유한체 GF(2^m)상에서의 빠른 역원계산 기법

박 정 식, 안 금 혁, 김 영 길, 장 청 룡
한국통신 연구개발본부

A Fast Inverse Calculation Method over Finite Field GF(2^m)

Jung-Sik Park, Keum-Hyug Ahn, Young-Kil Kim, Chung-Ryong Jang
Korea Telecom Research and Development Group

요 약

정보보호기법을 적용한 다양한 서비스의 구현에 있어서는 적용기법에서 채택한 암호학적 연산에 의해 그 실용성이 종속하게 되며 이러한 실용화를 위한 하드웨어 또는 소프트웨어적 구현기법에 관한 많은 연구가 진행되고 있다. 본 논문에서는 유한체 GF(2^m)상에서의 역원계산을 효율적이며 신속하게 처리할 수 있는 방법에 관해서 다루고 있다. 본 논문에서 제안하는 방법은 정규기저를 이용하여 임의의 유한체위에 적용 가능하도록 설계된 기법이다. 본 논문에서의 제안 방법은 이미 알려진 Itoh의 방법보다 대부분의 정수에 대하여 효율적임을 보인다.

1. 서 론

유한체 GF(2^m)상에서의 연산은 일반적으로 유한체 GF(p)상에서의 연산과는 다른 점이 있다. GF(p)에서의 연산은 정수 집합에 대해 소수 p의 법연산을 이용하지만 GF(2^m)에서의 연산은 GF(2)[x]에 대해 기약다항식 p(x)의 법연산을 이용한다. 하지만 GF(2^m)에서의 연산은 GF(2)[x]의 원소들에 대해서 이루어지는데 그 원소의 계수들이 모두 GF(2)의 원소이므로 하드웨어적으로 구현하기 편리하고 처리속도도 빠르다. 이러한 이점 때문에 최근에는 암호연산 기법설계시 사용될 체를 GF(2^m)으로 선택하여 암호계 처리에 걸리는 시간을 줄이고 있다. 특별히 전자서명과 같은 기법에서는 역원계산을 이용한 서명 생성과정등에서 이의 적용이 가능하다.

1986년 Itoh등은 정규기저를 이용하여 유한체 GF(2^m)상에서 고속으로 역원을 계산하는 알고리즘을 개발하였다[3]. Itoh의 방법은 계산 속도나 알고리즘 수행시 필요한 중간 메모리의 양에 있어서 상당한 이득을 볼 수 있다. 뿐만 아니라 Itoh는 그 논문에서 최적 정규기저를 이용하면 연산수행에 필요한 클럭의 수를 줄일 수 있음을 설명하였다. 그후 1993년 Vanstone등은 마찬가지로 같은 형태의 유한체 GF(2^m)에서의 역원 계산시 계산 속도면 에서는 Itoh의 방법보다는 느리지만 알고리즘 수행시 필요한 중간 메모리는 전혀 요구되지 않는 알고리즘을 제안하였다[1]. 다시 말하면, Vanstone의 방법은 연산 수행에 필요한 클럭의 수에 있어서 Itoh의 방법보다 훨씬 많이 요구되어 결국 계산 시간이 더 많이 걸리게된다. 그러나 제한된 메모리 공간을 사용해야만 하는 경우 보다 효과적일 수 있다.

본 논문에서 제안한 알고리즘은 Itoh의 방법을 개선한 것으로서 계산 속도 면에 있어서나 알고리즘 수행시 필요한 중간 메모리의 양을 줄일 수 있는 방법이다. 즉, 대부분의 정수 m에 관하여 유한체 GF(2^m)에서의 역원계산시 본 논문에서 소개된 기법이 보다 효율적이며 적어도 나쁘지는 않다. 그리고 정수 m이 커질수록 본 논문에서 제안한 방법이 필요로 하는 클럭 수는 Itoh의 방법에서 필요한 클럭의 수 보다 점점 더 작아진다. 다시 말하면, 큰 비트수 m을 사용할수록 본 논문에서 제안한 방법이 계산속도면에 있어서 더욱 유리하다. 뿐만 아니라 중간 메모리 양의 차이는 클럭 수의 차이보다 훨씬 유리한 결과를 얻을 수 있었다.

2장에서는 본 논문에 필요한 개념과 정의를 살펴보고, 3장에서는 Itoh가 제안한 유한체에서의 역원계산법을 소개하였다. 4장에서는 Itoh가 제안한 역원계산법보다 계산시간과 메모리면 에서 이익이 되는 개선된 알고리즘을 제안하고 이를 평가해본다. 마지막으로 5장에서 이를 정리하고 향후 연구과제를 검토해본다.

