Elliptic curve기반 센서네트워크 보안을 위한 곱셈 최적화 기법

서화정* · 김호원**

Multiplication optimization technique for Elliptic Curve based sensor network security

Hwa-jeong Seo* · Ho-won Kim**

이 논문은 부산대학교 자유과제 학술연구비(2년)에 의해 연구되었음

요 약

센서네트워크는 유비쿼터스 환경을 실현할 수 있는 기술로서, 최근 무인 경비 시스템이나 에너지 관리, 환경 모니터링, 홈 자동화, 헬스케어와 같은 다양한 분야에 적용 가능하다. 하지만 센서네트워크의 무선 통신 특성은 도청과 전송 메시지에 대한 위변조, 서비스 거부 공격에 취약하다. 현재 센서네트워크 상에는 안전한 통신을 위해 ECC(elliptic curve cryptography)를 통한 PKC(public key cryptography)암호화 기법이 사용된다. 해당 기법은 RSA에 비해 작은 키길이를 통해 동일한 암호화 강도를 제공하기 때문에 제한된 성능을 가진 센서상의 적용에 적합하다. 하지만 ECC에 요구되어지는 높은 부하 때문에 센서상의 구현을 위해서는 성능개선이 필요하다. 본 논문에서는 센서 상에서의 ECC를 가속화하기 위해 ECC연산의 핵심연산인 곱셈에 대한 최적화 기법을 제안한다.

ABSTRACT

Sensor network, which is technology to realize the ubiquitous environment, recently, could apply to the field of Mechanic & electronic Security System, Energy management system, Environment monitoring system, Home automation and health care application. However, feature of wireless networking of sensor network is vulnerable to eavesdropping and falsification about message. Presently, PKC(public key cryptography) technique using ECC(elliptic curve cryptography) is used to build up the secure networking over sensor network. ECC is more suitable to sensor having restricted performance than RSA, because it offers equal strength using small size of key. But, for high computation cost, ECC needs to enhance the performance to implement over sensor. In this paper, we propose the optimizing technique for multiplication, core operation in ECC, to accelerate the speed of ECC.

키워드

타원곡선, 암호, MSP430, 센서 네트워크, USN, 곱셈, PKC

Key word

elliptic curve cryptography, MSP430, Sensor Network, USN, Multiplication, PKC

* 부산대학교 컴퓨터공학과 석사과정(hwajeong@pusan.ac.kr)
** 부산대학교 컴퓨터공학과 교수

접수일자 : 2010. 06. 29
심사완료일자 : 2010. 08. 09

Ⅰ. 서 론

센서네트워크는 유비쿼터스 환경을 실현하는 핵심 기술로써 헬스케어, 홈 자동화, 환경・군사모니터링에 적용이 가능하여 u-city나 u-port와 같은 분야에 핵심 기술로 사용된다. 따라서 국내의 산업계와 연구소를 중심으로 핵심 기술과 응용 기술에 대한 개발이 활발히 진행되고 있다.

하지만, 센서 네트워크의 보안 분야에 대한 연구는 활발히 진행되고 있지 않다. 해당 분야는 센서네트워크의 상용화 및 산업화 관점에 있어서 중요하므로 연구 및 개발이 진행되어야 한다. 현재 센서 상에서의 암호화는 ECC를 기반으로 한 키교환, 전자서명이 많이 사용된다[1]. 하지만 ECC연산에 소모되는 높은 부하와 센서의 제한적인 성능으로 인해 실용적인 적용을 위해서는 효율적인 ECC연산 구현 기법이 제안되어야 한다.

본 논문에서는 ECC의 핵심연산인 곱셈연산을 대표적인 센서노드 프로세서인 MSP430 상에 최적화 구현한다. 논문의 구성은 2장에서 ECC에 대해 알아보며, 3장에서 구현환경과 사용된 프로세서 MSP430을 분석한다. 4장에서는 최적화 곱셈기법을 제안하며 마지막으로 5장에서는 제안된 기법의 성능을 보인다.

