ARM7TDMI 프로세서를 사용한 GF(2^m)상의 타원곡선 암호시스템 구현

신종훈, 박동진, 이필중

포항공과대학교, 전자공학과

Software Implementation of Elliptic Curve Cryptosystems over Binary Field for ARM7TDMI Processor

Jong Hoon Shin, Dong Jin Park and Pil Joong Lee

Department of Electronic & Electrical Engineering, POSTECH

요 약

본 논문은 ARM7TDMI 프로세서를 사용하여 유한체 GF(2^m) 상에 정의된 타원곡선 암호시스템을 구현한 결과를 제시한다. 타원곡선의 점을 표현하는 좌표계에 따른 비교를 하였고, 사전 계산과 사전 계산을 하지 않는 알고리즘의 구현 결과를 비교하고 있다.

I. 서론

최근 무선 통신 시스템 시장은 급속도로 성장하고 있다. 또한 스마트카드나 PDA 같이 일반적인 컴퓨터가 아닌 시스템에서의 보안 또한 큰 관심의 대상이 되고 있다. 하지만 스마트카드나 PDA, 휴대폰 같이 일반적인 컴퓨터가 아닌 특정 목적을 위해 만들어진 시스템의 경우 암호학적 기술을 구현하는데 여러 가지 제약이 따른다. 프로세서의 처리 속도와 사용할 수 있는 메모리양과 프로세서의 구조 같은 것을 고려할 필요가 있다.

ARM7TDMI 칩의 경우 32-bit RISC 마이크로 프로세서로 0.25mW/Mhz 정도의 저전력을 소모하며 32-bit 마이크로 프로세서이면서 동시에 16-bit 명령어(Thumb 명령어)도 지원을 한다. 또한 연산에 사용 가능한 레지스터는 31개가 있다.

본 논문에서는 유한체의 표현과 복합좌표계, 유한체의 연산에 대해 먼저 언급을 하겠다. 속도면에서 가장 큰 부분을 차지하는 상수 배 연산을 구현하는 알고리즘을 설명하며, 사전 계산량에 따라 구현한 결과를 비교할 것이다.

구현 환경은 ADS 1.1을 사용하여 C 언어로 구현하였으며, ARM7TDMI 10Mhz로 측정한 결과를 제시하였다.

II. 유한체 GF(2^m) 연산

1. 유한체 표현

본 논문에서는 다항식 기저(polynomial basis)에서 구현한 결과를 제시한다. GF(2^m)의 원소는 아래와 같이 다항식의 형태와 벡터 형태로 표현이 가능하다.[1]

$$a \in GF(2^m), a(x) = \sum_{i=0}^{m-1} a_i x^i \quad (1)$$

$$a(x) = (a_{m-1}, a_{m-2}, \ldots, a_0) \quad (2)$$

2. 유한체 덧셈

GF(2^m)에서의 덧셈은 $\lceil m/32 \rceil$ 번의 XOR연산으로 구현 가능하다. 또 XOR 연산 경우 가장 빠른 연산 중에 하나이므로 속도는 무시할 수 있을 정도로 빠르다.

3. 유한체 곱셈

c = a*b를 계산할 때 a(x)*b(x)를 먼저 계산한 후 기약 다항식 f(x)로 모듈러 감소를 하여 c(x)를 계산하게 된다.

1) 다항식 곱셈

알고리즘 1은 a(x)*b(x)를 계산하는 알고리즘이다[13]. 먼저 $0 \le u < 2^w$인 u에 대해 다항식 B_u = u*b(x)를 계산한다. 각각의 과정에서 a의 w 비트만큼 검색하여 미리 계산된 B_u를 이용하여 더해주면 c'(x)=a(x)*b(x)를 계산할 수 있다.