2. 개념 및 정의

정규기저는 $GF(2^m)$위의 기저로서 그 기저로 표시된 원소들의 제곱연산을 편리하게 해주는 성질을 갖고 있다.

[정의 1] $\alpha \in GF(2^m)$에 대하여 $\{ \alpha, \alpha^2, \alpha^{2^2}, \cdots, \alpha^{2^{m-1}} \}$가 유한체 $GF(2)$에 관하여 $GF(2^m)$의 기저가 될 때 $\{ \alpha, \alpha^2, \alpha^{2^2}, \cdots, \alpha^{2^{m-1}} \}$를 $GF(2^m)$의 정규 기저(Normal Base)라고 한다.

$GF(2^m)$의 어떤 원소 x가 $x = a_0 \cdot \alpha + a_1 \cdot \alpha^2 + \cdots + a_{m-1} \cdot \alpha^{2^{m-1}}$ 로 표현되었다고 가정하면 $x^2 = a_{m-1} \cdot \alpha + a_0 \cdot \alpha^2 + \cdots + a_{m-2} \cdot \alpha^{2^{m-1}}$가 되어서 x^2은 x의 계수 $(a_0, a_1, \cdots, a_{m-1})$을 오른쪽으로 한번 회전시킨 것과 같게 된다. 결국 x^{2^k}의 계수는 $(a_{m-k}, a_{m-k+1}, \cdots, a_{m-k-1})$가 된다. 따라서, $GF(2^m)$의 어떤 원소를 다항식 기저를 사용하여 제곱하면 $GF(2)$에서 m^2번의 곱셈을 수행하여야 하지만 정규기저를 사용하면 한번의 회전으로 연산이 가능해져 그 만큼 수행속도에 있어서 이익이 된다는 것을 알 수 있다.

모든 유한체는 정규기저를 가지는 것으로 알려져 있고[4] 특정한 m에 대해서는 $GF(2^m)$의 정규기저를 생성하는 α가 $GF(2^m)^*$의 생성자가 된다는 사실도 알려졌다[5].

먼저 α가 $GF(2^m)$의 0이 아닌 원소라고 할 때 α^{-1}는 다음의 식을 만족한다.

$$\begin{aligned} \alpha^{-1} &= \alpha^{2^m-2} \\ &= \alpha^{\sum_{i=1}^{m-1} 2^i} \\ &= \prod_{i=1}^{m-1} \alpha^{2^i} \end{aligned}$$

위의 방법을 이용하면 m-2번의 곱셈을 필요로 하게 되는데 다음에 소개할 방법들은 곱셈의 회수가 많이 절약된다.

3. Itoh의 역원계산 방법

Itoh에 의해 소개된 역원계산 방법은 곱셈회수를 줄이는 데 있어서 지금까지 알려진 가장 효율적인 기술이다.

$GF(2^m)$위에서 다음과 같은 소인수분해를 생각해 보자.

$$2^{m-1}-1 = \begin{cases} (2^{(m-1)/2}-1)(2^{(m-1)/2}+1), & m = \text{홀수}, \\ 2(2^{(m-2)/2}-1)(2^{(m-2)/2}+1) + 1, & m = \text{짝수}. \end{cases}$$

m이 홀수인 경우 $\alpha^{2^{(m-1)/2}-1}$는 이미 계산이 되었다고 가정하면, $\alpha^{2^{m-1}-1}$을 계산하기 위해서는 1번의 곱셈이 필요하다. m이 짝수인 경우는 홀수인 경우와 같이 가정하면 $\alpha^{2^{m-1}-1}$을 계산하기 위해서는 2번의 곱셈이 필요하다. 결국 $GF(2^m)$에서 α^{-1}를 계산하기 위해 연역적으로 이 과정을 사용할 때 요구되는 곱셈의 회수는 다음과 같이 된다[1].

$$nb(m-1) + \omega(m-1) - 2,$$

단, 여기서 $nb(x)$는 정수 x를 표현하기 위해 필요한 최소의 비트 수를 말하며, $\omega(m-1)$은 $m-1$의 이진수 표현에서 해밍 무게(Hamming weight)이다.

