다양한 운영체제상에서의 디지틀 서명 기본 알고리즘에 대한 구현 및 성능 비교 분석

최영철, 백동준, 김승주, 원동호

성균관대학교 정보공학과

Implementation and Benchmaking Test for the Cryptographic Library on Several OSs

Youngchul Choi, Dongjun Baek, Seungjoo Kim and Dongho Won

Department of Information Engineering
Sung Kyun Kwan University

요 약

디지틀 서명을 구현하기 위해서는 몇 가지 중요한 요소 알고리즘들이 필요하게 된다. 본 논문에서는 이러한 구성 요소 알고리즘들에 대해 고찰해 보며, 이러한 알고리즘들을 여러 가지 운영체제상에서 구현하여, 각 운영체제의 특성에 따른 구현결과를 비교해본다. 이러한 구현결과는 디지틀 서명 및 여러 가지 암호화 기술을 실제로 구현하는 경우 최적화가 가능한 운영체제 및 컴파일러 선택에 좋은 기준을 제공해 줄 수 있을 것이다.

1. 서 론

근래에 들어 인터넷을 비롯한 각종 컴퓨터 통신망의 발달로 인하여 인류는 최신의 정보 및 중요정보를 수초내에 얻을 수 있는 환경에 접해 있으며, 여러 가지의 문서처리나 상거래 또한 컴퓨터 통신망을 이용하는 단계에 접어들게 되었다. 이러한 컴퓨터 통신망의 이용은 사용자로 하여금 신속성, 정확성, 용이성 이라는 세 가지의 장점을 제공해 주게 되었지만, 반면에 디지틀 정보의 위조, 무단절취, 훼손 및 파괴라는 정보화의 역기능을 등장시키게 되었다. 이러한 정보화의 역기능 방지를 위해서 여러 가지 학문이 연구되고 있으며, 그 중의 하나로서 암호학에서의 디지틀 서명을 들 수 있다.

1976년 Diffie-Hellman[1]에 의해 발표된 공개키 암호 시스템을 기반으로 디지틀 서명에 관한 연구는 다방면으로 진행되어져 왔으며, 현재에는 여러 가지 방식의 디지틀 서명방식들이 있으며 또한 특수한 목적을 위한 디지틀 서명방식들도 많이 연구되고 있는 상황이다. 이러한 여러 가지 디지틀 서명방식들은 일반적으로 풀기 어려운 수학적 문제에 그 기반을 두고 있는데, 큰 수의 소인수 분해 문제의 어려움, 이산대수 문제의 어려움, 이차 및 고차 잉여류 문제의 어려움 등을 들 수가 있다. 이러한 수학적 배경을 가진 디지틀 서명들을 실제 환경에서 구현하기 위해서는 여러 가지 문제점들을 해결해야 하며, 또한 실시간 처리를 위한 다정도 연산의 효과적인 구현 및 각종의 컴퓨터가 사용하고 있는 여러 가지 운영체제와의 효율성 및 속도비례 관계 등에 대한 고려도

해야될 것이다.

본 논문에서는 이러한 여러 가지 연산방식들 중에서도 디지틀 서명 구현에 필요한 모듈라 연산에 관한 알고리즘들에 대해 고찰하며, 또한 그것들을 여러 가지 운영체제들상에서 C언어 및 C++ 언어로 구현하여 각 속도의 성능 비교 분석을 하였다. 이러한 비교 분석은 디지틀 서명을 실제 환경에서 구현하는 경우, 각종 운영체제 및 컴파일러들의 특성에 따른 디지틀 서명의 성능 및 효율성을 예측 가능케 하여 디지틀 서명 구현을 위한 적절한 환경 선택에 많은 도움을 줄 수 있으리라 사료된다.

2. 성능평가 방법

디지틀 서명을 구현하는 경우 일반적으로 그것의 안전성을 위해 512비트 이상의 모듈라 멱승연산을 필요로 하게 된다. 즉, 디지틀 서명의 구현을 위해서는 512비트의 큰 수를 표현할 수 있는 방식이 필요하게 되는데 일반적으로 b를 기수로 하는 n자리수 형태로 다음과 같이 표현하게 된다.

$$A = \sum_{i=0}^{n-1} A_i b^i$$

$$A_i \in [0, b-1] (i = 0, 1, 2, \ldots, n-1)$$

여기서, b는 일반적인 십진수의 경우 10으로 사용되는 반면, 512비트의 큰 수를 나타내야 하는 경우에는 보통 구현할 컴퓨터의 하드웨어 특성과 사용할 언어(컴파일러)에 따라 선택되어져야 한다. 본 논문에서는 MS-DOS 6.0, Windows3.1, Windows95, 그리고 UNIX 운영체제 환경 아래에서 TURBO-C 2.0, MS VISUAL C++ 1.52, MS VISUAL C++ 4.0, ANSI C를 사용하여 각 알고리즘들을 구현하였는데. 16비트 환경에서는 b를 2^{16}으로 32비트 환경에서는 b를 2^{32}로 설정하였다.

