情報保護學會論文誌
第16卷 第6號, 2006. 12

자원이 제약된 장치에서 효율적인 타원곡선 다중 상수배의 구현을 위한 유연한 접근*

서 석 충,† 김 형 찬, R.S. 라마크리시나‡
광주과학기술원 정보통신공학과

A Flexible Approach for Efficient Elliptic Curve Multi-Scalar Multiplication on Resource-constrained Devices

Seog Chung Seo,† Hyung Chan Kim, R.S. Ramakrishna‡
Gwangju Institute of Science and Technology,
Department of Information and Communications

요 약

타원곡선 암호시스템은 작은 키 길이로 인하여 스마트카드, 센서 모트와 같은 메모리, 계산 능력이 제약된 장치에서 사용하기에 적합하다. 본 논문에서는 이러한 장치에서 타원곡선 서명 알고리즘 검증 ($uP+vQ$, u,v: 상수, P,Q: 타원곡선 위의 점)의 주된 계산인 다중 상수배를 효율적으로 계산하기 위한 알고리즘을 제안한다. 제안 알고리즘은 부분 윈도우와 Interleave 방법에 기반을 둔 것으로서 어떠한 크기의 사전계산 테이블이라도 이용할 수 있을 뿐만 아니라, 해당 테이블에서 최적의 nonzero 밀도를 제공한다. 또한 상수 리코딩이 테이블 조회를 사용하지 않고 상수배 계산과 함께 진행되기 때문에 기존의 다른 알고리즘에 비하여 더욱 메모리를 절약할 수 있다. 실험을 통하여 163 비트의 u,v와, 233 비트의 u,v에 대하여 $uP+vQ$를 수행하는데 필요한 계산량을 사전계산 테이블의 크기에 따라 비교함으로써 최적의 테이블 크기는 각각 7, 15임을 알아낼 수 있었다.

ABSTRACT

Elliptic Curve Cryptosystem (ECC) is suitable for resource-constrained devices such as smartcards, and sensor motes because of its short key size. This paper presents an efficient multi-scalar multiplication algorithm which is the main component of the verification procedure in Elliptic Curve Digital Signature Algorithm (ECDSA). The proposed algorithm can make use of a precomputed table of variable size and provides an optimal efficiency for that precomputed table. Furthermore, the given scalar is recoded on-the-fly so that it can be merged with the main multiplication procedure. This can achieve more savings on memory than other recoding algorithms. Through experiments, we have found that the optimal sizes of precomputed tables are 7 and 15 when $uP+vQ$ is computed for u,v of 163 bits and 233 bits integers. This is shown by comparing the computation time taken by the proposed algorithm and other existing algorithms.

Keywords : *Elliptic Curve Cryptosystems, Multi-Scalar Multiplication, Fractional width-w MOF*

접수일: 2006년 8월 28일 : 채택일: 2006년 11월 15일
† 주저자, gegehe@gist.ac.kr
‡ 교신저자, rsr@gist.ac.kr

Ⅰ. 서 론

현재 유비쿼터스 (Ubiquitous) 컴퓨팅 환경은 스마트카드, 센서 모트와 같은 계산 능력과 메모리가 제약된 크기가 작은 장치들로 구성된다. 이러한 제약된 컴퓨팅 환경에서 충분한 보안성을 달성할 수 있는 암호 시스템이 필요하다. 타원곡선 암호 (Elliptic Curve Cryptosystem)는 타원 곡선상의 연산에서 정의되는 이산대수문제 (Elliptic Curve Discrete Logarithm Problem)의 어려움에 기반을 두는 암호시스템으로서 RSA나 ElGamal와 같은 기존의 공개키 암호 (Public-Key Cryptosystem)에 비하여 더 짧은 키 길이로도 비슷한 안전성을 달성할 수 있다는 장점을 가지고 있다[1,5]. 뿐만 아니라 타원곡선 암호는 키 길이의 증가에 따라 지수 (Exponential) 함수적으로 증가하는 안전성을 갖기 때문에 기존의 준지수 (Sub-exponential) 함수적인 증가의 안전도를 갖는 공개키 암호시스템에 비하여 키 길이의 증가에 따른 효율 면에서도 뛰어나다. 타원곡선 암호는 이러한 장점들로 인하여 스마트카드나 센서 모트와 같이 계산 능력과 메모리가 제한된 장치들에서 사용하기에 적합하다.

본 논문에서는 이러한 장치에서 타원곡선 서명 알고리즘 (Elliptic Curve Digital Signature Algorithm)의 검증 (Verification) 알고리즘의 주된 계산인 $uP + vQ$ (u, v는 상수, P, Q는 타원 곡선위의 점)와 같은 다중 상수배를 (Multi-scalar Multiplication) 효율적으로 구현하기 위한 알고리즘을 제안한다. 구현 알고리즘의 목표는 이러한 자원이 제약된 장치 상에서 계산 부하와 메모리 사용량을 최소화하는 것이다. 이를 위하여 여러 알고리즘들이 제안되었으나 위의 두 측면을 충분히 만족시키지 못하였다[4,7,9,10,11,12]. E. Dahmen의 알고리즘은 10개의 미리 계산된 점을 저장한 사전계산 테이블(precomputation table)을 이용하는 알고리즘들 중에서 최소의 nonzero 밀도를 제공하지만 다른 크기의 사전계산 테이블을 사용하지 못하기 때문에 확장성이 떨어진다는 단점이 있다[4]. 제안 알고리즘은 같은 조건에서 E. Dahmen의 알고리즘과 비교하여 거의 비슷한 nonzero 밀도를 제공하며, 메모리 절약과 확장성 측면에서는 보다 뛰어나다.

본 논문의 기여는 다음과 같다. 첫째, MSB로부터 LSB로 진행되는 (left-to-right) 리코딩 (recoding) 알고리즘의 사용으로 리코딩 시에 필요한 메모리의 낭비를 제거하였다. 또한 리코딩 알고리즘이 테이블 조회를 하지 않고, 가중치합 (weighted sum)을 이용하여 상수배 계산과 동시에 (on-the-fly) 계산이 되기 때문에 기존의 다른 리코딩 알고리즘들에 비하여 메모리 사용량을 더욱 줄일 수 있다. 둘째, 부분 윈도우 (fractional window)와 Interleave 방법을 결합시킴으로써 어떠한 크기의 사전계산 테이블도 사용할 수 있으며, 또한 이때에 최적의 nonzero 밀도를 제공한다. 셋째, 163 비트, 233 비트의 u, v에 대하여 제안 알고리즘과 기존의 알고리즘들이 $uP+vQ$를 수행하는데 필요한 계산량을 테이블의 크기에 따라 비교함으로써 테이블의 크기가 각각 7, 15일 때 제안 알고리즘이 기존의 알고리즘들과 비교하여 더욱 뛰어난 성능을 보임을 알 수 있었다. 넷째, [4]에서 제안한 알고리즘이 비록 크기 10의 사전계산 테이블을 사용하는 알고리즘들 중에서 가장 작은 nonzero 밀도를 제공하지만 $uP+vQ$에 대한 총 계산량을 비교할 경우에는 제안 알고리즘의 성능이 더 뛰어남을 알 수 있었다.

Ⅱ. 타원곡선 연산

본 절에서는 타원곡선 상수배의 기본이 되는 덧셈 연산 (Point Addition)과 두 배 연산 (Point Doubling)에 관하여 다룬다. 타원곡선 상에서 덧셈 연산과 두 배 연산은 여러 개의 필드연산을 이용하며 좌표계에 따라 계산되는 연산은 달라진다. 논의를 위하여 F_p (p 〉 3인 소수)상의 타원곡선이 사용되었다고 가정한다.

유한체 GF(p) (p 〉 3인 소수)상의 타원곡선은 다음과 같은 Weierstrass 방정식을 만족하는 GF(p)상의 모든 점 (x, y)와 가상의 무한원점 (Point at Infinity) O로 구성된다.

$$E: y^2 = x^3 + ax + b,\ a, b \in GF(p),\ 4a^3 + 27b^2 \neq 0$$

타원 곡선위의 해의 집합과 무한원점 O를 항등원으로 하여 타원곡선은 덧셈군을 형성한다. ECADD는 타원곡선 덧셈 연산 (Addition)을 의미하고 ECDBL은 타원곡선 두 배 연산 (Doubling)을 의미한다[1].

1) ECADD (Elliptic Curve Addition),
ECDBL (Elliptic Curve Doubling)

1. 아핀 좌표계에서의 연산

두 점 $P_1=(x_1,y_1)$과 $P_2=(x_2,y_2)$의 합 $P_1+P_2=P_3=(x_3,y_3)$은 다음과 같이 계산된다.

⊙ ECADD ($P_1 \neq \pm P_2$)

$$x_3=\lambda^2-(x_1+x_2)$$

$$y_3=\lambda(x_1-x_3)-y_1,\ where\ \lambda=\frac{y_2-y_1}{x_2-x_1}$$

⊙ ECDBL ($P_1=P_2$)

$$x_3=\lambda^2-2x_1$$

$$y_3=\lambda(x_1-x_3)-y_1,\ where\ \lambda=\frac{3x_1^{\ 2}+a}{2y_1}$$

t(A + A)와 t(2A)가 각각 덧셈 연산과 두 배 연산의 계산 시간을 나타낸다고 할 때, t(A + A) = I + 2M + S, t(2A) = I + 2M + 2S가 된다. (I = inversion, M = multiplication, S = squaring). 위의 아핀 좌표계에서의 덧셈과 두 배 연산은 유한체 상의 곱셈에 대한 역원의 계산을 요구한다. 역원의 계산은 곱셈에 비해 상당히 많은 계산량을 필요로 하기 때문에 자원이 제약된 장치에서 구현하기에 적합하지 않다. (일반적으로, GF(p)상에서 I ≥ 30M으로 가정된다[16].) 따라서 좀 더 많은 곱셈 연산을 이용하여 역원 계산을 상수배의 마지막 결과까지 미룰 수 있다. 즉, 덧셈과 두 배 연산에서 매번 역원 계산을 하는 것이 아니라, 상수배의 결과인 kP에 대해서만 역원 계산을 하면 된다. 이를 위하여 사영 좌표계 (Projective coordinate), Jacobian 좌표계, 혼합 좌표계 (Mixed coordinate)가 제안되었다[13,15].

