메모리를 적게 사용하는 모듈라 곱셈 알고리즘들의 비교

임승환*, 박근수
{shlim, kpark}@theory.snu.ac.kr
서울대학교 컴퓨터공학과

Comparison of Modular Multiplication Algorithms that Use Small Memory

Seung-Hwan Lim, Kunsoo Park
Department of Computer Engineering, Seoul National University

요 약

소인수 분해 문제 혹은 이산대수 문제의 어려움에 근거한 공개키 암호시스템에서는 큰 수에 대한 모듈러 멱승연산이 전체 시스템의 속도를 좌우하는 큰 요인이 된다. 모듈라 멱승 연산은 모듈라 곱셈으로 이루어진 연산이므로 모듈라 곱셈의 횟수를 줄이거나 빠른 모듈라 곱셈을 이용하면 모듈라 멱승 연산의 계산 속도가 향상된다 모듈라 곱셈 방법 중에서도 메모리를 적게 사용하면서도 고속인 방법들을 서로 비교하여 본다.

1. 서 론

소인수분해 문제 혹은 이산대수 문제의 어려움에 근거한 공개키 암호시스템에서는 큰 수에 대한 모듈라 멱승 연산이 필요하다 RSA[11]나 ElGamal 시스템[3]에서는 512비트 이상, 이산대수 문제에 근거한 전자서명 시스템[12]에서는 140비트 이상을 사용하며, 이렇게 큰 수의 모듈라 멱승 계산은 시간이 많이 소요되어 이를 줄이기 위한 연구가 진행되어 오고 있다

모듈라 멱승은 모듈라 곱셈으로 이루어져 있으므로 모듈라 곱셈의 횟수가 적은 멱승 방식을 개발하거나, 모듈라 곱셈의 속도를 향상시키는 방법으로 모듈라 멱승의 성능을 개선할 수 있다 모듈라 곱셈의 횟수가 적은 멱승 방식을 개발하는 것은 연결 사슬 방법과 같은 것이 있으며, 모듈라 곱셈 방식에는 고전적인 방법, Barrett 알고리즘[1], Montgomery 알고리즘[9] 등이 있으며, 일반적인 모듈러스에 대해서는 Montgomery 알고리즘이 효율적이라고 알려져 있다 또한, 모듈라 곱셈의 속도를 향상시키기 위해 시스템의 안전도를 해치지 않는 범위 내에서 모듈러스를 특정하게 선택하는 방법이 있다 DR(diminished radix)형태의 모듈러스를 이용하는 방법[8]이 이런 방법의 예이다 그리고, 하재철, 문상재가 제안한 Montgomery 알고리즘과 DR모듈러스를 이용하는 방법 두가지에 공통피승수 방법을 적용한 곱셈방법[4, 5]이 있다.

모듈러 곱셈은 두 가지의 방법이 가능한데, 첫번째는 두 정수를 곱한 후 모듈라 감소를 수행하는 방법이고 두번째는 두 정수를 곱하면서 모듈라 감소를 동시에 수행하는 방법이다 일반적으로 첫번째의 방법이 더 속도가 좋다는 것이 알려져 있다 하지만 두 번째 방법은 계산도중에 필요한 메모리의 양이 적은 특징이 있으므로 IC Card와 같이 메모리가 제한된 환경에서는 두번째의 방법이 적당하다.

일반적인 Smart Card의 경우 RAM이 512byte내외이며, 그 밖에 약간의 EEPROM이 있을 뿐이다. 따라서, 곱셈이나 멱승과정에서 추가로 사용할 수 있는 메모리의 양이 아주 적다. 이 논문에서는 제한된 메모리를 가진 환경에 적합한 모듈라 곱셈 알고리즘들에 대해 살펴보고 각 알고리즘들을 서로 비교해 보려고 한다. 먼저 2장에서는 각 알고리즘들에 대해 간단히 설명하고, 3장에서는 각 알고리즘에 대한 비교를 해본 후, 4장에서 결론을 내린다.