본 논문이하의 내용중에 나타나는 덧셈, 뺄셈, 곱셈, 나눗셈은 다정도 연산(즉 b가 2^{16}이거나 2^{32}인 수의 연산)에서의 사칙연산을 말하는 것이며, 일반적으로 컴퓨터에서 지원하게 되는 16비트 또는 32비트 사칙연산은 기본적인 덧셈, 뺄셈, 곱셈, 나눗셈이라고 말할 것이다.

본 논문에서는 여러 가지 운영체제들 및 컴파일러들을 사용하여 디지틀 서명에 필요한 기본적인 알고리즘들 구현하여 평가하게 되는데, 그러한 평가방법 및 분류를 표로 나타내면 <표 1>과 같다.

기존의 암호 알고리즘들에 대한 성능평가 논문들은 일반적으로 DOS환경에서의 각 암호 알고리즘들에 대한 유형별 평가가 주를 이루었다. 본 논문은 <표 1>에서 보는 바와 같이 그러한 관점을 바꾸어 급변하는 컴퓨터 사용환경에 따른 운영체제 및 컴파일러 유형에 따른 성능 분석에 그 초점을 맞추었다.

우리는 우선 알고리즘 구현 환경을 16비트와 32비트 환경으로 구분하여 PC 및 workstaion에서 테스트를 실시하였으며, PC는 인텔 펜티엄 100MHZ를 workstaion은 SPARCstation과 호환을 이루는 Unistation 20을 사용하였다. 여기서 DOS와 Windows 3.1에서는 실제 프로그램 구현시 필요로 하는 메모리 모델 설정이 알고리즘의 구현에 어떤 영향을 미치는 가를 조사하기 위해 메모리 모델을 소형(Small)과 대형(Large)으로 구분하여 성능비교를 하였다. 또한 각 알고리즘들을 실질적으로 윈도우즈 환경에서 많이 이용되는 DLL(Dynamic Link Library) 형태로 바꾸어 그것의 성능을 테스트 하였다.

3. 디지틀 서명의 구현 요소

본 절에서는 디지틀 서명의 구현에 필요한 여러 가지 알고리즘 및 그 종류들을 고찰할 것이다.

<표 1 : 사용되는 운영체제 및 컴파일러들의 사양 및 환경>

운영체제	메모리모델/DLL	기수 b의 단위	컴파일러	컴퓨터 기종	비고
MS-DOS 6.0	Small	2^{16}	TURBO-C	Intel Pentium 100MHZ	
	Large				
Windows 3.1	Medium	2^{16}	MS VISUAL C++ Ver. 1.52		
	Large				
	DLL				
Windows 95	None	2^{32}	MS VISUAL C++ Ver. 4.0		
	DLL				
UNIX (Solaris 2.3)	None	2^{32}	ANSI C	Unistation 20	

※ 상기표에서 Windows 3.1은 DOS를 기반 운영체제로 두기 때문에 엄밀한 의미에서는 운영체제라고 할 수 없지만 IBM PC 호환계열에서의 첫 Windows 환경이므로 운영체제 분류에 포함시켰다.

3. 1 다정도 연산

3. 1. 1 덧셈 및 뺄셈 연산

n자리의 수 A,B의 덧셈은 A와 B의 각 자리수끼리 더하여 C의 해당 자리수에 저장하는 방식을 취하며, 매 덧셈마다의 캐리(carry)는 다음 자리로 넘겨준다. 만약, A와 B의 덧셈이 b^n 이상이면, n번째 자리수의 덧셈에서는 항상 캐리가 발생하게 되며, C의 n번째 자리수에는 $A+B-b^n$을 저장하고 캐리 1을 되돌려 주게 된다.

n자리의 수 A,B의 뺄셈은 보수(complement)연산을 이용한다. r진법의 보수 연산은 (r-1)의 보수와 r의 보수 2가지가 존재하게 된다. 컴퓨터에서의 수체계는 이진법을 기반으로 하고 있기 때문에, A,B의 뺄셈에서는 1의 보수나 2의 보수를 이용하면 된다. 여기서 우리는 2의 보수를 이용하여 뺄셈을 수행하는데, 그 이유는 1의 보수를 이용하게 되면 0이나 1로만 이루어진 이진수가 두 가지 의미를 갖게 된다는 단점과 캐리 문제를 해결해 주어야 한다는 단점이 생기기 때문이다.