$GF(2^{593})$을 예로서 생각해보자.

$$\begin{aligned}
\alpha^{-1} &= \alpha^{2^{593}-2} \\
&= \alpha^{2(2^{592}-1)} \\
&= \alpha^{2(2^{37}-1)(2^{37}+1)(2^{74}+1)(2^{148}+1)(2^{296}+1)}, \\
\alpha^{2^{37}-1} &= \alpha^{2(2^9-1)(2^9+1)(2^{18}+1)+1}, \\
\alpha^{2^9-1} &= \alpha^{2(2+1)(2^2+1)(2^4+1)+1}.
\end{aligned}$$

위의 알고리즘을 하드웨어에 직접 적용하려면 위의 등식들을 다음의 몇 단계로 나누어야 한다.

단계 1. $\gamma = \alpha^{2(2^{37}+1)(2^{74}+1)(2^{148}+1)(2^{296}+1)}$ 를 계산한다.

어떤 수의 제곱은 한번의 회전에 해당되므로 γ는 4번의 곱셈과 556번의 회전을 필요로 한다.

단계 2. γ를 저장한다. $\beta = \gamma^{2(2^9+1)(2^{18}+1)}$.

β는 2번의 곱셈과 28번의 회전을 요구한다.

단계 3. β를 저장한다. $\delta = \beta^{2(2+1)(2^2+1)(2^4+1)}$.

δ는 3번의 곱셈과 8번의 회전을 필요로 한다.

단계 4. $\delta\beta\gamma$를 계산한다. 이 과정에서 2번의 곱셈을 더 필요로 하게된다.

결과적으로 α^{-1}를 구하기 위해서는 11번의 곱셈과 2개의 중간 값을 저장하기 위한 2개의 기억장소(593 비트)를 필요로 하게 된다. 만약 최적 정규기저(Optimal Normal Base)를 이용하면 1번의 곱셈은 m번의 회전으로 수행 가능하게 된다[2]. 그리하여 $GF(2^m)$에서 위의 알고리즘으로 역원을 구하기 위해서는 전체적으로 7115번의 회전을 필요로 하게된다. 참고로 $nb(592)=10$이며 $\omega(592)=3$이다.

4. 개선된 역원계산 방법

이제 Itoh의 방법을 개량한 본 발명의 알고리즘을 소개해 보고자 한다. 이 방법의 핵심적인 원리는 m이 짝수일 때는 Itoh의 방법을 그대로 이용하지만, m이 홀수일 때는 다시 한번 더 그 숫자가 3의 배수가 되는지를 알아본 다음 3의 배수가 되면 아래에서 소개할 방법을 적용하고 3의 배수가 아니면 Itoh의 방법을 그대로 적용하는 것이 그 대략의 방법이다.

$$2^{m-1}-1=\begin{cases}(2^{(m-1)/2}-1)(2^{(m-1)/2}+1), & m=\text{짝수},\\ (2^{(m-1)/3}-1)(2^{2(m-1)/3}+2^{(m-1)/3}+1), & m=3\text{의 배수인 홀수},\\ 2(2^{(m-2)/2}-1)(2^{(m-2)/2}+1)+1, & m=3\text{의 배수가 아닌 홀수}.\end{cases}$$

여기서 우리가 주목해야 할 것은 m이 3의 배수인 홀수일 때와 m이 3의 배수가 아닌 홀수일 때의 곱셈회수가 2번으로서 같지만 m이 4보다 큰 경우에는 $(m-1)/3 < (m-2)/2$이 항상 성립되므로, 이 알고리즘의 반복회수와 필요한 곱셈의 회수는 Itoh가 제안한 알고리즘보다 작거나 같아 질 수밖에 없다. 뿐만 아니라 중간에 임시로 값을 저장해야 할 메모리의 개수도 줄어들어 여러 가지면 에서 이익을 볼 수 있다.