Ⅱ. Elliptic Curve Cryptography

2.1 Elliptic curve cryptography의 특성

ECC는 1985년 Neal Koblitz와 Victor Miller에 의해 제안된 알고리즘으로 발견 당시 수학적으로는 이상적이지만 실용적이지는 못했다[2],[3]. 하지만 현재는 높은 보안 완결성과 실용성을 가진 효율적인 공개키 암호화 시스템이다. ECC의 응용은 키교환, 전자서명에 적용되며 수학적 문제인 ECDLP(Elliptic Curve Discrete Logarithm Problem)에 기저를 둔 암호화로 기존의 공개키 암호화 기법인 RSA/DSA와 비교해 보다 작은 키길이로 암호화 구현이 가능하다. ECDLP란 Prime filed F_q상에 정의된 elliptic curve E와 E상의 두 점 P, Q가 존재할 때 $lP = Q$를 만족하는 l을 찾는 것이 어렵다는 것을 의미한다. 따라서, 전자서명의 경우 160bit-ECDSA(Elliptic Curve Digital Signature Algorithm)는 1024bit-DSA(Digital Signature Algorithm)과 동일한 강도의 암호화를 작은 키길이로 제공한다[4]. 이는 제한적인 성능(memory, Computation performance)을 가진 센서 노드에 적합하다.

2.2 권장되어 지는 Elliptic Curve Domain

elliptic curve은 Prime fields $GF(p)$와 finite fields $GF(2^m)$상에 정의된다. ECC알고리즘은 표 1에서와 같이 권장되어지는 parameter domain이 있다. 이는 해당 Domain을 선택한 elliptic curve가 암호화적으로 안전하며 표 1의 RSA/DSA와 동일한 강도의 암호화 제공을 인증한다.

표 1. NIST에 의해 권장되어 지는 Prime[5]
Table 1. Prime recommended by NIST[5]

Size (bits)	Prime	RSA/DSA (bits)
192	$2^{192}-2^{64}-1$	1536
224	$2^{224}-2^{96}+1$	2048
256	$2^{256}-2^{224}+2^{192}+2^{96}-1$	3072
384	$2^{384}-2^{128}-2^{96}+2^{32}-1$	7680
521	$2^{521}-1$	15360

2.3 ECC Operations and performance

elliptic curve는 표 2에서와 같이 표기되며 a와 b는 $GF(P)$상에 정의되어 있으며 $b \neq 0$이다. $E(GF(P))$는 curve상의 모든 점을 포함하며 항등원인 O가 존재한다. 또한 curve상의 addition특성을 가진 임의의 두 점 $P=(x_1,y_1)$와 $Q=(x_2,y_2)$의 addition은 $P+Q=(x_3,y_3)$로 정의된다. k는 기울기를 의미하며 연산은 addition연산 또는 한 점에 대한 doubling연산으로 수행된다.

표 2. 정의된 elliptic curve 식
Table 2. defined elliptic curve expression

$$liptic\ curve: y^2 = x^3 + ax + b$$
$$P + O = O + P = P,$$
$$P + (-P) = 0, where\ -P = (x, x+y).$$
$$x_3 = k^2 - (x_1 + x_2)$$
$$y_3 = -y_1 + k(x_1 - x_3)$$
$$k = \frac{y_1 - y_2}{x_1 - x_2} \quad \text{if } P \neq Q \text{ (doubling operation)}$$
$$k = \frac{3x^2 + a}{2y} \quad \text{if } P = Q \text{ (addition operation)}$$

curve상의 addition연산은 curve상에서의 Scalar multiplication을 수행하기 위해 사용된다. 표 3에서와 같이 한 점에 대한 J곱은 addition을 J번하는 것으로 연산가능하다.

표 3. 점 P에 대한 곱셈
Table 3. Multiplication on point P

$Q = JP = P + ... + P(J\times)$

Scalar multiplication은 표 4에 나타난 알고리즘을 통해 구현한다. 이는 점에 곱하고자 하는 값 J를 bit로 표현한 뒤 J의 가장 상위 bit부터 1인지 0인지를 확인하여 0인 경우 doubling만을 수행하고 1인 경우 doubling과 addition을 취하는 방식으로 구현된다. 전체 수행은 $l/2$번의 point addition과 l번의 doubling으로 구성된다.