A-B를 계산하기 위해서는 먼저 B에 대한 그의 보수를 취하고, 그 결과 $\overline{B}+1$을 A에 더해주면 된다. 이 때 A가 B보다 크거나 같다면 항상 캐리가 발생하게 되며 이 캐리는 무시되어진다.

상기의 알고리즘들중 덧셈 알고리즘은 2n번의 덧셈연산, n번의 MOD 연산, n번의 DIV 연산이 필요하게 되며, 뺄셈 알고리즘은 덧셈 알고리즘과 같은 연산횟수를 가지며 더불어 n번의 비트반전 연산이 필요하게 된다.

3. 1. 2 곱셈 연산

n자리의 수 A,B의 곱셈을 하기 위해서는 일반적으로 세가지의 방법이 있을 수 있다. 첫 번째는 A×B의 연산을 행단위로 하는 방식이며, 두 번째는 각 열단위로 계산을 하는 방식이며, 세 번째는 n이 2의 배수인 경우에 취할 수 있는 Karatsuba의 곱셈방법[2]이 있다. 본 논문에서는 세 가지의 방법중에서 첫 번째 방법을 이용하여 곱셈을 수행하였다.

행단위로 계산하는 곱셈은 일반적으로 다음과 같이 나타낼 수 있다.

$$A\times B=\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}A_iB_jb^{i+j}$$

3. 1. 3 제곱 연산

n자리수를 가진 A를 제곱하는 방법은 세 가지로 나누어 볼 수 있다. 첫 번째는 곱셈 연산을 이용하는 방법이고, 두 번째는 제곱과정을 세 가지 단계로 나누어 계산하는 방법이고, 세 번째는 곱셈 연산에서 적용되었던 Karatsuba의 연산방법을 적용하는 것이다. 본 논문에서 사용한 제곱 연산 방법은 두 번째 방법이다. 두 번째 방법은 다음과 같은 식을 이용하게 되며, $A_i^2b^{2i}$, $A_iA_jb^{i+j}$, 2의 곱셈연산을 루프문을 사용하여 순차적으로 실행한다.

$$[\sum_{i=0}^{n-1}A_ib^i]^2=\sum_{i=0}^{n-1}A_i^2b^{2i}+2(\sum_{i<j}A_iA_jb^{i+j})$$

3. 2 모듈라 감소 알고리즘

모듈라 감소 알고리즘의 종류로는 고전적인 알고리즘(Classical Algorithm)[3,4], Barrett의 알고리즘[5], Montgomery의 알고리즘[6,7], Selby의 알고리즘[8], Takagi의 알고리즘 등이 있다. 본 논문에서는 고전적인 알고리즘과 montgomery의 알고리즘을 사용하였기에 여기에 대한 알고리즘들만 살펴보겠다.

3. 2. 1 고전적인 알고리즘 (Classical Algorithm)

곱셈의 출력값인 최대 2n자리 수의 C에 대해서 n자리 수의 N에 대해서 모듈라 감소를 하는 것은 C가 N보다 작아질 때까지 C에서 N을 계속해서 감산하는 과정을 반복함으로써 구할 수 있다. 그러나, 이것은 b(기수)와 n(자리수)이 커지는 경우 비효율적이게 되므로 나눗셈에 의해 구해진 몫을 가지고 감산을 하면 효과적이게 된다. 여기서 나눗셈은 다정도 나눗셈을 의미하게 되는데 이러한 다정도 나눗셈이 실제로 쉽지 않기 때문에 몫 추정 방법이라는 것을 사용하게 된다. 몫 추정 방법은 수행 연산 시간은 줄이는 대신 다소의 오차를 허용하는 방식이다. 몫 추정 방법은 일반적으로 다음 두 가지 정도의 방식으로 수행된다.

