Normal basis를 이용한 고속 타원곡선암호(ECC)시스템의 설계

Design of High-speed Elliptic Curve Cryptosystem using normal basis.

윤 여 준, 김 종 태

성균관대학교 정보통신공학부(전화:(031)290-7173, E-mail : iunii@orgio.net)

Abstract : This paper presents new hardware implementation of the ECC(Elliptic Curve Cryptography) algorithm that is improved in speed and stability. We proposed new datapath that changed square's position so that we can reduce required number of cycles for addition operation between two points by more than 30%. We used Massey-Omura parallel multiplier adopted Normal basis for fast scalar multiplications. Also the use of the window non-adjacent form (WNAF) method can reduce addition operation of each other different points. We implemented ECC system with GF(2^{196}), and this system was designed and verified by VHDL.

Keywords : ECC, normal basis, Massey-Omura parallel multiplier

Ⅰ. 서론

타원 곡선 암호 시스템 (elliptic curve cryptosystem : ECC)은 1985년에 N. Koblitz와 V Miller에 의해 독립적으로 제안되었다. 공개키 암호시스템의 한 종류인 ECC는 타원 곡선 이산 대수 문제(elliptic curve discrete logarithm problem : ECDLP)에 기반한다. 다른 공개키 암호시스템보다 비트 당 안전도가 더 높은 ECC는 서명, 인증 등 빠른 속도와 제한된 대역폭 등이 요구되는 분야에 적용가능하다.

ECC의 기본 연산은 타원 곡선 상의 한 점을 정수배 하는 스칼라 곱셈이며, ECC 실행시 이 부분이 가장 많은 시간을 소비한다.

본 논문에서는 스칼라 곱셈기가 가장 빠른 연산을 할 수 있도록 새로운 datapath를 제안하였으며 파라미터로는 GF(2^m)형태의 유한체 GF(2^{193}), 기저로는 normal basis를 사용하였다. 또한 스칼라 곱셈의 고속화를 위해 window non-adjacent form (WNAF) method를 적용하여 설계 및 검증하고, 이를 검증하였다.

본 논문에서는 먼저 구현의 핵심이 되는 유한체 연산과 스칼라 곱셈에 대해 분석한 다음 이러한 분석을 통해 연산을 위한 효율적인 하드웨어 구조를 제안한다. 그런 다음 도출된 이론을 바탕으로 ECC 암호 시스템설계 및 구현 결과에 대해 기술한다.

Ⅱ. 유한체 연산

1. Normal basis

유한체상의 연산은 근본적으로는 동일 하지만, 체의 원소 표현법에 따라 세부적인 계산법이 달라진다. 유한체의 원소 표현법은 크게 polynomial basis representation과 normal basis representation이 있는데 하드웨어 구현 시에는 비트단위의 연산이 간단해 지는 GF(2)상의 normal basis representation이 보다 적합하다.

GF(2)상의 GF(2^m)의 normal basis는 다음과 같은 형태를 가지며, 이는 m ≥ 1 일 때 모든 m에 대해서 존재한다[1].

$$\{\beta, \beta^{2}, \ldots, \beta^{2^{m-1}}\},\ \beta \in GF(2^m)$$

GF(2)의 한 원소 A는 다음과 같은 식으로 나타낼 수 있다.

$$A = \sum_{i=0}^{m-1} \alpha_i \beta^{2^i} = a_0\beta + a_1\beta^2 + \ldots + a_{m-1}\beta^{2^{m-1}}$$

where $a_i \in GF(2),\ 0 \le i \le m-1$ (1)

A에 대한 normal basis 표현은 다음과 같다.

$$A = (a_0, a_1, \ldots, a_{m-1})$$

이렇게 Normal basis를 이용했을 때 두 원소간의 덧셈 연산은 간단히 비트별 XOR이 되며 A의 제곱 연산은 A의 좌표를 한번씩 right cyclic shift를 하는 것만으로 구현되기 때문에 빠른 연산을 구현할 수 있다.

$$A^2 = (a_{m-1}, a_0, \ldots a_{m-2})$$

$$= a_{m-1}\beta + a_0\beta^1 + \ldots + a_{m-2}\beta^{2^{m-1}} \quad (2)$$

만약 GF(2^m)이 type I ONB[1]만을 가진다면 기약다항식 f(x)은 $f(x) = x^m + x^{m-1} + \ldots + x^2 + x^m + 1$이다.

2. 유한체 가산기

유한체 GF(2m)에서의 덧셈 연산은 각각 대응하는 비트들의 XOR 연산을 통하여 구현 할 수 있다. 두 입

력에 대한 덧셈은 다음과 같이 나타낼 수 있다.

