

효율적인 초기 블록 테이블 구축을 통한 모듈러 멱승 계산

김지은 김동규

부산대학교 컴퓨터공학과

(jekim, dkkim)@islab.ce.pusan.ac.kr

Computing Modular Exponentiations based on Constructing Efficient Initial Block Tables

Jieun Kim Dong Kyue Kim

Dept. of Computer Engineering

Pusan National University, Busan 609-735, Korea

요약

모듈러 멱승은 주어진 값 X, E, N 에 대하여 $X^E \bmod N$ 으로 정의된다. 모듈러 멱승은 대부분의 공개키 암호시스템과 전자서명에 사용되므로, 이 연산을 빠르게 수행하는 문제는 암호학 분야에서 중요하게 연구되고 있다. 모듈러 멱승을 계산하기 위해 가장 많이 사용되는 효율적인 알고리즘은 VLNW방법이다. 그러나 이 방법은 d, q 인자를 고정시키기 때문에 제한적인 결과를 제공한다. 일반화된 그래프 모델은 VLNW방법의 q 인자를 제거하여 그 성능을 향상시켰다. 그러나 d 인자는 여전히 VLNW방법과 동일하게 고정되어 있다. d 인자는 초기 블록 테이블을 결정하는 인자로써, 이것을 확장할 경우 전 처리 시간이 기하급수적으로 증가하게 된다. 그러나, 본 논문은 전 처리 시간의 기하급수적인 증가를 저지하면서도 d 인자를 확장하기 위해서 초기 블록 테이블을 만드는 몇 가지 효율적인 방법을 제시한다. 이 방법은 VLNW 방법과 일반화된 그래프 모델보다도 효율적이며, 그 성능은 실험을 통하여 검증하였다.

1. 서 론

전자 상거래, 인터넷 뱅킹처럼 인터넷상으로 중요한 정보들을 교환하는 일들이 많아지면서 암호학의 중요성이 부각되었고, 암호화 과정의 빠른 구현이 요구되었다. 특히, RSA 암호 시스템[2], Elgamal 서명 시스템[3], DSS[4]와 같이 널리 사용되고 있는 암호화 알고리즘의 핵심적인 연산이다. 암호화 과정을 빠르게 수행하기 위해서는 핵심적인 연산이 모듈러 멱승을 빠르게 계산하는 것이 중요한 문제이다.

모듈러 멱승은 양수 M, E, N 에 대하여 $M^E \bmod N$ 을 계산하는 연산 문제이다[1]. 모듈러 멱승을 계산하는 알고리즘은 크게 두 가지로 나눌 수 있다. 한 가지는 $M^E \bmod N$ 을 계산할 때 M 을 곱하는 과정을 빠르게 계산하는 방법이다[10, 11, 12]. 다른 한 가지는 $M^E \bmod N$ 에 필요한 곱셈의 수를 줄이는 것이다[1, 5, 6, 7, 8, 9]. 본 논문은 후자의 방향으로 이 문제에 접근한다. 후자의 방향으로 개발된 알고리즘들에는 이진 방법[1], M_ary 방법[1], 슬라이딩 윈도우 방법[8], 덧셈 사슬 방법이 있다. 이 중에서 가장 많이 사용되는 것이 슬라이딩 윈도우 방법

이다.

슬라이딩 윈도우 방법은 VLNW방법과 CLNW방법으로 나누어지는데, VLNW방법이 더 효율적이다. VLNW 방법은 d, q 인자를 고정시켜서 알고리즘을 진행하는 것이 핵심적인 내용이다. 그러나 VLNW방법의 단점은 d, q 인자를 고정함으로써 다른 결과들을 고려할 수 있는 여지가 없어 더 좋은 결과들을 얻을 수 있다는 것이다. VLNW방법의 한계를 뛰어 넘어, 다른 결과들을 고려할 수 있게 한 것이 일반화된 그래프 모델[14]이다. 이 방법은 q 인자를 제거하였기 때문에 고정된 d 인자에 대해서는 최적의 결과를 제공한다. 그러나 d 인자는 여전히 VLNW 방법과 동일하게 고정되어 있는 것이 단점이다. d 인자는 초기 블록 테이블을 결정해주기 때문에 q 인자처럼 제거 할 수 없으며, 확장하는 방법을 고안해야 한다. 그러나 d 인자를 확장하게 되면 초기 블록 테이블은 기하급수적으로 증가하며, 이것은 곧 전 처리 시간이 기하급수적으로 증가한다는 것과 같다. 이렇게 되면 d 인자를 확장하는 것이 아무런 의미가 없게 된다. 그래서 본 논문에서는 전 처리 시간의 증가를 저지하면서 d 인자를 확장할 수 있는 방법을 제시한다. 이 방법은 기존의 VLNW방법과 일반