(1) C의 최상위 두자리수와 N의 최상위 한자리수를 나누어서 몫 q'를 얻는다.
: 이 경우 q'는 최대 2의 오차를 가지게 되며, 이 때 $C=C+Nb^i$를 두 번 계산해야 한다.

$$q'=(C_{n+i}||C_{n-1+i})\ \mathrm{DIV}\ N_{n-1}$$
$$\text{if } N_{n-1}\geq b/2 \text{ then } q\leq q'\leq q+2$$

(2) C의 최상위 세자리수와 N의 최상위 두자리수를 나누어서 몫 q'를 얻는다.
: 이 경우 q'는 오차 확률 2/b로서 최대 1의 오차를 가지게 되며, 이 때 $C=C+Nb^i$를 한 번만 계산하면 된다.

$$q'=(C_{n+i}||C_{n-1+i}||C_{n-2+i})\ \mathrm{DIV}\ (N_{n-1}||N_{n-2})$$
$$\text{if } N_{n-1}\geq b/2 \text{ then } q\leq q'\leq q+1$$

(2)의 추정방법은 (1)의 추정방법보다 복잡하고 부분 계산속도가 더 느리지만 추정된 q'가 매우 정확하므로 교정에 필요한 시간이 거의 들지 않기 때문에 전체적인 속도면에서는 (1)의 방법보다 빠르게 된다. 그러므로, 본 논문에서는 (2)의 방법을 사용한다.

3. 2. 2 Montgomery의 알고리즘 (Montgomery's Algorithm)

Montgomery는 반복적인 나눗셈을 하는 대신 모듈라 N상에서의 잉여집합(residue class

modulo N)을 N-residue로 정의하고 이 집합 내에서 빠른 모듈라 감소를 하는 알고리즘을 제안하였다. 즉, 일반적인 모듈라 연산이 정의된 집합 Z_n에서 연산을 하는 대신 RZ_n상에서 모듈라 연산을 함으로써 속도를 개선시킨 것이다. 여기서 R은 N보다 크고 N과 서로 소이어야 하며, 또한 모듈라 감소가 쉽게 이루어져야 한다. 일반적으로 R은 b^n으로 정의되는데 그 이유는 b를 기수로 하는 수를 b^n으로 나누었을 때 몫과 나머지를 구하는 문제는 어려운 다정도 나눗셈을 필요로 하는 것이 아니라 단지 n자리 이상의 수를 얻는 것과 n자리 이하의 수를 얻는 것으로 가능하기 때문이다. Montgomery 알고리즘의 원리는 다음과 같다.

Montgomery는 A MOD N의 잉여류를 AR MOD N으로 표현하고, 그것의 모듈라 곱셈을 다음과 같이 정의하였다.

$$\text{Montgomery-Product}(A,B,N,R)=ABR^{-1} \text{ MOD } N$$

상기와 같은 Montgomery의 곱셈의 구현은 다음에 따르는 Montgomery의 정리에 그 기반을 두고 있다.

[정리 1] Montgomery (1985년)

N 과 R 을 서로 소라고 가정하고, $N' = -N^{-1}$ MOD R 이라고 하면, 모든 수 T 와 (T+MN)/R 은 다음 수식을 만족한다.

$$(T+MN)/R \equiv TR^{-1} \text{ MOD } N \text{ (여기서, } M=TN' \text{ MOD } R) \quad \text{---------- (1)}$$

<증명> 방정식 (1)은 M을 TN'으로 치환함으로써 직접적으로 증명된다.

상기 정리를 기반으로 하여 지금부터는 Montgomery-Product의 실제 구현에 대해서 살펴보겠다. [정리 1]을 기본으로 한 알고리즘은 다음과 같다.

Montgomery-Product (A,B,N,R)

단계 1 : $N' = -N^{-1}$ MOD R
단계 2 : T=AB
단계 3 : M=TN' MOD R
단계 4 : T=T+MN
단계 5 : T=T DIV R
단계 6 : if T≥N then T=T-N
단계 7 : return T

[알고리즘 1 : Mntgomery의 곱셈 알고리즘]

[알고리즘 1]은 Montgomery의 정리를 기본바탕으로 구현한 것이며, 실제 구현에서는 단계 3에서의 곱셈 연산을 위해 상기의 알고리즘을 약간씩 개선시켜야 하는데 다음은 그러한 알고리즘에 대한 고찰이며, 본 논문에서는 [알고리즘 2]를 사용하였다.