입력: a(x), b(x)
출력: c'(x) = a(x)*b(x)
1: 모든 u에 대해 $B_u(x)$ = u(x)*b(x)계산한다.
2: C' ← 0
3: for i = 7 down to 0 do
4: for j = 0 to s-1 do
5: u = (u_3, u_2, u_1, u_0)
u_k는 A[j]의 (4i+k) 번째 bit라 하면
6: C'{j} = C'{j}⊕B_u
7: end for
8: if i ≠ 0 then
9: C' ← C'x^4
10: end if
11: end for
12: Return c'(x)

알고리즘1. Lift-to-right Comb method[2]
(window size 4)

2) 다항식 모듈러 연산

기약 다항식 f(x)는 선택된 유한체에 따라 3항 다항식 또는 5항 다항식이다. 예를 들어 유한체 GF(2^m)에서 m이 163일 때 f(x)를 $x^{163} + x^7 + x^6 + x^3 + 1$ 라고 하면 모듈러 감소는 다음과 같은 방법으로 할 수 있다.

$$x^{163} = x^7 + x^6 + x^3 + 1 \bmod f(x)$$
$$x^{164} = x^8 + x^7 + x^4 + x \bmod f(x)$$
$$\vdots$$
$$x^{324} = x^{168} + x^{167} + x^{164} + x^{161} \bmod f(x)$$

따라서 c(x)의 상위 비트부터 검색하여 아래 비트에 XOR하는 식으로 모듈러 감소를 구현하는 것이다. 하지만 실제 구현 시 위의 알고리즘을 워드 단위로 적용하여 구현하였다[2].

4. 그 외의 필요한 연산

GF(2^m)에서 제곱의 경우 $a(x) = \sum_{i=0}^{m-1} a_i x^i$ 라면 $a^2(x) = \sum_{i=0}^{m-1} a_i x^{2i}$ 이고 이것은 $a^2(x)$이진 벡터 표현은 $a(x)$이진 벡터 표현 연속하는 비트 사이에 0을 넣음으로써 구현 가능하다. 역원을 구하는 것은 Almost Inversion 알고리즘으로 구현할 수 있다[2].

5. Timings

아래의 표1과 같이 덧셈은 무시가 가능할 정도로 빨랐다. 역원의 경우 곱셈 보다 16배 이상 느렸다.

유한체 연산	time(ms)
덧셈	0.010
곱셈	0.467
제곱	0.073
역원	7.633

표1. 유한체 연산 실행 시간
GF(2^{163})에서 $f(x) = x^{163} + x^8 + x^2 + x^1 + 1$

III. 타원곡선

1. 타원곡선 점의 표현

1) 아핀좌표계(Affine Coordinates)

표수가 2인 유한체 GF(2^m)에서 정의된 타원곡선은 아래의 Weierstrass 방정식을 만족하는 근의 집합으로 표현할 수 있다[1].

$$E : y^2 + xy = x^3 + ax^2 + b, \quad a, b (\neq 0) \square GF(2^m) \tag{3}$$

$P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$라고 할 때 타원곡선 상의 두 점을 더한 점 $P_3 = (x_3, y_3)$은 아핀 좌표계에서 아래와 같이 구할 수 있다.

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$$
$$y_3 = (x_1 + x_3)\lambda + x_3 + y_1$$
$$\lambda = \frac{y_1 + y_2}{x_1 + x_2} \text{ if } (P_1 \neq P_2) \tag{4}$$
$$\lambda = \frac{y_1}{x_1} + x_1 \text{ if } (P_1 = P_2)$$

2) 사영좌표계(Projective Coordinates)

GF(2^m)연산에서 역원을 구하는 것이 곱하기보다 훨씬 느릴 경우 사영좌표계를 사용할 수 있다. 아래의 표2는 각각의 좌표계에서 덧셈과 두배 연산의 계산량을 비교하고 있다.

좌표계	덧셈	덧셈 (복합좌표계)[4]	두배 연산
아핀좌표계	1I, 2M	-	1I, 2M
사영좌표계	14M	9M	4M

표2. 덧셈 연산과 두배 연산의 계산량[2]

사영좌표계의 점은 아핀좌표계의 점(X/Z, Y/Z^2)을 (X, Y, Z), $Z \neq 0$ 인 점에 대응시킨 것으로 타원곡선 방정식과 덧셈, 두배 연산 공식은 아래와 같다.

$$Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4 \tag{5}$$

4) 복합좌표계에서의 덧셈은 아핀좌표계와 다른 좌표계 (여기서는 사영좌표계) 간의 덧셈을 말한다.

