162 비트 Optimal Normal Basis상의 ECC Coprocessor의 구현

배상태° 백동근 김홍국
광주과학기술원 정보통신공학과
{stbae°, fromzero, hoonkook}@kjist.ac.kr

An Implementation of ECC Coprocessor over F_2^{162} Based on Optimal Normal Basis

Sangtae Bae° Dong Guen Paek Hong Kook Kim
Dept. of Information and Communications Kwangju Institute of Science and Technology

요 약

본 논문에서는 162bits의 Key Size를 가지고서도 RSA 1024bits의 암호학적 강도를 지니는 스마트카드용으로 적합한 ECC Coprocessor의 구현하고자 한다. ECC의 하드웨어 구현시의 적합성을 위해 162bit Optimal Normal Basis를 선택하였으며, Multiplication은 23 클록 사이클에 수행이 되도록 구현하였으며, Inversion은 Multiplication을 11번 사용하는 알고리즘을 선택하였다. 이때 한번의 점간의 덧셈 연산을 마치는데 331(335) 클록 사이클이 소요되며 클록의 최소주기는 3ns 이다. 또한 Area는 37,111를 기록했다.

1. 서 론

비즈니스와 일상생활에서 인터넷을 통한 정보교류가 활성화되기 시작하면서 정보교환의 안정성 문제가 대두되기 시작하였다. 안전한 데이터의 교환을 위해서는 보안 알고리즘을 이용하여 데이터의 암호화와 복호화를 하는 시스템을 사용하는 것이 일반적이다. 현재까지 널리 사용되고 있는 보안 시스템은 대칭키 암호시스템과 공개키 암호시스템의 두 가지로 구분이 된다.

이중 공개키 암호시스템은 현재에는 암호화, 복호화시의 비효율성으로 인해 일부 중요한 데이터만을 암, 복호화하거나 키 교환 시 등 일부에서만 사용이 되고 있다. 하지만 ITRC의 로드맵에 따르면 공정기술과 설계기술이 발달하기 시작하면서 칩 동작속도와 면적 등의 제약조건이 약화될 것으로 예측된다. 따라서 향후에는 대칭키 암호시스템에 비해 강한 암호학적 강도를 지니는 공개키 암호시스템의 비중이 더욱 커질 것으로 전망된다.[1]

ECC(Elliptic Curve Cryptography)는 현재까지 알려진 다른 공개키 암호시스템의 알고리즘에 비해 비트 당 가장 강력한 암호학적 강도를 지닌다고 알려져 있다. 이러한 특징으로 인해 ECC는 적은 키 사이즈를 가질 수 있는 장점이 있으며 많은 연구의 대상이 되고 있다. 이러한 ECC는 하드웨어로 구현 시 적은 메모리의 용량과 전력소모를 줄일 수 있게 된다. 따라서 하드웨어로 ECC를 구현 시 공개키 암호시스템의 이점인 강한 보안성과 적은 면적, 저 전력 등의 장점을 지니게 되므로 스마트카드와 같은 SoC환경에서의 보안모듈의 구현 등의 이득을 얻을 수 있다.

ECC는 공개키 암호시스템 중에서 타원곡선 내에서의 이산대수문제(Discrete Logarithm Problem)의 암호학적 기반을 두고 있다. 수학적인 관점에서 보면, 타원곡선위의 이산대수문제는 RSA알고리즘의 합성수의 소인수분해 문제, 유한체(Finite 체)의 이산대수문제등과 같은 다른 복잡한 수학적인 문제들을 해결하는데 드는 시간에 비해 훨씬 복잡하다.[2][3]

