$\mathbb{F}_{2^m}$ 상에서 곱셈에 대한 역원을 구하는 빠른 알고리즘

김이용[0] 김진욱 박근수
서울대학교 전기·컴퓨터공학부
{eykim,cwkim,kpark}@theory.snu.ac.kr

Fast Algorithms for Finding Multiplicative Inverses in $\mathbb{F}_{2^m}$

E-Yong Kim[0] Jin Wook Kim Kunsoo Park
School of Electrical Engineering and Computer Science, Seoul National University

요 약

타원 곡선이 정의되는 유한체의 연산 중 곱셈에 대한 역원을 빠르게 구하는 것은 타원 곡선 암호시스템의 성능 향상에 있어 중요한 요소이다. 본 논문에서는 이진체 $\mathbb{F}_{2^m}$ 상에서 다항식 기저를 사용하는 경우 곱셈에 대한 역원을 빠르게 구하는 알고리즘을 제시한다. 이 알고리즘은 기약 다항식으로부터 미리 계산 가능한 테이블을 만들어 테이블 참조 방식으로 속도 향상을 꾀한다. 이 방법을 사용할 경우 이전에 알려진 가장 빠른 방법보다 10 ~ 20% 정도 성능 향상이 있다.

1 서론

타원 곡선 암호시스템은 1985년 Koblitz [1]와 Miller [2]에 의해서 각각 독립적으로 제안되었다. 그 이후 타원 곡선 암호시스템의 효율적인 구현을 위한 여러가지 방법들이 제시되어 왔다. 타원 곡선이 정의되는 유한체의 연산 속도를 증가시키는 것은 그 중 한 방법이다. 유한체 연산 중에서 곱셈에 대한 역원을 구하는 것은 다른 연산에 비해 대단히 느린 연산으로서, 이것을 빠르게 구현하는 것은 전체 암호시스템의 성능 향상에 있어 중요한 요소이다.

이진체 $\mathbb{F}_{2^m}$ (m은 소수) 상에서 다항식 기저(polynomial basis)를 사용하여 원소를 표현하는 경우, 곱셈에 대한 역원을 구하는 방법으로는 확장 유클리드 알고리즘(Extended Euclid's Algorithm, EEA) [3]과 거의 역원 알고리즘(Almost Inverse Algorithm, AIA) [4]이 많이 사용되고 있다. 후자가 전자에 비해서 더 빠른 알고리즘으로 알려져 있다. 그러나 이 방법은 모듈러 약분(modular reduction)에 사용되는 기약 다항식(irreducible polynomial)에 낮은 차수항(상수항 제외)이 있을 경우 성능 향상이 그다지 많지 않은 것으로 알려져 있다 [5]. 한편, 표 1에서 볼 수 있듯이 ANSI X9.62 [6]에서 제안하는 기약 다항식에는 낮은 차수항을 갖는 것이 많다.

m	기약 다항식
163	$f(x) = x^{163} + x^7 + x^6 + x^3 + 1$
233	$f(x) = x^{233} + x^{74} + 1$
283	$f(x) = x^{283} + x^{12} + x^7 + x^5 + 1$
409	$f(x) = x^{409} + x^{87} + 1$
571	$f(x) = x^{571} + x^{10} + x^5 + x^2 + 1$

표 1. ANSI X9.62에서 제안하는 기약 다항식

본 논문에서는 AIA를 [3]에 나와있는 확장 이진 최대공약수 알고리즘(Extended Binary GCD Algorithm, EBGA)으로 변경한 후, 기약 다항식으로부터 미리 계산 가능한 테이블을 만들어 테이블 참조 방식으로 속도를 향상시키고자 한다. 두 가지 알고리즘을 제안하는데, 하나는 비교적 작은 크기의 테이블을 이용하여 이전 알고리즘들에 비해서 상당한 속도 향상을 보여주고, 또 다른 하나는 더 큰 크기의 테이블을 유지하여 더욱 좋은 성능을 보여준다.

2 이전 연구

본 논문에서는 다음과 같은 표현을 사용한다.