표 4. Algorithm 1. 포인트 곱셈을 위한 Left-to-right binary algorithm[6]
Table 4. Algorithm 1. Left-to-right binary algorithm for point multiplication[6]

$INPUT$: $P \in E$, and a positive intger $J = (J_{l-1}J_{l-2}\cdots J_1J_0)_2$ $OUTPUT$: $A = JP$ 1: $A \leftarrow O$ 2: For $i = l-1$ down to 0 do 3: $A \leftarrow 2A$ 4: If $J_i = 1$ then $A \leftarrow A + P$ 5: return A

ECC는 전체 소모되는 부하 중 69.5%는 Inversion연산, 25.7%는 Multiplication, 3.6%는 Squaring에서 소비된다. 여기서 대부분의 부하는 Inversion연산에 소비되지만 Inversion연산의 알고리즘은 대부분의 연산이 Multiplication으로 구성된다. 따라서 ECC의 성능은 Multiplication에 의해 결정된다.

표 5. Latency comparison of operation in ECC[6]
Table 5. ECC상에서의 연산비중비교[6]

Operations	Time
Multiplication	25.7%
Squaring	3.6%
Inversion	69.5%

Ⅲ. 타겟보드 및 구현 환경

3.1 16-bit 프로세서 상에서의 ECC구현

MSP430은 8.192Mhz로 동작하며 48KB의 ROM, 10KB의 RAM을 가진다. register는 총 16bit이며 이 중 사용이 가능한 general register의 수는 12개이다. MSP430은 Multiplication module이 있어 16bits Unsigned, Signed, MAC (Multiply and Accumulate)연산을 제공한다. 연산은 operand 1에 값을 인가하면 수행하고자 하는 곱셈의 mode가 결정되며 operand 2에 값을 인가 시 곱셈이 수행된다. 수행된 결과는 SumLo와 SumHi에 저장되며 만약 MAC 모드로 곱셈을 수행하는 경우 carry값은 SumExt에 저장되게 된다.

표 6. 곱셈에 사용되는 register 주소 및 기능[10]
Table 6. register address and function used multiplication[10]

이름	기능	주소
MPY	operand 1에 해당하며 값 인가 시 unsigned multiply mode가 된다.	130h
MPYS	operand 1에 해당하며 값 인가 시 signed multiply mode가 된다.	132h
MAC	operand 1에 해당하며 값 인가 시 unsigned multiply and accumulate mode가 된다.	134h
Op2	3가지 mode의 곱셈을 모두 지원하며 해당 register에 값을 인가하면 operand 1의 값과 곱셈을 수행한다.	138h
SumLo	곱셈이 수행되고 난 후의 결과 값 중 하위 16bits가 저장된다.	13Ah
SumHi	곱셈이 수행되고 난 후의 결과 값 중 상위 16bits가 저장된다.	13Ch
SumExt	MAC mode로 multiplication을 수행 시 발생하게 되는 carry값을 저장한다.	13Eh

3.2 구현환경

그림 1은 빛, 습도, 온도 센서가 장착된 MSP430보드이다. USB to Serial port를 통해 컴퓨터와 Serial통신이 가능하며 Tinyos-2.x상에 nesc를 통해 compile하였다. 성능은 IAR Simulator의 clock cycle을 통해 측정하였다.

그림 1. MSP430보드
Fig 1. MSP430 board

Ⅳ.성능 분석

4.1 이전 구현사례

현재 잘 알려진 곱셈 알고리즘에는 Comba method와 Hybrid method가 있다[8], [9]. 그림 속에 표시된 작은 박스는 곱셈을 취하게 되는 값의 단위를 나타내며 안의 값과 숫자는 변수의 이름과 자리수를 표현한다. 그림 2에 나타난 Comba method는 열을 기준으로 곱셈연산을 수행한다. 반면에 Hybrid 곱셈은 행과 열을 기준으로 묶어서 곱셈을 수행한다. 해당 연산은 register에 값을 저장하여 반복적으로 사용하여 memory access를 감소시킨다. 하지만 platform의 register가 충분하지 못한 경우 연산자를 저장할 수 없으므로 비효율적이다.