2. 사영 좌표계에서의 연산

사영 좌표계에서는 $x=X/Z,\ y=Y/Z$ 이며 이에 따라 다음과 같이 변형된 Weierstrass 방정식을 이용한다.

$$E_P:\ Y^2Z=X^3+aXZ^2+bZ^3.$$

두 점 $P_1=(X_1,Y_1,Z_1)$과 $P_2=(X_2,X_2,Z_2)$의 합 $P_1+P_2=P_3=(X_3,Y_3,Z_3)$은 다음과 같이 계산된다.

⊙ ECADD ($P_1 \neq \pm P_2$)

$$X_3=vA,\ Y_3=u(v^2X_1Z_2-A)-v^3Y_1Z_2,$$
$$Z_3=v^3Z_1Z_2,\ where\ u=Y_2Z_1-Y_1Z_2,$$
$$v=X_2Z_1-X_1Z_2,\ A=u^2Z_1Z_2-v^3-2v^2X_1Z_2$$

⊙ ECDBL ($P_1=P_2$)

$$X_3=2hs,\ Y_3=w(4B-h)-8Y_1^2s^2,\ Z_3=8s^3,$$
$$where\ w=aZ_1^2+3X_1^2,\ s=Y_1Z_1,$$
$$B=X_1Y_1s,\ h=w^2-8B$$

t(P + P)와 t(2P)가 각각 사영 좌표계에서의 덧셈 연산과 두 배 연산의 계산시간을 나타낸다고 할 때, t(P + P) = 12M + 2S, t(2P) = 7M + 5S가 된다. 만약 $Z_2=1,\ a=-3$인 경우 t(P + A = P)와 t(2P)는 각각 9M + 2S, 7M + 3S로 줄어든다. (P + A = P는 사영 좌표계로 표현된 한 점과 아핀 좌표계로 표현된 한 점의 덧셈 결과가 사영 좌표계로 저장된다는 것을 의미한다.)

3. Jacobian 좌표계에서의 연산

Jacobian 좌표계에서는 $x=X/Z^2,\ y=Y/Z^3$ 이며, 이에 따라 다음과 같이 변형된 Weierstrass 방정식을 이용한다.

$$E_J:\ Y^2=X^3+aXZ^4+bZ^6$$

두 점 $P_1=(X_1,Y_1,Z_1)$과 $P_2=(X_2,X_2,Z_2)$의 합 $P_1+P_2=P_3=(X_3,Y_3,Z_3)$은 다음과 같이 계산된다.

⊙ ECADD ($P_1 \neq \pm P_2$)

$$X_3=-H^3-2U_1H^2+r^2,$$
$$Y_3=-S_1H^3+r(U_1H^2-X_3),\ Z_3=Z_1Z_2H,$$
$$where\ U_1=X_1Z_2^2,\ U_2=X_2Z_1^2,\ S_1=Y_1Z_2^3,$$
$$S_2=Y_2Z_1^3,\ H=U_2-U_1,\ r=S_2-S_1$$

⊙ ECDBL ($P_1=P_2$)

$$X_3=T,\ Y_3=-8Y_1^4+M(S-T),\ Z_3=2Y_1Z_1,$$
$$where\ S=4X_1Y_1^2,\ M=3X_1^2+aZ_1^4,\ T=-2S+M^2$$

〔표 1〕 좌표계에 따른 타원곡선 연산량 비교

좌표계 (Coordinate)	타원곡선 연산	
	덧셈 연산	두 배 연산
아핀	t(A + A)=I + 2M + S	t(2A)=I + 2M + 2S
사영	t(P + P)=12M + 2S	t(2P)=7M + 5S
사영 (Z_2 = 1,a =-3)	t(P +A = P)=9M + 2S	t(2P)=7M + 3S
Jacobian	t(J + J)=12M + 4S	t(2J)=4M + 6S
Jacobian (Z_2 = 1,a =-3)	t(J + A = J)=8M + 3S	t(2J)=4M + 4S

t(J + J)와 t(2J)가 각각 Jacobian 좌표계에서의 덧셈 연산과 두 배 연산의 계산시간을 나타낸다고 할 때, t(J + J) = 12M + 4S, t(2J) = 4M + 6S가 된다. 만약 a값이 -3인 경우에 $3X_1^2 + aZ_1^4$가 $3X_1^2 - 3Z_1^4 = 3(X_1 - Z_1^2)(X_1 + Z_1^2)$와 같이 1M + 1S만으로 계산할 수 있기 때문에 t(2J) = 4M + 4S로 줄일 수 있다.

4. 혼합 좌표계에서의 연산[13]

Jacobian 좌표계를 이용하는 덧셈 연산의 성능은 혼합 좌표계로 변형시킴으로써 더욱 향상될 수 있다. 즉, 한 점 $P_1 = (X_1, Y_1, Z_1)$은 Jacobian 좌표계로 표현이 되고 한 점 $P_2 = (x_2, y_2, 1)$은 아핀 좌표계로 표현될 경우에 더욱 적은 계산 량으로 덧셈 연산을 할 수 있다. 성능이 향상된 덧셈 연산은 다음과 같다.

$X_3 = F^2 - (H + 2I)$,
$Y_3 = F(I - X_3) - Y_1H$, $Z_3 = Z_1E$,
where $A = Z_1^2$, $B = Z_1A$, $C = X_2A$,
$D = Y_2B$, $E = C - X_1$, $F = D - Y_1$,
$G = E^2$, $H = EG$, $I = X_1G$.

t(J + A = J) = 8M + 3S이므로 t(J + J)와 비교하여 4번의 곱셈과 한 번의 제곱 연산을 줄일 수 있다. 각 좌표계에 따른 덧셈 연산과 두 배 연산의 계산량은 〔표 1〕에 정리되어 있다. 〔표 1〕에서 볼 수 있듯이 한 점은 아핀 좌표계로 표현되고 다른 한 점은 Jacobian 좌표계로 표현된 혼합 좌표계의 경우에 덧셈 연산과 두 배 연산의 성능이 가장 뛰어남을 확인할 수 있다. 따라서 뒤에서 기술할 제안 알고리즘에서는 계산 효율성을 위하여 아핀 좌표계로 표현된 점들로 사전계산 테이블을 구성한다.

Ⅲ. 타원곡선 상수배 알고리즘

일반적으로 타원곡선 암호 시스템의 성능은 상수배의 구현에 달려있다. 타원곡선의 상수배는 RSA의 누승 (Exponentiation)에 대응하는 것으로서 kP와 같이 표현된다. 즉, 타원곡선 위의 점 P에 대하여 상수 k만큼의 덧셈을 수행하는 것이다. 상수 k를 이진확장으로 표현할 경우 상수배는 $\sum_{i=0}^{|k|-1} k_iP,\ k_i \in 0, 1$과 같이 표현되며 이는 누승의 계산을 위한 Square-and-Multiply 알고리즘의 타원 곡선 변형인 Double-and-Add 알고리즘을 이용하여 계산할 수 있다. 상수배의 Double-and-Add를 수행하기 위하여 기본적으로 위의 Binary L-R 알고리즘과 Binary R-L 알고리즘을 사용한다.

[알고리즘 1] Binary L-R algorithm

Input : k, P
Output : $Q = kP$
1: $Q \leftarrow P$
2: for $i = |k| - 2$ *to* 0 *do*
3: $Q \leftarrow ECDBL(X)$
4: if $d[i] \neq 0$ *then*
5: $Q \leftarrow ECADD(Q, P)$
6: *return* Q

[알고리즘 2] Binary R-L Algorithm

Input : k, P
Output : $Q = kP$
1: $Q \leftarrow O$, $T \leftarrow P$
2: for $i = 0$ *to* $|k| - 1$
3: if $k[i] \neq 0$ *then*
4: $Q \leftarrow ECADD(Q, T)$
5: $T \leftarrow ECDBL(T)$
6: *return* Q

즉, 위의 두 알고리즘에서 $d[i]$가 0이면 두 배 연산만 수행되고 그렇지 않으면 두 배 연산과 덧셈 연산이 수행된다. 비록 두 알고리즘의 연산 후의 결과는 같지만, Binary L-R의 경우 Binary R-L보다 적은 변수를 사용하고 wNAF나 wMOF와 같은 슬라이딩 윈도우를 사용하는 알고리즘으로 쉽게 변형이 가능하기 때문에 더욱 선호된다. 뿐만 아니라 슬라이딩 윈도우를 사용하는 경우에 윈도우가 가질 수 있는 값들을 먼저 계산하여 사전계산 테이블을 만들어두고, 실제 계산에서의 덧셈 연산은 이 테이블의 조회를 통하여 이루어지는데 〔알고리즘 1〕과는 달리 〔알고리즘 2〕의 경우에는 변수 T가 항상 새로운 값으로 업데이트되므로 이 방법을 적용시킬 수 없다. 따라서 본 논문에서는 메모리와 계산 효율성을 위하여 Binary L-R 알고리즘의 변형들을 다루기로 한다. 위의 두 알고리즘에서 볼 수 있듯이, 매번 두 배 연산이 수행되는 것에 반해 덧셈 연산은 $d[i]$의 값이 0이 아닌 경우에만 수행된다. 따라서 덧셈 연산의 수를 줄임으로써 상수배 알고리즘의 성능을 향상시킬 수 있다. 아래의 소단원에서는 덧셈 연산의 수를 줄임으로써 상수배 알고리즘의 성능을 향상시키기 위하여 제안된 리코딩 (Recoding) 알고리즘들을 살펴보고 각각의 메모리 요구량과 계산량을 분석한다.

