情報保護學會論文誌
第 21 卷 第 6 號, 2011. 12

페어링 암호 시스템을 위한 F_{3^m}에서의 효율적인 MapToPoint 방법*

박 영 호[1†], 조 영 인[2], 장 남 수[1‡]
[1]세종사이버대학교, [2]고려대학교

Faster MapToPoint over F_{3^m} for Pairing-based Cryptosystems*

Young-Ho Park[1†], Young In Cho[2], Nam Su Chang[1‡]
[1]Sejong Cyber University, [2]Korea University

요 약

페어링 암호 시스템에서 임의의 메시지 스트링을 타원곡선 위의 점으로 매핑하는 과정(MapToPoint)은 무시할 수 없는 연산량을 가지고 있으며 타원곡선 암호 시스템과 달리 페어링 암호 시스템에서는 F_{3^m} 위의 타원곡선도 이용하기 때문에 F_{3^m}에서의 MapToPoint 연산이 필요하다. Barreto 등이 F_{3^m} 위에서는 세제곱 계산이 선형연산인 것을 이용하여, x 좌표에 메시지를 대입하여 y 좌표를 계산하는 기존의 방법과 달리, y 좌표에 메세지를 대입하여 x 좌표를 계산하는 방법을 제안하였다. Barreto 등은 x 좌표의 계수들을 임의의 변수로 두고 이들로 이루어진 행렬을 이용하여 x 좌표를 계산했는데, 본 논문에서는 이 행렬의 크기를 줄여 보다 효율적으로 x 좌표를 계산할 수 있는 방법을 제안한다. 제안하는 방법은 Barreto 등의 방법의 44%의 메모리만으로 2 ~ 3배 빠른 MapToPoint 연산을 수행할 수 있다.

ABSTRACT

A hashing function that maps arbitrary messages directly onto curve points (MapToPoint) has non-negligible complexity in pairing-based cryptosystems. Unlike elliptic curve cryptosystems, pairing-based cryptosystems require the hashing function in ternary fields. Barreto et al. observed that it is more advantageous to hash the message to an ordinate instead of an abscissa. So, they significantly improved the hashing function by using a matrix with coefficients of the abscissa. In this paper, we improve the method of Barreto et al. by reducing the matrix. Our method requires only 44% memory of the previous result. Moreover we can hash a message onto a curve point 2 ~ 3 times faster than Barreto's Method.

Keywords: Map to point, Cube root, Finite field arithmetic

접수일(2011년 1월 5일), 수정일(2011년 6월 10일), 게재확정일(2011년 10월 5일)
* 본 연구는 지식경제부 및 한국산업기술평가관리원의 산업원천기술개발사업(정보통신)의 일환으로 수행하였음. [KI001810039140,스마트디바이스용칩(ARM7/9/11, UICC 등)에 최적화된 암호(ARIA, SEED, KCDSA 등)의 국가 인증 모듈 및 배포 체계 개발]
† 주저자, youngho@sjcu.ac.kr
‡ 교신저자, nschang@sjcu.ac.kr

I. 서 론

페어링 암호 시스템(Pairing-based crypto-systems)은 최근 가장 활발하게 연구되는 분야로서 페어링 기반 암호 시스템은 두 개의 타원곡선 위의 점을 하나의 유한체 원소로 보내는 양방향 겹선형성(Bilinearity)을 가지는 페어링 함수를 기반으로 구성되는 암호 시스템이다. 2001년 Boneh, Lynn 그리고 Shachaum은 처음으로 페어링 기반의 BLS 스킴이라 불리는 전자서명 스킴(Short signature scheme)을 제안하였다[2]. 이어 페어링을 이용한 여러 암호학적인 스킴들이 연구되었으며[3][7][11] 페어링을 효율적으로 구현하기 위한 연구도 이루어져왔다[8][9][12]. 한편, 페어링 연산 시간을 개선하려는 여러 노력에도 불구하고 페어링의 구현은 여전히 앞서 서술한 페어링 기반 암호 시스템들을 구현하는데 있어 가장 큰 연산 부담을 주고 있다.

페어링 함수는 소수 위수 q를 갖는 타원곡선 점들의 그룹 G_1과 G_2에서 유한체 원소들의 그룹 G_T로의 함수이다. P_1과 P_2를 각각 G_1과 G_2의 생성자라 할 때 G_1, G_2 그리고 G_T의 이산 대수 문제(DLP)를 풀기 어렵다고 가정한다. 그러면 아래의 세 조건을 만족하는 함수 $e: G_1 \times G_2 \rightarrow G_T$는 암호에 사용될 수 있는 겹선형 함수인 페어링 함수가 된다.

- Bilinearity : $A, B \in G_1, B', C \in G_2$일 때,
  $e(A+B, C) = e(A, C)e(B, C)$ 또는
  $e(A, B'+C) = e(A, B')e(A, C)$
- Non-degeneracy : 항등원이 아닌 모든 P에 대하여 $e(P, P) \neq 1$
- Computability : 페어링 함수 e는 효율적으로 계산되어야 함

이진체 또는 소수체 위의 타원곡선만을 사용하는 타원곡선 암호와 달리 페어링 암호는 위수가 3인 삼진체 위의 타원곡선도 이용한다. 삼진체 위에서의 페어링 연산이 다른 유한체 위에서의 연산보다 빠른 결과로는 [8]등이 있으며, F_{3^m}에서의 효율적인 페어링 연산 및 이에 사용되는 유한체 연산에 대한 연구가 주목을 받고 있다. 페어링 암호 기반의 암호 프로토콜 구현을 위하여 embedding degree가 상대적으로 작은 초특이 타원곡선이 매우 적합한 것으로 연구되었고[1], F_{3^m} 위의 페어링 연산을 위해서는 embedding degree가 6 이며, $E^{\pm}: y^2 = x^3 - x \pm 1$로 정의된 초특이 타원곡선이 사용된다.

