(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(51) Int. Cl.[6]
G06F 7/44

(11) 공개번호 특1998-083641
(43) 공개일자 1998년12월05일

(21) 출원번호 특1997-019018
(22) 출원일자 1997년05월16일

(71) 출원인 하재철
대구광역시북구대현동14-32
문상재
대구광역시 북구 북현동 주공아파트 204동 202호
(72) 발명자 하재철
대구광역시 북구 대현동 14-32
문상재
대구광역시 북구 북현동 주공아파트 204동 202호
(74) 대리인 이영필, 권석흠, 윤창일

*심사청구 : 있음*

# (54) 공통 피승수 모듈라 곱셈을 이용한 고속 멱승 방법

## *요약*

모듈라 곱셈시 일반적인 모듈러스(modulus)인 경우에는 몽고메리(Montgomery) 알고리듬이 속도면에서 가장 효과적이지만 모듈러스가 축소기(diminished radix;DR) 형태일 경우에는 DR 방법이 몽고메리 알고리듬보다 고속이다. 본 발명에서는 동일한 피승수를 가지는 두번의 모듈라 곱셈시 공통 계산 부분이 생기도록 함으로써 멱승에 적용할 경우 계산속도를 개선하고 있다. 나눗셈을 수행하지 않기 위하여 모듈러스 N의 상위자리 중 일부, 예를 들면 2-4자리가 모두 1이 되도록 하여, N을 DR 형태의 모듈러스를 사용한다. 또한, 본 발명은 이진 멱승방식, 윈도우 멱승방식, 지수 폴딩 기법을 사용한 멱승 방식에 적용되어 고속이면서 사용 메모리가 적게 되므로 IC카드와 같은 저메모리 환경에 효과적이다.

## *대표도*

*도3a*

## *명세서*

### *도면의 간단한 설명*

도 1a는 공통 피승수 모듈라 곱셈방법을 설명하기 위한 프로그램을 도시한 도면이다.

도 1b는 도 1a의 알고리듬을 플로우챠트형태로 도시한 도면이다.

도 2는 축소기(DR) 모듈러스를 사용한 모듈라 감소 프로그램을 도시한 도면이다.

도 3a는 라이트-투-레프트 이진 멱승방식을 설명하기 위한 프로그램을 도시한 도면이다.

도 3b는 도 3a의 알고리듬을 플로우챠트형태로 도시한 도면이다.

도 4는 라이트-투-레프트 윈도우 멱승방식을 설명하기 위한 프로그램을 도시한 도면이다.

도 5는 지수 폴딩기법을 사용한 멱승방식을 설명하기 위한 프로그램을 도시한 도면이다.

도 6은 본 발명에 의한 고속 멱승방법이 적용되는 하드웨어구성을 개략적으로 도시한 도면이다.

도면의 주요 부분에 대한 부호의 설명

70...PC, 워크스테이션, 또는 IC카드,72...입/출력 장치

74...CPU,76...RAM

78...ROM, 또는 EEPROM

### *발명의 상세한 설명*

#### *발명의 목적*

### 발명이 속하는 기술분야 및 그 분야의 종래기술

본 발명은 공통 피승수 곱셈을 이용한 고속 멱승 방법에 관한 것으로, 보다 상세하게는 두 번의 모듈라 곱셈시 공통계산부분이 생기도록 하여 공통부분을 한번만 계산할 수 있게 함으로써 멱승 계산속도를 개선하고, 메모리용량을 줄일 수 있는 고속 멱승 방법에 관한 것이다.

통상 소인수분해 문제(prime factorization problem)나 이산대수 문제(discrete logarithm problem)에 근거한 공개 키 암호시스템에서는 유한체 GF(p) 혹은 유한환 $Z_N$상에서 큰 수에 대한 모듈라 멱승 연산이 필요하다. 그러나, 멱승 연산은 시간이 많이 소요되므로 공개 키 암호 시스템의 설계시 단점이 될 수 있다. 그러므로 공개 키 암호 시스템에서 멱승 연산 시간을 줄이는 연구가 필요하다.

일반적으로 모듈라 멱승은 $A^E \bmod N$으로 표시할 수 있으며 A와 N은 안전도에 따라 가변적이나 512비트 이상의 큰 정수를 사용한다. 지수 E는 RSA 시스템(A method for obtaining digital signatures and public key crytosystems, Comm. of ACM, Vol. 21, No. 2, pp.120-126, February 1978, by R. Rivest, A. Shamir and L. Adleman)이나 ElGamal 시스템(A public key crytosystem and a signature scheme based on discrete logarithms, IEEE Trans. Inform. Theory, Vol. 31, No. 4, pp.469-472, July 1985, by T. ElGamal)에서는 512비트 이상, 이산 대수 문제에 근거한 디지털 서명 시스템에서는 140비트 이상을 사용한다. 멱승 연산은 모듈라 곱셈의 반복으로 이루어져 있으므로 고속 멱승 연산을 위해서는 모듈라 곱셈 횟수가 적은 멱승 연산방식을 선택하거나 모듈라 곱셈의 고속화가 필요하다. 멱승 연산방식으로는 이진 방식, m-ary 방식, 윈도우방식 등이 대표적이다. 모듈라 곱셈은 곱셈과 모듈라 감소 연산으로 나눌 수 있으며 모듈라 감소 방법으로는 고전적인 알고리듬, 바레트(Barrett )알고리듬 그리고 몽고메리 알고리듬등이 있다.

상술한 알고리듬들은 멱승연산중 동일한 피승수에 대해서 두 번의 모듈라곱셈을 독립적으로 수행하므로, 소요되는 메모리의 양이 많을 뿐만 아니라 멱승연산시간이 길어지는 문제점이 있었다.

### 발명이 이루고자 하는 기술적 과제

본 발명이 이루고자 하는 기술적 과제는 모듈라 곱셈연산시 동일한 피승수에 대해서 공통 계산 부분을 한번만 계산하는 고속 모듈라 멱승 방법을 제공하는 것이다.

### 발명의 구성 및 작용

본 발명은 상기 기술적 과제를 달성하기 위하여, 주어진 수 A, E, N에 대해서, 수 $C=A^E \bmod N$의 모듈라 멱승방법에 있어서, (a) 모듈라곱셈을 $C=SxM \bmod N$, $S=SxS \bmod N$이라 할 때, 상기 모듈라곱셈을 기수(radix) b에 대해서 다음 식으로 변형하는 단계,

$S \times M \bmod N = S(M_0 b^0 + M_1 b^1 + \cdots + M_{n-1} b^{n-1}) \bmod N$

$= (S_{b0} \cdot M_0 + S_{b1} \cdot M_1 + \cdots S_{bn-1} \cdot M_{n-1}) \bmod N$

$= T_m \bmod N$

$S \times S \bmod N = S(S_0 b^0 + S_1 b^1 + \cdots + S_{n-1} b^{n-1}) \bmod N$

$= (S_{b0} \cdot S_0 + S_{b1} \cdot S_1 + \cdots + S_{bn-1} \cdot S_{n-1}) \bmod N$

$= T_s \bmod N$ (여기서, $S_{bi}=Sb^i \bmod N$ $(0 \le i \le n-1)$이며 $T_m$은 $S_{b0} \cdot M_0 + S_{b1} \cdot M_1 + \cdots S_{bn-1} \cdot M_{n-1}$이고, $T_s = S_{b0} \cdot S_0 + S_{b1} \cdot S_1 + \cdots + S_{bn-1} \cdot S_{n-1}$이다.);

(b) 멱승 방법에 따라서 소정 횟수만큼 $S=SxS \bmod N$를 독립적으로 수행하거나, $C=SxM \bmod N$, $S=SxS \bmod N$를 동시에 수행하여 주어진 수 A에 대한 모듈라 멱승을 수행하는 단계를 포함하는 모듈라 멱승방법을 제공한다.

