情 報 保 護 學 會 誌
第 16 卷 第 3 號, 2006. 6

ONB 타원곡선 연산기와 Polynomial 기저 타원곡선 연산기 구현 및 분석

최 용 제*, 김 호 원*

요 약

본 논문에서는 ONB에서의 유한체 연산 및 타원곡선 암호 연산기의 효율성을 비교하고자, Type-I, Type-II로 구분되는 ONB용 유한체 연산기와 polynomial 기저용 유한체 연산기를 구현하고 이를 비교·분석하였다. 이때 구현되는 유한체 연산기는 하드웨어 면적과 성능을trade-off 할 수 있도록 hybrid 타입의 연산기를 구현하였으며, ONB용 유한체 연산기 구현 결과를 polynomial 기저의 유한체 연산기 구현 결과와 비교하여 ONB에서의 타원곡선 연산의 효율성을 검증하였다.

Ⅰ. 서 론

타원곡선 암호 시스템은 1985년 N. Koblitz와 V. Miller에 의해 제안된 공개키 암호 시스템으로 타원곡선 상에서 이산대수의 어려움에 안전성의 근거를 두고 있다. 타원곡선 상에서 이산대수 문제는 RSA 공개키 암호 시스템의 안전성의 근거인 소인수 분해 문제보다 더욱 어려운 것으로 알려져 있다. 암호 시스템의 안전성문제는 시스템에서 사용되는 키 길이와 밀접한 관계가 있으며, 암호 시스템의 안전성이 높을수록 짧은 키 값으로도 충분한 안전성을 보장하게 된다. 실제로 233 비트 타원곡선 암호 시스템은 2,048 RSA 암호 시스템정도의 안전성을 보장하는 것으로 알려져 있다[1].

타원곡선 암호 시스템에서는 prime field, binary field 등의 유한체가 주로 사용된다. Prime field에서 사용되는 유한체 연산은 RSA에서 이용되는 연산과 유사하다. Binary field에서 사용되는 유한체 연산은 기저의 표현에 따라 달라지며, 주로 사용되는 기저로는 polynomial 기저와 normal 기저가 있다. 하지만, 실제 타원곡선 암호 시스템에서는 일반적인 기저들은 연산의 복잡성 때문에 사용되지 않으며, polynomial 기저의 경우에는 trinomial이나 pentanomial을 최소다항식으로 가지는 특수한 경우가 주로 사용되며, normal 기저의 경우는 ONB (Optimal Normal Basis)와 같은 특수한 경우가 주로 사용된다. 특히 ONB에서의 유한체 연산은 곱셈 연산이 효율적으로 수행되고, 제곱 연산이 단순 쉬프트 연산으로 수행되며, 역승산 연산을 곱셈 연산의 반복 수행으로 수행할 수 있는 이점이 있어 polynomial 기저에서의 유한체 연산보다 효율적인 것으로 알려져 있다.

본 논문에서는 ONB에서의 유한체 연산 및 타원곡선 암호 연산기와 polynomial 기저에서의 유한체 연산과 타원곡선 암호 연산기의 효율성을 비교하고자, Type-I, Type-II로 구분되는 ONB용 유한체 연산기와 polynomial 기저 유한체 연산기를 설계하고 이를 이용한 타원곡선 암호 연산기의 성능을 예측해본다. 구현되는 유한체 연산기는 하드웨어 면적과 성능을 trade-off 할 수 있도록 hybrid 타입의 연산기를 구현한다.

본 논문은 다음과 같이 구성되어 있다. 세션2에서는 각 기저에서의 유한체 연산기를 설계하고, 세션3에서는 구현된 유한체 연산기를 이용한 타원곡선 암호 연산기 설계 및 이들의 설계 결과와 비교하며, 세션 4에서 결론으로 끝맺는다.

Ⅱ. ONB 및 polynomial 기저에서의 유한체 연산기 구현

타원곡선 암호 연산에 필요한 유한체 연산은 덧셈,

* 한국전자통신연구원 정보보호연구단 ({choiyj.khw}@etri.re.kr)

곱셈, 제곱, 역승산 등이 있다. 본 장에서는 구현하고자 하는 ONB 기저와 이에 따른 유한체 연산 구현을 살펴본다.