1. 부호화 이진 표현

타원곡선 덧셈군의 역원 연산은 매우 쉽게 계산될 수 있기 때문에 입력으로 들어온 상수가 0과 1 이외에 -1과 같은 음수 값을 갖도록 리코딩시켜 덧셈 연산의 수를 줄임으로써 상수배 알고리즘의 성능을 향상시킬 수 있다. 즉, 1^a과 같이 a번의 연속적인 덧셈을 $10^{(a-2)}\bar{1}$ ($\bar{1}=-1$)로 표현함으로써 두 번의 덧셈 연산으로 같은 계산을 수행할 수 있다. 대표적인 부호화 이진 표현으로 NAF (Non-Adjacent Form)가 있다. NAF의 특성은 어떠한 연속된 두 항에 대하여 $(1,1), (1,\bar{1}), (\bar{1},1), (\bar{1},\bar{1})$과 같이 둘 다 0이 아닌 수가 나오지 않도록 리코딩 하는 것이다. 리코딩의 결과 l비트의 상수 k는 기껏해야 $l+1$항이 있는 부호화된 이진 문자열 (signed binary string)로 변환되고, 변환된 k의 해밍 웨이트 (hamming weight)는 평균적으로 $\frac{(l+1)}{3}$이 된다. 이것은 일반적인 이진 확장 (hamming weight: $\frac{l}{2}$)에 비하여 향상된 성능을 제공한다. 예를 들어 127의 경우, 이진 전개하면 (1111111_2)가 되어 6번의 두 배 연산과 6번의 덧셈 연산을 필요로 한다. 하지만 NAF로 변환할 경우에는 $(1000000\bar{1}_2)$가 되어 7번의 두 배 연산과 2 번의 덧셈 연산을 요구하기 때문에, 4번의 덧셈 연산을 줄일 수 있다. NAF를 사용하는 Binary L-R 알고리즘은 〔알고리즘 1〕의 과정 4와 5를 변형시켜 $d[i]$의 값이 -1일 경우에 $ECADD(Q, -P)$를 수행하도록 하면 된다.

2. NAF에 적용된 슬라이딩 윈도우 방법

NAF는 기존에 존재하는 모든 부호화 이진 표현 중에서 최소의 해밍 웨이트를 제공한다[2]. 따라서, k의 이진 확장을 NAF로 리코딩한 결과에 슬라이딩 윈도우 (Sliding Window)를 적용함으로써 더욱 해밍 웨이트를 줄일 수 있다. 하지만 슬라이딩 윈도우를 적용하기 때문에 변환된 코드에서 하나의 항이 가질 수 있는 값의 범위가 더욱 커진다. 따라서 윈도우가 가질 수 있는 모든 값에 대한 테이블을 만들어 두어야 하기 때문에 계산 효율성과 메모리 소모량간의 tradeoff가 존재한다. 예를 들어 5245를 이진 확장으로 표현하면 $(1010001111101)_2$가 된다. 이것을 다시 NAF로 바꾸면 $(1010010000\bar{1}01)_2$가 된다. 이진 확장의 경우는 8번의 덧셈 연산을 필요로 하고 NAF의 경우는 5번의 덧셈 연산을 요구한다. 다시 NAF 표현에 슬라이딩 윈도우를 적용하면 $(\underline{1010}01000\underline{0\bar{1}01})$가 된다. $\underline{1010}, \underline{1000}, \underline{\bar{1}01}$ 각각은 십진수 10, 8, -3이며 상수배 계산을 시작하기 전에 이러한 값들을 계산하여 사전계산 테이블에 저장해두고 실제 연산 시에는 테이블에서 해당 값을 가져와 사용하기만 하면 되기 때문에 덧셈 연산의 수를 더욱 줄일 수 있다. NAF에 적용된 슬라이딩 윈도우 방법의 해밍 웨이트는 $\frac{l+1}{w+\frac{4}{3}+\frac{(-1)^{w+1}}{3\times 2^{w-2}}}$가 되며 사전계산 테이블의 크기는 $\frac{2^w+(-1)^{w+1}}{3}-1$ 가 된다[2,6]. 따라서 160비트의 상수에 $w=4$가 적용된 경우, 테이블의 크기는 4가 되고 덧셈 연산의 수는 약 30번 정도가 된다.

3. wNAF (Width-w Non-Adjacent Form)

Width-w NAF (wNAF) 리코딩은 NAF의 일반화로서 다음과 같은 특징을 가지고 있다.

1. 최상위 값 (0제외)은 언제나 양수이다.
2. w개의 연속적인 값들 중에서 기껏해야 하나의 값만이 0이 아니다.
3. 윈도우가 표현하는 값들은 홀수이고, 절대값으로 2^{w-1}보다 작은 값을 가진다.

부호화 이진 표현에 슬라이딩 윈도우를 적용하는 것과는 달리 wNAF 코드는 입력으로 들어온 이진 확장에 NAF리코딩의 일반화된 방법을 적용하여 직접적으로 계산될 수 있다. NAF는 w=2인 wNAF와 동일하다. wNAF는 〔알고리즘 3〕과 같이 이진 확장으로부터 $mods$ 연산자를 이용하여 계산된다. $mods$ 연산자는 부호화 나머지 (signed residue) 연산자로서 a = b $mods$ m의 결과 $-\left\lfloor\frac{m-1}{2}\right\rfloor \le a \le \left\lfloor\frac{m}{2}\right\rfloor$와 같은 특징을 갖는다. 예를 들어 14 $mods$ 5의 값은 -1이 되고, 10 $mods$ 6의 결과는 -2가 된다. 즉, 절대값이 작은 쪽이 선택된다.

〔알고리즘 3〕에 의하여 생성된 window는 $\underbrace{|00\ldots0x|}_{w}$

과 같이 윈도우의 최하위 위치에 nonzero 값이 저장된 형태이다. 따라서 w번의 연속적인 두 배 연산 후에 한 번의 덧셈 연산이 필요하다. wNAF의 평균적인 nonzero 밀도는 $\frac{1}{w+1}, (n\to\infty)$가 되며 사전계산 테이블의 크기는 $2^{w-2}-1$가 된다. 따라서, 만약 160 비트의 정수에 4NAF가 적용된 경우, 크기 3의 사전계산 테이블을 이용하여 약 32번의 덧셈 연산을 요구한다. NAF, Sliding+NAF, wNAF의 계산방향은 LSB로부터 MSB (right- to-left)로 진행된다. 이것은 상수배 알고리즘이 수행되는 것과는 정반대의 방향이다. 따라서 상수배 계산을 하기 위하여 NAF, Sliding+NAF, wNAF의 리코딩 결과가 메모리에 따로 저장되어야 한다. 예를 들어 160 비트의 정수에 대하여 리코딩를 했다면 그 결과를 저장하기 위하여 160 바이트의 메모리가 추가적으로 필요하다. 뿐만 아니라 키 크기가 커질수록 요구되는 메모리 역시 커진다. 이것은 자원이 제약된 장치에서는 큰 메모리 낭비이다. 이를 해결하기 위하여 리코딩의 방향이 MSB로부터 LSB (left-to-right)로 진행되는 알고리즘이 제안되었다[2].

[알고리즘 3] wNAF 알고리즘

Input : *width* w, *scalar* d
Output : *wNAF* $\delta_n|\delta_{n-1}|...|\delta_0$ *of* d
1 : $i\leftarrow 0$
2 : *while* $d \geq 1$ *do*
3 : if d *is even then*
4 : $\delta_i \leftarrow 0$
5 : *else*
6 : $\delta_i \leftarrow d \bmod 2^w$; $d \leftarrow d - \delta_i$
7 : $d \leftarrow d/2$; $i \leftarrow i+1$
8 : *return* $(\delta_n, \delta_{n-1}, ..., \delta_0)$

4. wMOF (Width-w Mutual-Opposite Form)

wMOF는 기존의 리코딩 알고리즘들의 단점인 상수배 알고리즘과의 수행 방향 불일치를 해결하기 위하여 제안되었다[2]. 즉, wMOF는 Binary L-R 알고리즘과 같은 상수배 알고리즘과 쉽게 병합될 수 있기 때문에 기존의 알고리즘들에서 낭비되던 메모리를 제거할 수 있다. wMOF는 MOF 코드에 MSB로부터 LSB로 슬라이딩 윈도우를 적용한 결과이며 MOF 코드는 다음과 같이 계산된다.

$$\mu = 2d \ominus d = d_{n-1}|d_{n-2} - d_{n-1}|...|d_1 - d_2|d_0 - d_1|-d_0.$$

MOF 코드에서는 근접한 0이 아닌 비트의 부호는 서로 상반되어야 하며, 또한 최상위 비트와 최하위 비트의 값은 각각 1과 $\bar{1}$이 된다. MOF의 평균적인 nonzero 밀도는 1/2 $(n\to\infty)$이 되며, 이를 wMOF로 확장시킴으로써 더욱 줄어든다. MOF 코드에서 wMOF를 계산하기 위하여 [2]에서는 테이블을 이용하였으나 [알고리즘 4]와 같이 가중치합을 이용하여 쉽게 계산을 할 수 있다. 연속한 두 비트의 값이 같을 경우에는 wMOF의 값 δ_i는 언제나 0이 된다. $d_i \neq d_{i-1}$일 경우에만 실제로 wMOF가 과정 6~13을 통하여 계산된다. wNAF에서 생성된 윈도우 $\underbrace{|00...0x|}_{w}$와는 달리 wMOF에서 생성된 윈도우는 $\underbrace{0...0}_{k}\ x\ \underbrace{0...0}_{w-k-1}$ 과 같이 nonzero x의 위치가 가변적이다. 이를 처리하기 위하여 [알고리즘 4]에서는 과정 8의 P (position)에 x의 위치를 기억시키고, 해당 위치에서 x가 적절한 값으로 바뀌도록 D (divisor)를 이용하여 S (sum)를 나눈다. 가중치합을 이용하여 MOF로부터 wMOF를 계산하는 것은 타원곡선 덧셈 및 두 배 연산에 비하여 극히 적은 계산량을 필요로 하기 때문에 [2]에서와 같이 테이블을 이용하는 방법보다 자원이 제약된 장치에 더욱 적합하다.