바람직하기로는 상기 모듈러스 N은 $N=b^n-N'(N'b^{n-\delta})$ (여기서, N은 상위자리 중 δ자리가 모두 1인 수)를 만족하며, $b^n=N' \bmod N$에 기초하여 모듈라감소를 수행함을 특징으로 한다.

바람직하기로는 이진 멱승방법인 경우에는 지수 E를 $E=\sum_{i=0}^{t-1} e_i 2^i$, $e_i \in \{0,1\}$로 표현할 때, $e_i=1$인 경우 동일한 S에 대해서 $C=SxM \bmod N$, $S=SxS \bmod N$의 공통피승수곱셈연산을 수행하고, 이외의 경우에는 $S=SxS \bmod N$의 곱셈연산을 수행하는 것을 특징으로 한다.

바람직하기로는 m진 멱승방법인 경우에는 지수 E를 t개의 자리수를 갖는 m진수 $E=\sum_{i=0}^{t-1} e_i m^i$, $e_i \in \{0,\ldots,m-1\}$로 표현할 때, $e_i=0$이 아닌 경우 동일한 S에 대해서 $C=SxM \bmod N$, $S=SxS \bmod N$의 공통피승수곱셈연산을 수행하고, 이외의 경우에는 $S=SxS \bmod N$곱셈연산을 수행하는 것을 특징으로 한다.

바람직하기로는 윈도우 멱승방식인 경우에는 지수 E를 $E=\sum_{i=0}^{t-1} e_i 2^i$, $e_i \in \{0,1\}$로 표현하여, w크기의

윈도우를 정하여 연산할 때 윈도우가 하나 발생할 때마다, 동일한 S에 대해서 C=SxM mod N, S=SxS mod N의 공통피승수곱셈연산을 수행하고, 이외의 경우에는 S=SxS mod N곱셈연산을 수행하는 것을 특징으로 한다.

바람직하기로는 지수폴딩 멱승방법인 경우에는 지수 E를 $E=\sum_{i=0}^{t-1} e_i 2^i$, $e_i \in \{0,1\}$로 표현하여, $E_H$은 상위반절, $E_L$은 하위반절이라 하면 $E=E_H 2^{t/2}+E_L$이므로 $E_H$과와 $E_L$이 공통적으로 1인 $E_{com}=E_H$ AND $E_L$, 상위반절만 1인 $E_{HXOR}=E_{com}$ XOR $E_H$, 하위반절만 1인 $E_{LXOR}=E_{com}$ XOR $E_L$를 얻는 과정; $E_{com}$, $E_{HXOR}$, $E_{LXOR}$이 각각 1인 경우, 동일한 S에 대해서 C=SxM mod N, S=SxS mod N의 공통피승수곱셈연산을 수행하고, 이외의 경우에는 S=SxS mod N곱셈연산을 수행하여, $A^{E_{com}}$ mod N, $A^{E_{HXOR}}$ mod N 그리고 $A^{E_{LXOR}}$ mod N을 먼저 구하는 과정; $A^{E_H}=A^{E_{com}}\cdot A^{E_{HXOR}}$ mod N과 $A^{E_L}=A^{E_{com}}\cdot A^{E_{LXOR}}$ mod N을 계산하는 과정; 및 최종적으로 $A^E=(A^{E_H})^{2^{t/2}}\cdot A^{E_L}$ mod N을 계산하는 과정;을 포함하는 것을 특징으로 한다.

더욱 바람직하기로는 상기 멱승 방법에서는 지수를 하위비트에서 상위비트쪽으로 탐색하는 라이트-투-레프트 방식을 채용함을 특징으로 한다.

더욱 바람직하기로는 고정된 값들은 프로그램내부에 두어 ROM이나 EEPROM에 저장하고, 변하는 값들은 프로그램외부로부터 읽어와서 RAM에 저장한 후 프로그램 동작시 불러와서 사용함을 특징으로 한다.

이하, 본 발명에 의한 공통 피승수곱셈을 이용한 고속 멱승방법의 바람직한 실시예들에 대해서 첨부된 도면을 참조하여 보다 상세히 설명하기로 한다.

통상 모듈라 곱셈은 두 수의 곱셈과 그 결과에 대한 모듈라 감소로 이루어지는 데 이 두 과정을 순차적으로 수행할 수도 있고 곱셈을 하면서 모듈라 감소를 수행할 수도 있다. 모듈라 감소 방법 중 일반적인 모듈러스인 경우에 대해서는 몽고메리 알고리듬이 속도면에서 가장 효과적인 것으로 알려져 있다. 그러나 모듈러스의 상위 자리중 일부가 모두 1인 축소기(DR) 형태의 모듈러스인 경우에는 몽고메리 알고리듬보다 고속으로 모듈라 곱셈을 수행하는 방법도 있다 (이하에서는 이를 DR 모듈라 곱셈 방법이라 한다.).

이하, 기존의 모듈라 곱셈 방법을 고찰하고 새로운 공통 피승수 모듈라 곱셈 방법을 제안한다.

기존의 고전적인 모듈라 곱셈 방법에서는 임의의 큰수 X를 기수(radix) b로 표현하면 $X=\sum_{i=0}^{n-1} X_i b^i$ 와 같이 n자리로 나타낼 수 있다. 여기서 $X_i \in \{0,1,\cdots,b-1\}$이다. n자리인 두 수 A와 B의 모듈라 곱셈 A×B mod N은 A와 B를 곱하는 곱셈과 이 결과에 대해 모듈라 감소 연산으로 이루어진다. n자리인 두 수의 곱셈은 n2번의 기수 b보다 작은 두 수의 곱셈으로 수행될 수 있다 (이하에서는 기수 b보다 작은 두 수의 곱셈을 작은 수 곱셈이라 한다.). 모듈라 감소는 A와 B를 곱한 결과 C에 대해 C-QN을 구하는 것이다. 여기서 Q는 C를 N으로 나눈 몫이다. 고전적인 모듈라 감소에서는 몫 Q를 구하는 것이 어려우므로 몫 추정 기법을 사용한다. 그러나 몫 추정시 나눗셈이 필요한데 이 연산은 덧셈이나 곱셈에 비해 구현이 복잡하고 계산 속도를 저하시키는 요인이 된다. 몫추정 기법을 사용하여 고전적인 모듈라 곱셈을 수행하는 데는 $2n^2+2.5n$번의 작은 수 곱셈과 n번의 나눗셈이 필요하다.

한편, 기존의 몽고메리 알고리듬에서는 먼저 N보다 크고 N과 서로 소인 R을 선택하는데 일반적으로 R에 대한 모듈라 연산(mod R)이나 나눗셈(div R)을 자리수 이동으로 간단히 계산할 수 있도록 $b^n$으로 한다. 이 알고리듬은 나눗셈을 하는 고전적인 방법과는 달리 임의의 정수 X의 R에 대한 N-잉여류 수를 XR mod N으로 정의하고 이 집합 내에서 빠른 모듈라 감소를 수행한다. 몽고메리 모듈라 감소 알고리듬은 N-잉여류수의 집합상에서 $0 \le T \le RN$인 임의의 정수 T에 대해 $TR^{-1}$ mod N을 계산하는 것과 같다. 여기서 T는 N-잉여류수 상의 A와 B를 곱한 결과로 볼 수 있다. 이 알고리듬의 핵심은 T에 대해 N의 일정한 배수를 더하여 하위 n자리가 0이 되도록 하는 것이다. 하위 n자리가 0인 수를 R로 나누면 $TR^{-1}$ mod N가 된다.

한편, 듀스(Dusse)와 칼리스키(Kaliski)는 몽고메리 알고리듬을 개선하여 N-잉여류수 A와 B를 곱하면서 모듈라 감소를 수행하는 모듈라 곱셈 방법을 제안하였다 (이하에서는 이를 몽고메리 곱셈이라 한다.). 몽고메리 곱셈은 $2n^2+n$번의 작은 수 곱셈이 필요하며 고전적인 방법이나 바레트(Barrett) 알고리듬에 비해 고속이면서 구현이 용이하다.