2. 모듈라 곱셈 알고리즘

모듈라 곱셈은 $A \times B \bmod N$ 단, A, B, N은 밑이 b인 n-digit 정수인 식을 계산하는 것이다. 여기서 A, B, N은 다음과 같이 나타낼 수 있다

$$A=\sum_{i=0}^{n-1} A_i b_i,\ B=\sum_{i=0}^{n-1} B_i b_i,\ N=\sum_{i=0}^{n-1} N_i b_i$$

이고, 여기서 A_i, B_i, N_i는 모든 i에 대해서{0, 1, , b-1}의 원소이다. 모듈라 곱셈을 위해서는 고전적인 방법, Barret의 알고리즘, Montgomery 알고리즘등이 사용된다 여기서는 고전적인 방법, Montgomery 방법, 그리고 DR모듈라를 이용한 방법 각각에 공통피승수 방법을 적용한 알고리즘[4, 5, 10]에 대해서 개략적으로 살펴보고 3절에서는 각각을 비교해 본다 이 논문에서는 고전적인 방법, Montgomery 방법, 그리고 DR모듈라를 이용한 방법 각각에 공통피승수 방법을 적용한 알고리즘들에 대해 중점적으로 살펴보겠다

2.1 Classical Algorithm

이 방법은 부분곱을 구한 뒤, 몫을 추정하여, 몫의 N배수를 부분곱에서 빼주는 것이 기본 아이디어이다 이 알고리즘은 전산과정 중에 추가로 필요한 메모리의 공간은 n+2 digit의 정수 하나이므로 필요한 메모리의 양이 적다 몫을 추정하는 방법에는 D. Knuth가 제안한 $\lfloor \frac{C_{n+1}\|C_n}{N_{n-1}} \rfloor$을 이용하는 방법이 있으며 이를 약간씩 변형한 것들이 존재한다

2.1.1 Morita-Yang 알고리즘

이 알고리즘은 고전적인 방법을 기본으로 하여 약간 다른 몫추정 방식을 사용하고, Lookahead determination기법을 추가로 사용을 하여 성능을 개선하였다 여기서 사용한 몫추정방식은 $\lfloor \frac{C_{r+1}\|C_r}{N_{n-1}+1} \rfloor$이다 이 방법은 D. Knuth가 제안한 방식의 단점인 C에서 추정된 몫의 N배수를 뺄 경우 음수가 나오는 경우를 없애서 2의 보수를 구하는 필요없는 연산을 제거하였다.

이 알고리즘에서 사용한 또다른 기법인 lookhead determination기법은 다음번 스텝에서 몫이 r-1이 되는 경우에는 미리 빼줘서 한번의 스텝을 생략하게 해준다 그림 1은 Morita-Yang의 알고리즘이다

2.2 Montgomery Algorithm

Montgomery 알고리즘은 b^n 을 N보다 크고 N과 서로 소인 정수 R로 잡은 뒤, Z_n을 $RZ_n=\{xR \bmod n\}$으로 변환시켜 RZ_n상에서의 모듈라 곱셈을 수행한 후 그 결과를 다시 Z_n으로 변환시켜서 원하는 결과를 얻는 것이 기본적인 아이디어이다. 이렇게 할 경우 몫을 추정할 필요가 없으므로 나눗셈을 사용하지 않아서 고전적인 알고리즘에 비

Step 1. $C \leftarrow 0$
$I \leftarrow n-1$
Step 2. $C \leftarrow r \times C + A \times B_i$
Step 3. $q \leftarrow \lfloor \frac{C_{n-1} \| C_n}{N_{n-1}+1} \rfloor$
Step 4. $C \leftarrow C - r \times q \times N$
Step 5. $q \leftarrow \lfloor \frac{C_{n+1} \| C_n}{N_{n-1}+1} \rfloor$
Step 6. if q = 1 then $C \leftarrow C - r \times N$
if q = 2 then $C \leftarrow C - r \times 2N$
Step 7. if $C_n \| C_{n-1} \geq (r-1)(N_{n-1}+1)$ then $C \leftarrow C - (r-1) \times N$
Step 8. $i \leftarrow i - 1$
if $i \geq 0$ then goto Step 2
Step 9. If C < N then return
Step 10. $q \leftarrow \lfloor \frac{C_n \| C_{n-1}}{N_{n-1}+1} \rfloor$
Step 11. $C \leftarrow C - q \times N$
Step 12. if C< N then return
Step 13. $C \leftarrow C - N$
Goto Step 12