[알고리즘 4] 수정된 wMOF 알고리즘

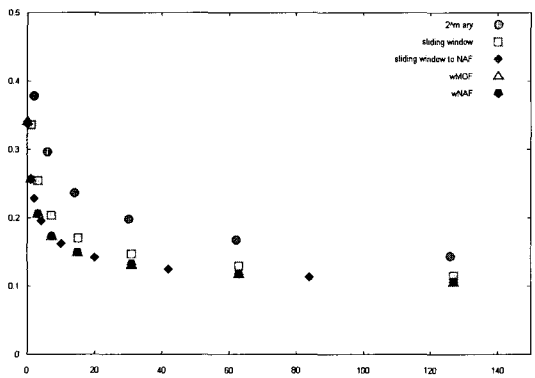
Input : *width* w, $d = d_{n-1}|d_{n-2}|...|d_1|d_0$
Output : *wMOF* $\delta = \delta_n|\delta_{n-1}|...|\delta_1|\delta_0$ *of* d
1 : $d_{-1} \leftarrow 0$; $d_n \leftarrow 0$; $i \leftarrow n-1$; $S \leftarrow 0$; $P \leftarrow 0$;
$M \leftarrow 1$; $D \leftarrow 1$
2 : *while* $i \geq 0$ *do*
3 : if $d_i = d_{i-1}$ *then*
4 : $\delta_i \leftarrow 0$; $i \leftarrow i-1$
5 : *else*
6 : if $i < w$ *then* $j = i$;
7 : *else* $j = w$;
8 : for j *to* 0
9 : if $d_{i-j} = d_{i-j-1}$ *then*
10 : $P \leftarrow i-j$; $D \leftarrow 2^{w-j}$
11 : $S \leftarrow S + M \times (d_j - d_{j-1})$
12 : $M \leftarrow M \times 2$
13 : $\delta_{i-j} \leftarrow 0$
14 : $S \leftarrow S/D$
15 : $\delta_P \leftarrow S$; $S \leftarrow 0$; $D \leftarrow 1$; $P \leftarrow 0$; $M \leftarrow 1$
16 : $i \leftarrow i-w$
17 : *return* $(\delta_n, \delta_{n-1}, ..., \delta_1, \delta_0)$

wMOF 알고리즘의 평균적인 밀도는 wNAF의 그것과 동일한 $\frac{1}{w+1}$이며 사전계산 테이블의 크기

역시 $2^{w-2}-1$로서 wNAF와 동일하다. 하지만 MSB로부터 LSB로 계산이 진행되는 특성으로 인하여 상수배 알고리즘과 통합될 수 있기 때문에, 추가적인 메모리에 대한 요구는 $O(w)$ (비교: wNAF의 경우는 $O(n)$)가 된다. 이것은 wNAF와 비교하여 낭비되는 메모리를 절약할 수 있을 뿐 아니라 입력으로 들어온 상수 k의 길이 (n)에 의존하지 않고 오직 윈도우의 크기 w에만 의존하기 때문에 메모리 측면에서 더욱 유연하다.

5. 부분 윈도우 방법 (Fractional Window Methods)

〔그림 1〕은 앞에서 제시한 윈도우에 기반을 둔 리코딩 알고리즘들의 nonzero 밀도와 사전계산 테이블 크기와의 상관관계를 나타낸다. 〔그림 1〕에서 볼 수 있듯이 윈도우의 크기가 커질수록 nonzero 밀도는 작아진다. 따라서 계산해야하는 타원곡선의 덧셈 연산의 수가 줄어들기 때문에 성능이 향상될 수 있다. 하지만 그에 따라 필요로 하는 사전계산 테이블의 크기는 지수 함수로 증가하고 있다. 두 개의 연속한 테이블 사이의 격차가 지수로 커진다는 것은 스마트카드나 센서모트, 모바일 장치와 같은 메모리 자원이 제약된 장치에서는 문제가 될 수 있다. 예를 들어 암호 알고리즘을 위하여 할당된 메모리가 50개의 미리 계산된 점을 저장할 수 있다면 기존의 알고리즘들은 이 메모리를 완전히 이용할 수 없다. 또한 낭비된 메모리는 성능의 저하로 이어진다. 부분 윈도우 방법 (Fractional Window Method)은 이러한 문제를 해결하기 위하여 제안되었다[3,6].



〔그림 1〕 리코딩 알고리즘들의 nonzero 밀도와 사전계산 테이블과의 상관관계 (x: 사전 계산 테이블 크기, y: nonzero 밀도) (만개의 랜덤 163 비트에 대한 평균)

Fractional wNAF (Frac-wNAF)는 어떠한 크기의 사전계산 테이블이라도 사용할 수 있도록 허용하는 wNAF의 일반화된 알고리즘이다. 사용이 가능한 사전계산 테이블의 크기 q로부터 Frac-wNAF를 적용하는 방법은 다음과 같다.

1. $2^{w_0-2} \le q < 2^{w_0-1}$ $(w_0 = \lfloor \log_2(q) \rfloor + 2)$를 만족하는 w_0을 찾아낸다. w_0은 일반적인 wNAF에서의 윈도우 크기이다. 이때의 사전계산 테이블의 크기는 2^{w_0-2} (기본점 포함)가 된다.
2. r을 일반적인 wNAF가 사용되었을 경우에 낭비되는 메모리의 크기라고 하자. 즉, $r = q - 2^{w_0-2}$가 된다.
3. 부분 윈도우 (fractional window) w_1는 $\frac{r}{2^{w_0-2}}$로 계산된다. 이때 r와 w_1은 $0 \le r < 2^{w_0-2}$, $0 \le w_1 < 1$을 만족한다. 이 때의 사전계산 테이블은 $\{1P, 3P, ..., (2^{w_0-1}-1)P, (2^{w_0-1}+1)P, ..., ((1+w_1)2^{w_0-1}-1)P\}$가 되고 각각의 원소들은 $\{1P, 3P, 5P, ..., (2q-1)P\}$의 원소와 일대일로 대응한다. 따라서 $q = (1+w_1)2^{w_0-2}$가 성립함을 알 수 있다. 이때 $((1+w_1)2^{w_0-1}-1)$는 Frac-wNAF에서 허용하는 한계값이 된다.

[알고리즘 5] Frac-wNAF 생성

Input : *scalar* d, *width* $w = w_0 + w_1$
where $w_0 = \lfloor w \rfloor$, $w_1 = r/2^{w_0-2}$ $r \in [0, 2^{w_0-2}-1]$
Output : *Frac-wNAF string* $(\delta_n, \delta_{n-1}, ..., \delta_1, \delta_0)$ *of* d,
where $\delta_i \in 0, \pm 1, \pm 3, ..., \pm(2^{w_0-1}(1+w_1)-1)$

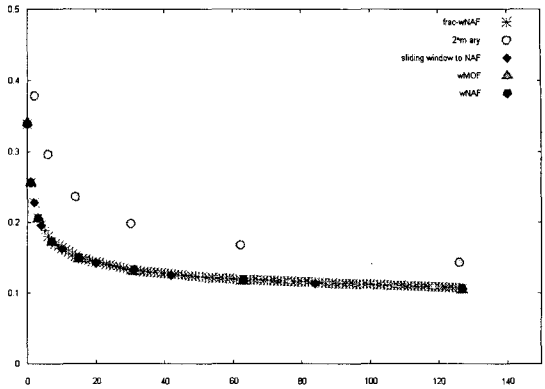
1 : $i \leftarrow 0$
2 : *while* $d > 0$
3 : if d *is odd then*
4 : $T \leftarrow d \bmod 2^{w_0+1}$
5 : if $|T| > ((1+w_1)2^{w_0-1}-1)$ *then* $\delta_i \leftarrow d \bmod 2^{w_0}$
6 : *else* $\delta_i \leftarrow T$
7 : $d \leftarrow d - \delta_i$
8 : *else* $\delta_i \leftarrow 0$
9 : $d \leftarrow d/2$; $i \leftarrow i+1$;
10 : *return* $(\delta_n, \delta_{n-1}, ..., \delta_1, \delta_0)$