본 발명에 적용되는 새로운 공통 피승수 모듈라 곱셈 방법은 동일한 피승수에 대해서 공통적인 부분을 추출하여 한번만 계산하도록 하는 것이다. 다시 말하면, 멱승연산은 모듈라 곱셈의 반복으로 구현할 수 있는데 라이트-투-레프트 형태의 멱승일 경우에는 C=S×M mod N과 S=S×S mod N과 같은 연산을 연속적으로 수행하는 경우가 있다. 이때 동일한 S가 공통적으로 곱해지므로 이 연산을 공통 피승수 모듈라 곱셈(common-multiplicand modular multiplication)이라 한다. 본 발명에서 제안하고자 하는 모듈라 곱셈의 기본 개념은 이 두 연산에서 공통적인 계산 부분을 추출하여 한번만 계산한다는 것이다. 이를 위해 모듈라 곱셈 S×M mod N 및 SxS mod N을 다음과 같이 표현한다.

[수학식 1]

$$S\times M \bmod N = S(M_0 b^0+M_1 b^1+\cdots+M_{n-1}b^{n-1}) \bmod N$$

$$= (Sb^0 M_0 \bmod N+Sb^1 M_1 \bmod N+\cdots+Sb^{n-1}M_{n-1} \bmod N) \bmod N$$

$$= (S_{b0}\cdot M_0+S_{b1}\cdot M_1+\cdots S_{bn-1}\cdot M_{n-1}) \bmod N$$

$= T_m \bmod N$

여기서 $S_{bi}=Sb^{i} \bmod N$ $(0\le i\le n-1)$이며 $T_m$은 $S_{b0}\cdot M_0+S_{b1}\cdot M_1+\cdots S_{bn-1}\cdot M_{n-1}$

이다. N이 k 비트인 경우 $T_m$값은 최대 k+b+[$\log_2 n$] 비트이고 이는 n+2 자리에 해당한다. 식 (1)과 같은 방법으로 S×S mod N을 나타내면 다음의 식(2)와 같다.

[수학식 2]

$S\times S \bmod N = S(S_0b^{0}+S_1b^{1}+\cdots+S_{n-1}b^{n-1}) \bmod N$

$= (Sb^{0}S_0 \bmod N+Sb^{1}S_1 \bmod N+\cdots+Sb^{n-1}S_{n-1} \bmod N) \bmod N$

$= (S_{b0}\cdot S_0+S_{b1}\cdot S_1+\cdots+S_{bn-1}\cdot S_{n-1}) \bmod N$

$= T_s \bmod N$

위의 식(1)과 (2)에서 $S_{bi}(0\le i\le n-1)$가 공통적으로 사용되므로 한번만 계산할 수 있다. 또한 $S_{b0}$는 S와 동일하며 나머지 $S_{bi}$는 다음과 같이 $S_{bi-1}$를 한자리 왼쪽으로 이동하여(shift) 모듈라 감소를 수행한 것과 같으며, 다음의 식 (3)으로 표시된다.

[수학식 3]

$S_{b1} = S_{b0}b \bmod N,\ S_{b2} = S_{b1}b \bmod N,\ \cdots,\ S_{bn-1} = S_{bn-2}b \bmod N$

결과적으로, 각 $S_{bi}$를 구할 때 고전적인 방법을 사용하면 n번의 작은 수 곱셈과 1번의 나눗셈이 필요하다. 그러나 N이 상위 몇 자리가 모두 1인 DR 형태이면 n+1자리인 수에 대한 모듈라 감소는 간단히 수행할 수 있다. 이 DR모듈라 곱셈방법은 모한(Mohan)과 애디거(Adiga)에 의해 처음 제안된 방법으로 먼저 다음 식 (4)을 만족하도록 모듈러스 N을 선택한다.

[수학식 4]

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

이러한 N을 DR 형태의 모듈러스라 하며 N은 상위자리 중 δ자리가 모두 1인 수이다. 이와 같은 N일 경우에는 합동식 $b^{n}=N' \bmod N$이 성립한다는 것에 기초하여 쉽게 모듈라 감소를 할 수 있다. DR 방법을 사용하여 2n자리인 C에 대해 C mod N을 수행하는 과정을 나타낸 것이 도 2이다. 여기서 C[i:j]는 $C_ib^{i-j}+\cdots C_j$를 나타낸다. 이로써, 나눗셈이 필요없이 모듈라 곱셈을 수행할 수 있다.

모한과 애디거는 $N'b^{n/2}$이고 $N=b^{n}-N'$인 모듈러스 N을 선택하였으나 마이스터(Meister)는 RSA 시스템의 경우 위의 조건을 만족하는 모듈러스를 선택하는 것은 시스템의 안전성에 영향을 줄 수 있음을 보였다. 논문(C.H.Lim and P.J.Lee, Sparse RSA secret keys and their generation, SAC'96, pp.117-131, 1996)에서는 이를 개선하여 δ를 2~4로 작게 하여 안전도를 유지하면서 계산 효율을 높일 수 있도록 하였다. RSA나 DSS 등 대부분의 공개 키 암호 시스템에서는 작은 δ를 가지는 N을 선택하는 것이 어려운 문제는 아니다. DR 방법은 N의 상위 δ자리가 모두 1일 경우 한번의 모듈라 곱셈을 수행하는 데 $2n^{2}-\delta n$번의 작은 수 곱셈이 필요하므로 몽고메리 곱셈에 비해 고속이다.

한편, S×M mod N을 계산한 후 S×S mod N을 계산할 경우에는 모든 $S_{bi}$을 저장해야 한다. 그러나 $S_{b0}$, $S_{b0}\cdot M_0$, $S_{b0}\cdot S_0$, $S_{b1}$ 등의 순서로 두 모듈라 곱셈의 중간 값을 번갈아 가면서 계산하면 $S_{bi}$를 하나의 임시 메모리에 저장하여 사용할 수 있다. DR 형태의 N을 사용할 경우 제안하는 공통 피승수 모듈라 곱셈을 나타낸 것이 도 1a 및 도 1b이다. 도 1a 및 도 1b에서 보는 바와 같이 첫번째 for문의 첫번째 줄에서 네번째 줄까지가 $S_{bi}$를 구하는 공통 계산 부분이고 두번째 for문이 n+2자리인 Tm과 Ts에 대한 모듈라 감소 부분이다. 한번의 모듈라 곱셈시에는 도면에 도시된 왼쪽부분만 수행한다.

도 1b에 도시한 바와 같이, 주어진 S, C, N'에 대해서 단계 10에서 변수 X를 S로 정의하고, 단계 11, 12에서 각각 $T_m$, $T_s$를 연산한다. 다음, 단계 13에서 기수 b에 대한 변수 X의 값을 정의하여 상기 변수 X를 구하는 데, 최종 캐리가 입력되면 상기 변수에 N'를 더하여 출력한다. 다음, 단계 14, 15에서 상기 $T_m$, $T_s$값을 상기 X를 이용하여 계산한다. 여기서, 단계13, 14, 15를 n-1회 수행한다. 다음, 단계 16, 17에서 상기 $T_m$, $T_s$값을 계산하는 데, 최종 캐리가 입력되면 상기 변수에 N'를 더하여 출력한다. 여기서 단계 16과 17은 2회 수행한다. 단계 18, 19에서 $T_m$, $T_s$값이 각각 N이상이면 $T_m=T_m-N$, $T_s=T_s-N$을 수행하여 원하는 모듈라 곱셈을 수행한다. 여기서 한번의 모듈라 곱셈을 수행할 때에는 왼쪽에 있는 부분만 수행한다.