1. ONB 기저 및 덧셈, 제곱 연산

GF(2^m)에 대한 normal basis 표현은 다음과 같다.

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

Normal basis는 임의의 양의 정수 m에 대해 항상 존재하며, GF(2m)에서의 연산이므로 덧셈 연산은 XOR로 구현된다. 또한 제곱 연산은 다음과 같다.

$$\{\beta, \beta^2, \beta^{2^2}, \cdots, \beta^{2^{m-1}}\}^2 = \{\beta^2, \beta^{2^2}, \cdots, \beta^{2^{m-1}}, \beta\}$$

위에서 보는 바와 같이 normal basis에서 제곱 연산은 순환 쉬프트 연산으로 구현되며, 이의 하드웨어 구현은 신호선 매핑으로 구현되어 추가적인 하드웨어 연산 로직은 필요하지 않는다. 이러한 normal basis의 유한체 제곱 연산 은 제곱 연산이 빈번하게 사용되는 타원곡선 암호 연산 구현시 시스템의 효율성을 향상시키는 요인이 된다.

이와 같이 normal basis에서의 제곱 연산은 매우 효율적이지만, 두 원소의 곱 연산은 그렇지 못하다. 예를 들어 b와 b2의 곱은 b3이 되지만, b3은 주어진 field에 존재하지 않아 이를 다시 주어진 field 내의 원소로 변환하는 연산이 필요하다. Normal basis에는 이러한 곱셈 연산을 보다 효율적으로 수행할 수 있는 Gaussian normal basis(GNB) class가 존재하는데, GNB가 되기 위한 필요충분 조건은 다음과 같다.

Type T GNB가 존재한다 ⟷
P = Tm + 1는 소수이고, gcd(Tm/k, m) = 1 이다.
(k는 modulo p에서의 2에 대한 곱셈 차수임)

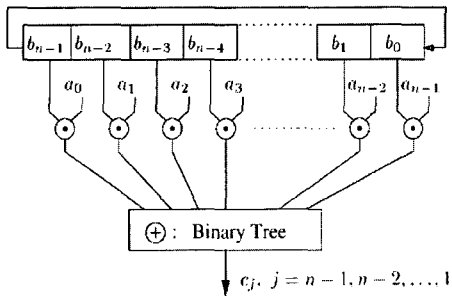
이때, 작은 type T에서 보다 효율적인 곱셈 연산을 수행할 수 있으며, 특히 type 1, type 2인 GNB는 모든 normal basis중에서 가장 효율적으로 곱셈 연산을 수행할 수 있어 이를 optimal normal basis(ONB)라고 부른다. 그리고, Type 1인 GNB를 type-I ONB라 하고, type 2인 GNB를 type-II ONB라 한다.

2. ONB 곱셈 연산

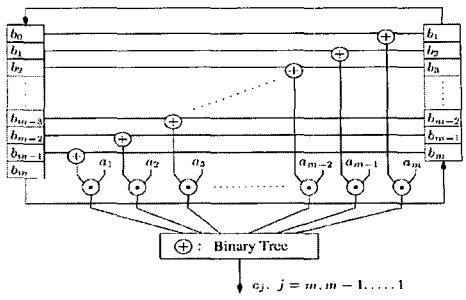
일반적인 normal 기저를 위한 곱셈 연산은 Massey-Omura[2] 곱셈 기법으로 구현된다. Massey-Omura 곱셈 기법에 의한 곱셈 연산기의 복잡도는 field를 생성하는 irreducible polynomial에 의해 결정된다. Type-I과 Type-II ONB에 대해서는 좀더 효율적인 구현이 가능한데, type-I의 경우Massey-Omura 곱셈 기법을 개선한 Modified Massey-Omura 곱셈 기법이 Hasan, Wang, Bhargava에 의해 제안되었다[3]. 이는 m^2개의 AND 게이트와 m^2-1개의 XOR 게이트로 구현되며, 연산 소요 시간은$T_A+(1+\log_2(m-1))T_X$이다. Type-II에 대한 최적화된 병렬 곱셈기는 Sunar와 Koc에 의해 제안되었으며, 이는 m^2개의 AND 게이트와 m(m-1)*3/2개의 XOR 게이트로 구현되며, 연산 소요 시간은 $T_A+(1+\log_2 m)T_X$이다[4]. 이때, T_A는 AND 게이트의 연산 지연 시간이며, T_X는 XOR 게이트의 연산 지연 시간이다.

