情報保護學會論文誌
第14券 第1號, 2004. 2

병렬 지수승에서 라운드 수 축소를 위한 알고리즘*

김 윤 정†
서울여자대학교 정보통신공학부

An Algorithm For Reducing Round Bound of Parallel Exponentiation

Yoonjeong Kim†
Seoul Women's University

요 약

지수승(exponentiation) 연산은 암호 관련 응용에서 널리 사용되고 있으며, 안전성을 위해 지수 n의 값을 크게 설정하여 이용하고 있다. 그런데, n의 값이 커짐에 따라 수행해야 하는 곱셈의 횟수도 따라서 증가하게 되고, 결과적으로 속도가 빠른 연산 알고리즘의 개발이 중요한 문제로 대두되고 있다. 본 논문에서는 정규 기저 표현(normal bases representation)을 갖는 $GF(2^n)$ 상의 병렬 지수승 연산에 있어서, 프로세서 수가 고정된 경우에 라운드 수를 개선할 수 있는 알고리즘을 제안하고 이의 성능분석을 수행한다. 제안하는 방안은 지수(exponent)를 특정 비트 수로 나누어 지수승을 수행하는 윈도우 방법(window method)를 이용하는 것으로, 윈도우 값 계산 단계에서 휴지 프로세서들로 하여금 윈도우들 간의 곱을 계산하도록 함으로써, 전체 라운드 수를 줄이는 효과를 갖는다.

ABSTRACT

Exponentiation is widely used in practical applications related with cryptography, and as the discrete log is easily solved in case of a low exponent n, a large exponent n is needed for a more secure system. However, since the time complexity for exponentiation algorithm increases in proportion to the n figure, the development of an exponentiation algorithm that can quickly process the results is becoming a crucial problem. In this paper, we propose a parallel exponentiation algorithm which can reduce the number of rounds with a fixed number of processors, where the field elements are in $GF(2^n)$, and also analyzed the round bound of the proposed algorithm. The proposed method uses window method which divides the exponent in a particular bit length and make idle processors in window value computation phase to multiply some terms of windows where the values are already computed. By this way, the proposed method has improved round bound.

Keywords: *exponentiation, GF(2^n), round bound, cryptography*

Ⅰ. 서 론

지수승(exponentiation)연산은 암호 관련 응용에서 폭넓게 이용되고 있으며, 지수 n의 값이 작으면 이산로그가 쉽게 구해지므로, 보통 안전한 시스템을 유지하기 위해 n 값을 크게 설정한다. 그런데, n의 값이 커짐에 따라 지수승 연산을 수행하는 시간도 따라서 증가하게 되고, 결과적으로 속도가 빠른 지수승 알고리즘의 개발이 대단히 중요한 문제로 대두되고 있다.

속도를 증진시키기 위한 일환으로 병렬 알고리즘이 제안되었는데, 이들은 필드 요소가 정규기저표현

접수일: 2004년 1월 15일; 채택일: 2004년 2월 5일
* 본 연구는 2002년도 서울여자대학교 교내연구비의 지원으로 이루어졌습니다.
† yjkim@swu.ac.kr

인 것[1~5]과 다항식 표현인 것들[7,8]로 나눌 수 있다. 이 중 $GF(2^n)$ 상에서 정규기저 표현을 갖는 병렬 지수승 연산에 대한 연구는 고정된 라운드에 대하여 프로세서 수를 줄이는 것과 프로세서 × 라운드의 바운드를 분석한 것 등이 있다.[1,2,4] 또한, 프로세서 수가 고정된 경우에 라운드 수의 바운드를 개선하는 알고리즘을 고려할 수 있는데, 이에 대하여는 Stinson이 제안한 방안이 가장 효율적인 것으로 알려져 있다.

본 논문에서는 윈도우 값 계산 단계에서 휴지 프로세서들로 하여금 윈도우들간의 값을 미리 계산하도록 함으로써 Stinson이 제안한 방안보다 더 적은 수의 라운드 수를 갖는 알고리즘을 제안하고 이의 성능분석을 수행한 내용을 소개한다.