Montgomery 감소는 전형적으로 Right-to-Left 연산이다. 그러므로, M_i는 단지 T_i에만 종속된다. 이것을 이용해서 단계 2의 T=AB를 단계 4의 바로 전에 놓아 T_i가 계산되면 즉시 M_i가 계산될 수 있도록 알고리즘을 개선할 수 있다. 물론 T=AB라고 놓는 것이 아니라 $T=T+A_iBb^i$라고 두어 A의 각 자리수 단위로 곱셈이 가능하게 해야 한다.

Montgomery_Product_B (A,B,N,R)

단계 1 : $N_0' = -N_0^{-1}$ MOD b
단계 2 : T=0
단계 3 : for i=0 to n-1 step 1
단계 4 : $T=T+A_iBb^i$
단계 5 : $M_i=T_iN_0'$ MOD b
단계 6 : $T=T+M_iNb^i$
단계 7 : endfor
단계 8 : T=T DIV R
단계 9 : if $T \geq N$ then T=T-N
단계10 : return T

[알고리즘 2 : Mntgomery의 개선된 모듈라 감소 알고리즘 B]

3. 3 모듈라 멱승 알고리즘

모듈라 멱승 알고리즘의 종류로는 이진 방식(Binary Method)[3], r진 방식(확장이진방식)[3], 누승 테이블 방식[9], Schnorr 방식, Montgomery 감소 알고리즘 및 이진 방식을 혼용한 방식 등이 있다. 본 논문은 여기서 이진 방식 및 Montgomery 감소 알고리즘 및 이진 방식을 혼용한 방식(이 방식은 모듈라 감소까지 행하는 알고리즘이다.)에 대하여 살펴보고, 이하 본절에서 사용되는 기호를 다음과 같이 정의하겠다.

$$L(d) = \lceil \log_d 2 \rceil$$

3. 3. 1 이진 방식 (Binary Method) 알고리즘

이진 방식은 x^d MOD n의 계산을 제곱연산(A^2 MOD N)과 곱셈연산(A MUL X MOD N)의 혼용으로 해결하는 방식이다. 구체적인 방법을 살펴보면 지수 d를 2진수 $d=(d_{L(d)}, d_{L(d)-1}, \ldots, d_0)$로 표현하여, 비트 값이 0이면 제곱연산을 수행하고, 1이면 제곱연산을 수행한 후 곱셈연산을 실행한다. 알고리즘은 다음과 같다.

Binary_Method (X,d)

단계 1 : A=X
단계 2 : for i=L(d)-1 to 0 step -1
단계 3 : $A=A \times A$
단계 4 : if d_i=1 then $A=A \times d$
단계 5 : endfor
단계 6 : return A

[알고리즘 3 : 이진 방식 알고리즘]

3. 3. 2 Montgomery 감소와 이진방식의 결합 알고리즘

이 방식은 이진 방식의 기본 개념인 지수 d를 2진수 $d=(d_{L(d)}, d_{L(d)-1}, \ldots, d_0)$로 표현하여, 비트 값이 0이면 제곱연산을 수행하고, 1이면 제곱연산을 수행하는 방식에 그 기반을 두고 있으며, 여기에 덧붙여 제곱연산의 속도를 개선시키기 위해서 3.2.2의 [알고리즘 2]를 사용하는 방식이다. 이 방식을 사용함으로써 속도 개선이외에도 모듈라 감소를 동시에 행할 수 있는 알고리즘이 된다. 알고리즘은 다음과 같다.

Montgomery_Exponentiation (X,d,N)

단계 1 : $X'=XR \bmod N$
단계 2 : n_0' =Modular_Inverse(N_1, b)
단계 3 : C=X'
단계 4 : for i=L(d) to 0 step -1
단계 5 : C=Montgomery_Product_B(C,C,N,R)
단계 6 : if d_i=1 then C=Montgomery_Product_B(C,M',N,R)
단계 7 : C=Montgomery_Product_B(C,1,N,R);
단계 8 : endfor
단계 9 : return C

[알고리즘 4 : Mntgomery 감소와 이진 방식 결합의 알고리즘]