화된 그래프 모델보다 성능이 더 좋으며, 그 타당성은 랜덤한 600개의 데이터에 대한 실험 결과로 제시한다.

본 논문은 총 5장으로 구성되어 있는데, 2장은 기존 알고리즘에 대한 분석이며, 3장은 그래프 모델에 대한 설명하고 4장은 초기 블록 테이블을 설정하는 방법을 제시하며 5장은 본 논문이 제시하는 알고리즘과 기존 알고리즘들을 비교한 실험결과를 제시한다.

2. 기존 알고리즘 분석

본 장에서는 모듈러 역승을 계산하는 몇 가지 알고리즘들을 분석한다. 모듈러 역승 알고리즘은 두 가지 방향으로 연구되는데, 한 가지는 한번의 곱셈을 빠르게 수행하는 것이고 다른 한 가지는 총 연산의 수를 줄이는 것이다. 여기서는 후자 방향의 알고리즘을 살펴본다.

먼저, 가장 간단한 알고리즘인 이진 방법[1]이다. 이 방법은 이진수로 표현된 지수 E 를 원쪽부터 한 비트씩 읽어가면서 곱셈과 제곱 연산을 수행한다. 이진 방법은 평균적으로 $1.5(\log E - 1)$ 번의 곱셈을 수행한다.

두 번째는 이진 방법을 일반화한 M -ary 방법[1]이다. 이 방법은 M 개의 초기 블록을 미리 계산한다. 초기 블록의 최장 길이는 $\log_2 M$ 이다. 알고리즘은 지수 E 를 $\log_2 M$ 씩 읽어가면서, 초기 블록에 계산한 값들 중 방금 읽은 $\log_2 M$ 에 해당되는 값을 곱해주면서 진행된다. 즉, $\log_2 M$ 비트를 한번 읽을 때 한번의 곱셈이 필요하게 된다.

세 번째는 M -ary 방법을 효율적으로 수정한 슬라이딩 윈도우(Sliding Window)방법[8]이다. 이 방법은 초기 블록 테이블에 시작과 끝이 '1'인 블록들만 저장한다. 실제로 저장되는 블록의 수를 M -ary 방법과 비교하면 $1/2$ 적고, 필요한 연산 수도 적다. 다음 과정은 지수 E 를 ZW (Zero Window)와 NW (NonZero Window)로 분할하면서 $M^E \bmod N$ 을 계산하는 것이다. NW 는 초기 블록 테이블에 있어서 한번의 곱셈으로 계산되고, ZW 는 곱셈이 수행되지 않는다. M -ary 방법이 만드는 윈도우의 수에 비하면 NW 의 수는 적으므로, 곱셈 수도 적음을 알 수 있다.

슬라이딩 윈도우 방법은 두 가지 방식이 있는데, 지수 E 를 어떤 방식으로 분할하는가에 따라 나누어진다. 첫 번째 방법은 CLNW(Constant Length Nonzero Window)이며, 특징은 ZW 의 길이가 가변적이고, NW 의 길이는 일정하다는 것이다. 알고리즘은 지수 E 를 한 방향으로 읽어가면서 수행된다.