Itoh의 방법을 사용하면 지수의 평균 반복 인수분해 회수는 m-1의 전체 비트수에 대하여 $[\log_2(m-1)]$이다. 또한 각 i 번째 비트에서 그 비트가 0일 확률 Pr(0 | i)은 1/2이며 필요한 곱셈회수는 2회가 되고 그 비트가 1일 확률 Pr(1 | i)도 1/2이며 필요한 곱셈회수는 1회가 되어서 평균 곱셈회수는 (Pr(0 | i) ×2 + Pr(1 | i) ×1) ×$[\log_2(m-1)]$ = 3/2 ×$[\log_2(m-1)]$이다. 한편 본 논문의 개선된 방법을 이용하면 m-1에 대한 지수부의 평균 반복 인수분해 회수는 (Pr(6k, 6k+1, 6k+2, 6k+4, 6k+5)$[\log_2(m-1)]$ + (Pr(6k+3))$[\log_3(m-1)]$이 됨을 알 수 있다. 여기서 Pr(6k, 6k+1, 6k+2, 6k+4, 6k+5)은 5/6이고 Pr(6k+3)은 1/6이다. 또한 각 i번째 단계에서 그 숫자가 2의 배수가 될 확률 Pr(6k, 6k+2, 6k+4 | i)은 3/6이며 필요한 곱셈의 회수는 1번이 되고 그 숫자가 3의 배수인 홀수일 확률 Pr(6k+3 | i)은 1/6이며 필요한 곱셈의 회수는 2번이 되고 다시 그 숫자가 3의 배수가 아닌 홀수일 확률 Pr(6k+1, 6k+5 | i)은 2/6이며 필요한 곱셈회수는 2번이 되어서 평균 곱셈회수는 (Pr(6k, 6k+2, 6k+4 | i) ×1 + Pr(6k+3 | i) ×2 + Pr(6k+1, 6k+5 | i) ×2)(5/6$[\log_2(m-1)]$ + 1/6$[\log_3(m-1)]$) = 3/2 ×(5$[\log_2(m-1)]$ + $[\log_3(m-1)]$)/6 가 된다. 따라서 본 제안방법의 평균 곱셈회수는 Itoh의 방법에 비하여 적게 요구됨을 알 수 있어 적은 계산능력을 갖는 환경에서는 보다 효율적으로 처리가 가능하다. 다시 말하면 3/2 ×$[\log_2(m-1)]$ ≥ 3/2 ×(5$[\log_2(m-1)]$ + $[\log_3(m-1)]$)/6이다. 여기서 알 수 있는 사실은 두기 법의 평균곱셈 회수의 차이는 ($[\log_2(m-1)]$ - $[\log_3(m-1)]$)/4이 된다.

$GF(2^{112})$의 예를 들어 생각해보자.

$$\begin{aligned}\alpha^{-1} &= \alpha^{2^{112}-2}\\ &= \alpha^{2(2^{111}-1)},\\ \alpha^{2^{111}-1} &= \alpha^{(2^{37}-1)(2^{74}+2^{37}+1)},\\ \alpha^{2^{37}-1} &= \alpha^{2(2^{36}-1)+1},\\ \alpha^{2^{36}-1} &= \alpha^{(2^{18}-1)(2^{18}+1)},\\ \alpha^{2^{18}-1} &= \alpha^{(2^{9}-1)(2^{9}+1)},\\ \alpha^{2^{9}-1} &= \alpha^{(2^{3}-1)(2^{6}+2^{3}+1)},\\ \alpha^{2^{3}-1} &= \alpha^{(2^{2}+2^{1}+1)}.\end{aligned}$$

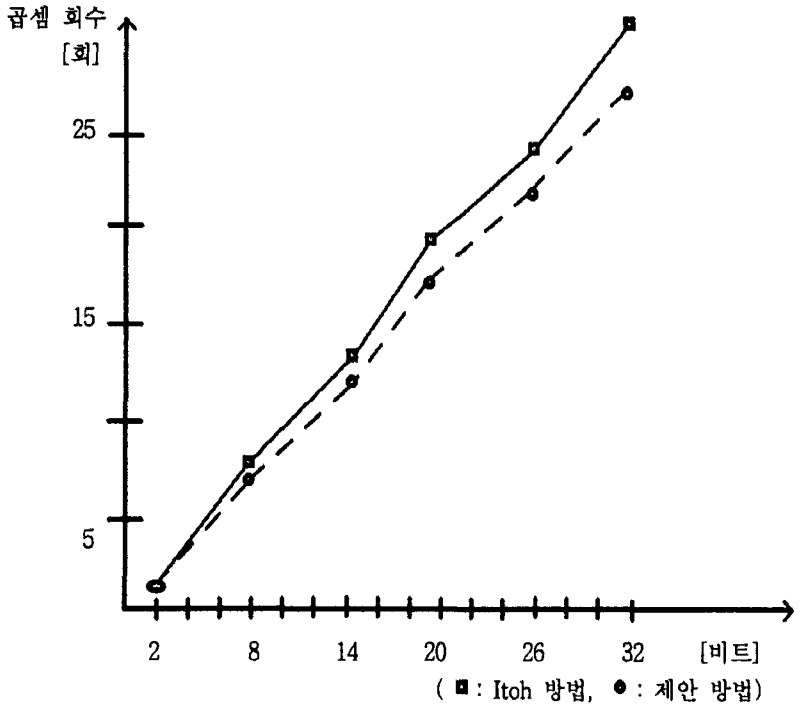
위의 예에서 보는 것처럼 곱셈의 회수는 9번이며 중간의 값을 저장하기 위한 메모리의 개수는 4

번째 등식에서 1개만 필요하다. 하지만 Itoh의 방법을 이용하면 $nb(111)+\omega(111)-2=11$번의 곱셈이 요구되며 중간 메모리는 5개가 필요하므로 그 차이를 명확히 알 수 있다. 이와 같이 m-1이 3의 배수일 때에는 본 제안 방법이 훨씬 유리하다는 것을 알 수 있으며, 뿐만아니라 m-1이 큰 수일수록 본 제안 알고리즘은 더욱 더 유용하다.