F_{3^m} 위에서 메시지 스트링을 초특이 타원곡선 $E^{\pm}$ 위로 매핑하는 과정을 [2]와 같이 MapToPoint이라 정의하며, 이 함수는 페어링 기반 암호시스템을 구성하는 함수 중 하나이다. 본 논문에서는 초특이 타원곡선을 이용하여 MapToPoint를 효율적으로 수행할 수 있는 방법을 제안한다. F_{2^m} 위에서의 타원곡선 암호에서는 메시지의 해쉬(Hash) 값을 x 좌표에 대입하고 타원곡선 식을 만족하는 y 좌표를 계산하는 것이 일반적이다. [2]에서는 F_{2^m} 위에서와 동일한 방법을 사용하므로 MapToPoint 연산을 위해 F_{3^m} 위에서 제곱근을 구해야 한다. 따라서 연산량이 $O(m^3)$으로 비교적 컸다. Kawahara 등은 F_{2^m} 위에서 곱셈연산 없이 $y^3 + y = c (c \in F_{2^m})$의 근 y를 찾을 수 있는 [10]의 방법을 F_{3^m} 위의 MapToPoint에 적용하여 [2]의 방법을 35% 개선하였다[21].

Barreto 등은 F_{3^m} 위에서는 세제곱이 선형연산으로서 비교적 가벼운 연산이므로 메시지의 해쉬 값을 y 좌표에 대입하고 타원곡선 식을 만족하는 x 좌표를 찾는 것이 [2]에 비해 훨씬 연산량이 적음을 보였다[15]. Barreto 등은 F_{3^m} 위에서의 삼차방정식을 풂으로써 해당 x 좌표를 찾을 수 있도록 하였다. 이 삼차방정식을 풀기 위해서는 F_3의 원소로 이루어진 $(m-1)\times(m-1)$ 행렬의 역행렬이 필요하고 이 역행렬을 $y^2 - b$의 계수들과 행렬 곱셈을 수행하여 x 좌표를 찾을 수 있게 된다. 따라서 연산량은 역행렬을 구하는 과정과 이 역행렬의 행렬 곱셈을 하는 과정에서 결정된다. 일단 역행렬을 찾은 뒤에는 주어진 y 좌표에 따라 행렬 곱셈만으로 x 좌표를 계산할 수 있으므로 연산량은 $O(m^2)$가 되고, 이미 계산된 역행렬은 다른 메시지 스트링의 MapToPoint 계산을 위해서 저장되어야 한다.

본 논문에서는 Barreto 등이 제안한 방법에서 사용하는 행렬 크기를 $(m-1)\times(m-1)$에서 $\lfloor (2/3)(m-1) \rfloor \times \lfloor (2/3)(m-1) \rfloor$으로 줄이는 방법을 제안한다. 이로 인하여 역행렬을 3~5배 빠르게 계산할 수 있으며 역행렬의 크기가 기존에 비해 44%로 작아졌으므로 이를 저장하기 위한 메모리 또한 기존의 44%만 필요하게 된다. 또한 역행렬과 $y^2 - b$의 계수의 행렬 곱셈 연산량도 줄어들게 되므로 결과적으로 역행렬이 주어졌을 때 MapToPoint 연산을 2~3배 정도 빠르게 할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 Barreto 등의 MapToPoint 방법을 좀 더 자세하게 서술하고 3장에서는 제안하는 MapToPoint 방법을 설명한다. 4장에서는 실험 결과를 통하여 제안하는 방법이 기존의 방법에 비하여 훨씬 연산 속도가 빠름을 보인다. 5장에서는 제안하는 MapToPoint 방법이 point compression에 적용될 수 있음을 서술하고 6장에서 결론을 맺는다.

II. 세제곱을 이용한 MapToPoint

본 장에서는 Barreto 등의 방법[15]을 간략히 서술한다. $\Gamma: F_{3^m} \to F_{3^m}$가 $\Gamma(x) = x^3 - x$로 정의될 때, Γ의 kernel이 F_3 이므로[19], Γ의 랭크는 $m-1$이 된다[13]. [15]의 MapToPoint 방법은 알고리즘 1과 같다.

알고리즘 1. MapToPoint[15]

1: $i = 0$으로 세팅한다.
2: (i, M)의 해쉬값 $h(i, M) = (y, t) \in F_{3^m} \times F_3$를 계산한다.
3: $c = y^2 - b$을 계산한다.
4: $\Gamma(x) = c$ 를 푼다.
5: 근이 없다면 i를 증가시키고 step 2로 이동한다.
6: 근이 있다면 ${}^0x, {}^1x, {}^2x$ 중에서 t를 이용하여 $({}^t x, y)$를 반환한다.

[15]에서와 같이 $u = (u_0, \cdots, u_{m-1}) \in GF(3^m)$에 대해 $\tilde{u} = (u_1, \cdots, u_{m-1})$ 그리고 $\tau(u) = u_0$으로 정의하자. 알고리즘 1의 마지막 step에서 표기된 ${}^0x, {}^1x, {}^2x$는 $\tau({}^0x) < \tau({}^1x) < \tau({}^2x)$를 기준으로 근을 구분한 것이다. 알고리즘 1의 step 4에서는 삼차방정식을 풀기위해 c의 trace $Tr(c) = c + c^3 + c^{3^2} + \cdots + c^{3^{m-1}}$ 가 0임을 확인함으로써 먼저 근이 있는지를 판단한다[19]. 근이 있다면, Γ의 랭크가 $m-1$이므로 $(m-1) \times (m-1)$ 크기의 행렬 A를 생성하고 A의 역행렬 A^{-1}를 계산한다. x 좌표는 $\tilde{x} = A^{-1} \cdot \tilde{c}$와 선택된 $x_0 \in F_3$으로 구한다.