본 논문은 다음과 같이 구성된다. 우선, 2 장에서는 연구의 배경으로서 정규기저표현을 갖는 $GF(2^n)$ 상의 연산과 지수승 연산, 병렬 지수승 연산의 개념에 대하여 기술한다. 그리고, 프로세서 수가 고정된 경우의 라운드 수에 대한 기존 연구 결과를 소개한다. 3 장에서는, 기존 제안 알고리즘에서 라운드 수를 향상 시킬 수 있는 가능성을 살펴보고 라운드 수를 개선하는 새로운 알고리즘을 제안한다. 그리고, 제안 알고리즘의 라운드 수 분석 내용을 기술한다. 4 장에서, 지수의 비트 길이와 프로세서 수를 다양하게 변화시키며 기존 알고리즘과 제안 알고리즘의 성능을 비교한 내용을 기술하고, 5 장에서 결론을 맺는다.

II. 배경 지식

여기서는 소단원에 관한 내용을 간단히 살펴보겠습니다. 게다가 소소단원에 관한 내용도 간단히 살펴보겠습니다.

2.1 정규기저표현을 갖는 $GF(2^n)$상의 연산

$GF(2^n)$은 $GF(2)$ 상에서의 n 차원 벡터 공간으로, 이 공간에서의 $\beta, \beta^2, \beta^4, \ldots\ldots, \beta^{2^{n-1}}$ 형태의 기저는모든 n에 대하여 정규 기저를 갖는다고 알려져 있다. 어느 한 필드 요소 a가 정규기저에 대한 계수로 표시될 때 이것을 "정규기저표현"이라 하는데, 정규기저표현으로 나타내어진 필드요소 a를 제곱하는 연산은 계수에 대한 환형 쉬프트만으로 이루어진다는 특성을 갖는다.[9]

예를 들어, x^3+x^2+1을 근저 다항식으로 갖는 $GF(2^3)$의 경우 α를 이 다항식의 근이라 할 때, $(\alpha, \alpha^2, \alpha^4)$은 정규기저이다. 또한, $\alpha^4=1+\alpha+\alpha^2$으로 표시될 수 있으므로, $(\alpha, \alpha^2, 1+\alpha+\alpha^2)$도 정규기저이다. 이 때, 값이 $a=\alpha$인 필드 요소 a의 정규기저표현은 (1,0,0)이며, a^2은 (0,1,0), a^4은 (0,0,1), a^8은 (1,0,0)으로 나타내진다. 이상과 같이, $GF(2^n)$의 정규기저표현에서 제곱은 환형쉬프트만으로 구성될 수 있으므로, 보통 제곱에 필요한 시간은 무시한다고 가정하며,[1~5] 이 가정은 본 논문에도 적용된다.

2.2 지수승 연산 알고리즘

지수승(exponentiation) 연산이란 주어진 a에 대하여 a^e을 계산하는 것으로 이 때, $e=\sum_{i=0}^{n-1} e_i 2^i$, $e_i=0$ 또는 1, $0\le i\le n-1$이다. 기본적인 지수승 연산 기법은 '이진 방법'으로 이 기법에서는 $0\le i\le n-1$인 $a^{e_i 2^i}$들을 모두 곱한다. 즉, $\prod_{i=0}^{n-1} a^{e_i 2^i}$를 계산한다. $GF(2^n)$상의 정규기저표현에서는 a^{2^i}의 비용을 무시할 수 있으므로, 이진방법에서의 곱셈의 개수는 값이 1인 $e_i(0\le i\le n-1)$의 개수가 된다. 결과적으로 이진 방법은 평균적으로 $\frac{n}{2}-1$ 개의 곱셈을 필요로 한다.

이진 방법보다 더 작은 수의 곱셈을 필요로 하는 '윈도우 방법(window method)'은 지수에 특정 패턴이 반복적으로 나타나는 특성을 이용한다. 윈도우 방법에서는 지수의 비트들을 k 개씩 나누어, 지수 e를 $e=\sum_{i=0}^{s-1} w_i 2^{ki}$ 형태로 표시한다. 여기서 $s=\lceil n/k \rceil$이며 $0\le w_i\le 2^k-1$, $0\le i\le s-1$이다. 이 때, k는 윈도우 길이라 불린다. 이런 표현 하에서, 지수승 연산은 2 가지 단계로 나뉘어 처리된다. 첫째는 윈도우 값 계산 단계이며, 둘째는 윈도우들 간의 곱셈을 수행하는 단계이다. 그림 1에 주어진 알고리즘 window()가 윈도우 방법을 설명해 준다.