ECC위에서의 암호화와 복호화는 타원곡선위의 점간의 산술연산과 유한체내에서의 산술연산으로 수행되어진다. 즉, 점사이의 덧셈 같은 산술연산은 유한체내의 산술연산으로 계산이 되어진다. 유한체내에서의 주된 연산은 덧셈과 곱셈이다. 덧셈연산의 경우 두 연산자간의 비트별 XOR 연산으로 계산된다. 또한 제곱 연산은 Cyclic Shift연산으로 처리가 된다. 따라서 유한체 내에서의 덧셈기와 제곱기는 간단히 구현된다. 하지만, 유한체 곱셈의 경우는 여러 연산의 조합으로 구현되므로 조금 더 복잡한 구조를 지니게 된다. 이는 계산 시간, 회로의 면적, 회로의 복잡도등의 측면 등, 곱셈기 설계 시 고려해야 할 요소가 많이 존재하기 때문이다.

본 논문에서는 이러한 제약 요소 중 스마트 카드위에 암호화 모듈로서 장착될 수 있는 ECC 프로세서를 제안하고자 한다.

2. ECC

ECC의 Elliptic Curve Discrete Logarithm Problem(이하 ECDLP)은 하나의 유한체위에 존재하는 Elliptic Curve를 E라고 하였을 때, P, Q ∈ E(Fq)에 대하여 $0 \le k \le \#P-1$, $k \in Fq$의 조건을 가지고 $Q = kP = P + P + P \cdots + P$를 만족시키는 k가 존재한다는 것이다. Elliptic Curve 위에서의 주된 연산은 점들 간의 덧셈연산이다. Elliptic Curve 위에서 점들 간의 덧셈 연산은 기반체의 산술연산에 의존한다.

본 논문에서는 F_2^m 형태의 Binary 유한체만을 취급하였다. 이러한 Binary 유한체는 산술연산 중 덧셈과 제곱 연산이 쉬워 하드웨어로의 구현이 쉽기 때문이다. Elliptic Curve에는 두 가지 형태의 곡선이 존재한다. 첫째가 Nonsupersingular Curve이고 둘째가 Supersingular Curve이다. 이중 MOV attack등에 더욱더 안전한 Nonsupersingular Curve를 선택한다.[4]

F_2^m 체위의 Nonsupersingular Curve는 $y^2 + xy = x^3 + a_2x^2 + a_6$(이하(1))의 공식을 만족시키는 해들의 쌍이며, a_2, $a_6 \in F_2^m$ $a_6 \neq 0$ 이다. 또한 무한점(Infinity Point)라고 불리는 특별한 형태의 점을 가진다. 이러한 Nonsupersingular Curve 위의 점들 간의 덧셈은 $P = (x_1, y_1) \in E$이고 $-P = (x_1, y_1+x_1)$로 정의가 된다. 또한, 만약 $Q = (x_2, y_2) \in E$ 이고 $Q \neq -P$ 이면 $P + Q = (x_3, y_3)$로 정의된다.

1) $P \neq Q$면

$$\lambda = \frac{y_1 + y_2}{x_1 + x_2}$$

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a_2$$

$$y_3 = (x_1 + x_3)\lambda + x_3 + y_1$$

2) P = Q 이면

$$\lambda = \frac{y_1}{x_1} + x_1$$
$$x_3 = \lambda^2 + \lambda + a_2$$
$$y_3 = (x_1 + x_3)\lambda + x_3 + y_1$$

따라서, 다른 두 점간의 덧셈연산에서는 2번의 체 곱셈과 1번의 체 Inversion이 필요하게 되고, 같은 점간의 덧셈연산에도 동일한 수를 요구하게 된다. 여기서의 체 Inversion은 상당히 복잡한 연산이다. 본 논문에서는 Inversion을 유한체의 특성을 이용, 설계된 곱셈기의 반복 사용하는 방법을 취한다. 이렇게 하면 Inversion의 경우 11번의 곱셈연산으로 대체할 수 있고 칩의 면적을 절약할 수 있다.