S×M mod N 연산시 계산량은 다음과 같다. 먼저 각 $S_{bi}$를 구하는데 $(n-1)(n-\delta)$번, 이를 $M_i$와 곱하는데 $n^{2}$ 그리고 $T_m$에 대한 모듈라 감소 시 $2(n-\delta)$번의 작은 수 곱셈이 필요하다. 그러므로 S×M mod N 연산시에는 약 $2n^{2}-\delta n+n$번의 곱셈이 필요하다. 그러나 S×S mod N 연산시에는 $S_{bi}$를 구할 필요가 없으므로 약 $n^{2}+2n$번의 곱셈이 필요하다. S×M mod N과 S×S mod N을 계산시 몽고메리 곱셈을 사용한다면 총 $2(2n^{2}+n)$번의 곱셈이 필요하지만 DR 방법을 사용한다면 $2(2n^{2}-\delta n)$번의 곱셈으로 가능하다. 반면 본 발명의 방

법으로는 $3n^2-\delta n+3n$번으로 수행할 수 있어 계산량을 약 0.75배로 줄일 수 있게 된다.

이상으로 기술한 공통 피승수 곱셈은 멱승연산방법에 적용되어 고속의 멱승연산을 달성한다.

멱승 방법에는 이진 방식, m진(m-ary) 방식, 윈도우 방식, 지수 폴딩 방식 등이 있으며, 지수를 상하위 비트 중 어느 쪽에서부터 탐색하는가에 따라 레프트-투-라이트 방식과 라이트-투-레프트 방식으로 구분할 수 있다. 그러나 레프트-투-라이트 형태는 공통 피승수 곱셈이 없으므로 변형한 모듈라 곱셈을 적용하기가 어렵다. 본 발명에서는 라이트-투-레프트 형태의 2진, m-ary, 윈도우 멱승 방식 그리고 지수 폴딩 기법을 사용한 멱승 방식에 변형한 모듈라 곱셈 방법을 적용한다.

이진 멱승 방식으로 $A^E$ mod N 연산시 먼저 k 비트인 E를 이진수로 변환한다. 즉, $E=\sum_{i=0}^{k-1} e_i 2^i$, $e_i \in \{0,1\}$로 표현할 수 있다. 라이트-투-레프트 이진 멱승 방식을 나타낸 것이 도 3a이다. 이 멱승 방식에서 지수 E가 k비트이고 해밍웨이트(Hamming weight)가 k/2라 가정하면 약 3k/2번의 모듈라 곱셈이 필요하다. 그런데 도 3a에서 보는 바와 같이 $e_i$가 1일 때마다 공통 피승수 곱셈이 생기므로 상기 제안한 공통 피승수 모듈라 곱셈 방법을 적용할 수 있다. 결국 이진 멱승 방식의 경우 평균 k/2번의 공통 피승수 모듈라 곱셈과 k/2번의 일반 모듈라 곱셈이 필요하다.

도 3b에 도시한 바와 같이, 주어진 A, E, N에 대해서 단계 31에서 E를 k비트의 이진수로 변환하고, 단계 32에서 축소기(diminished radix) 형태의 모듈러스 N을 정의하고, 단계 33에서 M을 1로 정의하고, 단계 34에서 S를 A로 정의한다. 다음, 단계 35에서 $e_i$=1인가를 체크하여 1이면, C=SxM mod N, S=SxS mod N를 동시에 수행하고, 1이 아니면 S=SxS mod N를 독립적으로 수행하여, 주어진 수 A에 대한 모듈라 멱승을 수행한다.

m-ary 멱승 방식은 E를 t개의 자리수를 가지는 m진수로 나타낸다. 즉, $E=\sum_{i=0}^{t-1} E_i m^i$, $E_i \in \{0,m-1\}$로 표현할 수 있다. 도면에는 도시하지 않았지만, 라이트-투-레프트 이진 멱승방식과 유사하지만, m-ary 멱승 방식의 기본 원리는 $A^{m^i}$를 계산한 후 각 자리의 계수인 $E_i$에 해당하는 만큼 멱승을 수행하는 것이다.

윈도우 멱승 방식은 E를 이진수로 표현한 후 적당한 크기의 윈도우를 잡고 멱승을 수행한다. 그러나 윈도우를 일률적으로 같은 비트씩 잡는 것이 아니라 윈도우의 크기가 w이하이면서 시작과 끝이 1이 되게 한다. 라이트-투-레프트 윈도우 멱승 방식을 나타낸 것이 도 4이다. 이 방식은 라이트-투-레프트 m-ary 멱승 방식의 원리와 비슷하게 윈도우가 생기는 위치에서 A에 관한 멱승을 수행한 후 각 윈도우 값에 해당하는 만큼 멱승을 수행하도록 한 것이다. 윈도우 멱승 방식에서 평균 윈도우 개수가 $[\log_2 E/(w+1)]$ 개이므로 약 $[\log_2 E/(w+1)] + [\log_2 E] + 2(2^{w-1}-1)+3$번의 모듈라 곱셈이 필요하다. 도 4의 while문내에서 보는 바와 같이 윈도우가 발생할 때마다 공통 피승수 곱셈을 사용할 수 있다. 윈도우 멱승 방식은 고속이면서 간단히 구현할 수 있지만 중간 계산 값들을 저장할 테이블용 메모리가 $2^{w-1}$개 필요하다.

멱승연산 중 지수 폴딩 기법을 이용한 멱승이란 지수 E를 반으로 접어서 연산함으로써 계산 효율을 높이는 방법이다. 이 방식은 슈노르(Schnorr)가 제시한 서명 검증식 계산방법(C.P.Schnorr, Efficient signature generation for smart cards, Advances in Cryptology CRYPTO '89 Proceedings, Springer-Verlag, pp. 239-252, 1990)에 기초한 것으로서 레프트-투-라이트 형태나 라이트-투-레프트 형태로 구현할 수 있다. 이 방식의 기본 개념은 지수 E는 $E_H 2^{k/2} + E_L$로 표현할 수 있고 $E_H$와 $E_L$을 이진수로 나타내었을 때 공통적으로 1인 것에 대한 연산을 한번만 하도록 한 것이다.

구체적인 지수 폴딩을 이용한 멱승 방법을 기술하면 다음과 같다. 먼저 식(5)와 같이 $E_H$와 $E_L$를 비트별로 AND시켜 공통으로 1인 부분만 추출한다. 그리고 $E_{com}$과 $E_H$을 배타논리합(XOR)시키고 $E_{com}$과 $E_L$을 XOR한다. 이 경우 $E_{com}$, $E_{HXOR}$ 및 $E_{LXOR}$는 각각 E를 반으로 접었을 때 상위 반절과 하위 반절이 공통적으로 1인 수, 상위 반절만이 1인 수, 하위 반절만이 1인 수를 의미하며, 이진수로 표현하면 다음 식(6)과 같다. 여기서 $e_{com\_i}$, $e_{HXOR\_i}$, $e_{HXOR_i} \in \{0,1\}$이다.

[수학식 5]

$E_{com}=E_H \wedge E_L$, $E_{HXOR}=E_{com}$ 1s $E_H$, $E_{LXOR}=E_{com}$ 1s $E_L$

[수학식 6]

$E_{com} = \sum_{i=0}^{\frac{k}{2}-1} e_{com\_i} \cdot 2^i$, $E_{HXOR} = \sum_{i=0}^{\frac{k}{2}-1} e_{HXOR\_i} \cdot 2^i$, $E_{LXOR} = \sum_{i=0}^{\frac{k}{2}-1} e_{LXOR\_i} \cdot 2^i$

그러므로, $E_H$와 $E_L$는 다음 식(7)과 같이 표현할 수 있다.

[수학식 7]

$E_H=E_{com}+E_{HXOR}$, $E_L=E_{com}+E_{LXOR}$

지수 폴딩 기법을 이용한 라이트-투-레프트 형태의 멱승 과정을 도 5에 나타내었다. 도 5의 첫 번째 for문에서 $A^{E_{com}}$ mod N, $A^{E_{HXOR}}$ mod N 그리고 $A^{E_{LXOR}}$ mod N을 먼저 구한 후,

$A^{E_H} = A^{E_{com}} \cdot A^{E_{HXOR}}$ mod N과 $A^{E_L} = A^{E_{com}} \cdot A^{E_{LXOR}}$ mod N을 계산한다. 그리고 두번째 for 문에서 $(A^{E_H})^{2^w}$ mod N을 구하고 최종적으로 $A^E = (A^{E_H})^{2^w}$ mod N을 계산한다.