- x를 곱한다 - 왼쪽으로 1비트 쉬프트 연산
- x로 나눈다 - 오른쪽으로 1비트 쉬프트 연산
- 다항식이 짝수다 - 상수항이 0
- 다항식이 홀수다 - 상수항이 1

2.1 확장 유클리드 알고리즘

알고리즘 1은 [3]의 EEA를 다항식에 맞게 변형한 것이다. 이 알고리즘은 $ba + df = u$와 $ca + ef = v$의 관계를 계산 중에 유지한다. 이 관계식에서 d와 e는 필요하지 않으므로 실제로 계산되지 않는다. 루프를 수행할 때마다 6번째 줄에서 u의 차수는 적어도 하나 이상 줄어들게 된다. b도 여기에서 차수가 조절되어 위의 관계가 유지되도록 한다. $\deg(u) = 0$, 즉 $u = 1$일 때 이 알고리즘은 b ($\equiv a^{-1} \bmod f(x)$)를 반환하고 종료한다.

알고리즘 1. 확장 유클리드 알고리즘 (EEA)
입력: $a \in \mathbf{F}_{2^m}$, $a \neq 0$
출력: $a^{-1} \mod f(x)$
1: $(b,u) \leftarrow (1,a)$, $(c,v) \leftarrow (0,f)$
2: **loop**
3: **if** $\deg(u) = 0$ **then return** b
4: $q \leftarrow \deg(u) - \deg(v)$
5: **if** $q < 0$ **then** $(b,u) \leftrightarrow (c,v)$
6: $(b,u) \leftarrow (b,u) + (c,v)x^{|q|}$
7: **end loop**

2.2 거의 역원 알고리즘

알고리즘 2는 [4]에서 Shroeppel 등이 제안한 방법으로 곱셈에 대한 역원을 직접 구하는 대신 $ab \equiv x^k \mod f(x)$를 만족하는 b, k를 구한다. 따라서 진짜 곱셈에 대한 역원을 구하려면 이 알고리즘 수행 후 b를 $x^k \pmod{f(x)}$로 나누는 과정이 필요하다.

이 알고리즘은 $ba + df = ux^k$와 $ca + ef = vx^k$라는 관계를 계산 중에 유지한다. EEA와 마찬가지로 여기서도 d와 e는 실제로 계산되지 않는다. 3-4의 while 루프를 수행하고 나면 u와 v는 홀수가 되고, 7번째 줄을 수행하고 나면 u는 다시 짝수가 된다. 따라서 x로 나눌 수 있게 되어 전체 루프를 한 번 수행할 때마다 u의 차수는 줄어들게 된다. 한편, 4번째 줄에서 k를 증가시킴으로써 위의 관계식이 유지되도록 한다. $u = 1$일 때 b, k를 반환하고 이 알고리즘은 종료한다.

알고리즘 2. 거의 역원 알고리즘 (AIA)
입력: $a \in \mathbf{F}_{2^m}$, $a \neq 0$
출력: b, k s.t. $ab \equiv x^k \mod f(x)$, $\deg(b) < m$, $k < 2m$
1: $k \leftarrow 0$, $(b,u) \leftarrow (1,a)$, $(c,v) \leftarrow (0,f)$
2: **loop**
3: **while** u가 짝수 **do**
4: $u \leftarrow u/x$, $c \leftarrow cx$, $k \leftarrow k+1$
5: **if** $u = 1$ **then return** b, k
6: **if** $\deg(u) < \deg(v)$ **then** $(b,u) \leftrightarrow (c,v)$
7: $(b,u) \leftarrow (b,u) + (c,v)$
8: **end loop**

EEA와 AIA의 차이점은 EEA의 경우 u와 v의 비트를 높은 차수에서 낮은 차수 방향으로 없애는 반면, AIA는 낮은 차수에서 높은 차수 방향으로 없앤다는 점이다.

2.3 확장 이진 최대공약수 알고리즘

알고리즘 3은 알고리즘 2를 약간 변형시켜서 역원을 바로 구할 수 있도록 한 것이다 [7]. 사실 이 방법은 [3]에 제시되어 있는 EBGA를 그대로 구현한 것이다. 이 알고리즘은 k를 유지하지 않고, u를 x로 나눌 때마다 b도 x로 나눈다. 만약 b가 홀수이면 $b + f$를 x로 나눈다. 따라서 이 알고리즘은 종료할 때 진짜 곱셈에 대한 역원을 반환하게 된다.

