능력이 제한된 환경에서 더욱 효과적으로 활용될 수 있을 것으로 기대된다.

참고 문헌

[1] N. Koblitz, "Elliptic curve cryptosystems", *Mathematics of Computation*, vol.48, pp.203-209, 1987.

[2] V. Miller, "Use of elliptic curve in cryptography", *Crypto'85*, pp.417-426, 1985.

[3] D. V. Bailey and C. Paar, "Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms", *Crypto'98*, pp.472-485, 1998.

[4] N. Koblitz, "CM-Curves with Good Cryptographic Properties", *Crypto'91*, pp.279-287, 1992.

[5] T. Kobayashi et al., "Fast Elliptic Curve Algorithm Combining Frobenius Map and Table Reference to Adapt to Higher Characteristic", *Eurocrypt'99*, pp.176-189, 1999.

[6] D. M'Raihi and D. Naccache, "Batch Exponentiation", *ACM CCS'96*, pp.58-60, 1996.

[7] D. M. Gordon, "A survey of fast exponentiation methods", *Journal of Algorithms*, vol.27, pp.129-146, 1998.