멱승 연산에 필요한 계산량을 보면 다음과 같다. E의 해밍웨이트가 k/2라면 $E_H$와 $E_L$의 해밍웨이트는 k/4이며 $E_{com}$, $E_{HXOR}$, $E_{LXOR}$의 해밍웨이트는 각각 k/8이 된다. 그러므로 k+3k/8+3번의 모듈라 곱셈이 필요하다. 이 방식에서는 첫 번째 for문에서 보는 바와 같이 3k/8번의 공통 피승수 모듈라 곱셈이 발생한다.

지금까지 본 발명의 고속 멱승 방법을 소프트웨어적인 방법면에서만 설명하였으나, 도 6을 참조하여 본 발명의 방법이 운영되는 하드웨어를 간단히 설명하기로 한다.

다음, 본 발명의 고속 멱승방법을 담고 있는 프로그램이 입/출력장치(72)를 통하여 입력되면 CPU(74)는 데이터 및 저장테이블들을 RAM(76) 이나 ROM 또는 EEPROM(78)으로부터 불러내어 멱승연산을 수행한다. 이러한, 하드웨어 환경하에서, 멱승연산시 메모리사용 부분에서 n자리의 수 A, B, N이 주어진 경우, 먼저 모듈라곱셈 AB mod N를 처리할 때 순수한 곱셈 C=AB를 계산한 후 그 결과에 대한 모듈라 감소 C mod N을 수행한다면 C의 크기는 A나 B의 자리수의 두배가 즉 2n자리가 된다. 이를 다시 모듈라 감소하면 n자리가 된다. 그러나, 본 발명에서는 곱셈을 하면서 모듈라 감소를 동시에 수행할 수 있으므로 가장 큰 중간값의 크기가 n+2자리수이므로 사용하는 메모리의 크기가 작게 된다. 또한, 멱승연산에서 필요한 메모리는 주로 중간값들을 저장하는 테이블의 수와 관계한다. 단, 하나의 테이블에는 n자리의 수가 저장된다. m-ary 멱승방식이나 윈도우 멱승방식에서 최적의 테이블의 수는 지수 E의 크기에 따라서 결정된다. 일반적으로 지수가 512비트일 때, m-ary 멱승방식에서는 15개, 윈도우 멱승방식에서는 16개가 최적의 테이블수이므로 IC카드와 같은 저메모리 환경하에서는 단점이 될 수 있으나, 본 발명에서는 변형된 몽고메리 모듈라 곱셈을 이용하면 m-ary 멱승방식에서는 7개, 윈도우 멱승방식에서는 8개에서 최적의 속도를 내므로, 전체적으로 메모리의 용량을 절반으로 줄일 수 있다. 결국, 모듈라곱셈에 사용되는 메모리의 양을 반으로 줄일 수 있을 뿐만 아니라 멱승방식에 따라서는 데이터저장용 테이블의 수도 반으로 줄일 수 있다. 한편, 데이터를 입력하는 것은 프로그램내에 저장하여 실행할 수도 있고 외부의 파일에 데이터를 저장한 후 그것을 입/출력장치(72)를 통해 읽어들여 사용할 수도 있다. 일반적으로 한 시스템을 운용할 때에는 고정되는 값이 있고 변하는 값이 있다. 예를 들어 ElGamal서명이나 DSS서명같은 경우에는 $A^E$ mod N를 계산하는 데 A와 N은 고정된 공통의 값이고 E는 변하는 값이다. 그러나, RSA서명에서는 E와 N이 자신의 정보로서 고정된 값이고 A는 변하는 값이다. 한편, RSA암호에서는 E와 N아 상대방의 정보이므로 변하는 값이고 A도 역시 변하는 값이다. 일반적으로 시스템을 운용할 때에는 고정된 값은 프로그램내부에 두고 변하는 값은 외부에서 독출하여 사용한다. 즉, 고정값들은 ROM이나 EEPROM에 저장해 둘 수 있고, 변하는 값은 RAM에 저장한 후 불러와서 사용한다.

이상으로 설명한 멱승방식들을 비교 분석하면 다음과 같다. 본 발명에서는 멱승 $A^E$ mod N을 구현시 A와 N을 512비트로 가정하였다. 기수 b는 $2^{16}$으로 하였는데 그 이유는 C 언어 컴파일러가 16비트의 곱셈을 지원하기 때문이다. 이 경우 A나 N을 기수 b로 표현하면 자리수 n은 32가 된다. 모듈러스 N은 다음과 같이 상위 4자리가 모두 1인 수로 하였다.

N = ffff ffff ffff ffff 9084 d56b f6bb 91f8 0c6f 5c4f f6c9 b0b9 b5fc a3f4 6270 961a bf56 1e3f d76b fb3d 7d74 7674 a2f7 3857 b791 ce54 e861 4682 48af d33f 225b

모듈라 곱셈 방법을 비교해 볼 때, 고전적인 방법은 몫 추정시 나눗셈은 없앨 수 있지만 뺄셈 연산이 많아 다른 방식에 비해 저속이다, 몽고메리 알고리듬은 N의 형태에 관계없이 나눗셈이 없으며 N이 일반적인 수일 경우에는 현재까지 가장 고속이다. DR 방법은 상기한 N의 조건을 만족할 경우 몽고메리 곱셈보다 고속이다. 본 발명에 의한 공통 피승수 모듈라 곱셈시 공통 계산 부분을 생략할 수 있으며 구체적인 내부 연산은 DR 방법에 기초하여 수행되므로 기존 방법에 비해 고속이다.

멱승 알고리듬의 계산량을 비교할 때에는 모듈라 곱셈수를 고려한다. 그러나 모듈라 곱셈은 기수 b보다 작은 수 곱셈으로 이루어지므로 결국 멱승에 필요한 작은 수 곱셈을 분석함으로써 전체 계산량을 비교할 수 있다. 특히, 본 발명의 모듈라 곱셈을 사용할 때에는 멱승에 필요한 모듈라 곱셈수도 고려해야 하지만 공통 피승수 곱셈의 발생 빈도에 따라 계산량이 달라진다. 이진 멱승 방식에서 E가 k비트이면 레프트-투-라이트 방식과 라이트-투-레프트 방식 모두 약 3k/2번의 모듈라 곱셈이 필요하다. 그러므로 몽고메리 알고리듬을 사용하면 $3k(2n^2+n)/2$번의 작은 수 곱셈이 필요하며 DR 방법으로는 $3k(2n^2-\delta n)/2$번이 필요하다. 그러나 라이트-투-레프트 멱승 연산에서는 k/2번의 공통 피승수 곱셈이 생기므로 본 발명의 방법을 사용하면 $3k(3n^2-\delta n+3n)/2+k(2n^2-\delta n+n)/2 = 2.5kn^2-k\delta n+2kn$번이 필요하다. 예로서 E가 512비트이고 $\delta$가 4이면 DR 방법은 몽고메리 알고리듬에 비해 약 7.7% 개선할 수 있는 반면 본 발명의 방법은 20% 개선할 수 있다. 즉, 본 발명의 방법은 한번 멱승이 768번의 몽고메리 모듈라 곱셈을 614번으로 감소시킨 것과 동일하다.