본 논문에서 구현하고자 하는 ONB에 대한 hybrid 형태의 최적화된 곱셈기는 [5]에서 Wu, Hasan, Blake 에 의해 제안되었다. [5]에서 제안된 곱셈 기법은 기저를 연산이 용이한 redundant 기저로 변환하여 최적화된 곱셈 연산을 수행한 후, 연산 결과를 다시 ONB 기저로 변환한다. 기저 변환은 단순 wiring으로 구현되므로, 하드웨어 구현시 추가적인 로직을 필요로 하지 않는다. Redundant 기저로 변환한 후, type에 따라 그림 1과 그림 2와 같은 연산으로 결과를 계산한다. 이과 같은 연산 방법을 이용하여 k번의 반복 연산으로 곱셈 연산을수행하는 경우 필요한 논리 게이트 수는 type-I ONB는 km개의 AND 게이트와 (m-1)k개의 XOR 게이트이며, 이때의 연산 소요 시간은$m/k(T_A+(\log_2 m)T_X)$이다. 또한 Type-II ONB의 경우에는 km개의 AND 게이트와 (2m-1)k개의 XOR 게이트가 필요하며, 이때의 연산 소요 시간은 $m/k(T_A+(1+\log_2 m)T_X)$이다.

이의 하드웨어 구현은 그림 3과 같은 구조로 구현할 수 있다. 그림에서 비트 곱셈 연산 모듈은 그림 1과 2의 논리 연산을 수행하며, 한비트의 출력값을 계산한다. 그림에서 보는 바와 같이 이 비트 곱셈 연산 모듈을 필요에 따라 확장함으로서 성능과 하드웨어 면적을

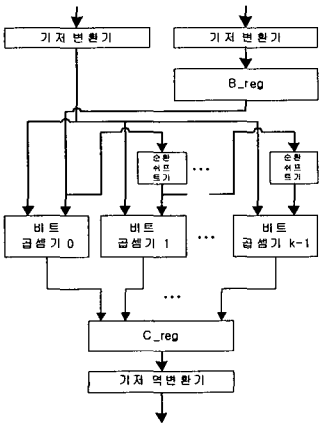


〔그림 1〕 Redundant 기저에서 곱셈 연산(Type-I ONB)



〔그림 2〕 Redundant 기저에서 곱셈 연산(Type-II ONB)

trade-off 할 수 있다.



〔그림 3〕 ONB용 hybrid 유한체 곱셈 연산기 구조

위에서 언급한 바와 같이 기저 변환기는 ONB 기저를 redundant 기저로 변환하는 연산을 수행하며, 다음과 같은 방법으로 각 계수를 변환한다.

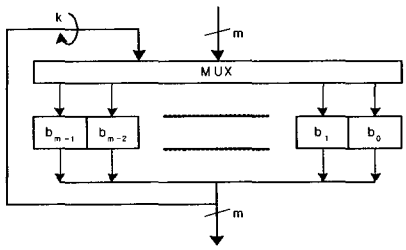
$$\text{ONB} \quad \longleftrightarrow \quad \text{Redundant Basis}$$

$$\{\beta, \beta^2, \beta^{2^2}, \cdots \beta^{2^{i-1}}, \cdots, \beta^{2^{m-1}}\} \leftrightarrow \{\beta, \beta^2, \beta^3, \cdots \beta^k, \cdots, \beta^m\}$$

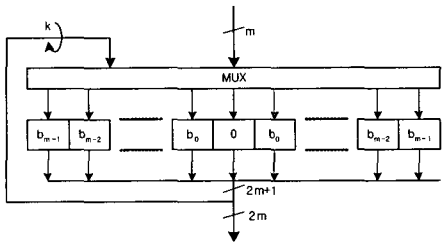
$$k \equiv 2^{i-1} (\text{mod} 2m+1), (i = 1,2,\cdots,m)$$

이러한 연산은 field가 결정되면 사전에 결정되며, 단순한 신호선의wiring 구현되어 추가적인 논리 연산기는 필요하지 않는다.