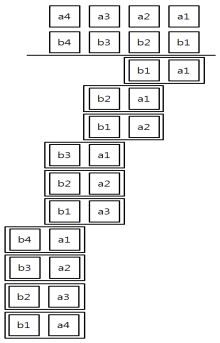

그림 2. Comba[8]
Fig 2. Comba[8]

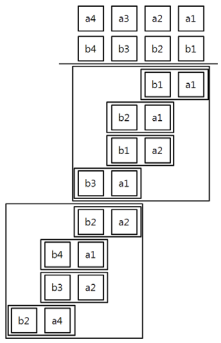

그림 3. Hybrid[9]
Fig 3. Hybrid[9]

Algorithm에 따른 곱셈기의 성능은 Comba와 하드웨어 곱셈기의 MAC 모드를 사용할 때 성능이 좋다. 그 이유는 Comba는 열을 기준으로 연산을 수행하는데 MAC 모드를 적용하면 결과 값을 중첩해서 계산하는 것이 가능하기 때문이다.

표 7. 곱셈 성능 비교[7],[11]
Table 7. Comparison of performance of multiplication[7],[11]

Algorithm	Clock Cycles	Time (ms)
Hybrid	1,746	0.22
Comba MAC	1,586	0.20

4.2 곱셈 알고리즘

4.2.1 Block comb method

표 8은 Block comb method을 적용하여 240-bit 곱셈을 수행한 것이다. 연산은 operand A와 operand B를 block 단위로 묶어 곱한 후 block 단위로 저장한다. 예를 들어 오른쪽에서 두 번째 열의 $A_6^6 \times B_0^6$과 $A_0^6 \times B_6^6$연산은 곱셈을 수행하면 결과값의 위치가 $C_{12:6}^6$로 동일하다. 사용된 인자들의 표기법은 밑수는 register의 시작위치를 나타내며 승수자리의 값은 register의 개수를 의미한다. 예를 들어 A_{12}^3은 3개의 register로 구성되며 범위는 register12에서 register15까지를 나타낸다.

표 8. size 3의 block 곱셈 (240-bit)[12]
Table 8. Block multiplication of size 3 (240-bit)[12]

			A_{12}^3 B_{12}^3	A_9^3 B_9^3	A_6^3 B_6^3	A_3^3 B_3^3	A_0^3 B_0^3
			A_{12}^3 B_0^3	A_9^3 B_0^3	A_6^3 B_0^3	A_3^3 B_0^3	A_0^3 B_0^3
		A_{12}^3 B_3^3	A_9^3 B_3^3	A_6^3 B_3^3	A_3^3 B_3^3	A_0^3 B_3^3	
	A_{12}^3 B_6^3	A_9^3 B_6^3	A_6^3 B_6^3	A_3^3 B_6^3	A_0^3 B_6^3		
A_{12}^3 B_9^3	A_9^3 B_9^3	A_6^3 B_9^3	A_3^3 B_9^3	A_0^3 B_9^3				
A_{12}^3 B_{12}^3	A_9^3 B_{12}^3	A_6^3 B_{12}^3	A_3^3 B_{12}^3	A_0^3 B_{12}^3				
C_{27}^3	C_{24}^3	C_{21}^3	C_{18}^3	C_{15}^3	C_{12}^3	C_9^3	C_6^3	C_3^3

표 9는 MSP430에 Block com method를 적용한 알고리즘을 나타낸다. Step 1~7에서는 인자를 불러오고 결과값을 저장하는 register($R_4 \sim R_9$)를 초기화시켜준다. Step 9에서는 Block Comb연산이 수행되는 경우를 계산하여 조건을 만족하는 경우에만 연산이 계속 수행된다.