4. 구현 결과 및 비교분석

본 절에서는 상기에서 고찰한 알고리즘들을 실제로 구현하여 그 결과를 비교하여 본다. 본 구현에서 사용된 운영체제는 16비트 환경인 MS-DOS 6.0과 Windows 3.1, 그리고 32비트 환경인 Windows 95와 UNIX(Solaris2.3)로 나뉘어지며, 16비트 환경에서는 기수 b를 2^{16}으로 설정하였고, 32비트 환경에서는 기수 b를 2^{32}으로 설정하였다. 속도비교는 데이터의 크기를 256비트, 512비트, 768비트, 1024비트별로 나누어 이루어졌으며, 이것을 b를 기수로 하는 자리수 n으로 나타내면 16비트 환경에서는 16,32,48,64이 되고, 32비트 환경에서는 8,16,24,32가 된다. 여기서 우리는 16비트 환경에서의 구현보다 32비트 환경에서의 구현이 속도가 더 빠를 것이라는 것을 쉽게 알 수 있으며, 이것은 <표2,3,4,5,>에서 증명이 되었다.

다음 구현 결과에서 우리는 세 가지의 관점에서 그 결과를 분석해 본다. 첫 번째는 MS-DOS 6.0의 TURBO-C 2.0과 Windows 3.1의 MS Visual C++ 1.52의 메모리 타입 설정이 알고리즘의 수행 속도에 어떤 영향을 미치는 가이다. 우리는 구현전 대형(large)모델에서의 구현이 소형(small)이나 중형(medium)모델에서의 구현에 비해 수행속도가 다소 느릴 것이라고 예상하였다. 아래의 구현결과는 이러한 예상이 옳음을 보여주는데, 그 이유는 대형모델이 원거리포인터(far pointer)를 사용하기 때문이며, 이것은 함수의 호출이나 데이터처리에 다소의 영향을 미치게 된다. 반면에 Windows 95와 UNIX는 메모리 모델 타입 설정이 없는데 이것은 운영체제가 메모리관리를 자동적으로 수행하기 때문이다.

두 번째로 우리는 DLL(Dynamic Link Library)에 관심을 두었다. DLL은 Windows3.1과 Windows95의 우수한 장점중의 하나로서, 이것의 기능은 강력한 동적 라이브러리 기능을 제공해 주며 폭넓은 호환성을 보장해 준다는 것이다. 그러므로, 디지틀 서명의 구현요소나 여러 가지 암호화 알고리즘들을 DLL화시키는 것은 아주 중요한 일이며, 이러한 경우 구현 알고리즘들의 수행속도 문제는 상당히 중요한 문제가 된다. 우리는 DLL이 주 프로그램 수행시에 동적으로 링크되

기 때문에 DLL로 구현하지 않은 알고리즘의 속도보다는 다소 느릴 것 이라고 예상했다. Windows 3.1에서는 예상된 결과가 도출되었으나, Windows 95에서는 컴파일러의 최적화 기능 및 성능 개선으로 인하여 DLL의 오버헤드가 거의 발생하지 않았다. 오히려 일부 알고리즘들에서는 DLL의 수행속도가 다소 더 빠르기도 하였다.

세 번째로 우리는 개인용 컴퓨터에서 실질적인 32비트 환경을 가져온 Windows 95에서의 32비트 알고리즘 수행속도 문제에 관심을 가졌다. 디지틀 서명의 구현 요소 알고리즘들은 큰 수를 다루어야 하기 때문에 기수 b가 속도에 미치는 영향은 아주 크다고 할 수 있다. 기수 b가 크게 되면 n자리수가 상대적으로 줄어들기 때문에 그 만큼 빠른 수행속도를 얻을 수 있는 것이다. 특히 n자리수가 커질수록 이것의 수행속도차는 커지게 되며, <표5>는 이것을 직접적으로 보여주고 있다. 또한 UNIX도 32비트 환경이지만 Workstation의 CPU가 Pentium 100MHZ에 비해 다소 성능이 떨어지기 때문에 예상했던 것 만큼은 좋은 수행속도 결과를 내지 못했다.