본 구현방법은 다음과 같은 세 가지에 의존한다. 첫째로, Supersingular 또는 Nonsupersingular 이냐의 Curve의 선택이다. 본 논문에서는 보다 높은 안전성을 위해서 Nonsupersingular Curve를 선택하였다. 둘째로, Optimal Normal Basis 상에서의 162비트 key size를 선택하였다. 이러한 선택은 MOV attack이나 현재의 컴퓨팅 능력에도 안전한 것으로 알려져 있다. 또한, 하드웨어로 구현 시 RSA 1024비트와 비교 시 오버헤드를 줄일 수 있게 된다. 셋째로, Reference Point인 P의 차수이다. P의 차수는 Pohlig Hellman Attack[2]에 대해서 안전하기 위해서 충분히 큰 Large Prime Number이어야 한다.

3. ECC의 구현

3.1 Optimal Normal Basis에서의 곱셈식 전개

Optimal Normal Basis위에서의 연산은 유한체(F_{2^m})의 두 원소를 A, B라고 하면,

$$A = \sum_{i=0}^{m-1} a_i\beta^{2^i},\ B = \sum_{j=0}^{m-1} b_j\beta^{2^j} \text{ 이고,}$$

$$C = A \times B \sum_{i=0}^{m-1}\sum_{j=0}^{m-1} a_i b_j \beta^{2^i}\beta^{2^j} = \sum_{i=0}^{m-1} c_i\beta^{2^i}$$

$\beta^{2^i}\beta^{2^j} = \sum_{k=0}^{m-1}\lambda_{ijk}\beta^{2^k}$, $\lambda_{ijk} \in \{0, 1\}$ ----> (3.1) 이 된다.

위의 곱셈연산에서 곱셈결과의 원소인 c_k는

$$c_k = \sum_{i=0}^{m-1}\sum_{j=0}^{m-1}\lambda_{ijk}a_i b_j \ ,\ 0 \le k \le m-1 \text{ ----> (3.2)}$$

로 나타내어 질 수 있다.

위의 3.1식의 양변에 2^{-i}승을 취하면

$$(\beta^{2^i}\beta^{2^j})^{2^{-i}} = \beta^{2^{-i}}\beta^{2^{j-i}} = \sum_{k=0}^{m-1}\lambda_{i-1,j-1,k}\beta^{2^k} = \sum_{k=0}^{m-1}\lambda_{ijk}\beta^{2^{k-i}}$$

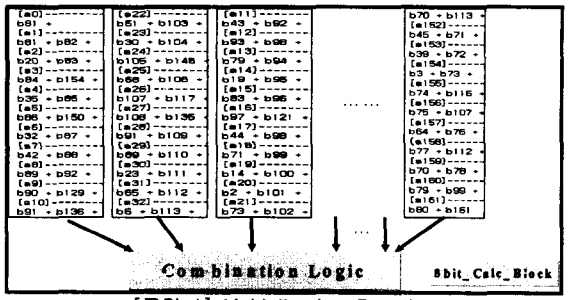
와 같이 계산되고, 위의 식에서 β^{2^0}를 계산하면,

$$\lambda_{ijl} = \lambda_{i-l,j-l,0} \text{ for all } 0 \le i, j, l \le m-1$$

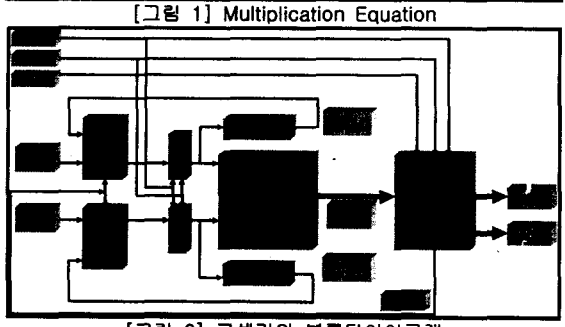
이 된다. 그러므로 3.2식은 다음과 같이 쓸 수 있다.