Step 10～14에서는 연산자와 피연산자를 register에 각각의 인자 3개씩 register로 불러온다. Step 17에서는 Block comb method 수행이 일어나는 경우를 계산한다. Step 18～20에서는 Block 단위의 곱셈을 16-bit 단위로 나누어 하드웨어곱셈기로 곱셈한다. 결과 값은 두 개의 인자의 곱이므로 16bit register두군데에 저장된다. Step 25～32에서는 곱셈연산이 끝난 후 register에 저장된 결과 값을 memory에 저장한다. 그 중 Step 25～28에서는 계산된 결과 값의 하위 3개의 register를 memory에 저장한다. 저장되지 않고 그대로 남아 있는 상위 3개의 register는 다음 연산 수행 시 하위 3개의 register로 중첩을 해서 사용이 가능하다. 이를 통해 memory에 대한 저장과 불러오기가 열을 기준으로 한번씩 일어나므로 효율적인 연산이 가능하다.

표 9. Algorithm 2. MSP430상에서의 size 3의 Block comb 곱셈 C=AB (240bits 곱셈)[12]
Table 9. Algorithm 2. Block comb method for computing C=AB (240bits multiplication) with block size 3 over MSP430[12]

```
Input : A = (A14, ..., A0), B = (B14, ..., B0) (∈ memory),
Output : (C29, ..., C0) = AB(to memory)
1: for i = 0 to 2 do
2:    R_{i+4} ← 0
3: end for
4: for i = 0 to 8 do
5:   for j = 3 to 5 do
6:     R_{j+4} ← 0
7:   end for
8:     for j = 0 to 4 do
9:       k = i − j;
10:      if 0 ≤ k and k ≤ 4 then
11:          for l = 0 to 2 do
12:            load R_{10+l} ← A_{3j+l}
13:            load R_{13+l} ← B_{3k+l}
14:          end for
15:          for q = 0 to 4 do
16:            for w = 0 to 2
17:              e = q − w;
18:              if 0 ≤ e and e ≤ 2 then
19:                R_{q+5} R_{q+4} = R_{10+w} × R_{13+e}
20:              end if
21:            end for
22:          end for
23:      end if
24:    end for
25:    for k = 0 to 2 do
26:      store C_{3i+k} ← R_{k+4}
27:      R_{k+4} ← R_{k+7}
28:    end for
29: end for
30: for i = 0 to 3 do
31:   store C_{27+i} ← R_{7+i}
32: end for
```

표 10. Algorithm 3에 사용된 명령어
Table 10. Operation used in Algorithm 3

명령어	동작 설명
load	memory에 있는 내용을 register로 저장한다.
store	register에 있는 내용을 memory로 저장한다.
move	register의 내용을 다른 register에 저장한다.

4.2.2 MAC을 이용한 Multiplication

MPY와 MPYS 방식은 Step 1에서 연산하고자 하는 방식을 Operand1의 주소로 값을 인가해 주면서 결정한다. Step 2에서 Operand2에 값을 넣어주면 곱셈연산이 수행한다. Step3～4에서는 계산된 결과 값을 SumLo와 SumHi로부터 memory에 저장하며 carry 발생시 adc instruction을 통해 다음 register로 전달해 준다.

표 11. Algorithm 3. MPY, MPYS 곱셈 기법[9]
Table 11. Algorithm 3. Multiplication techniques(MPY, MPYS)[9]

```
INPUT: First Operand1 r14, Second Operand r15
OUTPUT: The μltiplcation of the integer
and their accumulation into r4,r5,r6
1: mov  r14, MPY
2: mov  r15, OP2
3: add  SumLo, r4
4: addc SumHi, r5
5: adc  r6
```

MAC방식은 Step 1～2에서 MPY와 MPYS방식과 동일하게 Operand들을 인가해준다. 계산을 수행하면 결과 값은 하드웨어 상의 저장공간인 SumLo, SumHi에 저장되며 발생하는 carry값은 SumExt에 저장된다. 따라서 동일한 열에 대한 반복적인 연산수행시 carry값만을 SumExt로부터 불러와 저장하며 마지막에 한번만 SumLo, SumHi에 접근하여 값을 저장하므로 효율적이다.