III. 세제곱근을 이용한 MapToPoint

본 장에서는 제안하는 MapToPoint 방법을 설명한다. [15]의 방법과 달리 본 논문에서는 행렬 A를 생성하기 위해서 x의 세제곱근을 사용한다. 즉, Barreto 등은 [15]에서 $\Gamma(x) (= x^3 - x)$을 위해 x의 세제곱을 사용하여 $c = \Gamma(x)$의 근을 찾아 $\tilde{x}$을 구한다. F_{3^m}에서는 $a \mapsto a^3$이 permutation이므로 모든 원소는 유일한 세제곱근을 갖는다. 따라서 $(y^2 - b)^{1/3} = (x^3 - x)^{1/3} = x - x^{1/3}$을 만족하는 $d = (y^2 - b)^{1/3}$를 계산할 수 있다. 이 사실로부터 $\Gamma'(x) = x - x^{1/3}$으로 정의되는 함수 $\Gamma' : F_{3^m} \to F_{3^m}$을 이용하여 $d = \Gamma'(x)$을 풂으로써 $\tilde{x}$를 구한다.

$\Gamma'(x)$를 위해서 x의 세제곱이 아닌 세제곱근을 이용하는 이유는 생성 삼항기약다항식을 $f(t) = t^m + \alpha t^k + \beta$ $(\alpha, \beta \in F_3)$으로 정의하고 그 근을 s라 할 때, x의 세제곱을 계산할 때는 m과 k의 값에 따라 x의 계수가 서로 다르게 분포되지만 x의 세제곱근을 계산할 때는 $m \equiv k \pmod 3$ 인 경우에 같은 패턴으로 x의 계수가 분포되기 때문이다. 이로 인해 $\Gamma'(x)$의 행렬 표현 A를 보다 편리하게 변환할 수 있게 된다. 이어서 A의 변환 과정을 자세하게 서술한다.

$f(t) = t^m + \alpha t^k + \beta$ $(\alpha, \beta \in F_3)$의 한근을 s라 하자. f를 $F_3[t]$의 생성 삼항기약다항식이라고 할 때, $F_{3^m} = F_3[t]/(f)$라 하고 F_{3^m}에서의 세제곱근 연산을 살펴보면, 우선 임의의 자연수 u에 대하여 $m = 3u + r$ $(r \in 0, 1, 2)$라 하자. 그러면 $x \in F_{3^m}$에 대하여

$$x = \sum_{i=0}^{m-1} x_i s^i = \sum_{i=0}^{u-1+\lceil \frac{r}{2} \rceil} x_{3i} s^{3i} + s \sum_{i=0}^{u-1+\lfloor \frac{r}{2} \rfloor} x_{3i+1} s^{3i} + s^2 \sum_{i=0}^{u-1} x_{3i+2} s^{3i}$$

이고 이 식을 이용한 x의 세제곱근 연산식은 아래의 식 (1)과 같다.

$$x^{1/3} = \sum_{i=0}^{u-1+\lceil \frac{r}{2} \rceil} x_{3i} s^i + s^{1/3} \sum_{i=0}^{u-1+\lfloor \frac{r}{2} \rfloor} x_{3i+1} s^i + s^{2/3} \sum_{i=0}^{u-1} x_{3i+2} s^i. \quad (1)$$

$s^{1/3}$과 $s^{2/3}$가 사전계산 되었다면, 식 (1)을 이용하여 단지 두 번의 다항식 곱셈만으로 세제곱근을 계산할 수 있다. 이 다항식 곱셈을 할 때, $s^{1/3}$과 $s^{2/3}$의 다항식 표현이 작은 개수의 항을 갖는다면 보다 효율적으로 연산이 이루어진다. $m \equiv k \pmod 3$인 경우 $s^{1/3}$과 $s^{2/3}$의 다항식 표현이 매우 작은 개수의 항을 가지므로 가장 효율적으로 세제곱근 연산을 할 수 있다[17].

$m \equiv k \pmod 3$인 형태의 삼항 기약다항식을 cube

〔표 1〕 $f(s)=s^{13}+s^4-1$일 때, $d=\Gamma'(x)=x-x^{1/3}$의 다항식 표현

차수	s^0	s^1	s^2	s^3	s^4	s^5	s^6	s^7	s^8	s^9	s^{10}	s^{11}	s^{12}
d	d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}
x	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}
$-x^{1/3}$	$-x_0$	$-x_3$	$-x_6$	$-x_9$	$-x_{12}$	$-x_2$	$-x_5$	$-x_8$	$-x_{11}$	$-x_1$	$-x_4$	$-x_7$	$-x_{10}$
			$-x_2$	$-x_5$	$-x_8$	$-x_{11}$	x_1	x_4	x_7	x_{10}			
				$-x_1$	$-x_4$	$-x_7$	$-x_{10}$						

root-friendly trinomial이라 하며 이를 이용할 경우, 같은 값에 대하여 세제곱 보다 세제곱근 연산이 더 적은 덧셈 연산을 필요로 하게 된다〔14〕. 본 논문에서는 세제곱근 연산에 유리한 cube root-friendly trinomial을 대상으로 MapToPoint 방법을 전개하며 3.2 절에서는 non-cube root friendly irreducible trinomial에 대해서도 MapToPoint 방법을 대략적으로 기술한다.

3.1 Cube root friendly irreducible trinomials $(m\equiv k \bmod 3)$ 이용

본 절에서는 cube root-friendly trinomial을 이용하여 $\Gamma'(x)$의 행렬 표현 A를 변환하는 방법과 이것의 역행렬을 구하는 방법을 제안한다. $m\equiv 1 \pmod 3$ 또는 $m\equiv 2 \pmod 3$인 각 경우에 대하여 변환된 $\Gamma'(x)$의 행렬 표현 $\hat{A}$를 구한다.