$$c_k = \sum_{i=0}^{m-1}\sum_{j=0}^{m-1}\lambda_{i-k,j-k,0}a_i b_j = \sum_{i=0}^{m-1}\sum_{j=0}^{m-1}\lambda_{ij0}a_{i+k}b_{j+k}$$

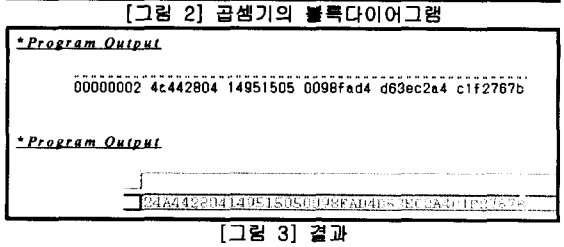
-----> (3.3) (모든 subscript는 modulo k)



[그림 1] Multiplication Equation



[그림 2] 곱셈기의 블록다이어그램



[그림 3] 결과

ONB는 3.3식에서 최소 개수의 0이 아닌 항들을 가진다. ($C_N = 2m-1$) 위의 식을 살펴보면 $k=0$일 때의 λ행렬, 즉 λ_{ij0}를 구하면 나머지 k에 대한 연산은 a, b의 rotate shift에 의해 구해질 수가 있다. 이런 특성 때문에 하드웨어 구현 시 최소한의 복잡도를 가지게 되는 것이다.

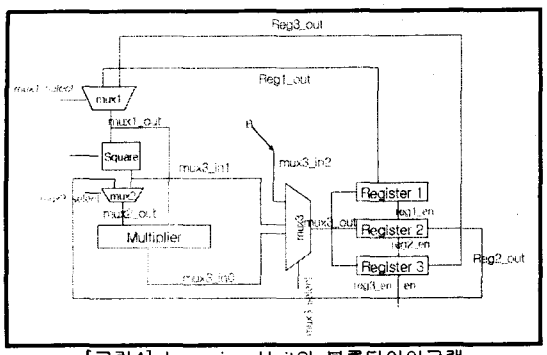
3.2 곱셈기의 구현

Type I에 해당되는 m=162의 크기를 가지고, MO(Massy Omura)곱셈기에 비해 연산을 수행하는 속도는 떨어지나 area 측면에서의 이득을 얻어 스마트카드에 적합하도록 곱셈기를 구현하였다. 이는 곱셈결과의 한 bit를 계산하는 조합회로를 8 bit의 연산이 한번에 가능하도록 8개로 구성하여 1 clock에 8bit의 결과를 계산하도록 하여 전체적으로 22 clock cycle후에 전체적인 곱셈의 결과가 출력에 나오는 구조이다.(그림 2)

두 입력 값에서 한 bit의 결과에 연관되는 bit위치를 알려주는 λ 행렬을 구하는 것이 가장 중요한 부분이다. 조건에 맞는 i,j의 값을 찾아 곱셈을 위한 곱셈식을 구하고 그 것을 조합회로로 구현하면 된다.

λ 행렬을 구하는 것은 C언어(그림 1)를 이용해 프로그램화 하였고, 그 결과를 검증하기 위한 검증용 프로그램을 설계하였다.(그림 3)

위의 곱셈기는 Synopsys Design Compiler를 통해 합성 시 14,632의 면적을 나타내었고 2.33ns의 critical path를 가졌다.

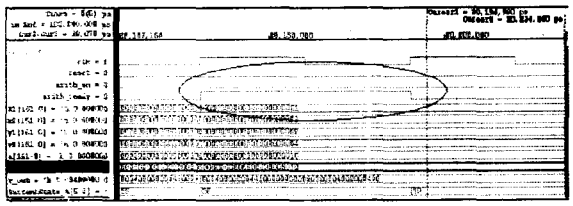


[그림4] Inversion Unit의 블록다이어그램

3.3 Inversion Unit

Inversion의 경우 그림4와 같은 구성을 지닌다. 이는 Inversion 연산의 경우 자주 사용되지 않으므로 따로 설계하는 것은 면적의 심각한 낭비를 초래한다. 따라서 수학적인 계산을 통하여 주어진 곱셈기를 재사용하여 Inversion 값을 계산할 수 있게 된다.[5] 이러한 Inversion Unit은 곱셈기를 11번 사용하는 알고리즘을 사용하였으며 이때 Elliptic Curve위의 두 점 간의 덧셈 연산 시 한번의 Inversion을 요하게 된다.