알고리즘 window (a,e,k)

// a^e을 계산한다

단계 1: 2^k-1 개의 $a^{w_i}(1\le w_i\le 2^k-1)$ 를 구한다.

단계 2: $(a^{w_i})^{2^{ki}}(0\le i\le s-1)$ 항들을 서로 곱한다.

그림 1. 지수승 연산 a^e을 수행하는 알고리즘

위 알고리즘의 예는 다음과 같다. $n=10, k=2$라 할 때, $e=631=1001110111(binary)$라 하자. 그러면, $s=5$가 된다. 단계 1에서는, 윈도우 값인 a^1, a^2, a^3을 계산하며 단계 2에서는, 윈도우 $(a^2)^{2^8}, (a^1)^{2^6}, (a^3)^{2^1}, (a^1)^{2^2}, (a^3)^{2^0}$ 들 간의 곱이 수행된다. 정규기저표현을 갖는 $GF(2^n)$ 상에서의 지수승 연산은 a^{2^r}가 비용이 없는 제곱 연산에 의하여 수행되므로, 단계 2는 단지 4 개의 곱셈 연산만을 필요로 한다.

2.3 병렬 지수승 연산

m개 요소의 곱셈을 병렬로 진행하는 가장 기본적인 방법은, 그림 2와 같이, $m/2$ 개의 프로세서로 각 2 요소를 곱하고, 이들 결과의 각 2 요소를 다시 $m/4$ 개의 프로세서로 곱하는 것이다. 이런 식으로 최종 결과가 나올 때까지 진행한다. 즉, m 개 요소의 곱을 진행하는데, m이 짝수인 경우에는 $m/2$ 개의 프로세서로 $\lceil \log_2 m \rceil$ 라운드가 필요하며, m이 홀수인 경우에는 $\lfloor m/2 \rfloor$ 개의 프로세서로 $\lceil \log_2 m \rceil$ 라운드가 필요하다.

그림 2의 방법을 이용하여, 알고리즘 window를 병렬로 진행하는 방법은 다음과 같다.[3] 우선, 단계 2에서 필요한 총 곱셈의 개수는 $s=\lfloor n/k \rfloor$이며, 이를 $\lfloor s/2 \rfloor$ 개의 프로세서로 그림 2의 방법으로 진행하면 $\lceil \log_2 s \rceil$ 라운드가 필요하다. 다음으로, 단계 1은 알고리즘 그림 3의 d-c-small-powers 에 의하여 수행될 수 있으며, $P(k)=(2^{k/2}-1)^2$(k가 짝수인 경우) 또는 $P(k)=(2^{(k+1)/2}-1)(2^{(k-1)/2}-1)$ (k가 홀수인 경우) 개의 프로세서로 $\lceil \log_2 k \rceil$ 라운드 동안 진행된다.

d-c-small-powers (a,5) 가 호출된 경우를 예로 들면 다음과 같다. $j=3$이 되고, 부단계 1에서 $a^1, a^2, a^3, a^4, a^5, a^6, a^7$을 계산한다. 부단계 2에서는 a^8, a^{16}, a^{24}를 계산한다. 부단계 3에서는 부단계 1과 부단계 2에서 계산된 요소들을 각각 곱하여

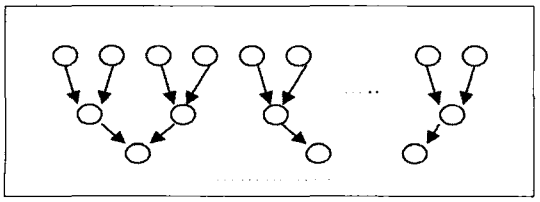

그림 2. m 개 요소 곱의 기본 병렬 연산: $m/2$ 개의 프로세서로 $\log_2 m$ 라운드가 필요함

알고리즘 d-c-small-powers (a,k)
// $a^w (1 \le w \le 2^k-1)$를 계산한다
부단계 1: 2^j-1개의 $a^w (1 \le w \le 2^j-1)$를 구한다. 이 때, $j=\lfloor k/2 \rfloor$이다.
부단계 2: 부단계 1에서 계산한 $a^u (1 \le u \le 2^{k-j}-1)$에 대하여, 를 계산한다.
부단계 3: 부단계 1에서 계산한 a^w 각 항과 부단계 2에서 계산한 $(a^u)^{2^r}$ 각 항을 곱한다.