〔알고리즘 5〕는 이러한 인자들을 이용하여 입력으로 들어온 상수 d에 대하여 Frac-wNAF를 계산한다. 일단 과정 4에서와 같이 w_0보다 한 치수 큰 w_0+1을 이용하여 $mods$ 연산을 수행한다. 만약 결과가 한계값 $((1+w_1)2^{w_0-1}-1)$ 보다 크면 윈도우의

크기를 한 치수 줄인 다음 다시 $mods$ 연산을 수행하고 그렇지 않으면 그대로 사용한다. (과정 5~6) Frac-wNAF의 nonzero 밀도는 $\frac{1}{w_0+w_1+1}=\frac{1}{w+1}$가 되고 사전계산 테이블의 크기는 $(1+w_1)2^{w_0-2}$가 된다. Frac-wNAF는 wNAF의 성질을 그대로 갖기 때문에 LSB에서 MSB (right-to-left)로 계산된다. 따라서 리코딩된 결과가 상수배 알고리즘을 시작하기 전에 메모리에 저장되어야 하므로 추가적인 메모리를 요구한다. 이를 해결하기 위하여 Frac-wMOF 알고리즘이 제안되었다[6]. Frac-wMOF 알고리즘은 계산 방향이 MSB에서 LSB로 진행한다는 것을 제외하면 nonzero 밀도와 사전계산 테이블의 크기가 Frac-wNAF의 그것과 완전히 동일하다. [6]에서는 Frac-wMOF 계산과 상수배 알고리즘의 진행 방향이 동일하다는 점을 이용하여 상수배 알고리즘과 Frac-wMOF가 합병된 형태의 알고리즘을 제시하였다. 하지만 MOF 코드에서 Frac-wMOF로의 변환이 테이블을 이용한 형태이기 때문에 개선의 여지가 남아있다. [표 2]는 본 절에서 제시한 알고리즘들의 nonzero 밀도와 사전계산 테이블의 크기 그리고 계산 방향을 비교한다. 각각의 알고리즘들이 가질 수 있는 테이블 크기가 다르기 때문에 효율성 측면에서는 어떤 것이 좋다고 논할 수는 없지만 메모리 측면에서는 Frac-wMOF가 다른 알고리즘에 비하여 더 큰 유연성을 제공한다. 즉, Frac-wNAF와 Frac-wMOF가 다른 알고리즘들에 비하여 자유로운 크기의 사전계산 테이블을 사용할 수 있기 때문에 메모리사용량과 계산 효율성의 tradeoff를 최적으로 조율할 수 있다. [그림 2]의 실험결과로서 이 사실을 확인할 수 있다.

[표 2] 리코딩 알고리즘의 nonzero 밀도와 사전계산 테이블의 크기

리코딩 알고리즘	nonzero 밀도	사전계산 테이블 크기	계산 방향
Sliding + NAF	$\frac{1}{w+\frac{4}{3}+\frac{(-1)^{w+1}}{3\times 2^{w-2}}}$	$\frac{2^w+(-1)^{w+1}}{3}-1$	right-to-left
wNAF	$1/(w+1)$	$2^{w-2}-1$	right-to-left
wMOF	$1/(w+1)$	$2^{w-2}-1$	left-to-right
Frac-wNAF	$\frac{1}{w_0+w_1+1}$	$(1+w_1)2^{w_0-2}-1$	right-to-left
Frac-wMOF	$\frac{1}{w_0+w_1+1}$	$(1+w_1)2^{w_0-2}-1$	left-to-right



[그림 2] 리코딩 알고리즘들의 nonzero 밀도와 사전계산 테이블과의 상관관계 (Frac-wNAF, Frac-wMOF 추가)

본 절에서는 상수배 알고리즘의 성능을 향상시키기 위한 윈도우에 기반을 둔 리코딩 알고리즘들을 살펴보았다. 이러한 알고리즘들은 다음 절에서 논할 타원 곡선 다중 상수배 계산의 성능을 높이기 위해 적용될 수 있다.

Ⅳ. 타원곡선 다중 상수배 알고리즘

타원곡선 다중 상수배는 타원곡선 서명 알고리즘의 검증과정 ($uP+vQ$, u,v는 상수, P,Q는 타원 곡선 위의 점)에서 주가 되는 계산이다. $uP+vQ$ 계산은 두 개의 상수배 계산 결과의 합의 형태로 되어 있기 때문에 각각의 상수배 계산을 일반적인 Binary L-R 알고리즘을 이용하여 순차적으로 처리한다면 $2nECDBL+(x+y+1)ECADD$ (x,y: 사용된 리코딩 알고리즘에 따른 해밍 웨이트, 1: 마지막 결과의 덧셈 연산)의 타원 곡선 연산이 필요하다. 따라서 다중 상수배 계산의 성능을 높이기 위하여 Binary L-R 알고리즘의 변형이 필요하다. 이를 위하여 Shamir 방법과 Interleave 방법이 제안되었다.

[알고리즘 6] Shamir 방법

Input: *Points* $P_j \in E(F_P)$,
$n-bits\ scalars\ d_j, j=1,...,k$

Output: $Multi-scalar\ Multiplication\ \sum_{j=1}^{k} d_jP_j$

1: $X \leftarrow O$
2: for $i=n-1$ *to* 0 *do*
3: $X \leftarrow ECDBL(X)$
4: if$(d_1[i],...,d_k[i]) \neq (0,...,0)$*then*
5: $X \leftarrow ECADD(X, d_1[i]P_1+...+d_kP_k)$
6: *return* X

[알고리즘 7] Interleave 방법

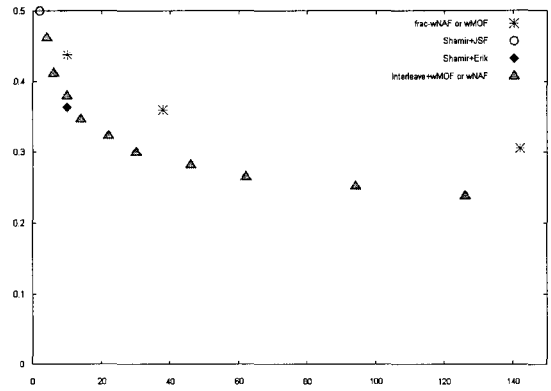
Input : *Points* $P_j \in E(F_p)$,
$n-bits\ scalars\ d_j, j = 1,...,k$

Output : $Multi-scalar\ Multiplication \sum_{j=1}^{k} d_j P_j$

1: $X \leftarrow O$
2: for $i = n-1\ to\ 0\ do$
3: $X \leftarrow ECDBL(X)$
4: for $j = 1\ to\ k\ do$
5: if $d_j[i] \neq 0\ then$
6: $X \leftarrow ECADD(X, d_j[i]P_j)$
7: $return\ X$

1. Shamir 방법

Shamir 방법의 기본 아이디어는 두 배 연산뿐만 아니라 덧셈 연산도 동시적으로 수행하는 것이다. [알고리즘 6]은 일반적인 다중 상수배 ($\sum_{j=1}^{k} d_j P_j$)를 효율적으로 계산하기 위한 Shamir 방법을 기술하고 있다. 과정 5의 덧셈 연산에서 $(d_1[i]P_1 + ... + d_k[i]P_k)$가 이용되기 때문에 Shamir 방법을 사용하기 위해서는 $(d_1[i]P_1 + ... + d_k[i]P_k)$가 가질 수 있는 모든 조합에 관한 사전계산 테이블을 필요로 한다. 예를 들어 k가 2이고 상수들이 이진확장으로 표현되었다고 가정할 때, 입력으로 들어온 P_1과 P_2를 제외한 (P_1+P_2)를 필요로 한다. 이때의 Shamir 방법은 $nECDBL + n\frac{3}{4}ECADD$ 연산을 요구한다. 이것을 k로 일반화하면 $\frac{(|D|^k - 1)}{2} - k$ ($|D|$ = digit set) 크기의 사전계산 테이블을 필요로 한다. Shamir 방법에서는 $(d_1[i], ..., d_k[i])$이 nonzero일 경우에 덧셈 연산을 수행하기 때문에 평균적으로 $nECDBL + n \cdot AJHD_k \cdot ECADD$의 연산을 필요로 한다. $AJHD_k$는 입력으로 들어온 k개의 상수에 대한 Average Joint Hamming Density를 의미한다. 따라서 Shamir 방법의 성능을 높이기 위해서는 $AJHD_k$를 최소로 해야 한다. 이를 위하여 JSF (Joint Sparse Form)와 이것의 변형 알고리즘들이 제안되었다[4,8,9]. 특히 [4]의 경우에는 w=3인 윈도우 방법에 기반을 둔 것으로서 크기가 10인 사전계산 테이블을 사용하는 다중 상수배 알고리즘 중에서 가장 작은 nonzero 밀도를 제공한다. 뿐만 아니라 LSB로부터 MSB로 계산되는 기존의 JSF와는 달리 MSB로부터 계산이 되기 때문에 메모리 절약 측면에서도 뛰어나다. 하지만 윈도우 크기 w가 커지거나 연관된 상수의 갯수 k가 커짐에 따라 사전계산 테이블의 크기가 지수함수로 증가하기 때문에 메모리에 대한 유연성과 확장성이 떨어진다는 단점이 있다. 뿐만 아니라 JSF의 확장 알고리즘들은 특정 윈도우 크기에 사용이 제한되어 있어 윈도우의 크기가 달라지면 새로운 알고리즘을 다시 만들어야 하기 때문에 자원이 제약된 장치에서는 사용하기에 적합하지 않다.