<표 2 : 256비트 구현결과 비교표>

(PC:Pentium 100MHZ, Workstation:Unistation20)

운영체제 / 알고리즘	MS-DOS ($b=2^{16}$ / n=16)		Windows 3.1 ($b=2^{16}$ / n=16)			Windows 95 ($b=2^{32}$ / n=8)		UNIX ($b=2^{32}$ / n=8)
	Small	Large	Medium	Large	DLL	Normal	DLL	(Solaris 2.3)
덧셈연산 (msec)	0.0054	0.0087	0.0050	0.0061	0.0127	0.0045	0.0045	0.0083
감산연산 (msec)	0.0082	0.0108	0.0071	0.0088	0.0329	0.0064	0.0050	0.0111
곱셈연산 (msec)	0.1923	0.2197	0.1100	0.1370	0.6480	0.0880	0.0880	0.1920
제곱연산 (msec)	0.1153	0.1645	0.0990	0.1100	0.3730	0.0307	0.0307	0.1140
Classical (msec)	0.0769	0.1098	0.0710	0.0820	0.2031	0.0412	0.0423	0.1400
Montgomery (msec)	0.4395	0.6593	0.2530	0.2980	1.3082	0.1710	0.1650	0.3940
이진방식 (sec)	0.329	0.384	0.192	0.247	0.703	0.0483	0.0483	0.315
혼합방식 (sec)	0.109	0.164	0.088	0.149	0.4122	0.0291	0.0291	0.137

※ 상기표에서 Montgomery 알고리즘은 모듈라 곱셈 감소 알고리즘이다.

<표 3 : 512비트 구현결과비교표>

(PC:Pentium 100MHZ, Workstation:Unistation20)

운영체제 / 알고리즘	MS-DOS ($b=2^{16}$ / n=32)		Windows 3.1 ($b=2^{16}$ / n=32)			Windows 95 ($b=2^{32}$ / n=16)		UNIX ($b=2^{32}$ / n=16)
	Small	Large	Medium	Large	DLL	Normal	DLL	(Solaris 2.3)
덧셈연산 (msec)	0.0082	0.0164	0.0094	0.0110	0.0225	0.0071	0.0071	0.0145
감산연산 (msec)	0.0137	0.0209	0.0126	0.0160	0.0621	0.0081	0.0079	0.0196
곱셈연산 (msec)	0.7362	0.9340	0.3960	0.4720	2.3952	0.1870	0.1860	0.6700
제곱연산 (msec)	0.4230	0.5494	0.2910	0.3350	1.3132	0.0929	0.0934	0.3950
Classical (msec)	0.3021	0.3296	0.1200	0.1370	0.3732	0.0686	0.0698	0.2570
Montgomery (msec)	1.4285	1.9780	0.8340	0.9560	4.8902	0.3730	0.3680	1.3470
이진방식 (sec)	2.142	2.802	1.346	1.713	5.075	0.2582	0.2587	1.904
혼합방식 (sec)	0.989	1.318	0.583	0.687	3.164	0.1279	0.1291	0.903

※ 상기표에서 Montgomery 알고리즘은 모듈라 곱셈 감소 알고리즘이다.

<표 4 : 768비트 구현결과 비교표>

(PC:Pentium 100MHZ, Workstation:Unistation20)

운영체제 / 알고리즘	MS-DOS ($b=2^{16}$ / n=48)		Windows 3.1 ($b=2^{16}$ / n=48)			Windows 95 ($b=2^{32}$ / n=24)		UNIX ($b=2^{32}$ / n=24)
	Small	Large	Medium	Large	DLL	Normal	DLL	(Solaris 2.3)
덧셈연산 (msec)	0.0137	0.0252	0.0131	0.0160	0.0308	0.0104	0.0110	0.0203
감산연산 (msec)	0.0219	0.0285	0.0176	0.0230	0.0917	0.0116	0.0115	0.0279
곱셈연산 (msec)	1.6208	2.0329	0.8620	1.0110	5.2891	0.3460	0.3510	1.4560
제곱연산 (msec)	0.9340	1.1538	0.5870	0.6700	2.8124	0.1930	0.1920	0.8400
Classical (msec)	0.2032	0.2747	0.1710	0.2090	0.5492	0.0980	0.0940	0.3540
Montgomery (msec)	3.1868	4.2857	1.8100	2.0300	10.601	0.7250	0.7140	3.0800
이진방식 (sec)	6.648	8.791	4.235	5.432	16.302	0.802	0.797	6.16
혼합방식 (sec)	3.131	4.285	1.851	2.164	10.403	0.379	0.384	2.92

※ 상기표에서 Montgomery 알고리즘은 모듈라 곱셈 감소 알고리즘이다.