$$2(X_1, Y_1, Z_1) = (X_3, Y_3, Z_3)$$
$$Z_3 = X_1^2 Z_1^2,\ X_3 = X_1^4 + bZ_1^4,$$
$$Y_3 = bZ_1^4 Z_3 + X_3 (aZ_3 + Y_1^2 + bZ_1^4) \quad (6)$$

$$(X_1, Y_1, Z_1) + (X_2, Y_2, 1) = (X_3, Y_3, Z_3)$$
$$A = Y_2 Z_1^2 + Y_1,\ B = X_2 Z_1 + X_1,$$
$$C = Z_1 B,$$
$$D = B^2 (C + aZ_1^2),\ Z_3 = C^2,$$
$$E = AC,\ X_3 = A^2 + D + E, \quad (7)$$
$$F = X_3 + X_2 Z_3,$$
$$G = X_3 + Y_2 Z_3,\ Y_3 = EF + Z_3 G$$

일반적으로 역원을 구하는 것이 곱셈을 하는 것 보다 시간이 많이 걸린다. 본 논문에서 구현한 결과는 한 번의 역원 계산이 16번의 곱셈 비슷한 시간이 나왔다. 따라서 역원을 구하는 것을 피하는 것이 효율적인 구현을 위한 방법이 될 수 있다. 따라서 아핀좌표계의 점과 사영좌표계의 점을 더하여 덧셈을 하고 두배 연산에 사영좌표계를 사용한다면 상대적으로 늦은 역원을 구할 필요도 없고 곱셈의 수도 적게 할 수 있다[2].

2. 상수배 알고리즘

1) 임의의 타원 곡선

가. 사전 계산을 하지 않는 경우

일반적인 GF(2^m)상의 타원곡선에서 사전 계산을 하지 않고 점 P를 상수 k배 만큼 하는 알고리즘은 여러 가지가 있다. k를 덧셈 사슬로 만들어 두배 연산과 덧셈 연산을 하는 방법의 경우 m번의 두배 연산과 m/2 정도의 덧셈 연산이 필요하다. 타원곡선의 경우 아핀 좌표계에서 P=(x, y)이면 -P=(x, x+y)이므로 -1을 곱하는 것은 유한체 덧셈 한번이므로 속도 면에서 무시할 수 있다. 이를 이용하여 k를 이진수로 전개할 때 1과 -1을 허용하여 덧셈-뺄셈 사슬을 만들 경우 두배 연산량은 덧셈사슬과 같지만 덧셈 연산을 m/3으로 줄일 수 있다. 다른 방법으로 몽고메리 방법과 Point halving[10]이 있는데 몽고메리 방법은 x-좌표만으로 상수배를 구하는 방법이다. 몽고메리 방법의 경우 상수배에 걸리는 시간이 상수의 비트 길이에 비례하고 상수배 하는 과정에서 덧셈과 두배 연산을 동시에 하기 때문에 side channel attack(SPA)를 방어 할 수 있는 장점이 있다[7]. Point halving의 경우 두배 연산을 사용하지 않고 두배 연산의 역함수라 할 수 있는 halving(1/2P)을 사용한다. 하지만 halving을 한 후 계산이 올바르게 되었는지 확인하는 단계가 필요하기 때문에 비교적 속도 면에서 큰 이점을 가지지 못할 것으로 예상된다. 특히 처리 능력이 제한되어 있는 프로세서의 경우 조건 처리문은 속도에 많은 저하를 가져 올 수 있어 구현에서 제외하였다.

알고리즘	time(ms)
덧셈 사슬	850.443
덧셈-뺄셈 사슬	731.556
몽고메리	525.667

표3. 임의의 타원 곡선 구현 결과
WTLS EC Curve 5[11]

나. 사전 계산을 하는 경우

상수배를 할 때 계산 과정에서 사용될 값을 미리 계산하여 저장할 수 있다면 계산량을 줄일 수 있다. 그리한 방법으로는 w-ary 방법과 signed sliding window 방법, Lim-Lee[5] 방법이 있으나 Lim-Lee 방법의 경우 사전 계산을 비교적 많이 해야 하기 때문에 구현에서는 제외하였다. w-ary 방법의 경우 사전 계산에 2^w-1개의 메모리가 필요하며 유한체 GF(2^m)에서 m-w번의 두배 연산과 m/w의 덧셈 연산이 필요하다. Signed sliding window 방법은 2^{w-1}개의 메모리가 필요하며 m-w번의 두배 연산과 m/(w+2)의 덧셈이 필요하다. w-ary에 비해 사전 계산에 사용되는 메모리양은 반 정도로 줄고 덧셈 연산량도 줄어들지만 w-ary방법에 비해 chain을 만드는 과정이 조금 어렵다.