[그림 3] Shamir 방법과 Interleave 방법 비교

2. Interleave 방법

Interleave 방법의 목표는 다중 상수배 계산에 필요한 두 배 연산을 동시적으로 계산함으로써 그 수를 줄이는 것이다. 하지만 Shamir 방법과는 달리 덧셈 연산을 입력으로 들어온 각각의 상수에 대하여 독립적으로 수행한다. ([알고리즘 7]의 과정 4~6) 따라서 각각의 상수배에 대하여 다른 윈도우 크기의 다른 리코딩 알고리즘을 사용할 수 있다는 장점이 있다. 예를 들어 $d_1P_1 + d_2P_2$를 계산한다고 할 때 크기가 10인 사전계산 테이블을 사용할 수 있다고 가정하자. 사용되는 리코딩 알고리즘이 wMOF라고 한다면, d_1P_1에 대해서는 3MOF를 적용하고 d_2P_2에 대해서는 4MOF를 적용할 수 있다. 3MOF와 4MOF가 요구하는 사전계산 테이블은 각각 $\{3P_1, 5P_1, 7P_1\}$과 $\{3P_2, 5P_2, 7P_2, 9P_2, 11P_2, 13P_2, 15P_2\}$가 되므로 테이블의 총 크기는 10이 된다. 이와 같이 사용이 가능한 메모리에 대하여 적절한 리코딩 알고리즘을 선택, 조합하여 사용할 수 있기 때문에 자원이 제약된 장치에서 Shamir 방법보다 메모리 활용

측면에서 더욱 뛰어나다. 뿐만 아니라 일반적으로 같은 크기의 사전계산 테이블을 사용할 경우 Interleave 방법의 효율성이 Shamir 방법보다 뛰어나다. [그림 3]은 Shamir 방법과 Interleave 방법에 각각 기반을 둔 다중 상수배 알고리즘의 nonzero 밀도와 사전계산 테이블의 크기를 비교하고 있다. [4]에서 제시한 알고리즘은 크기가 10인 사전계산 테이블을 사용하는 다중 상수배 알고리즘들 중에서 최소의 nonzero 밀도를 보이지만 Interleave 방법에 기반을 둔 알고리즘에 비하여 확장성이 떨어진다는 것을 알 수 있다.

V. 제안 알고리즘

앞 절에서 살펴보았듯이 Shamir 방법을 사용할 경우에는 리코딩 알고리즘으로 JSF와 이의 확장 알고리즘이 적합하였다. 하지만 특정 크기의 사전계산 테이블에서만 사용할 수 있기 때문에 메모리를 낭비할 수 있다는 단점이 있었다. Interleave 방법을 사용할 경우에는 상수의 해밍 웨이트를 최소로 하는 wNAF나 wMOF와 같은 슬라이딩 윈도우 계열의 리코딩 알고리즘이 적당하다. Frac-wMOF는 어떤 크기의 사전계산 테이블도 이용할 수 있으며 또한 해당 테이블 크기에서 최적의 nonzero 밀도를 제공하기 때문에 Interleave 방법에 가장 적합한 리코딩 알고리즘이다. 따라서 본 절에서는 효율적인 상수배 계산을 위하여 Frac-wMOF에 기반을 둔 Interleave 방법을 제안한다.

1. Frac-wMOF에 기반을 둔 Interleave 방법

Interleave 방법과 Frac-wMOF는 계산 방향이 같기 때문에 두 알고리즘을 합병할 수 있다. [알고리즘 8,9,10]은 일반적인 경우의 다중 상수배 계산 ($\sum_{j=1}^{k} d_j P_j$)을 위하여 통합된 Interleave + Frac-wMOF를 제시한다. 알고리즘의 간결함을 위하여 $q = km$으로 (k: 연관된 상수배 계산, m: Frac-wMOF의 사전계산 테이블 크기), 그리고 입력으로 들어온 상수들에 대하여 같은 크기의 윈도우를 사용하는 Frac-wMOF가 사용되었다고 가정한다. [알고리즘 8]의 과정 6에서 각각의 상수배 계산에서 필요한 두 배 연산을 동시적인 한 번의 두 배 연산으로 처리하였다. δ는 상수로부터 리코딩된 Frac-wMOF 코드를 의미한다. 과정 8에서 δ에 더 이상 계산할 값이 남아있지 않고 Frac-wMOF 계산 조건 ($d_j[i] \neq d_j[i-1]$)이 충족되면 과정 9~11을 거쳐 새로운 Frac-wMOF 코드를 계산한다. 과정 9에서는 Frac-wMOF를 적용하기 위하여 입력으로 들어온 기본 윈도우 w_0보다 큰 윈도우 (w_0+1)를 설정하였다. 과정 10에서는 *comp* 함수를 통하여 w_0+1 크기의 윈도우를 적용하였을 때 Frac-wMOF가 갖는 값이 한계치를 벗어나는 가를 평가한다. *comp* 함수에서는 계산된 값이 한계치 (U)보다 클 경우에 *TRUE* 값을 반환하고 윈도우의 크기를 w_0로 줄인다. 과정 11에서는 설정된 윈도우 크기를 바탕으로 *fracwMOF* 함수를 이용하여 크기 s의 윈도우를 사용하는 Frac-wMOF 코드를 계산한다. *fracwMOF* 함수에서는 가중치합을 이용하여 입력으로 들어온 s 비트의 MOF 코드가 표현하는 총 값 (S)를 구하고 이 값이 저장될 위치 P에 $\frac{S}{2^P}$로 저장한다. 과정 12와 13에서는 계산된 Frac-wMOF 코드를 바탕으로 사전계산 테이블을 이용하여 실제 덧셈 연산을 수행한다. 과정 13에서 덧셈 연산을 하기 위하여 윈도우의 MSB인 $\delta[s-1]$를 이용하므로 덧셈 연산 후에는 윈도우에 왼쪽 쉬프트 (left shift)를 수행하여 값을 옮겨준다. 제안 알고리즘은 각각의 상수배에 대하여 같은 윈도우 크기의 Frac-wMOF가 적용되었다고 가정할 때 $\sum_{j=1}^{k} d_k P_k$를 계산하기 위하여 $k\{(1+w_1)2^{w_0-2}-1\}$ 크기의 사전계산 테이블을 이용하여 $nECDBL + \frac{kn}{w_0+w_1+1} ECADD$의 연산을 수행한다.

[알고리즘 8] Interleave 방법

Input : *Points* $P_1, ..., P_k$, n *bits scalars* $d_1, ..., d_k$,
a fractional width $w = w_0 + w_1$ *where* $w_0 = \lfloor w \rfloor$
and $w_1 = r/2^{w_0-2}$, $r \in [0, 2^{w_0-2}-1]$

Output : $\sum_{j=1}^{k} d_j P_j$

1: $d_j[n] \leftarrow 0$, $(d_j[-1], ..., d_j[-w]) \leftarrow (0, ..., 0)$, $\forall j = 1, ..., k$
2: *Compute* iP_i for *all* $i \in [3, (1+w_1)2^{w_0-1}-1]$
3: $U \leftarrow (1+w_1)2^{w_0-1}-1$
4: $X \leftarrow O$
5: for $i = n$ *to* 0 *do*
6: $X \leftarrow ECDBL(X)$

```
7:  for j = 1 to k do
8:   if δ_j = 0 and d_j[i] ≠ d_j[i−1]
9:    s ← w_0 + 1
10:   if comp(U, (d_j[i−1] − d_j[i], ..., d_j[i−s] − d_j[i−s+1]))
        then s ← w_0
11:   δ_j ← fracwMOF(d_j[i−1] − d_j[i], ..., d_j[i−s] − d_j[i−s+1])
12:   if δ_j[s−1] ≠ 0 then
13:     X ← ECADD(X, δ_j[s−1]P_j)
14:   δ_j << 1
15: return X
```

[알고리즘 9] comp 함수

```
Input: upper bound U, MOFstring (μ[s], μ[s−1], ..., μ[1], μ[0])
Output: TRUE or FALSE
1: T ← 0, M ← 1
2: for i = 0 to s do
3:   T ← T + μ[i] × M; M ← M × 2
4: if T > U then return TRUE
5: else return FALSE
```

[알고리즘 10] fracwMOF 함수

```
Input: MOFstring (μ[s], μ[s−1], ..., μ[1], μ[0])
Output: fracwMOF (δ[s], δ[s−1], ..., δ[1], δ[0])
1: S ← 0, M ← 1, P ← 0, D ← 1
2: for i = 0 to s do
3:   if μ[i] = 0 then P ← i; D ← 2^i
4:   S ← S + μ[i] × M; M ← M × 2
5:   δ[i] ← 0
6: δ[P] ← S/D
7: return (δ[s], δ[s−1], ..., δ[1], δ[0])
```

Ⅵ. 비 교

1. 확장성과 유연성 측면

〔그림 4〕는 $uP+vQ$ 계산을 위한 기존의 알고리즘들 (Shamir+JSF[8], Shamir+(wMOF or wNAF), Shamir+Erik[4], Interleave+(wMOF or wNAF)[10,11])과 제안 알고리즘 (Interleave+Frac-wMOF)의 nonzero 밀도와 사전계산 테이블 크기간의 상관관계를 표현한다. Shamir+(wNAF or wMOF)의 nonzero 밀도는 같은 크기의 테이블을 사용하는 다른 알고리즘들의 nonzero 밀도와 비교하여 상대적으로 높다. 이 결과로부터 Shamir 방법은 wNAF나 wMOF와 같은 HD (Hamming Density)를 줄이는 것을 목적으로 하는 리코딩 알고리즘 보다는 JSF 혹은JSF 확장 알고리즘과 같이 JHD (Joint Hamming Density)를 최소로 하는 것을 목표로 하는 알고리즘과 더욱 잘 어울린다는 것을 알 수 있다. 따라서 Interleave 방법은 wMOF,