표 12. Algorithm 4. MAC 곱셈 기법[7]
Table 12. Algorithm 4. Multiplication technique(MAC)[7]

$INPUT$: $First\ Operand1\ r14, Second\ Operand\ r15$ $OUTPUT$: $The\ multiplcation\ of\ the\ integers$ and $their\ accumulation\ into\ SumLo, SumHi, r6$ 1: $mov\ r14, MAC$ 2: $mov\ r15, OP2$ 3: $add\ SumExt, r6$

표 13. Algorithm 3, 4에 사용된 명령어
Table 13. Operation used in Algorithm 3, 4

명령어	동작 설명
$mov\ R_s,\ R_d$	$R_s \rightarrow R_d$
$add\ R_s, R_d$	$R_d = R_d + R_s$
$addc\ R_s, R_d$	$R_d = R_d + R_s + Carry\ Bit$
$adc\ R_d$	$R_d = R_d + Carry\ Bit$

4.3. 곱셈 구현 기법

4.3.1 memory access 최소화 기법

그림 4는 MAC기법을 사용하여 곱셈을 수행하게 된다. 첫 번째 단계에서 곱셈이 끝나면 결과값이 하드웨어 곱셈기에 저장된다. 두 번째 단계에서는 SUMEXT의 carry을 register 6에 저장하며, 다음으로 a1과 b0의 연산을 중첩하여 수행한 후 결과 값을 한번에 모두 저장한다. 그림 5는 $a0 \times b1$와 $a0 \times b2$를 수행할 때 a0의 값은 첫 번째 operand에 저장되어 있으므로 memory로부터 다시 불러오지 않고 b2만을 operand 2에 불러와서 연산을 수행한다. 해당 구현이 가능한 이유는 첫 번째 operand의 값은 변하지 않으며 두 번째 operand가 인가될 때 수행되는 하드웨어 곱셈기의 특성을 이용한 것이다.

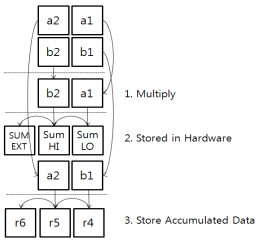

그림 4. MAC 기법
Fig 4. MAC technique

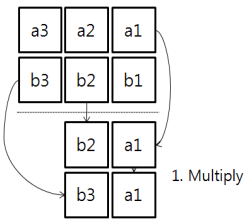

그림 5. 인자재사용 기법
Fig 5. Operand reusing technique

4.3.2 올림연산 중첩기법

Block comb method 곱셈과정에서는 올림이 계속해서 발생한다. 그 이유는 결과값을 저장하는 register를 중첩해서 사용하는 특징 때문이다. 그림 6은 일반적인 곱셈에서의 올림 구현을 나타내고 있다. 여기서는 올림이 발생할 경우 모든 register의 올림을 계산해주고 있다. 하지만 그림 7에서는 중간에 하나의 register를 두어 발생하는 올림들을 저장해 두었다가 마지막에 한번만 연산하게 된다.

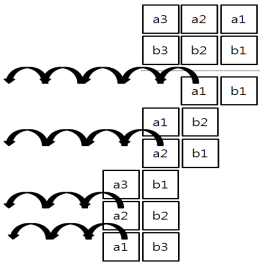

그림 6. 기존 기법
Fig 6. Previous technique

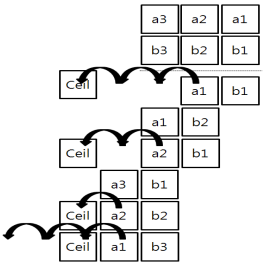

그림 7. 제안 기법
Fig 7. Proposed technique

4.3.3 곱셈순서정렬을 통한 operand 재사용 기법

block comb method에서는 각각 3개의 operand를 묶어서 연산을 한다. 표 14에서와 같이 operand의 순서를 조절하면 memory access중복을 줄여준다. 표 14의 회색으로 색칠된 부분은 register값을 변경하지 않고 이전 값을 사용하여 곱셈을 수행한다.