$m\equiv k\equiv 1 \pmod 3$인 경우 : $m\equiv 1 \pmod 3$, $k\equiv 1 \pmod 3$이라 하면 적당한 양수 u와 v에 대하여 $(u>v)$ $m=3u+1$와 $k=3v+1$로 표현할 수 있다. F_{3^m}에서 $s^{3u+1}+as^{3v+1}+b=0$이므로 〔17〕에 의하여 $s^{1/3}=s^{2u+1}-\alpha s^{u+v+1}+s^{2v+1}$과 $s^{2/3}=-\beta s^{u+1}-\alpha\beta s^{v+1}$를 구할 수 있다. 따라서

$$\begin{aligned}\Gamma'(x)&=x-x^{1/3}=C-C_0-s^{1/3}\cdot C_1-s^{2/3}\cdot C_2\\&=C-C_0-C_1(s^{2u+1}-\alpha s^{u+v+1}+s^{2v+1})\\&=C-C_0-C_1(s^{2u+1}-\alpha s^{u+v+1}+s^{2v+1})\\&\quad+C_2(\beta s^{u+1}+\alpha\beta s^{v+1})\end{aligned}\quad(2)$$

이고 이때, $C=\sum_{i=0}^{3u}x_i s^i$, $C_0=\sum_{i=0}^{u}x_{3i}s^i$, $C_1=\sum_{i=0}^{u-1}x_{3i+1}s^i$

그리고 $C_2=\sum_{i=0}^{u-1}x_{3i+2}s^i$이다.

예제를 통하여 $\hat{A}$를 생성하기 위한 사전계산과정을 설명한다.

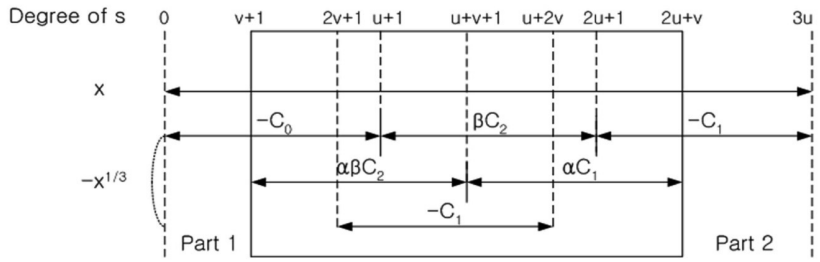
예제) 생성 삼항기약다항식이 $f(t)=t^{13}+t^4-1$이라 하고 s를 $f(t)$의 한 근이라고 하면 $u=4, v=1$이고 $s^{1/3}=s^9-s^6+s^3$과 $s^{2/3}=s^5+s^2$를 구할 수 있다. $d=\Gamma'(x)$의 계수 분포는 〔표 1〕과 같다. 〔표 1〕에서 $d_0=0$이고 각 $d_1, d_{10}, d_{11}, d_{12}$는 x의 계수 중 두 원소의 뺄셈으로 이루어졌다. 이제 이들을 제거하여 행렬 A를 변환하는 과정을 살펴본다. 먼저 $(d_0', \cdots, d_{12}')$에 $\tilde{d}$를 대입한다. x_1은 d_1, d_3, d_6, d_9에 나타나므로 $d_3'=d_3+d_1'$, $d_6'=d_6-d_1'$ 그리고 $d_9'=d_9+d_1'$을 계산하고 (d_3, d_6, d_9)를 (d_3', d_6', d_9')으로 대체함으로써 〔표 1〕에서 x_1을 삭제한다. 다음으로 x_{10}은 d_6, d_9, d_{10}, d_{12}에 나타나므로 x_{10}을 삭제하기 위해 $d_6'=d_6+d_{10}'$, $d_9'=d_9-d_{10}'$, $d_{12}'=d_{12}+d_{10}'$을 계산한다. 그리고 (d_6, d_9, d_{12})를 (d_6', d_9', d_{12}')으로 대체한다. 유사한 과정을 x_{11}과 x_{12}에 수행하면 $x_1, x_{10}, x_{11}, x_{12}$가 사라진 다음의 수식을 얻는다.

$$\begin{aligned}&d_2'=-x_6, && d_3'=-x_5-x_9,\\&d_4'=-x_4-x_8, && d_5'=-x_2+x_5+x_7,\\&d_6'=x_3-x_4-x_5+x_6, && d_7'=x_4+x_7-x_8,\\&d_8'=x_8, && d_9'=-x_3+x_4+x_9.\end{aligned}$$

따라서 $(d_2', \cdots, d_9')^T=\hat{A}\cdot(x_2, \cdots, x_9)^T$을 만족하는 변환된 행렬 $\hat{A}$을 구할 수 있고 아래의 행렬 곱셈을 통하여 $(x_2, \cdots, x_9)$을 구할 수 있다 :

$$\widehat{A^{-1}}\cdot\tilde{d}'=\begin{pmatrix}1&1&1&2&1&1&2&1\\2&1&2&0&2&0&2&1\\0&0&2&0&0&0&2&0\\1&1&0&0&1&0&0&1\\2&0&0&0&0&0&0&0\\0&0&1&0&0&1&2&0\\0&0&0&0&0&0&1&0\\2&1&0&0&2&0&0&2\end{pmatrix}\cdot\begin{pmatrix}d_2'\\d_3'\\d_4'\\d_5'\\d_6'\\d_7'\\d_8'\\d_9'\end{pmatrix}=\begin{pmatrix}x_2\\x_3\\x_4\\x_5\\x_6\\x_7\\x_8\\x_9\end{pmatrix}.$$

마지막으로 $(x_1, x_{10}, x_{11}, x_{12})$을 복원하기 위하여 단



〔그림 1〕 $m \equiv k \equiv 1 (\mathrm{mod}\ 3)$일 때 s의 차수에 따른 $\Gamma'(x) = x - x^{1/3}$의 표현

지 $x_1 = d_1 + x_3$, $x_{10} = d_{10} + x_4$, $x_{11} = d_{11} + x_7$, $x_{12} = d_{12} + x_{10}$만을 계산하면 된다. 여기서 크기가 12×12인 A의 대신에 크기가 8×8인 $\hat{A}$을 사용함으로써 역행렬 연산 시간 및 저장 공간과 행렬 곱셈 연산량을 줄일 수 있게 된다.