〔표 3〕 테이블의 크기에 따른 $uP+vQ$의 계산량 비교 (u, v: 163 비트 랜덤 정수, 기본 단위는 M)

테이블 크기	Shamir + JSF[8] or Erik[4]			Interleave + wMOF[10,11]			Interleave+Frac-wMOF		
	사전계산	주계산	총계	사전계산	주계산	총계	사전계산	주계산	총계
0	-	-	-	0	2298.44	2298.44	0	2298.44	2298.44
2	65.6	2014.0	2079.6	66.4	2017.94	2084.34	66.4	2017.94	2084.34
4	-	-	-	-	-	-	99.2	1924.20	2023.40
6	-	-	-	132.0	1849.32	1981.32	132.0	1849.32	1981.32
7	**-**	**-**	**-**	**-**	**-**	**-**	**132.0**	**1833.54**	**1965.54**
8	-	-	-	-	-	-	164.8	1817.45	1982.25
10	328.8	1779.21	2108.01	-	-	-	197.6	1789.03	1986.63
12	-	-	-	-	-	-	230.4	1761.18	1991.58
14	-	-	-	263.2	1736.73	1999.93	263.2	1736.73	1999.93
16	-	-	-	-	-	-	296	1725.47	2021.47
18	-	-	-	-	-	-	328.3	1715.41	2044.21
20	-	-	-	-	-	-	361.6	1704.54	2066.14
22	-	-	-	-	-	-	394.4	1694.41	2088.81
24	-	-	-	-	-	-	427.2	1684.73	2111.93
26	-	-	-	-	-	-	460.0	1674.88	2134.88
28	-	-	-	-	-	-	492.8	1665.34	2158.14
30	-	-	-	525.6	1656.89	2182.49	525.6	1656.89	2182.49

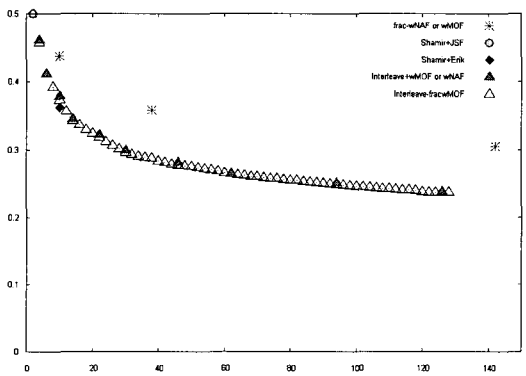
〔표 4〕 테이블의 크기에 따른 $uP+vQ$의 계산량 비교 (u, v: 233 비트 랜덤 정수, 기본 단위는 M)

테이블 크기	Shamir + JSF[8] or Erik[4]			Interleave + wMOF[10,11]			Interleave+Frac-wMOF		
	사전계산	주계산	총계	사전계산	주계산	총계	사전계산	주계산	총계
0	-	-	-	0	4100.8	4100.8	0	4100.8	4100.8
2	65.6	2889.2	2954.8	66.4	2892.41	2958.81	66.4	2892.41	2958.81
4	-	-	-	-	-	-	99.2	2759.09	2858.29
6	-	-	-	132.0	2650.89	2782.89	132.0	2650.89	2782.89
8	-	-	-	-	-	-	164.8	2605.51	2770.30
10	328.8	2553.59	2882.39	-	-	-	197.6	2564.24	2761.84
12	-	-	-	-	-	-	230.4	2526.38	2756.78
14	-	-	-	263.2	2491.14	2754.34	263.2	2491.14	2754.34
15	**-**	**-**	**-**	**-**	**-**	**-**	**263.2**	**2482.31**	**2745.51**
16	-	-	-	-	-	-	296.0	2474.68	2770.68
18	-	-	-	-	-	-	328.8	2459.30	2788.09
20	-	-	-	-	-	-	361.6	2443.98	2805.58
22	-	-	-	-	-	-	394.4	2429.37	2823.77
24	-	-	-	-	-	-	427.2	2415.27	2842.47
26	-	-	-	-	-	-	460.0	2401.74	2861.73
28	-	-	-	-	-	-	492.8	2388.74	2881.54
30	-	-	-	525.6	2376.64	2902.24	525.6	2376.64	2902.24

wNAF, Frac-wMOF 계열의 리코딩 알고리즘과 Shamir 방법은 JSF 계열의 리코딩 알고리즘과 함께 사용될 때 최적의 nonzero 밀도를 제공한다. 크기가 10인 사전계산 테이블을 사용할 경우에 〔4〕(Shamir+Erik)에서 제안한 알고리즘의 nonzero 밀도가 제안 알고리즘보다 근소하게 낮았지만 특정 크기의 테이블에서만 사용이 가능하기 때문에 확장성이 떨어진다. 그에 비하여 제안 알고리즘은 어떤 크기의 사전계산 테이블에서도 사용이 가능하고 또한 해당 테이블에서 최적의 성능 (최적의 nonzero 밀도)을 제공한다[2].

2. 성능 측면

본 소절에서는 Ⅱ절에서 논의한 타원곡선에서의 연산을 바탕으로 타원곡선 서명 알고리즘 검증의 주된 계산이 되는 $uP+vQ$를 계산하는 경우 제안 알고리즘의 성능을 기존의 다른 알고리즘들과 비교한다. 실제로 제안 알고리즘의 성능은 같은 크기의 사전계산 테이블을 사용하였을 경우에 Interleave+(wMOF or wNAF)의 성능과 같기 때문에 기존의 알고리즘들이 낭비하는 메모리를 제안 알고리즘을 통하여 낭비 없이 사용하였을 경우 얻을 수 있는 성능 향상에 대하여 논하겠다.



〔그림 4〕 기존의 방법과 제안 알고리즘의 비교

사전계산 (Precomputation)

$uP+vQ$를 계산할 때 점 P는 타원곡선 연산의 기본점 (Base Point)이고, Q는 상대방의 공개키이

2) wNAF와 wMOF은 같은 크기의 윈도우를 사용하는 부호화된 표현 (Signed representation) 중에서 최적의 nonzero 밀도를 제공한다고 증명된바 있다〔2,4,6〕. 같은 크기의 테이블을 사용할 경우 제안 알고리즘의 nonzero 밀도가 wNAF와 wMOF의 그것과 일치하는 것을 통하여 최적의 성능을 보인다고 말할 수 있다. 또한 이 사실은 〔표 2〕에서도 제시되었다.

다 ($Q=dP$). 따라서 P에 관해서는 장치를 구동하기 전에 미리 사전계산 테이블을 구성할 수 있으나, Q에 관한 사전계산 테이블은 암호 연산중에 구성해야한다. Interleave 방법의 경우 uP와 vQ의 계산이 서로 독립적으로 계산되기 때문에 Q에 관한 사전계산 테이블 만을 구성하면 된다. 반면 Shamir 방법의 경우에는 uP와 vQ가 서로 연관성을 맺고 동시에 계산되기 때문에 Interleave 방법에 비하여 사전계산 테이블을 구성하는 것에 더 많은 연산을 요구한다. 위의 사실을 바탕으로 기본점 P에 관한 연산은 제외하고 상대방의 공개키인 Q에 관한 사전계산 테이블을 구성하는데 필요한 연산만을 고려할 것이다. 실질적인 계산 과정에서 효율적인 덧셈 연산을 위하여 혼합 좌표계를 사용할 것이기 때문에 테이블에 아핀좌표계로 계산된 점들을 저장한다. ([표 1] 참조) 아핀 좌표계에서 덧셈 연산은 (I+2M+S) =(30M+2M+0.8M) =32.8M의 계산을 필요로 하고, 두 배 연산의 경우에는 (I+2M+2S) = (30M+2M+1.6M)=33.6M의 계산을 요구한다[13,15].

주계산 (Evaluation)

여러 표준에서 권고하는 타원곡선 방정식의 a 값은 효율성을 위하여 대부분 -3을 취한다[15,17]. 또한 테이블을 아핀 좌표계의 점들로 구성하였기 때문에 혼합 좌표계를 이용한 덧셈 연산이 가능하다. $Z_2=1, a=-3$인 경우에 덧셈 연산과 두 배 연산은 각각 (8M+3S)=10.4M, (4M+4S)=7.2M의 계산을 한다.