- ZW (Zero Window) : 한 비트를 확인한다. '0'이면 ZW 에 머물러 있고, '1'이면 NW 를 시작한다.
- NW (Nonzero Window) : d 비트를 읽는 동안 NW 에 머물러 있다. 다음 한 비트를 읽어 '0'이면 ZW 를 시작하고, '1'이면 NW 를 시작한다.

$$\text{예) } E = \underline{\underline{111}} \underline{\underline{00}} \underline{\underline{101}} \underline{\underline{0}} \underline{\underline{001}} \quad (d=3)$$

두 번째 방법은 VLNW (Variable Length Nonzero

Window)방법이다. 이 방법은 ZW 와 NW 의 길이를 모두 가변적이라 하는 방법이다. 이 알고리즘을 기술하기 위해서는 몇 가지 인자(parameter)가 필요하다.

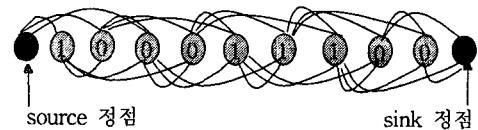
- d : NW 의 최대 길이
 - q : 현재의 NW 를 끝내는 최소한의 영의 수
 - k 와 r : $d = 1 + kq + r$ ($1 \leq r < q$)를 만족하는 정수
- 예) $E = \underline{\underline{101}} \underline{\underline{10111}} \underline{\underline{0000}} \underline{\underline{11}} \underline{\underline{00}} \underline{\underline{111}} \underline{\underline{00}} \underline{\underline{10011}}$ ($d=5$, $d=3$)

위의 방법들을 이용하여 모듈러 역승을 계산할 때 필요한 연산 수는 다음 세 가지의 합이다. 첫 번째는 초기 블록들을 계산하는 데 필요한 연산 수이고, 두 번째는 NW 의 개수에서 1을 뺀 수이며, 마지막으로 전체 길이에서 첫 번째 윈도우의 길이를 뺀 수이다.

3 일반화된 그래프 모델

본 장은 기존의 알고리즘들을 일반화하는 그래프 모델[14]을 소개한다. 일반적으로 그래프는 정점의 집합 V 와 간선의 집합 E 로 정의된다. 간선의 가중 값은 가중 값 함수로 정의된다. 일반화된 그래프 모델은 모듈러 역승을 문제를 다음과 같이 모델링 한다.

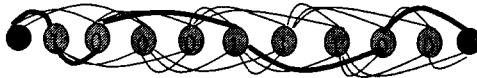
- $V = \{ v \mid \text{이진수로 표현된 지수 } E \text{의 각 bit} \text{로 인덱스를 가진 정점, source 정점, sink 정점} \}$
- $E = \{ e=(v,u) \mid | \text{정점 } v \text{의 인덱스} - \text{정점 } u \text{의 인덱스} | \leq d, \forall v \in E \}$
- $W(e) = \{ w(e) \mid e \text{가 표현하는 윈도우를 계산하는데 필요한 곱셉의 수}, \forall e \in E \}$



[그림 1] $d=3$ 때 모델링된 그래프

[그림 1]은 정의에 따라 그래프를 모델링한 것이다. 각 정점마다 3개의 간선의 가진다. 초기 블록 테이블이 d 의 값을 3으로 정하고 만들었을 때, 간선의 가중 값은 '1'로 모두 동일하다.

그래프에는 수많은 경로(path)가 존재한다. 그 중에서 source 정점에서 시작해서 sink정점에서 끝나는 각각의 경로가 있다. 이것은 $M^E \bmod N$ 을 계산하는 곱셈의 순서를 나타낸다. 그래프에는 일어날 수 있는 모든 경우의 곱셈의 순서는 각각의 경로로 모두 표현되어 있다. 각각의 경로가 가지는 가중 값의 합은 초기 연산을 제외하고, 모듈러 역승을 계산하는데 필요한 곱셈의 수를 나타낸다. 모든 경로 중에서 가중 값의 합이 가장 작은 경로가 곧 최소의 곱셈 수를 가지는 곱셈의 순서이다. 이것을 구하기 위해서 그래프에 최단 경로를 구하는 알고리즘[13]을 적용하면 된다. 최소의 곱셈의 순서를 가지는 경로는 [그림 2]의 굵은 선으로 표현된 경로이다.