B_reg는 승수 입력값을 ONB type에 따라 Type-I ONB는 입력을 그대로 저장하며, type-II ONB는 입력을 확장하여 저장한다. Type에 따른 레지스터는 각각 그림 4와 그림 5와 같은 구조로 구현된다. 입력단의 MUX는 초기 입력값 m과 중간 연산에 필요한 k비트 순환 쉬프트 되는 내부 연산값을 선택한다. 저장된 승수값은 곱셈 연산이 반복 수행되면서 k 비트씩 쉬프트 되어 출력된다.



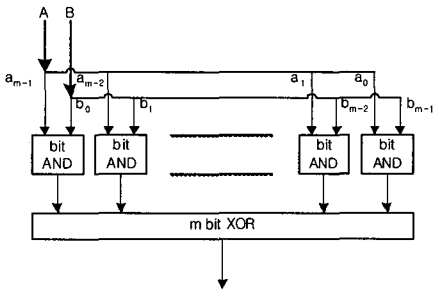
〔그림 4〕 Type-I ONB B_reg 구조



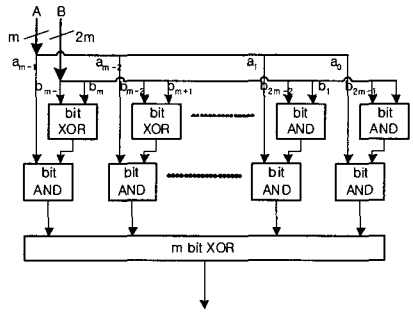
〔그림 5〕 Type-II ONB B_reg 구조

비트 곱셈연산 블록은 그림 1과 그림 2의 로직 연산을 수행하며, ONB type에 따라 그림 6과 그림 7과 같은 구조로 구현된다. 그림에서 보는 바와 같이 type-II ONB 연산에는 한 단의 m개의 AND 연산기들이 더 필요하다.

순환 쉬프트 연산기는 단순히 한 비트씩 순환 쉬프트 연산을 수행하는 블록으로 신호선의 wiring으로 구현되어 추가적인 하드웨어 로직은 필요하지 않는다. C_reg는 k 비트씩 계산되는 곱셈 연산 중간값들을 저장하여 최종 곱셈 연산 출력을 위한 레지스터이다. 기



〔그림 6〕 Type-I ONB 비트 곱셈 연산기



〔그림 7〕 Type-II ONB 비트 곱셈 연산기

저 역변환 연산기는 곱셈 연산 결과를 redundant 기저에서 다시 ONB 기저로 변환하는 연산을 수행하며, 기저 변환기의 연산을 역으로 수행하며, 역시 추가적인 로직없이 신호선의 wiring으로 구현된다.

3. ONB 연승산 연산

Normal 기저에서의 역승산 연산은 type에 상관없이 IT(Itoh and Tsujii) 알고리즘으로 효율적으로 구현될 수 있다[6]. IT 알고리즘은 다음과 같이 역승산 연산을 곱셈 연산과 제곱 연산을 반복 사용함으로써 수행하며, normal 기저에서는 제곱 연산이 단순 쉬프트이므로 결과적으로 곱셈 연산만을 반복 사용하여 연산하게 된다.