[표 3]과 [표 4]는 각각 u,v가 163 비트, 233비트의 정수인 경우에 $uP+vQ$ 연산을 수행하기 위하여 Shamir+(JSF[8] or Erik[4]), Interleave+wMOF[10,11], Interleave+Frac-wMOF (제안 알고리즘)가 필요로 하는 계산량의 변화를, 사전계산 테이블의 크기에 따라 비교한다. 사전계산은 테이블을 구성하는데 필요한 계산량을, 주계산은 미리 구성된 테이블을 이용하여 실제로 $uP+vQ$를 수행하는데 필요한 계산량을 의미한다. [표 3]과 [표 4]를 통하여 제안 알고리즘이 기존의 알고리즘에 비하여 어떠한 크기의 사전계산 테이블에도 적용이 가능하다는 것을 알 수 있다. 반면 Shamir 방법은 크기가 2와 10를 제외한 사전계산 테이블을 사용할 경우에 성능향상 없이 메모리를 낭비하는 결과를 초래한다. 비록 [4]에서 제안한 알고리즘이 크기가 10인 사전계산 테이블을 사용하는 경우에 가장 작은 nonzero 밀도를 가지기 때문에 주계산 과정에서의 계산량은 제안 알고리즘 보다 작지만, 사전계산 테이블을 구성하는데 사용된 계산량과 함께 고려할 경우 제안 알고리즘의 성능이 뛰어나다. 이때의 성능향상은 163 비트와 233비트에서 각각 5.75%, 4.18%였다. Interleave-wMOF 역시 특정 크기 이외의 테이블을 사용할 경우 성능향상 없이 메모리만 낭비하는 결과를 얻는다. 따라서 제안 알고리즘은 메모리 활용 측면에서도 기존의 알고리즘에 비하여 뛰어나며 이는 향상된 성능으로 이어진다. 163 비트, 233 비트의 u,v에 대하여 $uP+vQ$를 계산하는데 최적의 테이블 크기는 각각 7, 15이며 이때의 계산량은 1965.54M, 2745.51M이 된다. 163비트 보다 233 비트 u,v의 경우에 더 높은 성능 향상을 얻은 사실로부터 키의 길이가 늘어날수록 제안 알고리즘을 이용하여 더 높은 성능 향상을 기대할 수 있다. 반면 큰 크기의 사전계산 테이블을 사용하는 경우에는 주계산 과정의 계산량은 줄일 수 있지만 테이블 구성에 필요한 계산량이 커지기 때문에 결과적으로는 성능 저하로 이어진다.

163 비트에서 7개의 점을 저장하는 테이블의 크기는 $21\times2\times7=294$ 바이트, 233 비트에서 15개의 점을 저장하는 테이블의 크기는 $30\times2\times15=900$ 바이트의 메모리를 요구한다. 스마트카드와 센서모트의 메모리 크기는 적어도 4 킬로 바이트이기 때문에 제안 알고리즘은 충분히 위와 같은 장치에서 사용될 수 있다.

3. 가능한 성능 개선

본 논문에서 제안한 알고리즘에서는 아핀 좌표로 구성된 precomputation 테이블을 사용하기 때문에 사전계산 과정에서 여러 번의 역원 계산을 필요로 한다. Montgomery trick을 이용하여 사전계산에서 요구되는 역원 계산의 수를 줄일 수 있다[14]. Montgomery trick을 이용하여 n개의 입력에 대하여 각각에 대응하는 역원을 구하는 경우, 필요한 연산의 수는 $3(n-1)M+I$이다. $GF(p), p\geq 3$ 상에서 $I=30M$이므로 Montgomery trick을 이용하여 사전계산 테이블을 구성함으로써 사전계산에 필요한 연산의 수를 줄일 수 있다. 예를 들어, [표 3]의 Interleave+Frac-wMOF에서 크기가 7인 사전계산 테이블은 132M (=3(I+2M+S)+(I+

2M+2S))의 연산을 필요로 하나, Montgomery trick을 적용할 경우, 51M (=3(2M+S)+(2M+2S)+3(4-1)M+I)의 계산이 필요하다. Montgomery trick을 이용함으로써, 81M의 연산을 절약할 수 있으며, 이는 사전계산 테이블의 크기가 커질수록 더욱 많은 연산을 절약할 수 있다. [표 4]의 제안 알고리즘이 크기가 15인 사전계산 테이블을 이용하는 경우 263.2M의 연산을 요구하나, Montgomery trick을 이용할 경우 74.2M (=7(2M+S)+(2M+2S)+3(7-1)M+I)를 필요로 하며, 189M의 연산을 절약할 수 있다. 본 논문의 주된 목적은 부분 윈도우 방법과 Interleave 방법을 이용한 효율적인 다중 상수배 알고리즘을 제시하는 것이기 때문에 Montgomery trick은 사전계산 테이블의 계산 부하를 줄일 수 있는 방안으로서 제시한다.

V. 결 론

본 논문에서는 자원이 제약된 장치 상에서, 타원곡선 서명 알고리즘 검증과정의 주된 계산인 다중 상수배 계산 ($uP+vQ, u, v$ 상수, P, Q 타원곡선상의 점)을 효율적으로 구현하기 위한 알고리즘을 제시하였다. 제안 알고리즘은 Interleave 방법과 FracwMOF에 기반을 두었기 때문에 어떠한 크기의 사전계산 테이블도 사용할 수 있으며 해당 크기의 테이블을 이용하여 최적의 nonzero 밀도를 제공한다. 실험을 통하여 163 비트, 233 비트의 u,v에 대하여 제안 알고리즘과 기존의 알고리즘들이 $uP+vQ$를 수행하는데 필요한 계산량을 테이블의 크기에 따라 비교함으로써 테이블의 크기가 각각 7과 15 일 때 제안 알고리즘이 기존의 알고리즘들과 비교하여 더욱 뛰어난 성능을 보임을 알 수 있었다. 또한 크기 10인 사전계산 테이블을 사용하는 E. Dahmen의 알고리즘 (2108.01M, 2882.39M)과 비교하여 각각 5.75%, 4.18%의 성능 향상을 보였다. 또한 제안 알고리즘의 성능은 사전계산 과정에서 Montgomery trick을 적용함으로써 더욱 향상될 수 있다. 163 비트의 7개의 점을 저장하는 테이블의 크기는 294 바이트, 233 비트의 15개의 점을 저장하는 테이블의 크기는 900 바이트이기 때문에 제안 알고리즘은 스마트카드나 센서모트와 같은 자원이 제약된 장치에서 충분히 사용이 가능하다.

참 고 문 헌

[1] V.S. Miller, "Use of Elliptic Curves in Cryptography," CRYPTO'85, LNCS 218, pp. 417-426, 1986.

[2] K. Okeya, and et al, "Signed Binary Representation Revisited," *CRYPTO 2004, LNCS 3152*, pp. 123-139, 2004.

[3] B. Möller, "Fractional Windows Revisited: Improved Signed-Digit Representation for Efficient Exponentiation," *ICISC 2004, LNCS 3506*, pp. 137-153, 2004.

[4] E. Dahmen, and et al, "An Advanced Method for Joint Scalar Multiplications on Memory Constraint Devices," *ESAS 2005, LNCS 3813*, pp. 189-204, 2005.

[5] J. Solinas, "Efficient Arithmetic on Koblitz Curves," *Design, Codes and Cryptography*, 19:195-249, 2000.

[6] K. Schmidt-Samoa, and et al. "Analysis of Fractional Window Recoding Methods and Their Application to Elliptic Curve Cryptosystems," *IEEE Transaction on computers*, Vol. 55, 2006.

[7] Y. Sakai and K. Sakurai, "Algorithms for Efficient Simultaneous Elliptic Scalar Multiplication," *ISC 2002, LNCS 2433*, pp. 484-499, 2002.

[8] J. Solinas, "Low-Weight Binary Representations for Pairs of Integers," *CACR Technical Reports, CORR 2001-41 University of Waterloo*, 2001.

[9] B. Kuang, Y. Zhu, and Y. Zhang, "An Improved Algorithm for uP + vQ Using JSF3," *ACNS 2005, LNCS 3089*, pp. 467-478, 2004.

[10] B. Möller, "Algorithms for Multi-exponentiation," *SAC 2001, LNCS*

2259, pp. 165-180, 2001.
[11] B. Möller, "Improved Techniques for Fast Exponentiation," *ICISC 2002, LNCS 2587*, pp. 298-312, 2003.
[12] X. Ruan and R.S. Katti, "Left-to-Right Optimal Signed-Binary Representation of a Pair of Integers," *IEEE Transaction on Computers*, Vol. 54, 2005.
[13] H. Cohen, A. Miyaji, and T. Ono, "Efficient Elliptic Curve Exponentiation Using Mixed Coordinates," *ASIACRYPT'98, LNCS 1514*, pp. 51-65, 1998.
[14] K. Okeya and K. Sakurai, "Fast Multi-scalar Multiplication Methods on Elliptic Curves with Precomputation Strategy Using Montgomery Trick," *CHES 2002, LNCS 2523*, pp. 564-578, 2003.
[15] IEEE P1363: Standard Specifications for Public Key Cryptography (Draft 13) Annex A, 1999.
[16] E. De Win, et al, "On the Performance of Signature Schemes Based on Elliptic Curves," *ANTS*, pp.252-266, 1998.
[17] SEC 2-Recommended Elliptic Curve Domain Parameters. *Standards for Efficient Cryptography*. 1999.

〈著者紹介〉

서 석 충 (Seog Chung Seo) 학생회원
2005년 2월: 아주대학교 정보 및 컴퓨터공학과 졸업
2005월 3월~현재: 광주과학기술원 정보통신공학과 석사과정
<관심분야> 암호학, 타원곡선암호, 센서네트워크

김 형 찬 (Hyung Chan Kim) 비회원
2001년 8월: 경북대학교 컴퓨터과학과 졸업
2003년 8월: 광주과학기술원 정보통신공학과 석사 졸업
2003년 9월~현재: 광주과학기술원 정보통신공학과 박사과정
<관심분야> 접근통제, 보안운영체제, 프로그램 보안

R.S. 라마크리시나 (R.S. Ramakrishna) 비회원
1979년: Indian Institute of Technology(Kanpur), 전자공학과 박사
1980년~1986년: Indian Institute of Technology (Bombay), 조교수
1986년~1990년: Indian Institute of Technology (Bombay), 부교수
1990년~1996년: Indian Institute of Technology (Bombay), 교수
1996년~현재: 광주과학기술원 정보통신공학과 교수
<관심분야> 컴퓨터 그래픽스, 분산 시스템, 퀀텀 컴퓨터 이론