$A = (a_0, a_1, \ldots, a_{m-1})$, $B = (b_0, b_1, \ldots, b_{m-1})$일 때

$$A+B = (a_0 + b_0, a_1 + b_1, \ldots + a_{m-1} + b_{m-1}) \quad (3)$$

3. 유한체 곱셈기

유한체 GF(2m)에서 두개의 입력 A, B에 대한 곱C는 다음과 같이 나타낼 수 있다.

$$c_k = \sum_{i=0}^{m-1}\sum_{j=0}^{m-1} a_{i+k} b_{j+k} \lambda_{ij}, \quad 0 \leq k \leq m-1 \quad (4)$$

설계시 곱셈기의 구조는 속도 이득을 최대화하기 위해 Massey-Omura parallel multiplier를 사용하였다[2].

4. 유한체 역원기

임의의 체 K에서 정의된 곱셈 연산에 대한 역원은 다음과 같이 나타낼 수 있다.

$$a \times a^{-1} = 1 \text{ 단 } a \in K \quad (5)$$

따라서 a가 GF(2m)일 경우에도 이 식은 성립한다. 이를 구하기 위한 방법은 여러 가지가 있으나 이 논문에서는 제곱연산이 간단하다는 normal basis의 특징을 이용하여 역원을 구하는 방법으로 페르마의 정리를 사용하였다. 페르마의 정리는 식(6)과 같다.

$$x^{2^m} = x \quad x \in GF(2^m) \quad (6)$$

여기서,

$$x^{-1} = x^{2^m-2} = x^2 \times x^{2^2} \times x^{2^3} \times \ldots \times x^{2^{m-1}}$$

$$x \in GF(2^m),\ x \neq 0 \quad (7)$$

본 논문에서는 하드웨어 구현을 통해 연산속도를 높이는 것을 목적으로 하고 있으므로 모든 항을 병렬로 계산하는 것이 이상적이나 이렇게 할 경우 하드웨어 면적 및 지연이 지나치게 커져 구현에 적합하지 않다. 따라서 식(7)을 변경하여 몇 개의 단위항으로 유도함으로써 역원 연산을 보다 빠르게 구현하였다.

이때 m = pk+1을 만족하는 p와 k에 대해 식(7)은 다음과 같이 전개할 수 있다[3].

$$x^{-1} = \prod_{i=1}^{k=(m-1)/p} (X)^{2^i},\ X = \prod_{j=0}^{p-1} x^{2^{j(m-1)/p}} \quad (8)$$

이때 기저체가 GF(2^{193})인 경우 k=24, p=8을 사용하며 이때의 알고리즘은 그림1과 같다.

```
Input : the element of FG(2^m) that is to be
        inverted
Output : point x^-1
        A,B := x;
        For i:=1 to 7 Do
        { A := A^24; B := A · B; }
        D := B;
        For j:=1 to 24 Do
        { C := B^2; D := C · D; ]
Return D;
```

그림 1. Fermat의 정리를 이용한 역원 연산

이 경우 역원을 구하기 위해서 31사이클을 소모하게 된다.

Ⅲ. 스칼라 연산

1. Points addition

유한체 K가 타원곡선 E상에 존재 할 때 서로 다른 두 점 P = (x_1, y_1), Q = (x_2, y_2)를 더하기 위한 식은 식(9)와 같다.

$$\begin{aligned} &P = (x_1, y_1),\ Q = (x_2, y_2), \\ &E = y^2 + xy = x^3 + ax^2 + b \\ &P + Q = (x_3, y_3) \\ &x_3 = \theta^2 + \theta + x_1 + x_2 + a \\ &y_3 = \theta(x_1 + x_3) + x_3 + y_1 \\ &\theta = \frac{y_1 + y_2}{x_1 + x_2} \end{aligned} \quad (9)$$

그림 2는 Points Addition을 구하는 과정이며 역원을 구하기 위해 그림 1을 사용하였다.

```
Input : P(x1, y1), Q(x2, y2), a
Output : (x3, y3)
        r3, r4 := 1
        r1 := y1 + y2;
        r2 := x1 + x2;
        For i:=1 to 7 Do
        { r2 := r2^24;
          r3 := r3 × r2; }
        For i=1 to 24 Do
        { r3 := r3^2;
          r4 := r4 × r3; }
        θ := r1 × r4;
        r1 := θ^2;
        r1 := θ + r1;
        r2 := x1 + x2
        r2 := r1 + r2
        x3 := r2 + a;
        r1 = x1 + x3;
        r1 = r1 × θ;
        r2 = x3 + y1;
        y3 = r1 + r2;
Return (x3, y3);
```

그림 2. Points Addition

이 경우 points addition에는 총 74 사이클을 필요로 한다.