이제 일반적인 경우에 대하여 행렬 $\hat{A}$를 생성하는 방법을 살펴본다. 식 (2)의 차수에 따른 $\Gamma'(x)$의 도표는 〔그림 1〕과 같다. 〔그림 1〕에 따르면 *Part* 1과 *Part* 2의 각 d_i들은 x의 계수 중 두 원소의 뺄셈으로 이루어진다. 다시 말해서 *Part* 1에는 $d_i = x_i - x_{3i}\ (i = 0, \cdots, v)$, *Part* 2에는 $d_{2u+v+j} = x_{2u+v+j} - x_{3v+3j+1}\ (j = 1, \cdots, u-v)$가 존재한다. 앞서 보인 예제처럼 변환된 행렬 $\hat{A}$을 생성하기 위한 사전 계산은 $\Gamma'(x)$의 행렬 표현인 A에서 *Part* 1과 *Part* 2에 존재하는 x의 계수들을 제거하는 것이다. 예를 들어 *Part* 1의 d_1은 $x_1 - x_3$이고 x_1은 d_{2v+1}, d_{u+v+1} 그리고 d_{2u+1}에 존재한다. $(d_0', \cdots, d_{m-1}')$에 $\tilde{d}$를 대입한 후 $d_{2v+1}' = d_{2v+1} + d_1'$, $d_{u+v+1}' = d_{u+v+1} - \alpha \cdot d_1'$ 그리고 $d_{2u+1}' = d_{2u+1} + d_1'$을 계산함으로써 x_1을 삭제할 수 있다. 이러한 과정을 $(d_2, \cdots, d_v, d_{2u+v+1}, \cdots, d_{3u})$에 대하여 반복하고 $(d_1, \cdots, d_v, d_{2u+v+1}, \cdots, d_{3u})$을 $(d_1', \cdots, d_v', d_{2u+v+1}', \cdots, d_{3u}')$으로 대체한다. 알고리즘 2는 *Part* 1에 존재하는 x의 계수들을 제거하는 과정을 알고리즘으로 표현한 것이다. *Part* 2에 존재하는 x의 계수 제거 또한 매우 유사하게 이루어진다.

위의 사전계산 전에 추가적인 연산이 필요할 수 있는데, 만약 $d_1 = x_1 - x_3$과 $d_3 = x_3 - x_9$가 모두 *Part* 1에 속해있다면 우리가 x_1을 삭제하는 과정에서 x_3이 d_{2u+1}, d_{u+v+1} 그리고 d_{2v+1}에 더해지거나 빼지게 된다. 따라서 예상하지 않던 위치에 x_3이 생기게 되고 그러면 d_3을 제거하는 과정에서 x_3의 삭제가 복잡하게 된다. 이런 상황을 방지하기 위하여 $d_1' = d_1 + d_3 = x_1 - x_9$

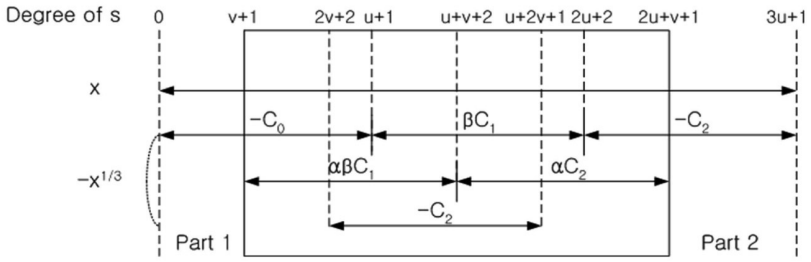
알고리즘 2. $m \equiv k \equiv 1 (\mathrm{mod}\ 3)$일 때 사전계산(*Part* 1)

입력: $\tilde{d} = (d_1, \cdots, d_{m-1})^T$
출력: $\tilde{d}' = (d_{v+1}', \cdots, d_{2u+v}')^T$

```
1: (d_1', ..., d_{m-1}')에 d̃를 대입한다.
2: for i = ⌈v/3⌉ to 1 do
3: if 3i ≤ v then
4: d_i' = d_i + d_{3i}
5: end if
6: end for
7: for i = 1 to v do
8: if i ≡ 1 (mod 3) then
9: d_{2u+i}' = d_{2u+i} + d_i'
10: d_{u+v+i}' = d_{u+v+i} − α · d_i'
11: d_{2v+i}' = d_{2v+i} + d_i'
12: else if i ≡ 2 (mod 3) then
13: d_{u−1+i}' = d_{u−1+i} − β · d_i'
14: d_{v−1+i}' = d_{v−1+i} − αβ · d_i'
15: end if
16: end for
```

을 계산한 후 사전계산을 시작한다. 사전계산 전의 이러한 추가적인 연산은 알고리즘 2의 step 2 ~ 6에 해당한다.

사전계산 후에 $\tilde{d}' = (d_{v+1}', \cdots, d_{2u+v}')$과 이들의 해당 x의 계수를 얻어 $\hat{A}$를 생성한다. 다음으로 $\widehat{A^{-1}} \cdot \tilde{d}'$을 계산함으로써 $(x_{v+1}, \cdots, x_{2u+v})$을 구한다. 마지막으로 알고리즘 3의 step 8 ~ 13에서 처럼 $(x_1, \cdots, x_v, x_{2u+v+1}, \cdots, x_{3u})$을 복원함으로써 x 의 전체 계수를 구한다. 이때, A의 크기가 $(3u+1) \times (3u+1)$인 반면 $\hat{A}$의 크기는 $2u \times 2u$로 작아, $\widehat{A^{-1}}$ 및 $\widehat{A^{-1}} \cdot \tilde{d}'$의 연산량이 〔15〕에서처럼 A^{-1} 및 $A^{-1} \cdot \tilde{c}$를 연산할 때 보다 훨씬 줄어들게 된다.