그림 3. 윈도우 값을 계산하는 알고리즘(알고리즘 window의 단계 1 수행)

21 개의 값을 얻는다.

위에서 기술한 바와 같이, Stinson은 알고리즘 window를 병렬로 진행하는 경우, 프로세서 수가 $\max P(k), \lfloor s/2 \rfloor$일 때 최대 $\lceil \log_2 k \rceil + \lceil \log_2 s \rceil$ 라운드가 필요함을 밝히고 있다.[3]

2.4 프로세서 수가 고정된 경우의 라운드 수

알고리즘 window가 수행될 때, 프로세서 수가 2^k (이 때, $2^k \le \lfloor s/2 \rfloor$)로 제한된 경우의 최대 라운드 수에 대한 연구 결과는 다음과 같다.[3] 우선, 단계 1은 알고리즘 d-c-small-powers를 통하여 수행되는데, k가 짝수인 경우와 홀수인 경우 모두 2^k가 2.3 절에서 기술한 $P(k)$보다 크므로 최대 $\lceil \log_2 k \rceil$ 라운드가 필요하다. 다음으로 s 개의 요소를 곱하는 단계 2는 다음과 같이 진행된다. p_0를 2의 제곱 값들 중 p를 넘지 않는 최대값이라 할 때, 단계 2의 마지막 $\log_2 2p_0 = \log_2 p_0 + 1$ 라운드는 그림 2의 기본 병렬 연산을 통하여 $2p_0$ 개의 요소를 곱한다. 이 $\log_2 p_0 + 1$ 라운드에 수행되는 곱셈의 개수는 $2p_0 - 1$이다. 따라서, 단계 2의 앞부분 라운드에서는 $s-1-(2p_0-1)=s-2p_0$ 개의 곱셈을 필요로 한다. $s-2p_0$ 개의 곱셈은 p 개의 프로세서로 $\lceil (s-2p_0)/p \rceil$ 라운드에 수행될 수 있다. 따라서, 프로세서 수가 $p=2^k$ 인 경우에 단계 2의 필요한 라운드 수는 $\lceil (s-2p_0)/p \rceil + \log_2 p_0 + 1 = \left\lceil \frac{s}{2^k} \right\rceil - 2 + k + 1$ 이다. 결과적으로, 2^k 개의 프로세서로, $GF(2^n)$ 상에서 정규기저표현을 갖는 요소의 병렬 지수승 연산은 $\lceil \log_2 k \rceil + \left\lceil \frac{s}{2^k} \right\rceil + k - 1$ 라운드에 수행됨이 알려져 있다.

$a^1\ a^2\ a^3\ a^4\ a^5\ a^6\ a^7$ (7항)
$a^1\ a^2\ a^3$
a^1
a^2
a^3 ①
a^4
$a^5\ a^6\ a^7$ ②
$a^8\ a^{16}\ a^{24}$ (3항)
$a^9\ a^{10}\ a^{11}\ a^{12}\ a^{13}\ a^{14}\ a^{15}\ a^{17}\ a^{18}\ a^{19}\ a^{20}\ a^{21}\ a^{22}\ a^{23}\ a^{25}\ a^{26}\ a^{27}\ a^{28}\ a^{29}\ a^{30}\ a^{31}$ (21항) ③

④ $\boxed{e_1 \cdot e_2}$ $\boxed{e_3 \cdot e_4}$ ..... $\boxed{e_{63} \cdot e_{64}}$ (←32개)
⑤ $\boxed{\times e_{65}}$ $\boxed{\times e_{66}}$ ..... $\boxed{\times e_{96}}$ (←32개)
⑥ $\boxed{\times e_{97}}$ ... $\boxed{\times e_{117}}$ $\boxed{\times e_{118}}$ $\boxed{\times e_{119}}$ (←23개) $\boxed{e_{88} \cdot e_{89}}$ .... $\boxed{e_{94} \cdot e_{95}}$ (←4개+위라운드요소1)
⑦ 2개씩 곱하기 (←14개)
⑧ (←7개)
⑨ (←3개+위라운드 요소1)
⑩ (←2개)
⑪ (←1개)