[그림 2] 그래프의 최단 경로를 적용한 예

이 그래프는 단방향 그래프이므로 DAG(Directed Acyclic Graphs)의 최단 경로 구하는 알고리즘을 적용할 수 있어서 시간 복잡도는 $O(d \times \text{비트수})$ 가 된다.

4. 초기 블록 테이블과 간선의 가중 값

본 장은 초기 블록 테이블을 생성하는 새로운 방법을 소개한다. 사용된 방법은 크게 세 가지로 나누어진다. 첫 번째는 기존의 알고리즘의 방법을 그대로 반영하여 기본 블록들을 저장하는 것이고, 두 번째는 지수 E의 접두사(Prefix)를 블록으로 추가하는 것이고, 세 번째는 덧셈 사슬을 적용하여 간선에 변화를 줌으로써 새로운 블록들을 추가하는 것이다.

4.1 초기 블록 테이블의 기본 블록

초기 블록 테이블의 기본 블록들은 슬라이딩 윈도우 방법의 초기 블록 테이블과 동일한 블록들이다. 슬라이딩 윈도우 방법은 M_ary 방법을 초기 블록 테이블에서 블록의 시작과 끝이 '1'인 블록들과 중간 계산과정을 위해서 필요한 블록들을 저장하는 것이다. 초기 블록 테이블의 기본 블록들은 [표 1]과 같은 방식으로 만들어진다.

예) d=3

bits	w	M^w
001	1	M
010	2	$M \cdot M = M^2$
011	3	$M \cdot M^2 = M^3$
101	5	$M^3 \cdot M^2 = M_5$
111	7	$M^5 \cdot M^2 = M'$

[표 1] 기본 블록 테이블

4.2 접두사 블록 확장

모듈러 멱승을 계산하는 중간 과정에 얻어진 결과물은 뒤에서 다시 사용될 수도 있기 때문에 중간 결과물을 초기 블록 테이블에 저장하는 방법이다. 접두사 블록이라고 이름이 붙혀진 것은 얻어지는 중간 결과물은 항상 지수 E의 접두사이기 때문이다. 모듈러 멱승은 지수 E의 왼쪽 끝에서부터 시작하여 오른쪽 방향으로 계산하여 지수 E의 부분적인 계산 결과가 중간 결과물로 나올 수 없고, 지수 E의 접두사가 중간 결과물로 나올 수밖에 없다.

지수 E의 접두사를 저장할 때에 지수 E의 모든 접두사를 저장할 필요는 없다. 200개의 랜덤한 데이터에 대해서 실험을 한 결과 접두사가 다시 나타날 수 있는 데이터의 길이는 $5\log N$ (N은 지수 E의 길이)을 넘지 않을음을 알 수 있었다. 접두사의 길이 $5\log N$ 까지는 CLNW방법으로 계산하여 접두사 블록들을 초기 블록 테이블에 저장한다. [표 2]는 추가되는 접두사 블록들을 나타낸다.

예) E=..110101111100101010011...(d=3)

bits	w	M^w
110	6	$M^3 \cdot M^3 = M^6$
1100	12	$M^3 \cdot M^6 = M^{12}$
11000	24	$M^{12} \cdot M^{12} = M^{24}$
110000	48	$M^{24} \cdot M^{24} = M^{48}$
110101	53	$M^{48} \cdot M^5 = M^{53}$
1101010	106	$M^{53} \cdot M^{106} = M^{106}$
11010100	212	$M^{106} \cdot M^{212} = M^{212}$
110101000	424	$M^{212} \cdot M^{212} = M^{424}$
110101111	431	$M^{424} \cdot M^7 = M^{431}$