〔그림 2〕 $m \equiv k \equiv 2 \pmod 3$ 일 때 s의 차수에 따른 $\Gamma'(x) = x - x^{1/3}$의 표현

알고리즘 3. $m \equiv k \equiv 1 \pmod 3$일 때 MapToPoint

입력: $M, y, \tilde{d}'$
출력: $\tilde{x} = (x_1, \cdots, x_{m-1})^T$
1: $i = 0$으로 세팅한다.
2: (i, M)의 해쉬값 $h(i, M) = (y, t) \in F_{3^m} \times F_3$를 계산한다.
3: $d = (y^2 - b)^{1/3}$을 계산한다.
4: 사전계산 (알고리즘 2)
5: $\tilde{d}' = (d_{v+1}', \cdots, d_{2u+v}')^T$로 이루어진 행렬 $\hat{A}$를 생성한다.
6: $\widehat{A^{-1}}$를 계산한다.
7: $\widehat{A^{-1}} \cdot \tilde{d}' = (x_{v+1}, \cdots, x_{2u+v})^T$를 계산한다.
8: for $i = 1$ to v do
9: $x_i = d_i + x_{3i}$ ($Part\ 1$복원)
10: end for
11: for $i = 0$ to $u - v - 1$ do
12: $x_{2u+v+1+i} = d_{2u+v+1+i} + x_{3v+1+3i}$ ($Part\ 2$복원)
13: end for
14: 근이 없다면 i를 증가시키고 step 2로 이동
15: 근이 있다면 ${}^0x, {}^1x, {}^2x$ 중에서 t를 이용하여 $({}^t x, y)$를 반환한다.

이어서 $m \equiv k \equiv 2 \pmod 3$인 경우에 대해서도 $d = \Gamma'(x)$의 근을 찾도록 한다. 이는 앞서 설명한 과정과 유사하다.

$m \equiv k \equiv 2 \pmod 3$인 경우 : $m \equiv 2 \pmod 3$, $k \equiv 2 \pmod 3$이라 하면 임의의 양수 u와 v에 대하여 $(u > v)$ $m = 3u + 2$이고 $k = 3v + 2$라 하자. F_{3^m}에서 $s^{3u+2} + as^{3v+2} + b = 0$이므로 〔17〕에 의하여 $s^{1/3} = -\beta s^{u+1} - \alpha\beta s^{v+1}$과 $s^{2/3} = s^{2u+2} - \alpha s^{u+v+2} + s^{2v+2}$를 구할 수 있다. 따라서

$$\begin{aligned}\Gamma'(x) = x - x^{1/3} &= C - C_0 - s^{1/3} \cdot C_1 - s^{2/3} \cdot C_2 \\ &= C - C_0 - C_1(\beta s^{u+1} + \alpha\beta s^{v+1}) \\ &\quad + C_2(s^{2u+2} - \alpha s^{u+v+2} + s^{2v+2})\end{aligned}$$

이고 이때,

$$C = \sum_{i=0}^{3u+1} x_i s^i,\ C_0 = \sum_{i=0}^{u} x_{3i} s^i,\ C_1 = \sum_{i=0}^{u} x_{3i+1} s^i$$

그리고 $C_2 = \sum_{i=0}^{u-1} x_{3i+2} s^i$이다.

$m \equiv k \equiv 2 \pmod 3$일 때의 차수에 따른 $\Gamma'(x)$의 도표는 〔그림 2〕와 같으며 알고리즘 4는 $Part\ 1$ 제거 과정을 나타내고 알고리즘 5는 사전계산을 마친 후 MapToPoint 연산을 수행하는 알고리즘이다.

3.2 나머지 Irreducible Trinomials ($m \neq k \pmod 3$) 이용

본 절에서는 생성 삼항기약다항식이 cube root-friendly trinomial이 아닌 경우 $\Gamma'(x)$의 행렬 표현 A를 변환하는 방법을 대략적으로 설명한다.

$m \neq \pm k \pmod 3$인 경우에 대하여 〔20〕에서는 Shifted polynomial basis(SPB)〔5〕를 이용하여 cube root-friendly trinomial을 사용했을 때와 같은 연산량으로 세제곱근을 구할 수 있는 방법을 제안하였다. 〔20〕의 결과에 따르면 cube root-friendly trinomial을 사용했을 때와 같이 $s^{1/3}$과 $s^{2/3}$의 다항식 표현을 매우 작은 개수의 항으로 표현할 수 있으므로 3.1절의 행렬 생성과 변환 방법을 그대로 적용할 수 있다. $m \equiv -k \pmod 3$인 경우에는 $s^{1/3}$과 $s^{2/3}$의 다항식 표현이 많은 개수의 항을 가지므로 세제곱근 연산량이 비교적 크게 된다. 이는 행렬 A로부터

$\widehat{A}$를 얻는 것이 매우 복잡하기 때문에 세제곱을 이용한 방법을 이용하는 것이 유리하다는 것을 의미한다. 세제곱을 이용한다고 하면 생성 삼항 기약다항식에 따라 사전계산이 달라져야 하는 단점이 있으나 3.1절에서와 마찬가지로 행렬을 변환할 수는 있다.

알고리즘 4. $m \equiv k \equiv 2 \pmod 3$일 때 사전계산(*Part* 1)

입력: $\tilde{d}=(d_1, \cdots, d_{m-1})^T$
출력: $\tilde{d}'=(d_{v+1}', \cdots, d_{2u+v+1}')^T$
1: $(d_1', \cdots, d_{m-1}')$에 $\tilde{d}$를 대입한다.
2: for $i=\lceil v/3 \rceil$ to 1 do
3: if $3i \le v$ then
4: $d_i' = d_i + d_{3i}$
5: end if
6: end for
7: for $i=1$ to v do
8: if $i \equiv 1 \pmod 3$ then
9: $d_{u+i}' = d_{u+i} - \beta \cdot d_i'$
10: $d_{v+i}' = d_{v+i} - \alpha\beta \cdot d_i'$
11: else if $i \equiv 2 \pmod 3$ then
12: $d_{2u+i}' = d_{2u+i} + d_i'$
13: $d_{u+v+i}' = d_{u+v+i} - \alpha \cdot d_i'$
14: $d_{2v+i}' = d_{2v+i} + d_i'$
15: end if
16: end for