(a) Stinson의 방법

$a^1\ a^2\ a^4\ a^8\ a^{16}$
① $a^{3(=2+1)}$ $a^{5(=4+1)}$ $a^{6(=4+2)}$ $a^{9(=8+1)}$ $a^{10(=8+2)}$ $a^{12(=8+4)}$ $a^{17(=16+1)}$ $a^{18(=16+2)}$ $a^{20(=16+4)}$ $a^{24(=16+8)}$
(←10개)
$e_1 \cdot e_2$ $\boxed{e_3 \cdot e_4}$ ..... $\boxed{e_{15} \cdot e_{16}}$ (←8개)
② $a^{7(=4+3)}$ $a^{11(=8+3)}$ $a^{13(=8+5)}$ $a^{14(=8+6)}$ $a^{19(=16+3)}$ $a^{21(=16+5)}$ $a^{22(16+6)}$ $a^{25(=16+9)}$ $a^{26(=16+10)}$ $a^{28(16+12)}$ (←10개)
$\times e_{17}$ ... $\times e_{24}$ (←8개) $e_{25} \cdot e_{26}$ ... $e_{51} \cdot e_{52}$ (←14개)
③ $a^{15(=8+7)}$ $a^{23(=16+7)}$ $a^{27(=16+11)}$) $a^{29(16+13)}$ $a^{30(=16+14)}$) (←5개)
$\times e_{53}$ ... $\times e_{74}$ (←22개) $e_{75} \cdot e_{76}$ ... $e_{83} \cdot e_{84}$ (←5개)
④ $a^{31(=16+15)}$) (←1개) $\times e_{85}$ ... $\times e_{111}$ (←27개) $e_{112} \cdot e_{113}$ ... $e_{118} \cdot e_{119}$ (←4개)
⑤ (←15개+위 라운드 요소 1)
⑥ (←7개 + 1개 (위 라운드 남은 요소)
⑦ (←4개)
⑧ (←2개)
⑨ (←1개)

(b) 라운드 수 향상 가능성

그림 4. n=593, k=5 일때, Stinson 알고리즘에 의한 라운드수와 라운드 수 향상 가능성

Ⅲ. 새로운 라운드 수 향상 알고리즘

3.1 라운드 수 향상 가능성

Stinson이 제시한 알고리즘 window가 n=593, k=5 일때 진행되는 과정이 그림 4 (a)에 나타나 있다. 이 때 알고리즘 window의 단계 1은 알고리즘 d-c-small-powers로 진행되며, 단계 2는 그림 2의 방법으로 윈도우 값들을 곱하여 진행된다. 이 경우 k=5로 프로세서 수가 32로 고정되어 진행되는데, 주목할 점은 알고리즘 window의 단계 2를 수행하는 라운드 ④~⑪에 비하여 단계 1이 수행되는 라운드 ①~③까지는 유휴상태인 프로세서가 많다는 점이다. 특히 a^1, a^2, a^3을 계산하는 라운드 ①에서는 a^2 계산이 환형 쉬프트 연산만으로 구성될 수 있어서, 하나의 프로세서를 이용하여 a^3 만을 계산하면 된다. 즉, 전체 32 개의 프로세서 중 나머지 31 개의 프로세서는 유휴상태이다.

이러한 유휴상태의 프로세서들로 하여금 이후단계에서 수행되어야 할 작업을 진행시킨다면 전체 라운드 수는 줄어들 수 있다. 이를 위한 기본 가정은 지수값의 구성이 window 알고리즘의 단계 1과 단계 2를 중첩해도 진행에 무리가 없는 경우이다. 예를 들어, 지수의 윈도우 값이 31 인 경우가 나타나지 않고, 윈도우들간의 곱을 진행시에, 해당 윈도우 값이 미리 계산되어질 수 있다면, 그림 4 (b) 방법과 같이, 단계 1과 단계 2를 중첩하여 사용할 수 있다. 즉, 전체 라운드수를 11 라운드에서 9 라운드로 축소할 수 있다.

실제로, n=593인 경우와 n=512인 경우, k=5일 때 지수의 윈도우 값으로 31이 나타나지 않는 경우를 실험을 통해 조사한 결과 부록 1과 같이 대략 5~6% 임을 알 수 있었다.

3.2 제안 알고리즘

본 절에서는 단계 1과 단계 2를 양분하지 않고 단계 1 수행 중 유휴 상태에 놓인 프로세서들로 하여금 단계 2의 곱셈을 일부 미리 계산하도록 할 수 있는 알고리즘 구성 결과를 제안한다.