그림 1 Morita - Yang의 알고리즘

해 속도기 빠르다 Montgomery 곱셈을 하는 방법중 Dusse와 Kaliski가 제안한 개선된 방법[2]은 곱셈과 모듈러감소를 병행적으로 수행할 수 있다 이 방법 역시 계산과정에서 n+2 digit 정수 한 개만 더 필요로 하며, 추가로 필요한 메모리의 양이 적다. 여기서는 Dusse와 Kaliski가 제안한 방법을 약간 수정하여 이진 멱승방법에 Montgomery 곱셈을 이용하였을 경우 공통피승수를 한번만 계산하여 속도를 개선한 방법[5]에 대해 살펴보겠다

2.2.1 Montgomery 알고리즘에 대한 공통피승수 방법

그림 2는 이진 멱승 방법에 Montgomery 곱셈을 적용할 경우의 계산과정이다 Step 2에서 REDC(SC), REDC(SS)는 공통피승수를 가지고 있다 이 공통피승수를 한번만 계산하도록 REDC(AB)를 약간 수정하면 REDC(SC), REDC(SS)를 동시에 계산할 수 있다 이런 방법은 이진 멱승 방법 뿐만아니라 공통피승수가 생기는 모든 멱승방법에 적용할 수 있다 그림 3은 Montgomery 알고리즘에 대해 공통피승수 방법을 적용한 알고리즘이다

Step 1 $S = AR \bmod N$, $C = R \bmod N$
Step 2. for I = 0 to k-1 do {
if(e_i = 1) then
{
C = REDC(SC), S = REDC(SS)
}
else S = REDC(SS)
}
Step 3. C = REDC(C)
Step 4. return (C)

그림 2 Montgomery 곱셈을 이진 멱승법에 적용했을 때

2.3 DR Modular Algorithm

Mohan과 Adiga에 의해 처음 제안된 방법으로 먼저 다음 식을 만족하도록 모듈러스 N을 선택한다.

$$N = b^n - N' (N' < b^{n-\delta})$$

이러한 N을 DR형태의 모듈러스라 하며 이런 형태의 N은 상위 δ bit가 모두 1 이다 DR 모듈라 곱셈은 이와 같은 N일 경우 $b^n \equiv N' \bmod N$ 이 성립한다는 것에 기초하여 고들라 감소를 쉽게 할 수 있다 DR 방법을 사용하여 2n자리인 C에 대해 C mod N을 수행하는 과정은 그림4 와 같다

Mohan과 Adiga가 선택한 N' (N' < $b^{n/2}$)는 후에 RSA System에 적용할 경우 시스템의 안전성에 영향을 줄 수 있다는 것이 알려졌고[7], 그 후에 δ를 2~4로 작게 하면 안전하다는 연구결과[6]가 있다 DR방법은 n의 상위 δ자리가 모두 '1'일 경우 한번의 모듈라 곱셈을 수행하는 데 $2n^2 - \delta n$의 단정도 곱셈이 필요하므로 Montgomery 곱셈에 비해 고속이다

Step 1 X = 0 Y = 0
Step 2 T = S
for I = n - 3 to 0 do {
m = $T_0 N'_0$ mod b
T = (T+mN)/b
X = X + TC_i Y = Y + TS_i
for I = 0 to 1 do {
X = X + $C_{n-2+i}S$ Y = Y + $S_{n-2+i}S$
m_X = $Y_0 N'_0$ mod b m_Y = $Y_0 N'_0$ mod b
X = (X + m_XN)/b Y = (Y + m_YN)/b
Step 3 while (X≥N) X = X - N while(Y≥N) Y = Y - N
Step 4 return (X, Y)

그림 3 Montgomery 곱셈에 공통피승수 방법을 적용한 알고리즘

2.3.1 DR 모듈라 곱셈 알고리즘에 대한 공통피승수 방법

DR 모듈라 곱셈방법을 멱승에 이용할 때에 이진 멱승방법처럼 공통피승수가 발생하는 경우에 좀 더 효율적인 알고리즘[4]이 제안되었는데, 이 방법은 공통 피승수를 저장하기 위해 n-2 digit의 정수 하나가 더 필요하지만, 공통 피승수가 발생하는 멱승 방법인 경우 공통피승수를 한번만 계산하기 때문에 효율적인 멱승을 수행할 수 있다 DR 형태의 N을 사용할 경우의 공통 피승수 모듈라 곱셈은 그림 4와 같다 그림 4에서 $A[n-1 : n-2]$는

$$A[n-1\ n-2] = A_{n-1} \times b + A_{n-2}$$

를 뜻한다

X = S
T' = M_0X T'' = S_0X
for I=1 to n-1
X = Xb
X[n-1:0] = X[n-1:0] + X_nN'
if (final carry)
X[n-1:0]=X[n-1:0] + N'
T''=T''+S_iX
T'=T'+M_iX
for I=0 to 1
T'[n-1:0] = T'[n-1:0]+T'$_{n-1+i}$N' T''[n-1:0]=T''[n-1:0]+T''$_{n-1+i}$N'
if (final carry) if (final carry)
T'[n-1:0]=T'[n-1:0]+N' T''[n-1:0]=T''[n-1:0]+n'
if (T'≥N) T' = T' - N if(T''≥N) T'' = T'' -N
return (T', T'')