알고리즘(w=3)	time(ms)
w-ary	700.500
signed sliding window	652.667

표4. 사전 계산을 허용할 때 임의의 타원곡선 구현 결과(표3과 같은 타원곡선)

2) Koblitz 곡선

유한체 GF(2^m)에서 정의된 타원곡선 중 b = 1 이고 a = 0 또는 1인 것을 특별히 이진 비정규곡선 또는 Koblitz 곡선이라 부른다. Koblitz 곡선에서도 위에서 언급한 임의의 타원 곡선에서의 방법을 사용할 수 있지만 Frobenius 사상을 이용하여 상수 k를 τ-adic NAF로 전개하여 상수배 할 경우 일반적인 방법보다 2배 정도 빠르게 할 수 있다. 이 방법은 상수배 연산에서 큰 부분을 차지하는 두배 연산 대신 τ연산을 사용하는 것으로 τ의 경우 아핀 좌표계에서 다항식 기저에서는 2번의 제곱으로 정규 기저에서는 2번의 쉬프트로 구현이 가능하다. 또한 window 방법을 사용할 경우 속도 향상이 가능하다[9].

방법	time(ms)
TNAF	648.000
Reduction & TNAF	330.000
w-TNAF	435.333
Reduction & w-TNAF	250.667

표5. Koblitz 구현 결과
SECG SEC2 sect163k1 Curve[12]

IV. 결론

본 논문에서는 ARM7TDMI 10Mhz 프로세서를 사용하여 유한체 $GF(2^m)$상에 정의된 타원곡선 암호시스템을 효율적으로 구현할 수 있는 몇 가지 알고리즘 구현 결과를 제시하고, 그 수행 속도를 비교하였다. 임의의 타원곡선에서는 사전 계산을 하지 않은 몽고메리 방법이 사전 계산을 한 다른 방법에 비해 빨랐다. 물론 몽고메리 방법은 m의 크기에 비례하여 수행속도가 늘어나므로 m의 크기가 크면 사전 계산을 한 signed sliding window 방법이 더 빠를 것이다. Koblitz 곡선은 몽고메리 방법에 비해 최대 2배 정도 빠른 것으로 나왔다.

참고문헌

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

[2] D. Hankerson, J. L. Hernandez and A. Menezes. "Software Implementation of Elliptic Curve Cryptography Over Binary Fields," *CHES 2000*, LNCS 1965 1-24, 2000

[3] N. Koblitz, "CM-curves with good cryptographic properties," *CRYPTO'91*, LNCS 576, 279-287, 1992

[4] K. Koyama and Y. Tsuruoka, "Speeding up elliptic curve cryptosystems by using a signed binary window method," *CRYPTO'92*, LNCS 740, 345-357, 1993

[5] C. Lim and P. Lee, "More flexible exponentiation with precomputatoin," *CRYPTO'94*, LNCS 839, 95-107, 1994

[6] J. López and R. Dahab, "Improved Algorithms for Elliptic Curve Arithmetic in $GF(2^n)$," *SAC'98*, LNCS 1556, 201-212, 1999

[7] J. López and R. Dahab, "Fast multiplicatoin on Elliptic Curves over $GF(2^n)$ without precomputation," *CHES'99*, LNCS 1717, 316-327, 1999

[8] A.J. Menezes, P.C. van Oorschot and S.A. Vanstone, "Handbook of Applied Cryptography," CRC Press, 1996

[9] J. A. Solinas, "Efficient Arithmetic on Koblitz Curves," Designes, Codes and Cryptography, 19(2/3), 195-249, 2000

[10] E. Knudsen, "Elliptic scalar multiplicatoin using point halving," *Asiacrypt'99*, LNCS1716, 135-149, 1999

[11] WAP Forum, "Wireless Transport Layer Security," Approved version 06-Apr-2001, 2001

[12] SECG SEC2, "Recommended Elliptic Curve Cryptography Domain Parameters," 2000

[13] J. López and R. Dahab, "High-speed software multiplicatoin in F_{2^m}", preprint, 2000