위에서는 특정한 m에 대해서만 본 제안 방법이 보다 효율적이라고 했지만 실제로 적당한 크기의 여러 숫자들에 대해 검증해본 결과로는 위에서 이론적으로 예측한 결과와 일치함이 (그림 1)에서 잘 보여지고 있다.

* 단위: 회(소요 곱셈 평균 회수)

methods / m (bits)	제안 방법	Itoh	Difference (Itoh-제안 방법)
2	1.000	1.000	0.000
4	2.145	2.225	0.080
6	3.690	3.985	0.295
8	5.445	5.930	0.485
10	7.150	7.885	0.735
12	9.010	9.910	0.900
14	10.780	11.955	1.175
16	12.505	13.975	1.470
18	14.430	16.090	1.660
20	16.050	18.005	1.955
22	17.850	19.990	2.140
24	19.850	22.100	2.250
26	21.225	23.880	2.655
28	23.395	26.180	2.785
30	25.075	28.075	3.000
32	26.665	29.760	3.095



(그림 1) GF(2^m)상에서의 역원계산시 요구되는 평균 곱셈연산 회수

여기서 선택된 정수의 개수는 m의 크기에 따라 다르지만 대략 큰 수에 대해서는 200개정도로 잡았다. 위의 단위는 필요한 곱셈 회수의 평균값이다. m의 크기가 32비트인 경우 곱셈 회수에 있어서 대략 3회 정도의 차이가 있다. 이 값은 위에서 예측한 수식 ($[\log_2(m-1)] - [\log_3(m-1)]$)/4에 $m= 2^{32}$에서 $m= 2^{32} -1$까지의 임의의 숫자를 대입한 결과와 일치함을 알 수 있다. 예를 들어 m이 10비트 정도의 정수라고 하면 Itoh의 방법과 본 제안방법과의 차이는 대략 0.735회 정도의 곱셈회수에서 차이를 보이나 이것을 연산수행시 필요한 클럭 수의 차이는 적어도 $0.735 \times 2^{10} = 752$ 클럭이 된다[2].

5. 결론

본 논문에서는 유한체 $GF(2^m)$상에서의 역원계산을 위한 빠른 알고리즘을 제안하였다. 여기서 제안된 알고리즘은 기존 Itoh의 알고리즘 보다 계산속도 및 연산수행시 필요한 중간 메모리의 수에 있어서 효율적임을 보일 수 있었다. m의 크기가 증가할수록 그 효율성의 차이는 점점 증가하여 향후 보다 큰 비트의 수를 요구할 때 더욱 유리할 것이다.

본 논문에서 자세히 언급하지는 않았지만 유한체위에서의 최적정규기저의 존재성은 곱셈에 대한 연산수행시 필요한 클럭으로의 결과로 바꿀 때 계산속도면에 있어서 적지 않은 도움을 준다. 최적정규기저에 대한 최근의 연구결과들이 나오긴 했지만 아직 구체적인 최적정규기저를 찾을 수 있는 알고리즘은 없다. 향후 여기에 대한 계속적인 연구가 요구된다.

참 고 문 헌

[1] G. B. Agnew, T. Beth, R. C. Mullin and S. A. Vanstone, "Arithmetic Operations in $GF(2^m)$", J. Cryptology, 6(2), pp. 3-7, 1993.

[2] G. B. Agnew, R. C. Mullin and S. A. Vanstone, "An implemention for a first public key crytosystem", J. Cryptology, 3(2), pp. 63-79, 1990.

[3] T. Itoh, O. Teechai, and S. Tsujii, "A fast algorithm for computing mulplicative inverses in $GF(2^k)$ using normal bases", J. Soc. Electron. Comm. (Japan), 44(1986), pp. 31-36.

[4] Serge Lang, Algebra, Addison Wesley, pp. 344-345, 1984.

[5] H. W. Lenstra, Jr., and R. J. Schoof, "Primitive normal bases for finite fields", Math. Comp., 48, pp. 217-232, 1987.