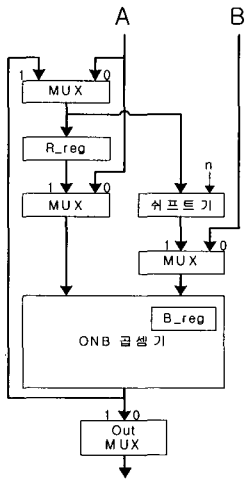
Algorithm 1. IT algorithm

Input : $a \in F_{2^m}, a \neq 0.$

Output : a^{-1}

1. $m-1=[1\, m_{q-2}...m_1\, m_0]_2$
2. $\gamma \leftarrow a$
3. *for i from q-2 downto 0 do*
   3.1 $\gamma \leftarrow \gamma \times \gamma^{2^{2^i}}$
   3.2 *If* $m_i = 1$ *then* $\gamma \leftarrow \gamma^{2^{2^i}} \times a$
4. Return (γ^2).

IT 알고리즘은 Chang, Takagi 등에 의해 반복 사용되는 곱셈 연산 횟수가 줄어 들도록 효율성이 개선되었는데, 이는 GF(2^m)의 몇몇 특별한 경우로 한정된 경우에만 적용된다. 본 논문에서는 다양한 GF(2^m)에 적용 가능하도록 IT 알고리즘을 이용하여 유한체 역승산기를 구현하였다. 구현된 역승산기 구조는 그림 4와 같다. 그림에서 보는 바와 같이 ONB 역승산 연산은 곱셈기를 반복 사용하여 구현되며, MUX와 레지스터 추가로 유한체 곱셈 연산과 역승산 연산을 모두 수행하는 연산기를 구현할 수 있다. 이때, 쉬프트 연산기에는 IT 알고리즘의 3.1과 3.2의 제곱 연산에 필요한 순환 쉬프트 연산을 수행하며, 실제 사용되는 m 값이 250 내이므로 순환 쉬프트 연산 해야 하는 값은 64, 32, 16, 8, 4, 2, 1 중에 하나이며, 입력값 n은 이들의 로그값이 된다. R_reg는 ONB 역승산의 중간 연산값을 저장하며, MUX는 역승산 연산의 데이터 패스를 선택하거나, ONB 곱셈과 역승산을 선택한다. 역승산에 필요한 곱셈 반복 회수는 field가 결정되면 고정적으로 정해진다. 예를 들어 GF(2^{233})에서는 역승산을 위하여 10번의 곱셈 연산이 반복 사용된다.

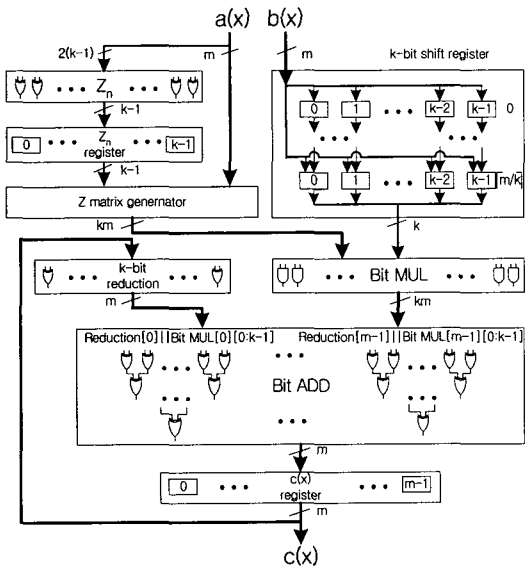


〔그림 8〕 ONB 곱셈기/역승산기 구조

4. Polynomial 기저 곱셈 연산

GF(2^m)에서의 polynomial 기저에 대한 곱셈연산은 bit-serial 기법[9, 12]과 bit-parallel 기법[10, 11, 13] 등에 의해 구현된다. Bit-serial 기법은 작은 하드웨어로 구현할 수 있는 이점이 있지만, m번 이상을 반복 수행하므로 시스템 성능을 저하시킬 수 있다. 반면 bit-parallel 기법은 높은 성능을 기대할 수 있지만, m이 커짐에 따라 하드웨어 면적은 제곱배로 커져 m값이 큰 시스템의 경우 구현에 어려움이 있다. 실제로 [10]의 경우 233비트 유한체 곱셈기를 구현하기 위해서는 54,288개의 XOR 게이트와 54,289개의 AND 게이트가 필요하며, 이러한 곱셈기를 이용하여 암호 시스템을 구현하기에는 어려움이 있다. 이에 우리는 [7]에서 Sunar와 Koc의 bit-parallel 곱셈기의 면적과 성능을 trade-off할 수 있는 hybrid 버전의 곱셈기를 제안하였다.