알고리즘 3. 확장 이진 최대공약수 알고리즘 (EBGA)
입력: $a \in \mathbf{F}_{2^m}$, $a \neq 0$
출력: $a^{-1} \mod f(x)$
1: $(b,u) \leftarrow (1,a)$, $(c,v) \leftarrow (0,f)$
2: **loop**
3: **while** u가 짝수 **do**
4: $u \leftarrow u/x$
5: **if** b가 짝수 **then** $b \leftarrow b/x$
6: **else** $b \leftarrow (b+f)/x$
7: **if** $u = 1$ **then return** b
8: **if** $\deg(u) < \deg(v)$ **then** $(b,u) \leftrightarrow (c,v)$
9: $(b,u) \leftarrow (b,u) + (c,v)$
10: **end loop**

3 제안 방법

x로 나누는 연산은 쉬프트 연산이므로 x로 $i(\geq 0)$번 나누는 것보다 한 번에 x^i으로 나누는 것이 더 빠르다. 본 논문에서 제안하는 방법은 이 사실에 기초하여 고안되었다.

3.1 제안 알고리즘 1

알고리즘 3에서 3-6의 while 루프를 보면 u와 b를 여러 번 x로 나누는 연산을 수행한다. u의 하위 0비트의 개수를 s라고 하면 알고리즘 3의 while 루프는 s번 반복하게 되는데, 그 대신 u를 한 번에 x^s으로 나누는 것이 가능하다. 그런데 b의 경우는 하위 s비트가 모두 0이 아니기 때문에 약간의 처리가 필요하다. b의 하위 $i(\geq 0)$번째 비트를 b_i라고 하자. $b_i = 1$인 i에 대해서 기약 다항식 $f(x)$에 x^i을 곱한 것을 b에 더해 주면 $b_i = 0$이 된다. 이 작업을 s번 반복하면 b의 하위 s비트는 모두 0이 된다. 이런 방식으로 수정한 것이 다음의 알고리즘 4이다.

알고리즘 4. 개선된 확장 이진 최대공약수 알고리즘 1
(Modified Extended Binary GCD Algorithm 1, MEBGA1)
입력: $a \in \mathbf{F}_{2^m}$, $a \neq 0$
출력: $a^{-1} \mod f(x)$
1: $(b,u) \leftarrow (1,a)$, $(c,v) \leftarrow (0,f)$
2: **loop**
3: $s \leftarrow u$의 하위 0비트 개수
4: $u \leftarrow u/x^s$
5: **for** $i = 0$ **to** $s-1$ **do**
6: **if** $b_i = 1$ **then** $b \leftarrow b + fx^i$
7: $b \leftarrow b/x^s$
8: **if** $u = 1$ **then return** b
9: **if** $\deg(u) < \deg(v)$ **then** $(b,u) \leftrightarrow (c,v)$
10: $(b,u) \leftarrow (b,u) + (c,v)$
11: **end loop**

$0 \leq i < s$에 대하여 fx^i을 미리 테이블로 만들어 놓으면 계산 속도를 더 향상시킬 수 있다. 기약 다항식은 미리 정해지므로 이 테이블 역시 미리 계산해 놓을 수 있다. 32비트 구조를 가정하였을 때, m차 기약 다항식 하나를 저장하는데 필요한 메모리의 크기는 $\lceil m/32 \rceil$워드이므로 $0 \leq i < w$에 대한 테이블을 저장하는데는 $\lceil m/32 \rceil \times w$워드만큼의 메모리가 필요하게 된다.

3.2 제안 알고리즘 2

보다 메모리를 제약 없이 사용할 수 있다면 알고리즘 4의 5-6 for문을 덧셈 연산 한 번으로 줄일 수 있다. 다음 알고리즘 5는 b의 하위 s비트를 0으로 만들기 위하여 b에 더해질 기약 다항식들을 미리 더하여 그 결과를 테이블로 유지한다.

알고리즘 5. 개선된 확장 이진 최대공약수 알고리즘 2
(Modified Extended Binary GCD Algorithm 2, MEBGA2)

입력: $a \in \mathbf{F}_{2^m}$, $a \neq 0$
출력: $a^{-1} \mod f(x)$
미리 계산:
for $v = 0$ **to** $2^w - 1$ **do**
  $t \leftarrow 0$
  for $i = 0$ **to** $w - 1$ **do**
    if $v_i = 1$ **then** $t \leftarrow t + fx^i$
  $T[t_{w-1} \ldots t_1 t_0] \leftarrow t$
1: $(b, u) \leftarrow (1, a)$, $(c, v) \leftarrow (0, f)$
2: **loop**
3: $s \leftarrow u$의 하위 0비트 개수
4: $u \leftarrow u/x^s$
5: $b \leftarrow b + T[b_{s-1} \ldots b_1 b_0]$
6: $b \leftarrow b/x^s$
7: **if** $u = 1$ **then return** b
8: **if** $\deg(u) < \deg(v)$ **then** $(b, u) \leftrightarrow (c, v)$
9: $(b, u) \leftarrow (b, u) + (c, v)$
10: **end loop**

테이블 T를 유지하는데 필요한 메모리의 크기는 $\lceil m/32 \rceil \times 2^w$워드이다.