표 14. register에 저장되는 (피)연산자의 순서
Table 14. Order of operands saved in register

곱셈순서	operand A			operand B		
1	2	1	0	2	1	0
2	5	4	3	2	1	0
3	2	1	0	5	4	3
4	2	1	0	8	7	6
. . .						
24	11	10	9	14	13	12
25	14	13	12	14	13	12

V. 결 론

본 논문은 Block comb MAC method를 $GF(P)$ 상에서 MSP430에 적용하여 성능을 개선한다. 표 15에는 기존의

가장 높은 성능을 내는 Comba MAC method와 논문에서 제안한 Block comb MAC의 성능을 비교한 것이다. 기존 기법은 288-Bit이하일 때 좋은 성능을 보인다. 반면에 제안된 기법은 336-Bit 이상에서는 기존 기법에 비해 1~3% 향상된 성능을 보인다. 그 이유는 도메인의 크기가 커짐에 따라 register 중첩의 이득이 증가하기 때문이다. 따라서 해당 기법은 NIST에 의해 권장되는 ECC도메인의 384, 521-Bit 키길이에 적용할 경우 기존기법에 비해 성능이 향상된다.

표 15. 알고리즘 성능비교[7][11]
Table 15. Performance comparison[7][11]

Bits	Comba MAC (clock)	Block comb MAC (clock)	성능비교 (제안 / 기존)
48	156	148	0.948718
96	519	523	1.007707
144	1120	1211	1.08125
192	2050	2120	1.034146
240	3212	3312	1.031133
288	4688	4701	1.002773
336	6452	6430	0.99659
384	8504	8423	0.990475
432	10814	10660	0.985759
480	13442	13162	0.97917
528	16388	15928	0.971931

참고문헌

[1] Certicom Research, "Certicom ECC Challenge," 1997.

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

[3] V. Miller, "Uses of elliptic curves in cryptography," Advances in Cryptology: proceedings of Crypto'85, pp. 417-426, 1986.

[4] "National Institude of Standards and Technology, Digital Signature Standard," FIPS Publication 186-2, 2000.

[5] Daniel R. L. Brown, "Recommended Elliptic Curve Domain Parameters," 2010.

[6] Zhijie Jerry Shi, Hai Yan, "Software Implmentations of Elliptic Curve Cryptography," International Journal of Network Security, vol. 7, no. 2, pp. 157-166, 2008.

[7] Conrado Porto Lopes Gouvêa, Julio López "Software Implementation of Pairing-Based Cryptography on Sensor Networks Using the MSP430 Microcontroller," Progress in Cryptology - INDOCRYPT 2009 pp 248-262, 2009.

[8] Comba, "Exponentiation cryptosystems on the IBM PC," IBM Systems Journal vol 29, no. 4, pp 526 - 538, 1990.

[9] Gura, N., Patel, A., Wander, A. Eberle, H. Shantz,"Comparing elliptic curve cryptography and RSA on 8-bit CPUs," CHES 2004. LNCS, vol. 3156, pp. 925 - 943, 2004.

[10] Texas Instruments, "The MSP430 Hardware Multiplier Function and Applications," Application Report, 1999.

[11] Scott, M., Szczechowiak, "Optimizing multiprecision multiplication for public key cryptography," Cryptology ePrint Archive, Report 2007, pp. 299, 2007.

[12] Masaaki Shirase, Yukinori Miyazaki, Tsuyoshi Takagi, Dong-Guk Han, Dooho Choi, "Efficient Implementation of Pairing-Based Cryptography on a sensor Node," IEICE TRANSACTIONS on Information and Systems Vol.E92-D No.5 pp. 909-917, 2009.

저자소개

서화정(Hwa-jeong Seo)

2004.3~2010.2 : 부산대학교 정보컴퓨터공학과 학사
2010.2~현재 : 부산대학교 컴퓨터공학부 석사
※관심분야 : 정보보안, RFID/USN, 암호 이론, VLSI 설계

김호원(Ho-won Kim)

1989. 3~1993. 2 : 경북대학교 전자공학과 학사
1993. 3~1995. 2 : 포항공과대학교 전자전기공학과 공학석사
1995. 2~1999 .2 : 포항공과대학교 전자전기공학과 공학박사
1998. 12~2008. 2 : 한국전자통신연구원(ETRI) 정보보호연구단 선임연구원 / 팀장
2008. 3~현재 : 부산대학교 정보컴퓨터공학부 조교수
※관심분야 : 스마트그리드 보안, RFID/USN 정보보호 기술, PKC 암호, VLSI 설계, embedded system 보안