레프트-투-라이트 윈도우 멱승 방식에서는 512 비트 멱승시 윈도우 크기가 5일 때 최적이며 평균 609번의 모듈라 곱셈이 필요하다. 라이트-투-레프트 윈도우 멱승 방식에서는 $[\log_2 E/(w+1)]+[\log_2 E]+2(2^{w-1}-1)+3$번의 모듈라 곱셈이 필요하다. 이 경우 2w-1개의 저장용 테이블이 필요하다. 그러나 라이트-투-레프트 윈도우 멱승 연산에서는 윈도우 개수 만큼 공통 피승수 곱셈이 생기므로 모듈라 곱셈 방법을 적용하면 $([\log_2 E]-[\log_2 E/(w+1)])(2n^2-\delta n+n)+([\log_2 E/(w+1)])(3n^2-\delta n+3n)+(2(2^{w-1}-1)+3)(2n^2-\delta n+n)$ 번의 작은 수 곱셈이 필요하다. 이는 위의 경우와 동일한 조건일 때 556번의 모듈라 곱셈을 수행한 것과 같다. 그러나 윈도우 크기를 4로 하면 $(512-103)(2n^2-\delta n+n)+ 103(3n^2-\delta n+3n)+17 (2n^2-\delta n+n)$이고 이것은 약 550번의 모듈라 곱셈을 수행하는 결과와 같다. 지수가 512비트이고 변형한 곱셈을 사용하는 경우에는 윈도우 크기가 4일때 속도도 빠르고 메모리 사용을 절반정도로 감소할 수 있다. 그 이유는 윈도우 크기가 작으면 평균 윈도우 개수는 증가하므로 윈도우가 생기는 위치의 멱승 값을 계산하는 연산은 많아지나 모듈라 곱셈으로 보상할 수 있고, 반면 저장용 테이블이 감소하게 되어 테이블을 멱승하는 계산량이 줄어들기

때문이다.

지수 폴딩 기법을 이용한 멱승은 레프트-투-라이트 방식이나 라이트-투-레프트 방식으로 구현할 수 있으며 각각 k+3k/8+1와 k+3k/8+3번의 모듈라 곱셈이 필요하다. 그러나 라이트-투-레프트 방식에서는 3k/8번의 공통 피승수 곱셈이 생기므로 모듈라 곱셈을 사용하면 $3k/8(3n^2-\delta n+3n)+(k-3k/8+3)(2n^2-\delta n+n)$번이 필요하다. 이는 E가 512비트이고 δ가 4일 때 768번의 몽고메리 모듈라 곱셈을 584번으로 감소시킨 것과 동일하다.

표 1은 지수 E가 512 비트 및 160 비트인 경우에 대해, 모듈라 곱셈과 멱승 방식에 따른 성능을 나타낸 것으로서 몽고메리 곱셈을 사용한 이진 방식을 기준으로 상대적이 모듈라 곱셈수를 나타낸 것이다. 단, 0(E0(는 E의 비트 수를 의미한다.

[표 1]

| 멱승방식 / 모듈라 곱셈방식 | 2진(R-to-L) | | 폴딩(R-to-L) | | 윈도우(R-to-L) | |
|---|---|---|---|---|---|---|
| | \| E\| =5 | \| E\| =16 | \| E\| =5 | \| E\| =16 | \| E\| =5 | \| E\| =16 |
| 몽고메리 | 768 | 240 | 707 | 223 | 631 | 209 |
| DR | 709 | 222 | 653 | 206 | 582 | 193 |
| 본 발명 | 614 | 192 | 584 | 184 | 550 | 183 |

일반적으로 멱승 방식에서 레프트-투-라이트 형태가 라이트-투-레프트 형태보다 계산 측면에서 다소 유리하지만 라이트-투-레프트 형태에서는 공통 피승수 곱셈이 생기므로 이를 효과적으로 처리하면 고속 멱승을 실현할 수 있다. 특히, 이진 방식이나 지수 폴딩 기법을 이용한 방식에서는 공통 피승수 곱셈의 발생 비율이 높으므로 개선효율이 크다. 윈도우 방식은 이들보다는 고속이지만 데이터 저장용 테이블이 필요하므로 메모리가 확보된 시스템에서 사용할 수 있다.

상기한 멱승 방식들을 퍼스널컴퓨터(PC) 및 워크스테이션(workstation)에서 C 언어로 구현하였다. 몽고메리 알고리듬, DR 방법 그리고 본 발명의 방법으로 모듈라 곱셈을 구현하여 각 멱승 방식에 적용하였다. PC는 펜티엄급(Pentium(586)/133MHz)을 사용하였고 DOS 환경 하에서 볼란드(Borland) C로 컴파일 하였다. 워크스테이션은 솔라이스(Solais) 2.5.1환경 하에서 썬울트라(SUN Ultra) 2/200MHz를 사용하였다.

표 2와 표 3은 지수가 512 비트와 160비트인 경우 PC 및 워크스테이션에서 한번의 멱승을 수행하는데 소요되는 평균 시간을 나타낸 것이다. 이 결과는 각 모듈라 곱셈을 사용하여 멱승을 1000번씩 수행 후 이를 평균한 것이다. 표의 ( )는 몽고메리 곱셈을 사용한 이진 방식을 기준으로 한 상대적인 모듈라 곱셈수를 나타낸 것이다.

[표 2]

| 멱승방식 / 모듈라 곱셈방식 | 2진(R-to-L) | | 폴딩(R-to-L) | | 윈도우(R-to-L) | |
|---|---|---|---|---|---|---|
| | \| E\| =512 | \| E\| =160 | \| E\| =512 | \| E\| =160 | \| E\| =512 | \| E\| =160 |
| 몽고메리 | 0.777(768) | 0.244(240) | 0.715(707) | 0.227(223) | 0.637(530) | 0.212(216) |
| DR | 0.715(707) | 0.224(220) | 0.674(666) | 0.213(210) | 0.599(502) | 0.197(194) |
| 본 발명 | 0.638(631) | 0.198(196) | 0.610(603) | 0.192(189) | 0.580(573) | 0.191(188) |

[표 3]

| 멱승방식 / 모듈라 곱셈방식 | 2진(R-to-L) | | 폴딩(R-to-L) | | 윈도우(R-to-L) | |
|---|---|---|---|---|---|---|
| | \| E\| =512 | \| E\| =160 | \| E\| =512 | \| E\| =160 | \| E\| =512 | \| E\| =160 |
| 몽고메리 | 0.306(768) | 0.096(240) | 0.282(600) | 0.090(220) | 0.251(622) | 0.083(203) |
| DR | 0.293(735) | 0.090(225) | 0.272(683) | 0.085(213) | 0.240(602) | 0.078(195) |
| 본 발명 | 0.246(617) | 0.076(190) | 0.232(582) | 0.075(188) | 0.220(552) | 0.073(183) |

표2와 3에서 보는 바와 같이 몽고메리 곱셈을 각 멱승 방식에 적용했을 경우 이론적인 소요시간과 구현 결과가 거의 일치함을 볼 수 있다. 또한 제안 방법을 적용했을 경우가 몽고메리 곱셈을 사용하는 것보다

고속 멱승을 구현할 수 있었고 그 비율도 표 1의 분석결과와 비슷하였다. 예로서 이진 멱승의 경우 DR 방법이 몽고메리 곱셈에 비해 PC에서 약 0.92배, 워크스테이션에서는 0.95배의 시간이 소요된다. 반면 본 발명의 방법은 PC에서 약 0.82배, 워크스테이션에서는 0.79배의 시간이 소요됨을 알 수 있다. 위의 결과는 δ가 4일 경우이고 δ가 2나 3인 경우에도 시뮬레이션한 결과 이론적인 산출량과 거의 일치함을 확인하였다. 다른 PC나 워크스테이션에서도 수행한 결과, 멱승 연산의 속도는 시스템 기종의 성능, 컴파일러의 선택, 혹은 최적화 기능의 추가 등에 따라 약간씩 달랐지만 본 발명의 방법이 몽고메리 곱셈이나 DR 방법에 비해 고속으로 멱승을 수행할 수 있다.