알고리즘 5. $m \equiv k \equiv 2 \pmod 3$일 때 MapToPoint

입력: $M, y, \tilde{d}'$
출력: $\tilde{x}=(x_1, \cdots, x_{m-1})^T$
1: $i=0$으로 세팅한다.
2: (i, M)의 해쉬값 $h(i, M)=(y, t) \in F_{3^m} \times F_3$계산
3: $d=(y^2-b)^{1/3}$을 계산한다.
4: 사전계산 (알고리즘 4)
5: $\tilde{d}'=(d_{v+1}', \cdots, d_{2u+v+1}')^T$로 이루어진 행렬 $\widehat{A}$를 생성한다.
6: $\widehat{A^{-1}}$를 계산한다.
7: $\widehat{A^{-1}} \cdot \tilde{d}'=(x_{v+1}, \cdots, x_{2u+v+1})^T$를 계산
8: for $i=1$ to v do
9: $x_i = d_i + x_{3i}$ (*Part* 1복원)
10: end for
11: for $i=0$ to $u-v-1$ do
12: $x_{2u+v+2+i} = d_{2u+v+2+i} + x_{3v+2+3i}$ (*Part* 2복원)
13: end for
14: 근이 없다면 i를 증가시키고 step 2로 이동
15: 근이 있다면 ${}^0x, {}^1x, {}^2x$ 중에서 t를 이용하여 $({}^tx, y)$를 반환한다.

IV. 실험 결과

3장에서는 $\Gamma'(x)$의 행렬 표현을 변환하여 MapToPoint 연산을 효율적으로 수행할 수 있는 방법을 제안하였다. 이때, *Part* 1과 *Part* 2에 존재하는 x의 계수들을 제거하기 위한 덧셈과 x 좌표의 일부를 복원하기 위한 덧셈은 매우 가벼운 연산이므로 MapToPoint의 연산량은 $\widehat{A}$의 역행렬 계산과 $\widehat{A^{-1}} \cdot \tilde{d}$의 행렬 곱셈에서 결정된다. Harrison 등은 F_{3^m}에서의 유한체 연산을 구현하기 적합한 원소 표현 방법 두 가지를 제안하였다[12]. 이중 Type Ⅱ 표현은 F_{2^m}에서의 원소 표현과 유사하며 각 $a_i \in F_3$을 $a_i = a_i^L - a_i^H$를 만족하는 (a_i^L, a_i^H) 두 비트로 유일하게 표현하였다. 여기서 a_i^L과 a_i^H 동시에 1이 아니다. 본 논문에서는 이 Type Ⅱ 표현 방법을 이용하여 Barreto 등의 방법과 함께 제안하는 의 초특이 타원곡선 MapToPoint 방법을 구현하였다. $F_{3^{167}}$과 $F_{3^{193}}$ 위 $E^-: y^2 = x^3 - x - 1$, $F_{3^{97}}$ 위의 초특이 타원곡선 $E^+: y^2 = x^3 - x + 1$과 [14]에 제시된 cube root-friendly trinomial : $f(t) = t^{97} - t^{16} + 1$, $f(t) = t^{167} - t^{71} + 1$, $f(t) = t^{193} - t^{64} + 1$을 이용하였다. C 언어로 Intel(R) Core(TM) i7 CPU 860 @ 2.80GHz 프로세서 환경에서 구현되었으며 역행렬 연산은 일반적인 가우스 소거법보다 약 $m/4$배 빠른 LU 분해 방법을 사용하였다[4]. [15]의 결과와 비교했을 때, 역행렬은 3 ~ 5배 가량 빨리 연산 할 수 있으며 이 역행렬을 저장하기 위해서는 3.1절에서 언급한 바와 같이 4/9(44%)의 메모리만이 필요하였다. 또한 역행렬이 주어졌을 때 MapToPoint 연산을 위한 행렬 곱셈은 2 ~ 3배 정도 빠르다. 이들의 구현 결과는 [표 3]과 같으며 각 연산 시간은 [표 2]의 연산이 수행되는 동안 측정되었다.

V. Point Compression

타원곡선 위의 점을 전송할 때 대역폭 절약을 위해 (x, y) 좌표 모두를 전송하는 대신 둘 중 하나만을 전송한 후 다른 하나는 수신자가 복원하도록 하는 방법을 point compression이라 한다. F_{2^m} 위의 타원곡선 암호에서는 수신자에게 160 비트 이상의 x좌표와

〔표 2〕 MapToPoint 연산 비교

	Barreto 등[15]	제안 방법
Mat_Inv	A^{-1} 계산	Precomputation $\widehat{A^{-1}}$ 계산
MapToPoint	y^2-b 계산 $A^{-1}\cdot\tilde{c}$ 계산	y^2-b 계산 $(y^2-b)^{1/3}$ 계산 $\widehat{A^{-1}}\cdot\tilde{d}'$ 계산 나머지 x 좌표의 계수 복원

〔표 3〕 연산 비용 비교

m	Barreto 등[15]			제안 방법		
	Mat_Inv (ms)	MapToPoint (μs)	Memory (byte)	Mat_Inv (ms)	MapToPoint (μs)	Memory (byte)
97	12.06	53.02	2304	2.58	19.65	1024
167	42.27	102.82	6889	13.79	58.46	3081
193	79.64	135.06	9216	22.58	63.00	4096

1 비트의 y 좌표 부호가 주어지고 수신자는 타원곡선 이차식을 만족하는 y 좌표를 구하게 된다. F_p 위에서는 x 좌표와 1 비트의 $y'=y \pmod 2$가 주어진다. F_{2^m}과 F_p에서의 point compression 방법은 〔6〕에 자세하게 설명되어있다.

제안하는 MapToPoint 방법에 사용된 cube root -friendly trinomial을 이용하는 기술은 F_{3^m}에서의 point compression에 효과적으로 적용할 수 있는데 이때는 y 좌표와 x_0 및 복원할 x 좌표의 부호까지 모두 3 비트가 추가로 주어져야 하며, point decompression시에 적합한 x 좌표를 연산하는데 있어 가장 빠른 알고리즘을 제공한다.

VI. 결 론

본 논문에서는 임의의 메시지 스트링을 F_{3^m} 위의 타원곡선 점으로 보다 효율적으로 대응시키는 방법을 제안하였다. 덧셈 연산이 다른 연산에 비해 훨씬 가벼우므로 그 횟수를 늘리는 대신 MapToPoint 연산을 위한 행렬의 크기를 줄였으며, 이로 인하여 저장 공간은 기존의 방법에 비해 44%로 줄이고 MapToPoint 연산은 2~3배 빠르게 연산할 수 있었다. 따라서 제안하는 MapToPoint 방법은 유비쿼터스 환경 등의 제한된 저장 공간과 컴퓨팅 파워를 갖는 장비에 효과적으로 적용가능하다.