4. 실험결과 및 분석



[그림5] 전체연산의 결과파형

위의 그림 5는 Elliptic Curve위의 두 점간의 덧셈 시의 결과 파형을 나타낸다. 이때 한번의 점간의 덧셈 연산을 마치는데 걸리는 시간은 두 점이 같을 경우와 같지 않을 경우의 입력 값이 주어질 때 각각 331, 335 클록 사이클이 소요되며 클록의 최소주기는 3ns 이다. 또한 Area는 37,111를 기록했다.

이는 실제적으로 3ns의 클록 속도와 하나의 점간의 덧셈연산이 소요되는데 걸리는 시간과의 곱을 취하면, 약 1μs정도의 시간이 소요된다고 볼 수 있다. 또한 점간의 덧셈을 위해 필요한 최소의 prime number인 k가 충분히 클 경우에는 MO구조와 같은 곱셈기에서와 1bit단위로 결과 값을 도출하게 되면 스마트카드 내에서의 전체계산 시간이 너무 오래 걸리게 된다.

따라서, 현재의 공정기술을 고려해 보면, 어느 정도의 하드웨어의 중복을 허용하더라도 스마트카드의 계산시간의 요구조건을 맞추는 것이 중요하다.

ECC 시스템에서의 오버헤드의 대부분은 곱셈기의 설계에 있으므로 곱셈기의 병렬화 구조를 채택하면 이러한 문제를 해결할 수 있게 된다.

하지만, 곱셈기 최적화 구조를 판별하기 위해서 병렬화도(비트단위로 표기) * 처리시간 (ns)의 조건을 가해보면 8비트단위로 입력 값을 모듈화시에 가장 최소의 값을 얻을 수 있게 된다.

5. 결론 및 향후연구

본 논문에서는 162bits의 Key Size를 가지고서도 RSA 1024bits의 암호학적 강도를 지니는 스마트카드용으로 적합한 ECC Coprocessor를 구현하였다. ECC의 하드웨어 구현시의 적합성을 위해 162bit Optimal Normal Basis를 선택하였으며, Multiplication은 23 클록 사이클에 수행이 되도록 구현하였으며, Inversion은 Multiplication을 11번 사용하는 알고리즘을 선택하였다. 이때 한번의 점간의 덧셈 연산을 마치는데 331(335) 클록 사이클이 소요되며 클록의 최소주기는 3ns가 소요되었다. 또한 Area는 37,111를 기록했다.

향후 주어진 ECC 프로세서를 버스를 통해 스마트카드에 연결 후 전체 시스템의 구현을 행할 예정이다.

6. 참고문헌

[1] IEEE P1363, "Standard Specifications for Public Key Cryptography", 1999, Institute of Electrical and Electronics Engineers, Inc, 1999

[2] A. Menezes, "Elliptic Curve Public Key Cryptosystems", Kluwer Academic Publishers, Boston, 1993

[3] A. Menezes, "Applications of Finite Field", Kluwer Academic Publishers, Boston, 1993

[4] A. Menezs, T. Okamoto and S. Vanstone, "Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field", IEEE Transactions on Information Theory, 39(1993), 1639-1646

[5] Feng, G. -L, *"A VLSI Architecture for fast inversion in GF(2^m)"*, IEEE Transactions On Computers, Vol.38, No. 10 October 1989

[6] Don Johnson, *"The Elliptic Curve Digital Signature Algorithm (ECDSA)."* Certicom Research Technical Paper.