세부 내용은 그림 5의 알고리즘 window_cowork에 기술되어 있다. 여기서, 단계 0와 단계 $j(1\le j\le k-1)$는 알고리즘 window의 단계 1에 대응하고, 단계 $j(k\le j\le k+\left|\frac{s-1}{2^k}\right|)$는 단계 2에 대응한다. 단계 $j(1\le j\le k-1)$에서 프로세서들은 우선 윈도우

알고리즘 window_cowork (a,e,k)
// a^e을 계산한다
단계 0. k 개의 a^{2^i} $(0\le i<k)$구함
단계 $j(1\le j\le k-1)$
[윈도우값 계산] 각 단계 j 에서는, $j\le i<k$인 a^{2^i}에 대하여 단계 $j-1$에서 계산된 a^h 중 $h<2^i$를 이용하여 a^{2^i+h}를 구한다.
[윈도우들간의 곱] 유휴 프로세서들로 하여금 $(a^w)^{2^i}(0\le i\le s-1)$ 들간의 곱을 수행하도록 한다.
단계 $j(k\le j\le k+\left|\frac{s-1}{2^k}\right|)$
[윈도우들간의 곱] 곱해지지 않고 남아있는 $(a^w)^{2^i}(0\le i\le s-1)$ 항들을 그림 2의 방법으로 구한다

그림 5. 라운드 수가 개선된, 지수승 연산 a^e을 계산하는 알고리즘

값을 계산하고 나머지 프로세서들이 윈도우 간의 곱을 계산하게 된다. 이 때, 지수는 (윈도우값, 비트위치)가 윈도우값이 작은 순으로 정렬되어 있다고 하자. 이것은 각 단계에서 윈도우들간의 곱을 수행할 때 지수의 해당 윈도우 승 값이 미리 계산되어 있을 확률을 높이기 위해서이다. 지수가 윈도우 값들 순으로 정렬되어 있는 것은 부록 2에 기술한 바와 같이, 소프트웨어 구현시에는 별도의 오버헤드가 없다.

3.3 제안 알고리즘 성능분석

프로세서 수가 2^k로 고정된 경우에, 알고리즘 window_cowork의 전체 라운드 수를 분석하면 다음과 같다. 우선, 처음 $k-1$ 라운드 동안 $(k-1)\cdot 2^k$개의 프로세서를 이용할 수 있는데, 이 중 2^k-1개의 프로세서로 윈도우 값을 계산하고 나머지 $(k-1)2^k-(2^k-1)=(k-2)2^k+1$개의 프로세서로 윈도우 간의 곱셈을 수행하도록 한다. 그러면, 이후의 라운드에서 $s-((k-2)2^k+1)$개 요소의 곱을 수행하면 된다. 2^k개의 프로세서로 t개 요소를 곱하는데 필요한 라운드 수는 2.4 절에서 기술한 바와 같이, $\left|\frac{t}{2^k}\right|+k-1$이다. 그러므로, 윈도우 간의 곱을 수행할 때 각 윈도우 값이 모두 전 단계에서 계산된 것이라면, $s-((k-2)2^k+1)$개 요소의 곱셈을 수행하는데 필요한 라운드 수는 $\left|\frac{s-((k-2)2^k+1)}{2^k}\right|+k-1=\left|\frac{s-1}{2^k}\right|-(k-2)+k-1=\left|\frac{s-1}{2^k}\right|+1$이다. 즉, 제안하는 방법의 전체 라운드 수는 $(k-1)+\left|\frac{s-1}{2^k}\right|+1=\left|\frac{s-1}{2^k}\right|+k$ 이다.

Ⅳ. 성능비교

정규 기저 표현(normal bases representation)을 갖는 $GF(2^n)$ 상의 병렬 지수승 연산에 있어서, 프로세서 수가 고정된 경우의 라운드 수는 Stinson이 제안한 알고리즘을 이용하면 $|\log_2 k|+|s/2^k|+k-1$이고, 본 논문에서 제안한 알고리즘을 이용하면 $\left|\frac{s-1}{2^k}\right|+k$이다. 즉, 제안하는 알고리즘은 기존 방법에 비하여 $|\log_2 k|$ 만큼 라운드 수를 축소시킬 수 있다.