4 실험 결과

표 1에 나와 있는 ANSI X9.62에서 제안하는 기약 다항식들에 대하여 실험을 수행하였다. Pentium III 866MHz × 2 워크스테이션에서 C로 구현하였으며 어셈블러는 사용하지 않았다. 다음 표 2는 각 경우에 대하여 1000번을 반복 실험한 후 이를 평균낸 것이다. 여기서 AIA의 수행 시간은 마지막 결과를 x^k으로 나누는 과정까지 포함한 시간이다.

m	163	233	283	409	571
EEA	62.20	94.10	117.56	202.40	317.79
AIA	62.91	107.75	139.85	231.16	379.91
EBGA	59.72	109.04	131.66	238.16	349.59
MEBGA1	54.63	94.79	117.08	197.49	306.18
MEBGA2	48.77	87.13	106.02	183.15	286.55

표 2. 각 알고리즘의 수행 시간(μs)

위의 표에서 볼 수 있듯이 제안 알고리즘들은 기존에 빠르다고 알려진 알고리즘들(AIA, EBGA)에 비하여 대략 20% 정도의 성능 향상을 보여준다.

미리 계산하여 유지해야 할 테이블의 크기를 실험적으로 살펴보면, u의 연속적인 하위 0비트의 개수는 16개를 넘지 않는다. 따라서 $w = 16$일 때 필요한 메모리 양을 계산해 보면 다음 표 3과 같다.

m	163	233	283	409	571
MEBGA1	0.384	0.512	0.576	0.832	1.152
MEBGA2	1536	2048	2304	3328	4608

표 3. 테이블 저장에 필요한 메모리 크기 (KB)

이 실험에서 한 가지 짚고 넘어갈 점은 우리의 구현 결과에서는 [4]에 나와 있는 결과와는 다르게 EEA가 AIA보다 더 빠르게 나왔다는 점이다. 이것은 [7]의 결과와 일치하는 것이다. EEA는 MEBGA1보다 약간 느리고 MEBGA2보다는 대략 10% 정도 느리다.

5 결론 및 토의

우리는 타원 곡선 암호시스템의 성능 향상에 중요한 요소인 타원 곡선이 정의되는 유한체 상에서 곱셈에 대한 역원을 보다 빨리 구할 수 있는 방법을 제시하였다. 이 방법은 기존의 방법들보다 10 ~ 20% 정도 빠른 결과를 보여주었다.

본 논문은 이진체 상에서 다항식 기저를 사용하여 원소를 표현할 경우 적용할 수 있는 방법이므로, 다른 체 또는 정규 기저(normal basis)와 같은 다른 기저를 사용할 경우에도 곱셈에 대한 역원을 빠르게 구하는 방법이 연구되어야 할 것이다.

참고 문헌

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

[2] V. Miller, "Uses of elliptic curves in cryptography", *Advances in Cryptology - Crypto '85*, LNCS 218, 1986, pp. 417-426.

[3] D. Knuth, *The Art of Computer Programming - Seminumerical Algorithms*, Addison-Wesley, 3rd edition, 1998.

[4] R. Schroeppel, H. Orman, S. O'Malley and O. Spatscheck, "Fast key exchange with elliptic curve systems", *Advances in Cryptology - Crypto '95*, LNCS 963, 1995, pp. 43-56.

[5] E. De Win, S. Mister, B. Preneel and M. Wiener, "On the performance of signature schemes based on elliptic curves", *Algorithmic Number Theory, Proceedings Third International Symposium, ANTS-III*, LNCS 1423, 1998, pp. 252-266.

[6] ANSI X9.62, *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*, 1999.

[7] D. Hankerson, J. Hernandez and A. Menezes, "Software implementation of elliptic curve cryptography over binary fields", *CHES 2000*, 2000.