제안된 hybrid 유한체 곱셈기는 피승수를 임의의 k비트로 나누어 최적화된 m*k 부분곱 연산을 수행하며, 이를 $\lceil m/k \rceil$ 번 반복 수행함으로써 m*m 곱셈연산 수행한다. 이의 전체 구조는 그림9와 같다. 이는 입출력 값 저장을 위한 레지스터들과 km 개의 AND 게이트, km+2k-1 개의 XOR 게이트로 구현된다. 전체 연산 지연 시간은 $T_X + \lceil m/k \rceil (\max\{T_A, T_X\} + \lceil \log_2(k+1) \rceil * T_X)$ 이다.



[그림 9] Polynomial 기저 hybrid 곱셈기 구조

5. Polynomial 기저 역승산 연산

Polynomial 기저에서 역승산 연산 구현을 위한 알고리즘으로는 EEA(Extended Euclidean Algorithm), AIA(Almost Inverse Algorithm) 등이 있다. AIA는 비트로 단위 연산이 수행되는 점과 연산 예측이 용이하다는 점에서 하드웨어 구현에 더 효율적이다. AIA는 역리덕션 연산을 별도로 수행하지 않도록 한 MAIA (Modified AIA)로 효율성이 개선되었다. 우리는 [8]에서 MAIA를 연산 예측성과 병렬 연산을 확장하여 효율성을 더욱 개선하였다. 그림 4는 이의 하드웨어 구조이다. GF(2^{233})에서 연산 시간은 입력값에 따라 차이가 있지만, 대략 250여 사이클 정도이다.

Ⅲ. 연산기 성능 분석

ONB와 polynomial 기저용 연산기는 GF(2^{233})에서 구현되었으며, xilinx virtex-2 8000 FPGA에서 합성 및 검증하였다. GF(2^{233})는 type-II ONB가 존재하며, SEC2에서 추천하는 trinomial minimal polynomial을 가지는 field이다. Type-I ONB용 곱셈기가 더 작은 면적으로 구현되며 성능도 더 우수하지만, type-I ONB 기저와 SEC2에서 추천하는 polynomial 기저를 동시에 만족하는 field가 존재하지 않아 GF(2^{233})에서 구현 비교하였다.

1. 하드웨어 구현 분석

GF(2^{233})에서의 type-I, type-II ONB와 polynomial 기저 곱셈기의 이론상 필요한 논리 게이트와 연산 소요 시간은 표 1과 같다. GF(2^{233})는 type-I ONB가 존재하지 않지만, 비교를 위해 type-I ONB의 조건을 그대로 적용하였으며, 117비트씩 나누어 연산하는 곱셈기를 가정하였다.

[표 1] 곱셈기 필요 논리 게이트와 연산 소요 시간

	117 Bit		
	Type-I ONB	Type-II ONB	Polynomial
XOR Gate	27,144	54,405	27,494
AND Gate	27,261	27,261	27,261
Operation Time	$2T_A + 17T_X$	$2T_A + 18T_X$	$2\max\{T_A, T_X\} + 15T_X$

표에서 곱셈기의 경우 type-I ONB와 polynomial 기저는 거의 성능과 면적이 유사하게 구현될 것으로 보인다. 하지만, type-II ONB의 경우에는 XOR 게이트가 늘어나면서 그에 따른 성능차이가 나타난다. 실제로 GF(2^{233})에서 위와 같은 조건으로 구현한 결과 polynomial 기저 곱셈기는 13,300 slices에 20ns의 지연 시간을 가지며, type-II ONB의 경우에는 17,600 slices에 21ns의 지연 시간을 가졌다.

GF(2^{233})에서의 역승산기 구현은 type-II ONB 경우 곱셈 연산기에 1,200 slices정도를 추가하여 구현 가능하였으며, polynomial 기저의 경우 2,200여 slices를 가지는 별도의 연산기로 구현되었다. 역승산 연산에 필요한 시간은 type-II ONB는 30클럭이며, polynomial 기저는 입력에 따라 다르지만 평균적으로 250클럭이 소요되었다.