### *발명의 효과*

상술한 바와 같이, 본 발명은 모듈러스 N이 축소기 형태일 경우 고속 멱승을 위한 새로운 모듈라 곱셈 방법을 제공하는 것으로, 동일한 피승수를 가지는 두 번의 모듈라 곱셈시 공통 계산 부분을 한번에 계산할 수 있도록 한 것으로서 이를 라이트-투-레프트 형태의 멱승에 적용하면 멱승 계산속도를 개선하고, 메모리용량을 줄일 수 있다. 특히, IC 카드와 같은 저 메모리 환경 하에서는 본 발명의 지수 폴딩 기법을 사용한 멱승 방식을 사용하는 것이 효과적이다. 따라서 본 발명은 RSA나 디지탈 서명 등 공개키 암호시스템에 효과적으로 사용할 수 있다.

### *(57) 청구의 범위*

#### 청구항 1

주어진 수 A, E, N에 대해서, 수 $C=A^E \bmod N$의 모듈라 멱승방법에 있어서,

(a) 모듈라곱셈을 $C=SxM \bmod N$, $S=SxS \bmod N$이라 할 때, 상기 모듈라곱셈을 기수(radix) b에 대해서 다음 식으로 변형하는 단계,

$S \times M \bmod N = S(M_0 b^0 + M_1 b^1 + \cdots + M_{n-1} b^{n-1}) \bmod N$

$= (S_{b0} \cdot M_0 + S_{b1} \cdot M_1 + \cdots S_{bn-1} \cdot M_{n-1}) \bmod N$

$= T_m \bmod N$

$S \times S \bmod N = S(S_0 b^0 + S_1 b^1 + \cdots + S_{n-1} b^{n-1}) \bmod N$

$= (S_{b0} \cdot S_0 + S_{b1} \cdot S_1 + \cdots + S_{bn-1} \cdot S_{n-1}) \bmod N$

$= T_s \bmod N$

(여기서, $S_{bi}=Sb^i \bmod N$ $(0 \le i \le n-1)$이며 $T_m$은 $S_{b0} \cdot M_0 + S_{b1} \cdot M_1 + \cdots S_{bn-1} \cdot M_{n-1}$이고, $T_s = S_{b0} \cdot S_0 + S_{b1} \cdot S_1 + \cdots + S_{bn-1} \cdot S_{n-1}$이다.);

(b) 멱승 방법에 따라서 소정 횟수만큼 $S=SxS \bmod N$를 독립적으로 수행하거나, $C=SxM \bmod N$, $S=SxS \bmod N$를 동시에 수행하여 주어진 수 A에 대한 모듈라 멱승을 수행하는 단계를 포함하는 모듈라 멱승방법.

#### 청구항 2

제1항에 있어서, 상기 모듈러스 N은 $N=b^n-N'(N'b^{n-\delta})$ (여기서, N은 상위자리 중 δ자리가 모두 1인 수)를 만족하며, $b^n=N' \bmod N$에 기초하여 모듈라감소를 수행함을 특징으로 하는 모듈라 멱승방법.

#### 청구항 3

제1항 또는 제2항에 있어서, 상기 멱승방법이 이진 멱승 방법인 경우에는 지수 E를 $E=\sum_{i=0}^{t-1} e_i 2^i$, $e_i \in \{0,1\}$로 표현할 때, $e_i=1$인 경우 동일한 S에 대해서 $C=SxM \bmod N$, $S=SxS \bmod N$의 공통피승수곱셈연산을 수행하고, 이외의 경우에는 $S=SxS \bmod N$의 곱셈연산을 수행하는 것을 특징으로 하는 모듈라 멱승방법.

#### 청구항 4

제1항 또는 제2항에 있어서, 상기 멱승방법이 m진 멱승방법인 경우에는 지수 E를 t개의 자리수를 갖는 m진수 $E=\sum_{i=0}^{t-1} e_i m^i$, $e_i \in \{0,\ldots,m-1\}$로 표현할 때, $e_i=0$이 아닌 경우 동일한 S에 대해서 $C=SxM \bmod N$, $S=SxS \bmod N$의 공통피승수곱셈연산을 수행하고, 이외의 경우에는 $S=SxS \bmod N$곱셈연산을 수행하는 것을 특징으로 하는 모듈라 멱승방법.

#### 청구항 5

제1항 또는 제2항에 있어서, 상기 멱승방법이 윈도우 멱승방식인 경우에는 지수 E를 $E=\sum_{i=0}^{t-1} e_i 2^i$, $e_i \in \{0,1\}$로 표현하여, w크기의 윈도우를 정하여 연산할 때 윈도우가 하나 발생할 때마다, 동일한 S에 대해서 $C=SxM \bmod N$, $S=SxS \bmod N$의 공통피승수곱셈연산을 수행하고, 이외의 경우에는 $S=SxS \bmod N$곱셈연산을 수행하는 것을 특징으로 하는 모듈라 멱승방법.

**청구항 6**

제1항 또는 제2항에 있어서, 상기 멱승방법이 지수폴딩 멱승방법인 경우에는 지수 E를 $E=\sum_{i=0}^{t-1} e_i 2^i$, $e_i \in \{0,1\}$로 표현하여, $E_H$은 상위반절, $E_L$은 하위반절이라 하면 $E=E_H 2^{t/2}+E_L$이므로 $E_H$과와 $E_L$이 공통적으로 1인 $E_{com}=E_H$ AND $E_L$, 상위반절만 1인 $E_{HXOR}=E_{com}$ XOR $E_H$, 하위반절만 1인 $E_{LXOR}=E_{com}$ XOR $E_L$를 얻는 과정;

$E_{com}$, $E_{HXOR}$, $E_{LXOR}$이 각각 1인 경우, 동일한 S에 대해서 C=SxM mod N, S=SxS mod N의 공통피승수곱셈연산을 수행하고, 이외의 경우에는 S=SxS mod N곱셈연산을 수행하여, $A^{E_{com}}$ mod N, $A^{E_{HXOR}}$ mod N 그리고 $A^{E_{LXOR}}$ mod N을 먼저 구하는 과정;

$A^{E_H}=A^{E_{com}}\cdot A^{E_{HXOR}}$ mod N과 $A^{E_L}=A^{E_{com}}\cdot A^{E_{LXOR}}$ mod N을 계산하는 과정; 및

최종적으로 $A^E=(A^{E_H})^{2^{t/2}}\cdot A^{E_L}$ mod N을 계산하는 과정;을 포함하는 것을 특징으로 하는 모듈라 멱승방법.

**청구항 7**

제3항 내지 제6항에 있어서, 지수를 하위비트에서 상위비트쪽으로 탐색하는 라이트-투-레프트 방식을 채용함을 특징으로 하는 모듈라 멱승방법.