그림 4 공통피승수 곱셈방법을 적용한 DR 모듈라 곱셈 알고리즘

3. 각 알고리즘의 비교

고전적인 방법의 경우에는 2n(n+1)번의 단정도 곱셈과 n번의 나눗셈이 필요하게 된다. 보통 나눗셈은 곱셈보다 훨씬 시간이 많이 걸리는 연산이므로, 지수의 크기가 큰 경우 Montgomery 방법보다 시간이 많이 걸릴 것으로 예상된다 고전적인 방법은 n+2 digit 정수 한 개가 추가로 필요하다

Morita-Yang의 방법은 최대 n(n+1)번의 단정도 곱셈과 2n번의 나눗셈이 필요하다 b배 혹은 b-1배등은 shift연산으로 쉽게 구현이 가능하므로 단정도 곱셈이 필요하지 않다. 또한, Step 6에서는 곱셈을 이용하지 않고, 뺄셈을 사용하였는데, 뺄셈이 수행될 확률은 실험적으로 약50%~25%정도이다. 뺄셈은 곱셈보다 비용이 적게 드는 연산이므로 뺄셈이 수행될수록 속도는 향상된다 또한, Lookahead방법을 사용하였는데, Lookahead방법이 걱용될 조건이 성립할 확률은 매우 낮긴 하지만 만약 해당 조건이 성립한다면 다음번의 몫을 바로 추정할 수 있기 때문에 속도의 개선이 가능하다. 추가로 필요한 메모리의 양은 고전적인 방법과 동일하게 n+2 digit 정수 한 개가 필요하다

Dusse 와 Kaliski가 제안한 Montgomery 곱셈 알고리즘은 $2n^2$ + n번의 단정도 곱셈이 필요하며, N의 모듈라 b에 대한 역원을 구하기 위해 Euclidian Algorithm을 쓸 경우 $\log^2(b)$정도의 시간이 걸린다

Montgomery 알고리즘에 대해 공통 피승수 곱셈방법을 적용한 알고리즘은 (n-2)(n+1)번의 단정도 곱셈으로 공통피승수를 계산하게 되고,이 공통피승수 계산을 포함하여 동시에 REDC(SC), REDC(SS)를 계산하는데에는 각각 약 $2n^2+n$, n^2+2n의 단정도 곱셈이 필요하다 REDC(SS)이 계산에 필요한 단정도 곱셈횟수가 줄어드는 것은 공통피승수를 REDC(SC)에서 미리 계산하여 저장해 놓았기 때문에 만약 k-bit 지수를 이용하여 이진 멱승을 하는데에 Montgomery 곱셈방법을 이용하게 되면 $1.5k(2n^2+n)$번의 단정도 곱셈이 필요하겠지만, 공통피승수 방법을 이용하면 $0.5k(5n^2+4n)$번의 단정도 곱셈이 필요하게 된다 공통피승수 방법을 이용한 몽고메리 곱셈은 공통피승수 계산을 할 때에 중간계산결과와 공통피승수를 저장해야 하므로 동시에 3개의 n+2 digit 정수가 필요하다

또한 DR모듈라를 이용한 방법은 $2n^2-\delta n$의 단정도 곱셈이 필요하며 이에 공통피승수 곱셈방법을 적용한 경우에는 공통피승수 곱셈을 할 경우에는 $3n^2-\delta n+3n$의 단정도 곱셈이 필요하고, 일반 곱셈일 경우에는 $2n^2-\delta n$이 필요하게 된다 따라서 이진 멱승 방식에 적용했을 경우에는 $0.5k(3n^2-\delta n+3n)+0.5k(2n^2-\delta n+n)$ 번의 단정도 곱셈이 필요하다 그리고, DR 모듈라를 이용한 방법에 공통피승수 방법을 적용한 알고리즘의 경우 동시에 3개의 n+2 digit정수가 필요하다