이와 같은 연산기들을 이용하여 타원곡선 암호 시스템을 구현하면, 사용하고자 하는 좌표와 스칼라 곱셈 연산 기법에 따라 두 기저가 유사하게 구현된다. 하지만, 기본 연산기들의 성능 차이로 기저에 따라 타원곡선 암호 시스템의 성능차이는 크게 나타나는데, 다음과 같은 가장 기본적인 affine 좌표와 binary 스칼라 곱셈 기법을 이용하는 경우 두 기저에 따른 타원곡선 암호 시템의 성능 차는 표 2와 같다.

Algorithms 2. Affine 좌표 타원곡선 연산

Input : $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$.
Output : $P_3 = P_1 + P_2 = (x_3, y_3)$.
1. If $P_1 = P_2$ (doubling)
   $x_3 = \lambda^2 + \lambda + a$,
   $y_3 = x_1^2 + (\lambda + 1)\, x_3$
   where ($\lambda = x_1 + y_1 / x_1$)
2. Else if P_1 P_2 (point addition)
   $x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$,
   $y_3 = \lambda(x_1 + x_3) + x_3 + y_1$
   where ($\lambda = (y_2 + y_1) / (x_2 + x_1)$)
3. Return (x_3 , y_3)

Algorithm 3. Binary 스칼라 곱셈 기법

Input : $k = (k_{t-1}, \ldots, k_2, k_1, k_0)_2$, P $GF(2^m)$.
Output : kP.
1. Q ← O.
2. For i from t-1 downto 0 do
2.1 Q ← 2Q.
2.2 If $k_i = 1$ then Q ← Q + P.
3. Return (Q)

〔표 2〕 기저에 따른 ECC 시스템 구현

	Area (slices)				Operation Time
	MUL	INV	ETC	Total	
Polynomial Basis	13,300	2,214	1,500	17,000	88,500 clocks
Optimal Normal Basis (Type-II)	18,800		1,200	20,000	11,500 clocks

표 2에서 보는 바와 같이 affine 좌표와 binary 스칼라 곱셈 기법을 이용하는 경우 기저에 따른 성능 차이는 매우 크게 나타났다. 특히 type-I ONB는 제곱 연산기가 필요하지 않아 더욱 polynomial 기저보다 더 작은 면적으로 구현되면서 성능은 더욱 우수할 것으로 보인다. 하지만, 소요 연산 시간의 차이는 역승산 연산시간의 차이로 인하여 나타난 것으로 이 차이는 projective 좌표와 같이 역승산 연산을 최소로 하는 좌표를 이용하면, 두 기저에 따른 성능 차이는 최소한 번의 역승산 연산 시간 차이로 현저히 줄일 수 있다.

Ⅳ. 결 론

본 논문에서는 ONB와 polynomial 기저에서의 유한체 연산 및 타원곡선 암호 연산기의 효율성을 비교하고자, Type-I, Type-II로 구분되는 ONB용 유한체 연산기와 polynomial 기저용 유한체 연산기들을 구현하였다. 유한체 연산기들은 하드웨어 면적과 성능을 trade-off 할 수 있도록 hybrid 타입의 연산기를 ONB용 유한체 연산기 구현 결과를 polynomial 기저의 유한체 연산기 구현 결과와 비교하여 각 기저에 따른 타원곡선 연산의 효율성을 검증하였다.

GF(2^{233})에서 구현된 type-II ONB 유한체 연산기는 polynomial 기저의 유한체 연산기에 비하여 많은 논리 게이트를 필요로 하지만, 역승산 연산기의 성능 차이로 인하여 기본적인 기법을 이용한 타원곡선 암호 시스템의 성능은 더욱 우수하게 구현되었다. 이는 type-I ONB의 경우에는 polynomial 기저에 비하여 더욱 작은 면적으로 구현되면서 성능도 현저히 우수할 것으로 예상된다. 하지만, projective 좌표를 사용하는 경우에는 역승산 회수의 감소로 성능 차이는 현저히 줄어들 것으로 예상된다. 이에 고속 곱셈 연산기를 많이 필요로 하는 고속 공개키 암호 시스템에서는 각 기저의 선택적 사용이 가능할 것으로 보이며, 저전력/저면적의 공개키 암호 하드웨어 시스템을 필요로 하는 시스템에서는 성능과 면적에서 모두 우수한

type-I ONB가 가장 효율적으로 구현될 것으로 보인다.