[표 2] 접두사가 추가된 초기 블록 테이블

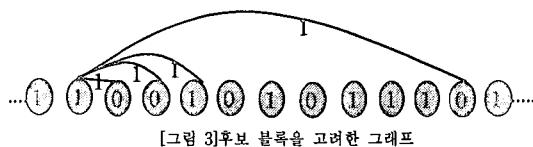
4.3 덧셈 사슬을 이용한 블록 확장

덧셈 사슬이란 곱셈들의 순서를 지수들의 덧셈을 이용해서 나타낸 수열이다. 초기 블록 테이블에 있는 블록들을 덧셈사슬로 연결하면 새로운 블록을 만들 수 있다. 앞으로 이런 블록을 후보 블록이라고 부를 것이다. 후보 블록들은 초기 블록 테이블에는 없지만 마치 있는 것처럼 그래프의 간선을 만들 때 사용한다. 후보 블록으로 만든 간선에도 가중값을 주어야 하는데 가중값을 주기 위해서는 두 가지 요소를 고려해야 한다. 한 가지는 후보 블록을 초기 블록 테이블에 추가하기 위해 필요한 연산 수이고 다른 한 가지는 전체 연산 수에 미치는 궁정적인 영향인데, 두 가지 요소를 모두 반영하는 값을 찾는 것은 어려운 문제이다. 이 문제를 단순화하여 유용한 후보 블록들을 초기 블록 테이블에 추가하기 위해서 다음과 같은 방법을 제시하였다.

- ① 초기 블록 테이블의 블록들과 덧셈 사슬이 '1'인 후보 블록들만 간선을 만들고, 이 간선들의 가중값을 모두 동일하게 '1'로 정한다.
- ② 일반화된 그래프 모델을 이용하여 모듈러 멱승을 계산하는 곱셈의 순서를 구한다.
- ③ 사용된 각 후보 블록들이 총 곱셈의 수를 줄이는데 공헌한 블록들만 초기 블록 테이블에 저장한다. 후보 블록이 ((블록 길이/d) \times (블록이 사용된 횟수))값이 3이상일 때 총 곱셈의 수를 줄이는데 공헌했다고 판단한다. 후보 블록이 3번 이상 사용되면 총 곱셈의 수를 줄이는 데 공헌했다고 판단할 수 있는데, 길이가 긴 블록인 경우 사용된 횟수에 비해 끼치는 영향을 크기 때문에, 블록 길이와 사용된 횟수 모두 고려하여 수식을 만들었다.
- ④ 초기 블록 테이블로만 그래프를 생성하고 모듈러 멱승을 계산하는 곱셈의 순서를 구한다. 이 때의 초기 블록 테이블에 필요한 곱셈 수와 모듈러 멱승을 계산하는 필요한 곱셈의 합이 총 곱셈 수이다.
- ⑤ ①~④의 과정을 반복하여 ④과정에서 얻어진 총 곱셈 수가 더 이상 줄어지지 않을 때 반복을 멈춘다.

[그림 3]은 [표2]의 초기 블록 테이블과 후보 블록들로 만들어진 그래프를 나타내는데, 다른 정점들에 대한 간선은 생략하고 하나의 정점에 대해서만 표현한다. [그림 3]의 한 정점에서 나타난 후보 블록은 “110101000”과 “110101111”的 덧셈 사슬로 만들어지는 블록 “1001010111”이다.

예) $E = \dots 1\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 1\dots$ ($d=3$)



4. 실험 결과

200개의 랜덤 데이터에 대해서 VLNW방법과 그래프 모델을 이용한 방법, 본 논문에서 제시한 새로운 방법을 비교하였다. 결과는 다음과 같다.

	512 bit (d=5)	1024 bit (d=6)	2048 bit (d=7)
VLNW	607	1195	2363
q를 최적화한 방법	604	1192	2361
새로운 방법	601	1189	2354

[표 3]

[표 2]는 d 를 각각의 방법에서 최적의 결과를 얻는 값으로 설정하였다. q 를 최적화한 방법이 일반화된 그래프 모델인데, 이 방법은 고정된 d 에 대해서 최적인 결과를 제공한다. 그래서 VLNW보다는 효율적이었다. 그러나 q 를 최적화한 방법은 초기 블록 테이블에 대해서는 VLNW와 동일한 방법으로 설정하고 있기 때문에 d 를 확장한 본 논문의 방법이 q 를 최적화한 방법보다 더 효율적인 결과를 얻을 수 있었다.