4. 구현결과 및 결론

그림 5는 각 알고리즘에서 계산과정중에 필요한 메모리양이다. 그림 6은 Morita-Yang의 알고리즘과, 몽고메리 곱셈방법, 몽고메리 곱셈방법에 공통피승수 방법을 적용한 알고리즘, DR모듈라 곱셈방법, DR모듈라 곱셈방법에 공통피승수 방법을 적용한 알고리즘을 구현하여 실험한 결과이다 멱승을 1000번 수행하여 그 평균값을 구한 것이다 실험은 SUN Ultra Sparc I을 사용하였다 A^E mod N에서 A, N은 모두 512 bit이고, 지수 E는 512bit, 160 bit두가지를 사용하였다. 단위는 sec이다. DR모듈라 방법을 위해서는 상위 2자리가 1인 수를 N으로 사용하였다

구현결과 공통피승수방법을 적용한 곱셈방법들이 멱승에 적용하였을 때 공통피승수 방법을 적용하지 않은 곱셈방법보다 좋은 성능을 보였다 또한 고전적인 알고리즘을 변형한 방법인 Morita-Yang의 알고리즘도 지수 크기가 160 bit일 경우에는 몽고메리 곱셈방법과 성능이 비슷한 것으로 나타났다.

알고리즘	계산과정중 추가로 필요한 메모리 양
고전적인 알고리즘	n+2 digit 정수 1개
Morita-Yang	n+2 digit 정수 1개
몽고메리 곱셈	n+2 digit 정수 1개
공통피승수 몽고메리곱셈	n+2 digit 정수 3개
DR모듈라 곱셈	n+2 digit 정수 1개
공통피승수 DR모듈라 곱셈	n+2 digit 정수 3개

그림 5 각 알고리즘에서 추가로 필요한 메모리 양

알고리즘	|E|= 512 bit	|E|=160 bit
Morita-Yang	0.612	0.170
몽고메리 곱셈	0.530	0.167
공통피승수 몽고메리곱셈	0.433	0.136
DR 모듈라 곱셈	0.496	0.156
공통피승수 DR모듈라 곱셈	0.412	0.126

그림 6 SUN Ultra Sparc 에서의 모듈라 멱승 속도 (단위 sec)

참고 문헌

[1] Paul Barrett, "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor," *Advances in Cryptology* - CRYPTO '86, pp 311-323, Springer-Verlag, 1986

[2] Dusse, S R and Kaliski Jr, B S, "A cryptographic Library for the Motorola DSP56000," *Advances in Cryptology* - EUROCRYPT'90, pp 230-244, Springer-Verlag, 1991

[3] T ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans Inform Theory*, Vol 31, No. 4, pp.469-472 1985

[4] 하재철 문상재, "공통 피승수 모듈라 곱셈을 이용한 고속 멱승," 정보과학회논문지(C) 제 3권 제 5호, pp491-497, 1997.

[5] Jae-Cheol Ha, Sang-Jae Moon, "A common-multiplicand method to the Motgomery algorithm for speeding up exponentiation," *Information Processing Letters* 66, pp 105-107, 1998.

[6] C H Lim and P J Lee, "Sparse RSA secret keys and their generation," *SAC'96*, pp 117-131, 1996

[7] G Meister, "On an implementation of the Mohan-Adiga Algorithm," *Advances in Cryptology* - EUROCRYPT '90, pp 496-500, Springer-Verlag, 1991

[8] S. B. Mohan and B S Adiga, "Fast algorithms for implementing RSA public key cryptosystem," *Electronics Letters*, Vol 21, NO. 7, p.761, 1985

[9] Peter L. Montgomery, "Modular multiplication without trial division," *Math. of Comp.* Vol. 44, No 170, pp 519-521, 1985

[10] Hikaru Morita and Chung-Huang Yang, "A modular-multiplication algorithm using lookahead determination," *IEICE Trans. Fundamentals* Vol. E76-A No. 1, pp.70-77, 1993.

[11] R. Rivest, A Shamir and L Adleman, "A method for obtaining digital signatures and public key cryptosystems," *Comm. of ACM*, Vol. 21, No. 2, pp.120-126, Feb. 1978.

[12] C. P. Schonorr, "Efficient signature generation for smart cards," *Advances in Cryptology* - CRYPTO '90, pp.239-252, Springer-Verlag, 1990.