참 고 문 헌

[1] Certicom research , The Elliptic Curve Cryptosystem, Certicom, April 1997.

[2] J. Omura and J. Massey, "Computational Method and Apparatus for Finite Field Arithmetic" U.S. Patent Number 4,587,627

[3] M. A. Hasan, M. Z. Wang, V. K. Bhargava, "A Modified Massey-Omura Parallel Multiplier for a Class of Finite Fields", IEEE Tans on Comp, Vol 42, No 10, October 1993.

[4] C. K. Koc, and B. Sunar, "Low-Complexity Bit-Parallel Canonical and Normal Basis Multipliers for a Class of Finite Fields", IEEE Trans on Comp. Vol 47, No 3, March 1998.

[5] Huapeng Wu, Anwar Hasan, and lan F. Blake,"Finite Field Multiplier Using Redundant Representation", IEEE Transactions on Computers, Vol 51, No 11, November 2002 .

[6] T. Itoh, S. Tsujii, "Effective recursive alogirthm for computing multiplicative inverses in GF(2^m)", Electronics letters 17th march 1988, Vol. 24. No. 6.

[7] Y.J. Choi, K.-Y. Chang, D.W. Hong and H.S. Cho, "Hybrid multiplier for GF(2^m) defined by some irreducible trinomials" Electronics Letter, Volume 40 852-853, Number 14, 8th July 2004.

[8] HoWon Kim, Thomas Wollinger, YongJe Choi, Kyoil Chung, and Christof Paar, "Hyperelliptic Curve Coprocessors on a FPGA," The 5th International Workshop on Information Security Applications (WISA 2004), Aug. 23-25, 2004, JeJu, Korea (LNCS: SCI-E)

[9] Mastrovito, E. D. : 'VLSI architectures for computations in Galois fields' PhD Thesis, Linkoping University, Department of Electrical Engineering, Linkoping, Sweden, 1991.

[10] Sunar, B. and Koc, C. K.: 'Mastrovisto multiplier for all trinomials', IEEE Trans. Comput. 1999, 48, (5), pp. 522-527.

[11] Wu, H. : 'Bit-parallel finite field multiplier and square using polynomial basis', IEEE Trans. Comput., 2002, 51, (7), pp. 750-758.

[12] Chiou, C. W, Chou F. H. and Shu S. F : 'Low-complexity finite field multiplier using irreducible trinomials', Electron. Lett., 2003, 39, (24), pp. 1709-1711

[13] Elia, M. Leone, M. and Visentin C. : 'Low complexity bit-parallel multipliers for GF(2m) with generator polynomial xm + xk + 1', Electron. Lett., 1999, 35, (7), pp. 551-552.

〈著 者 紹 介〉

최 용 제 (Yong-Je Choi)
정회원

1996년 8월 : 전남대학교 전자공학과 졸업
1999년 2월 : 전남대학교 전자공학과 석사
1999년 8월~현재 : 한국전자통신연구원 정보보호연구단 선임연구원
관심분야 : 전자공학, 통신공학, 정보보호, VLSI 설계, RFID/USN 보안

김 호 원 (Ho-Won Kim)
정회원

1993년 2월 : 경북대학교 전자공학과 졸업
1995년 2월 : 포항공과대학교 전자전기공학과 석사
1999년 2월 : 포항공과대학교 전자전기공학과 박사
1998년 12월~현재 : 한국전자통신연구원 정보보호연구단 팀장/선임연구원
2003년 7월~2004년 6월: Ruhr University Bochum, Post-doctorial, 독일
관심분야 : 정보보호, RFID/USN 보안, 프라이버시 보호