2. Point doubling

한 점 P = (x_1, y_1)를 두 배 하기 위한 식은 식 (10과 같다.

$P = (x_1, y_1)$
$E = y^2 + xy = x^3 + ax^2 + b$
$2P = P + P = (x_3, y_3)$ (10)
$x_3 = \theta^2 + \theta + a$
$y_3 = \theta(x_1 + x_3) + x_3 + y_1$
$\theta = x_1 + \frac{y_1}{x_1}$

그림 3은 Point Double을 구하는 과정이다.

Input : $P(x_1, y_1)$, a
Output : (x_3, y_3)
$r_3, r_4 := 1$
$r_1 := x_1^2;$
$r_1 := y_1 + r_1;$
$r_2 = x_1^{24}$
$r_3 := r_3 \times r_2$
For i:=1 to 6 Do
{ $r_2 := r_2^{24};$
$r_3 := r_3 \times r_2;$ }
For i=1 to 24 Do
{ $r_3 := r_3^2;$
$r_4 := r_4 \times r_3;$ }
$\theta := r_1 \times r_4;$
$r_1 := \theta^2;$
$r_1 := \theta + r_1;$
$x_3 := r_1 + a;$
$r_1 = x_1 + x_3;$
$r_1 = r_1 \times \theta;$
$r_2 = x_3 + y_1;$
$y_3 = r_1 + r_2;$
Return (x_3, y_3);

그림 3. Point double

이 경우 point doubling에는 총 72 사이클을 필요로 한다.

3. 제안된 Datapath

스칼라 연산을 위한 하드웨어 구조는 그림4와 같으며 ALU datapath의 구조는 그림5와 같다. 여기서 squarer는 barrel shift 구조로써 ^2 연산과 ^24 연산을 하도록 설계하였다.

이 datapath를 쓰는 경우 points addition의 경우 72 사이클, point doubling의 경우 70 사이클을 필요로 한다. 하지만 이 데이터 패스의 경우 multiplier의 latency가 adder, squarer의 latency보다 훨씬 크기 때문에 비효율적이다. 따라서 이 논문에서는 새로운 datapath를 제안한다. 제안된 datapath의 블록 다이어그램은 그림 3과 같다.

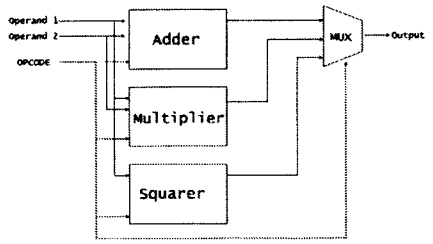

그림 4. 기존 datapath의 블록 다이어그램

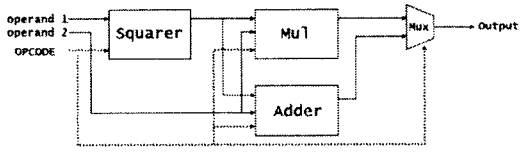

그림 5. 제안된 datapath

제안된 datapath에서는 곱셈 연산기와 덧셈 연산기 앞에 squarer 연산기에서 나온 결과를 입력으로 취함으로서 두 가지 연산을 한 클럭 내에 취할 수 있도록 하였다.

위의 Datapath를 사용하였을 때 그림 2, 그림 3은 그림 6, 그림 7과 같이 변형된다.

Input : $P(x_1, y_1)$, $Q(x_2, y_2)$, a
Output : (x_3, y_3)
$r_1 := y_1 + y_2$
$r_2 := x_1 + x_2$
$r_3 := r_2 \times r_2^{24}$
For i:=1 to 6 Do
{ $r_3 := r_3 \times r_4^{24};$ }
For i:=1 to 24 Do
{ $r_3 := r_3 \times r_4^2;$ }
$\theta := r_1 \times r_3$
$r_1 := \theta^2 + \theta$
$r_1 := r_1 + r_2$
$x_3 := r_1 + a$
$r_1 := x_1 + x_3$
$r_1 := r1 \times \theta$
$r_2 := x_3 + y_1$
$y_3 := r_1 + r_2$
Return (x_3, y_3);

그림 6. 최적화된 Points Addition algorithm

최적화된 알고리즘을 사용하였을 때 points addition과 point doubling에는 각각 41사이클, 39사이클이 필요하다.