표 1에 $n=593$인 경우와 $n=512$인 경우에 대한 실제 라운드 수를 계산한 예가 나타나 있다. $k=5$인

표 1. Stinson 방법과 제안방법의 라운드 수 비교($n=593$, 512인 경우)

(a) $n=593$인 경우

k	s	2^k	Stinson방법 $\lfloor \log_2 s \rfloor + \lfloor s/2^k \rfloor + k - 1$	제안방법 $\left\lfloor \frac{s-1}{2^k} \right\rfloor + k$
2	297	4	77 (=1+75+2-1)	76 (=74+2)
3	198	8	29 (=2=25+3-1)	28 (=25+3)
4	149	16	15 (=2+10+4-1)	14 (=10+4)
5	119	32	11 (=3+4+5-1)	9 (=4+5)

(b) $n=512$인 경우

k	s	2^k	Stinson방법 $\lfloor \log_2 s \rfloor + \lfloor s/2^k \rfloor + k - 1$	제안방법 $\left\lfloor \frac{s-1}{2^k} \right\rfloor + k$
2	256	4	66 (=1+64+2-1)	66 (=64+2)
3	171	8	26 (=2+22+3-1)	25 (=22+3)
4	128	16	13 (=2+8+4-1)	12 (=8+4)
5	102	32	11 (=3+4+5-1)	9 (=4+5)

표 2. Stinson 방법과 제안방법의 라운드 수 비교 (다양한 n, k에 대한 결과)

n	k	Stinson 방법	제안 방법	n	k	Stinson 방법	제안 방법
512	2	66	66	768	2	98	98
	3	26	25		3	36	35
	4	13	12		4	17	16
	5	11	9		5	12	10
	6	10	8		6	10	8
593	2	77	76	896	2	114	114
	3	29	28		3	42	41
	4	15	14		4	19	18
	5	11	9		5	13	11
	6	10	8		6	11	9
640	2	82	82	1024	2	130	130
	3	31	30		3	47	46
	4	15	14		4	21	20
	5	11	9		5	14	12
	6	10	8		6	11	9

경우, 지수 값이 3.1절에 기술한 바와 같이 단계를 병합하는 조건이 만족되는 경우에는, $n=593$과 $n=512$인 두 경우 모두 라운드 수를 11 단계에서 2 단계 줄인 9 단계에 수행됨을 알 수 있다. 표 2에는 다양한 n에 대하여 k 값을 변화시키며 분석한 라운드 수가 비교되어 나타나 있다.

V. 결 론

본 논문에서는 $GF(2^n)$ 상의 병렬 지수승 연산에서 프로세서 수가 2^k로 제한된 경우의 라운드 수 향상을 위한 새로운 알고리즘을 제안하고 이의 성능분석을 수행한 내용을 소개하였다. 제안 방안은 지수의 값이 단계를 병합할 수 있는 조건을 가질 때 이용될 수 있는 것으로, 실험을 통하여 이 조건의 발생 경우도 확인하였다. 본 논문에서 제안한 알고리즘은 n이 커짐에 따라 속도가 더욱 중요한 의미를 갖는 지수승 연산을 이용한 암호 응용의 유용성에 기여하리라 기대된다.

부 록

1. 지수의 윈도우 값 발생 빈도

지수의 윈도우 값에 특정 패턴이 나타나는 빈도를 계산하기 위하여는 Linux 시스템 상의 openssl-0.9.7b 공개소스 중 RSA 비밀키 생성부를 수정하여 이용하였다. 임의의 비밀키를 생성하고 이를 윈도우 길이에 따라 나눈 후, 나타나는 비트값을 조사하는 방법을 사용한 결과 n=593이고 k=5일 때 지수의 윈도우 값으로 31이 나타나지 않는 경우는 조사한 100 개의 비밀키 중 5 가지 경우였다. n=512, k=5인 경우에는 100 개의 비밀키 중 6 개가 윈도우 값으로 31을 갖지 않았다.

n=593인 경우의 비밀키 값 하나와 이의 윈도우 값 발생빈도를 첨부하면 다음과 같다.