<표 5 : 1024비트 구현결과 비교표>

(PC:Pentium 100MHZ, Workstation:Unistation20)

운영체제 / 알고리즘	MS-DOS ($b=2^{16}$ / n=64)		Windows 3.1 ($b=2^{16}$ / n=64)			Windows 95 ($b=2^{32}$ / n=32)		UNIX ($b=2^{32}$ / n=32)
	Small	Large	Medium	Large	DLL	Normal	DLL	(Solaris 2.3)
덧셈연산 (msec)	0.0192	0.0329	0.0171	0.0160	0.0412	0.0110	0.0126	0.0262
감산연산 (msec)	0.0301	0.0384	0.0230	0.0230	0.1203	0.0138	0.0143	0.0363
곱셈연산 (msec)	2.8461	3.5714	1.5050	1.0110	9.2992	0.5760	0.5710	2.5500
제곱연산 (msec)	1.6483	2.1978	0.9770	0.6700	4.8671	0.3190	0.3240	1.4390
Classical (msec)	0.2747	0.3296	0.2250	0.2090	0.7302	0.1270	0.1260	0.9060
Montgomery (msec)	5.4945	7.6923	3.0700	2.0300	18.562	1.1750	1.1650	5.0800
이진방식 (sec)	14.780	19.560	9.448	12.133	36.78	1.695	1.681	6.76
혼합방식 (sec)	7.987	9.615	4.120	4.806	23.59	0.774	0.791	13.51

※ 상기표에서 Montgomery 알고리즘은 모듈라 곱셈 감소 알고리즘이다.

5. 결 론

본 논문에서는 디지틀 서명 구현에 필요한 모듈라 연산에 관한 알고리즘들을 여러 가지 운영체제들상에서 C언어 및 C++언어로 구현하여 성능을 비교 분석하였다. 사용된 운영체제는 16비트 환경인 MS-DOS 6.0과 Windows 3.1, 그리고 32비트 환경인 Windows 95와 UNIX(Solaris2.3)이며, MS-DOS 환경에서는 메모리 모델을 달리하여 보았고, Windows 환경에서는 DLL로 구현하여 보았다. 이러한 구현결과는 디지틀 서명 및 여러 가지 암호화 기술을 실제로 구현하는 경우 최적화가 가능한 운영체제 및 컴파일러 선택에 기준을 제공해 줄 수 있을 것으로 사료된다.

향후에는 암호기술에 전반적으로 필요로되는 여러 가지 암호화 기본 알고리즘들을 라이브러리화하여 Windows NT 및 OS/2 운영체제 등에서 확장된 성능평가를 할 계획이다.

참 고 문 헌

[1] W. Diffie and M. Hellman, New Directions in cryptography. IEEE Trans, on Informati-

on Theory, Vol. 22. No. 6. pp.644-654, November 1976

[2] Naofumi Takaki, A modular Multiplication Algorithm with Triangle Additions, Proceeings 11th Symposium on Computer Arithmetic, pp. 272-276, 1993.

[3] D. E. Knuth, The Art of Programming, Vol.2 : Seminumerical Algorithms, 2nd Ed., Chap. 4, Addison Wesley, 1981.

[4] Antoon Bosselaers, Rene Govaerts and Joos Vandewalle, Comparison of three modular reduction functions, Advances in Cryptography-Crypto '93, pp. 175-186, Santa Barbara, California, August 1993.

[5] Paul Barrett. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signature Processor, Advances in Cryptology-Crypto '86, LNCS 263, pp. 311-323

[6] Peter L. Montgomery, Modular Multiplication Without Trial Division, Math. of Comp., Vol. 44, No. 170, pp.519-521, April 1985.

[7] S. R. Dusse and B. S. Kaliski Jr., A Cryptographic Library for the Motorola DSP 560-00, Advances in Cryptology-Eurocrypt '90, LNCS 437, pp.230-244.

[8] A. Selby and C.Mitchell, Algorithms for Software implementations of RSA, IEEE Proceedings, Vol. 136 Pt. E. No. 3, pp. 166-170, May 1989.

[9] 최성민,김병천,정지원,원동호, 모듈라 멱승 알고리즘의 구현에 관한 연구, 한국통신정보보호학회 '93 종합학술발표회 논문집, 1993.

[10] 황효선,임채훈,이필중, PC에서 모듈라 곱셈 연산의 구현과 비교분석,통신정보보호학회 학회지, 제4권 3호, 1994.

[11] 황효선,임채훈,이필중, 멱승 알고리듬의 구현과 분석, 통신정보보호학회 학회지, 제5권 1호, 1995. 3.