Input : $P(x_1, y_1)$, a
Output : (x_3, y_3)

$r_1 := y_1 + x_1^2$
$r_3 := x_1 \times x_1^{24}$
For i:=1 to 6 Do
{ $r_3 := r_3 \times r_4^{24}$; }
For i:=1 to 24 Do
{ $r_3 := r_3 \times r_4^2$; }
$\theta := r_1 \times r_3$
$r_1 := \theta^2 + \theta$
$x_3 := r_1 + a$
$r_1 := x_1 + x_3$
$r_1 := r1 \times \theta$
$r_2 := x_3 + y_1$
$y_3 := r_1 + r_2$
Return (x_3, y_3);

그림 7. 최적화된 point doubling algorithm

3. WNAF method

스칼라 곱셈은 입력 값 정수 k를 변환하고, 변환값에 의해 연산을 수행하는 방법에 따라 binary method, non-adjacent form(NAF), window NAF(WNAF) 등의 3가지 방법이 있다[4]. WNAF는 스칼라 곱셈 전에 사전 연산을 통하여 k를 $NAF_w(k)$의 배열로 변환하여 사용하는 알고리즘으로 보다 빠른 연산을 할 수 있다.[5] 본 논문에서는 WNAF를 택하여 보다 빠른 연산 속도를 갖도록 하였다.

Window w=4인 경우 $NAF_w(k)$는 {0, ±1, ±3, ±5, ±7}의 값을 갖는다. 스칼라 곱셈 연산은 $NAF_w(k)$의 값에 따라 서로 다른 두 점을 더하는 points addition 연산과 같은 점을 더하는 point doubling 연산으로 이루어진다. $NAF_w(k)$의 값이 (-)인 경우 레지스터에 저장된 점 P(x, y)는 다음과 같이 변환되어 연산에 사용된다[6].

$$-(x, y) = (x, x+y) \qquad (11)$$

그림 3은 points addition과 point doubling 연산 등을 하기위한 스칼라 곱셈기 블록도이다[7].

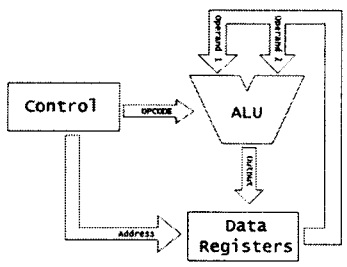

그림 8. 스칼라 연산의 블록다이어그램

Ⅳ. 결론

본 논문에서는 타원곡선 암호 시스템을 구현하기 위한 일환으로 타원 곡선 상의 점을 고속으로 연산 할 수 있는 전용의 스칼라 곱셈기를 하드웨어로 구현하였다. 스칼라 곱셈기의 고속화를 위해 normal basis를 채택 하였고 parallel 구조의 massey omura를 사용하였다. 또한 포인트 간의 연산 시에 최적화된 datapath를 사용함으로써 보다 빠른 연산을 가능케 하였다. 기존의 datapath를 사용하였을 때와 제안된 datapath를 사용하였을 때의 points addition과 point doubling의 필요 사이클을 비교한 결과를 표1에 나타내었다. 최적화된 datapath는 기존의 datapath보다 30%이상 필요 사이클 수를 줄일 수 있다. 제안된 구조는 VHDL로 설계, 검증하였다.

	기존의 datapath	제안된 datapath
Points Addition	74	41
Point Doubling	72	39

(단위 : cycle)

< 표 1 스칼라 연산에 필요한 사이클의 수 >

참고문헌

[1] I. F. Blake "Elliptic Curves in Cryptography" Cambridge University Press. 1999

[2] Reyhani-Masoleh, A. Hasan, M.A. "A new construction of Massey-Omura parallel multiplier over GF(2m)" Computers, IEEE Transactions on , Volume: 51 Issue: 5 ,Page(s): 511 -520 May 2002

[3] Sung-Ming Yen "Improved normal basis inversion in GF(2m)" Electronics Letters , Volume: 33 Issue: 3 ,Page(s): 196 -197 30 Jan. 1997

[4] M. Rosing, "Implementing Elliptic Curve Cryptography", Manning, pp 120-126, 1999

[5] 안경문, 김종태 "Window Non-Adjacent form을 스칼라 곱셈기를 이용한 타원곡선 암호 시스템 설계" 대한 전자 공학회 하계 종합 학술대회 논문집 제25권 제 1호, 2002

[6] J.Solinas, "Improved Algorithms for Arithmetic on Anomalous Binary Curves", corrected and updated version of CRYPTO'97 paper, Technical Report of CACR, University of Waterloo, 1999

[7] Leung, K.H. "FPGA implementation of a microcoded elliptic curve cryptographic processor " Field - Programmable Custom Computing Machines, 2000 IEEE Symposium on , 17-19 Page(s): 68 -76 April 2000