참고문헌

〔1〕 D. Boneh and M. Franklin, "Identity based encryption from the Weil pairing," SIAM J. of Computing. vol. 32. no. 3, pp. 586-615, 2003.

〔2〕 D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," Journal of Cryptology, vol. 17, no. 4, pp. 297-319, 2004.

〔3〕 F. Hess, "Exponent group signature schemes and efficient identity based signature schemes based on pairing," SAC 2002, LNCS 2595, pp. 310-324, 2002.

〔4〕 G.H. Golub and C.F.V. Loan, "Matrix computations 3rd.," Johns Hopkins studies in the mathematical sciences. Johns Hopkins University Press, Baltimore, 1996.

〔5〕 H. Fan and Y. Dai, "Fast bit-parallel $GF(2^n)$ multiplier for all trinomials," IEEE Transactions on Computers, vol. 54, no. 4, pp.485-490, 2005.

〔6〕 IEEE P1363. Standard specifications for public key cryptography, 2000. http://gro-

uper.ieee.org/groups/1363/index.html.

〔7〕 J.C. Cha and J.H. Cheon, "An identity-based signature from gap Diffie-Hellman groups," PKC 2003, LNCS 2567, pp. 18-30, 2003.

〔8〕 J.L. Beuchat, E. López-Trejo, L. Martínez-Ramos, S. Mitsunari, and F.R. Henríquez, "Multi-core implementation on the Tate pairing over supersingular elliptic curves," CANS 2009, LNCS 5888, pp. 413-432, 2009.

〔9〕 J.L. Beuchat, J. Detrey, N. Estibals, E. Okamoto, and F. R.-Henríquez, "Fast architectures for the η_T pairing over small-characteristic supersingular elliptic curves," IEEE Transactions on Computers, vol. 60, no. 2, pp. 266-280, 2011.

〔10〕 K. Fong, D. Hankerson, J. López, and A. Menezes, "Field inversion and point halving revisited," IEEE Transactions on Computers, vol. 53, no. 8, pp. 1047-1059, 2004.

〔11〕 K.G. Paterson, "ID-based signature from pairing on elliptic curves," Electronics Letters, vol. 38, no. 18, pp. 1025-1026, 2002.

〔12〕 K. Harrison, D. Page, and N.P. Smart, "Software implementation of finite fields of characteristic three, for use in pairing-based cryptosystems," LMS Journal of Computation and Mathematics, vol. 5, pp. 181-193, 2002.

〔13〕 K. Hoffman and R. Kunze, "Linear algebra," Prentice Hall, New Jersey, USA, 2nd edition, 1971.

〔14〕 O. Ahmadi and F. R.-Henríquez, "Low complexity cubing and cube root computation over F_{3^m} in polynomial basis," IEEE Transactions on Computers, vol. 59, no. 10, pp.1297-1308, 2010.

〔15〕 P.S.L.M. Barreto and H.Y. Kim, "Fast hashing onto elliptic curves over fields of characteristic 3," Cryptology ePrint Archive, Report 2001/098.

〔16〕 P.S.L.M. Barreto, H.Y. Kim, B. Lynn, and M. Scott, "Efficient algorithms for pairing-based cryptosystems," CRYPTO 2002, LNCS 2442, pp. 354-368, 2002.

〔17〕 P.S.L.M. Barreto, "A note on efficient computation of cube roots in characteristic 3," Cryptology ePrint Archive: Report 2004/305, 2004.

〔18〕 P.S.L.M. Barreto, S.D. Galbraith, C. Ó h'Eigeartaigh, and M. Scott, "Efficient pairing computation on Supersingular Abelian Varieties," Des. Codes Cryptography 42, pp. 239-271, 2007.

〔19〕 R. Lidl and H. Niederreiter, "Finite fields," Number 20 in Encyclopedia of Mathematics and its Applications. Cambridge University Press, Cambridge, UK, 2nd edition, 1997.

〔20〕 조영인, 장남수, 김창한, 박영호, 홍석희, "페어링 암호 연산을 위한 F_{3^m}에서의 효율적인 세제곱근 연산 방법," 정보보호학회논문지, 21(2), pp. 3-11, 2011년 4월.

〔21〕 Y. Kawahara, T. Kobayashi, G. Takahashi, and T. Takagi, "Faster MapTo-Point on supersingular elliptic curves in characteristic 3," IEICE Transactions on Fundamentals, vol. E94-A, no. 1, pp. 150-155, 2011.

〈著者紹介〉

박 영 호 (Young-Ho Park) 종신회원
1990년 2월: 고려대학교 수학과 이학사
1993년 2월: 고려대학교 수학과 이학석사
1997년 2월: 고려대학교 수학과 이학박사
2002년 3월 ~ 현재: 세종 사이버 대학교 부교수
〈관심분야〉 정수론, 공개키 암호, 암호 프로토콜, 부채널 공격

조 영 인 (Young In Cho) 학생회원
2006년 2월: 한양대학교 수학과 이학사
2009년 2월: 고려대학교 정보보호 대학원 공학석사
2009년 9월 ~ 현재: 고려대학교 정보보호 대학원 박사과정
〈관심분야〉 페어링 암호, 암호칩 설계 기술, 부채널 공격, 공개키 암호 알고리즘

장 남 수 (Nam Su Chang) 정회원
2002년 2월: 서울 시립대학교 수학과 이학사
2004년 8월: 고려대학교 정보보호 대학원 공학석사
2010년 2월: 고려대학교 정보경영공학전문대학원 공학박사
2010년 7월 ~ 현재: 세종사이버대학교 전임강사
〈관심분야〉 암호칩 설계 기술, 부채널 공격, 공개키 암호 알고리즘, 공개키 암호 암호분석