5. 결론

모듈러 연산을 빠르게 계산하는 알고리즘을 개발하는 것은 암호학 분야에서 중요한 문제이다. 많은 알고리즘들이 개발되어 왔는데, 그중에서 가장 많이 사용되는 것이 VLNW방법이다. 그러나 VLNW방법은 d , q 인자를 고정시키기 때문에 최적의 곱셈결과를 찾기기에 한계가 있다. 그래서 VLNW방법의 q 인자를 제거하면서 성능을 개선한 것이 일반화된 그래프 모델을 이용한 방법이다. 그러나 이 방법 역시 VLNW방법과 동일하게 d 인자를 고정시키고 있다. 본 논문에서는 일반화된 그래프 모델을 기반으로 하여 d 인자를 확장하기 위한 새로운 방법을 제시한다. d 인자는 초기 블록 테이블을 결정하는 인자이다. d 를 확장한다는 것을 초기 블록 테이블의 블록들의 길이를 확장하는 것이다. 블록을 확장하기 위해 본 논문에서 제시한 방법과 그 효과는 다음과 같다.

첫 번째는 지수 E 의 접두사를 저장하는 것인데, 이 방법은 블록을 계산하기 위한 연산이 필요 없기 때문에 전처리 시간을 증가 시키지 않으면서 d 인자를 확장시킨다. 두 번째는 곱셈 수를 줄이는 유용한 후보 블록들을 초기 블록 테이블에 있는 블록들의 덧셈 사슬을 이용하여 만들어 주는 것이다. 이 방법은 후보 블록을 계산하는데 필요한 전 처리 연산이 한번만 필요하므로, 후보 블록을 추가함으로써 곱셈의 수가 줄어드는 횟수가 더 많다. 즉, 총 곱셈 수를 줄여줄 수 있다.

본 논문에 제시된 방법은 VLNW방법보다 효율적이며, 일반화된 그래프 모델을 이용한 방법보다도 효율적인 실험 결과를 보인다.

[참고문헌]

- [1] D.E Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, Addison-Wesley, U.S.A(1981)
- [2] R.L Rivest, A. Shamir and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, *Communications of the ACM* 21(2) 120-126 (1978)
- [3] T. Elgamal, A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Transactions on Information Theory* 31(4)469-472(1985)
- [4] National Institute for Standards and Technology, Digital Signature Standard(DSS), *Federal Register* 56 169(1991)
- [5] J. Bos and M. Coster, Addition chain heuristics, In *Proc. Crypto'89*, Lecture Notes in Computer Science Vol. 435, pp.400-407, Springer-Verlag,(1990)
- [6] P.Downey, B. Leong and R.Sethi, Computing sequences with addition chains, *SIAM Journal on Computing* 10(3) 638-646 (1981)
- [7] C. K. Koc, "High-radix and bit recoding techniques for modular exponentiation", *International Journal of Computer Mathematics*, 40(3+4) pp.139-156 (1991)
- [8] C.K. Koc, Analysis of sliding window techniques for exponentiation, *Computers & Mathematics with Applications* 30(10)17-247(1995)
- [9] S.Hong, S. Oh, and H. Yoon, New modular multiplication algorithms for fast modular exponentiation, In *Proc. Eurocrypt'96* Lecture Notes in Computer Science Vol. 1070, pp.166-177, Springer-Verlag,(1996)
- [10] P.L.Montgomery, Modular multiplication without trial division, *Mathematics of Computation* 44(170) 519-521(1985)
- [11] H. Morita, A fast modular-multiplication algorithm based on a higher radix, In *Proc. Crypto'89*, Lecture Notes in Computer Science Vol. 435, pp.387-399, Springer-Verlag,(1990)
- [12] C.D.Walter, Faster modular multiplication by operand scaling, In *Proc. Crypto'91*, Lecture Notes in Computer Science Vol.596,pp.313-323, Springer-Verlag,(1992)
- [13] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, *Introduction to Algorithms* pp.514-531(1990)
- [14] 김지은, 김동규 모듈러 연산을 계산하는 일반화된 모델* 멀티미디어 춘계 학술 대회 6권 1호 pp.1-4 (2003)