▸ 593 비트 길이의 RSA 비밀키와 이를 5 비트 윈도우로 분할했을때의 윈도우 발생빈도

C9905F113E4D6B413F51B361120EE1C1C
FA9D264E66024A6BB19965A591124CA014A
0BC618C2B6A9C235CC8C26F6758478B22B7
2F2B17E3F69BB3D23272994122O6CB25D5F
B0D4D1

윈도우값	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
발생빈도	3	3	4	3	6	9	3	3	6	6	1	3	3	4	0	3
윈도우값	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
발생빈도	5	4	3	4	6	4	2	2	6	7	5	2	4	2	4	0

2. 지수승 연산의 소프트웨어 구현

지수승 연산 a^x은 활용 경우에 따라 2 가지 부류로 나눌 수 있다. 하나는, 밑 a의 값은 고정되고 지수 x의 값이 변하는 경우이며, 다른 하나는 지수 x의 값이 고정되고 밑 a의 값이 변하는 경우이다. 전자는 일반적인 지수승 연산에서 밑을 고정시키고 지수승 연산 값을 구하는 것이 해당되며 후자는 RSA 등과 같은 공개키 암호 알고리즘에서 키인 지수가 고정되었을때 데이터인 밑을 변화시키는 것이 해당된다.

안전한 암호 알고리즘을 구성하기 위하여 지수 x는 256 또는 512 비트 이상을 이용하게 된다. 보통의 경우 기본 정수형은 32 비트 (4바이트)인 관계로, 지수는 정수형의 배열로 구성되게 된다. 공개 소프트웨어인 openssl의 경우도 기본 데이터형(C 언어의 경우 long 형)의 배열로 이들을 표현하고 있다.

지수승 연산의 코드 구성은 이러한 정해진 데이터형을 이용하여 지수 값 등을 저장하고 또한 정해진 데이터 형을 대상으로 지수승 연산 코드를 구성하면 된다. 3 장에서 제안한 알고리즘은 지수가 값 및 위치의 정보로 구성된 배열을 이용한다.

이렇게 기존의 방법들이 사용하는 지수의 데이터형과는 다른 것을 사용하더라도, 소프트웨어 및 하드웨어 등의 구현에서는 데이터형을 따르는 연산 코드를 구성하면 되므로 연산수행시의 추가오버헤드는 없다.

참 고 문 헌

[1] M.Lee, Y.Kim, K.Park, and Y.Cho, "Efficient Parallel Exponentiation in GF(q^n) using Normal Basis Representations," *Journal of Algorithms*, Academic Press, Inc., Accepted for Publication.

[2] G.B. Agnew, R.C. Mullin, S.A. Vanstone, "Fast exponentiation in GF(2^n)," *Advances in Cryptology EUROCRYPT '88*, Lecture Notes in Computer Science, vol. 330, pp. 251-255, 1988.

[3] D.R. Stinson, "Some Observations on parallel algorithms for fast exponentiation in GF(2^n)," *SIAM Journal on Computing*, vol. 19, no. 4, pp. 711-717, August, 1990.

[4] Joachim von zur Gathen, "Processor-efficient exponentiation in finite fields," *Computational Complexity*, vol. 1, pp. 360-394, 1991.

[5] Joachim von zur Gathen, "Processor-efficient exponentiation in finite fields," *Information Processing Letters*, vol. 41, pp. 81-86, 1992.

[6] Daniel M. Gordon, "A survey of fast exponentiation methods," *Journal of Algorithms*, vol. 27, pp. 129-146, 1998.

[7] C.K.Koc, T.Acar, "Montgomery multiplication in GF(2^k)," *Designs, Codes and Cryptography*, vol. 14, no. 1, pp. 57-69, April, 1998.

[8] A. Haalbutogullari, C.K.Koc, "Parallel multiplication in GF(2^k) using polynomial residue arithmetic," *Designs, Codes and Cryptography*, vol. 20, no. 2, pp. 155-173, June 2000.

[9] Rudolf Lidl, *Introduction to finite fields and their applications*, Cambridge University Press, London, 1994.

〈著者紹介〉

김 윤 정 (Yoonjeong Kim) 종신회원

1991년: 서울대학교 컴퓨터공학과 졸업(학사)
1993년: 서울대학교 대학원 컴퓨터학과 졸업(석사)
2000년: 서울대학교 대학원 전기컴퓨터공학부 졸업(박사)
2000~2001년: (주)엔써커뮤니티 제품개발연구소 차장
2001~2002년: (주)데이타게이트 인터내셔널 보안기술연구소 차장
2002~현재: 서울여자대학교 정보통신대학 정보통신공학부 교수
〈관심분야〉 암호학, 시스템 보안, 암호 응용