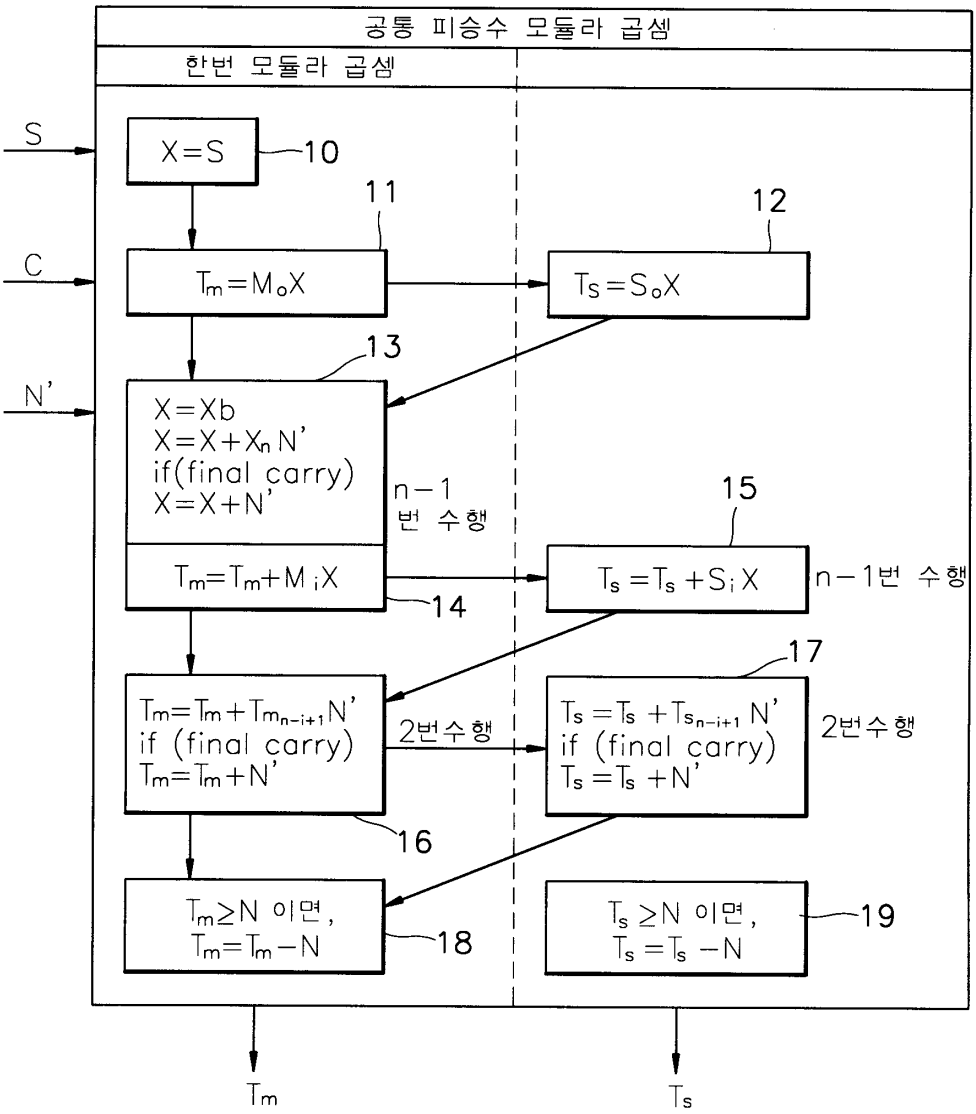
*도면*

*도면1a*

Common-multiplicand modular multiplication :

| | |
|---|---|
| Tm=S×M mod N, | Ts =S×S mod N |
| X=S | |
| Tm=$M_0$X ; | Ts=$S_0$X ; |
| for i=1 to n-1 step 1{ | |
| X=Xb; | |
| X[n-1:0] =X[n-1:0]+$X_n$N'; | |
| if(final carry) | |
| X[n-1:0] = X[n-1:0]+N'; | |
| Tm= Tm + $M_i$X; | Ts= Ts + $S_i$X; } |
| for i=0 to 1 step 1{ | |
| Tm[n-i:0] =Tm[n-i:0]+$Tm_{n-i-1}$N'; | |
| if(final carry) | |
| Tm[n-i:0] = Tm[n-i:0]+N'; | |
| | Ts[n-i:0] =Ts[n-i:0]+$Ts_{n-i-1}$N'; |
| | if(final carry) |
| | Ts[n-i:0] = Ts[n-i:0]+N'; |
| } | |
| if (Tm≥N) Tm=Tm-N; | If (Ts≥N) Ts=Ts-N; |
| return (Tm,Ts); | |

도면1b



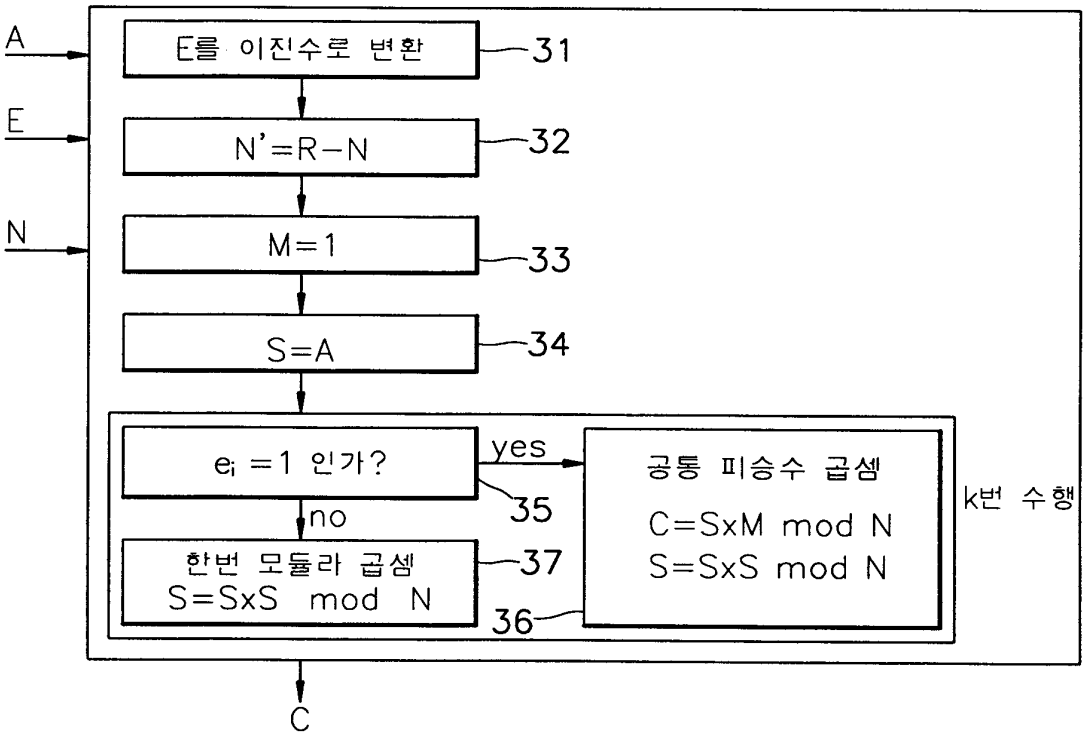

도면2

```
Diminished radix : C mod N
 for i= 2n-1 to n step -1
     C[i-1:i-n] = C[i-1:i-n]+CiN'
 if(final carry) C[i-1:i-n] = C[i-1:i-n]+N'
 if(C≥N) C= C-N;
 return (C);
```

도면3a

```
Right-to-left binary exponentiation : C=A^E mod N
 M= 1;
 S= A;
 for i=0  to  k-1 step 1
    if( ei=1) then { C=S×M mod N; S=S×S mod N;}
    else S=S×S mod N;
 return(C);
```

도면3b




도면4

```
Window exponentiation : C=A^E mod N

for i=0 to 2^(w-1)-1 step 1 { T[i]= 1; }
S=A;
i=0 ;
while i≤k do{
    if e_i=1 then
        j=MIN(k,i+w-1) ;
        while e_j=0 do
            j=j-1 ;
        e=(e_j...e_i)_2 /2 ;
        T[e]=S×T[e] mod N;
        S=S×S mod N;
        for k=i+1 to j step 1 { S=S×S mod N; }
        i=j+1 ;
    else
        S=S×S mod N;
        i=i+1;
}
for i=2^(w-1)-2 to0 step -1 { T[i]=T[i]×T[i+1] mod N; }
for i=2^(w-1)-2 to1 step -1 {T[2^(w-1)-1]=T[2^(w-1)-1]×T[i] mod N; }
T[2^(w-1)-1]=T[2^(w-1)-1]×T[2^(w-1)-1] mod N;
C=T[2^(w-1)-1]×T[0] mod N;
return (C);
```

도면5

```
Folding : C=A^E mod N

C_com = C_L = C_H = 1;    S=A;
for i=0 to k/2-1 step 1 {
    if( e_com_i = 1)  { C_com = S×C_com mod N; S=S×S mod N;}
    else if( e_HXOR_i = 1) { C_H = S×C_H mod N; S=S×S mod N;}
    else if( e_LXOR_i = 1) { C_L = S×C_L mod N; S=S×S mod N;}
    else S=S×S mod N; }
C_H = C_H×C_com mod N;    C_L = C_L×C_com mod N;
for i=0 to k/2-1 step 1 { C_H = C_H×C_H mod N; }
C = C_H×C_L mod N;
return(C);
```

도면6