공개키 암호시스템의 고속 구현

황효선°, 임채훈

퓨쳐시스템 정보통신연구소

Fast Software Implementation of Public key Systems

Hyo Sun Hwang, Chae Hoon Lim

Inform. & Commun. Research Center, Future Systems, Inc.

요 약

공개키 암호시스템의 구현을 위해서는 큰 수들 간의 모듈라 연산 라이브러리를 구축해야 한다. 본 논문에서는 C 언어 및 부분적인 어셈블리어를 사용하여 구축된 모듈라 연산 라이브러리를 이용하여 모듈라 멱승 연산에 바탕을 둔 대표적인 공개키 암호시스템인 RSA, Diffie-Hellman 및 한국 디지탈서명 표준(안)인 KCDSA 등을 다양한 컴퓨터 기종에서 구현한 결과를 제시한다.

제 1 절 서 론

소인수분해의 어려움을 이용한 RSA 암호시스템[14]과 이산대수문제의 어려움을 이용한 Diffie-Hellman / ElGamal 암호시스템[3, 4]들은 큰 수의 모듈라 멱승을 기본연산으로 한다. 그러므로 이러한 암호시스템의 구현속도는 모듈라 멱승 연산의 속도에 전적으로 의존한다. 모듈라 멱승을 빠르게 하기 위해서는 빠른 모듈라 곱셈 알고리듬으로 기본연산의 속도를 향상시키고, 또한 멱승에 필요한 모듈라 곱셈수를 가능한 최소한으로 줄여야 한다.

본 논문에서는 Montgomery[13] / L2[11] 모듈라 감소 알고리듬, 윈도우 멱승 알고리듬, 사전계산테이블을 이용한 LL 멱승 알고리듬[8] 등 지금까지 개발된 최선의 알고리듬들로 구성된 모듈라 연산 라이브러리를 이용하여 RSA, Diffie-Hellman, KCDSA[10] 등의 공개키 암호 시스템들을 구현한 결과를 제시하고 이를 비교 분석한다.

큰 수의 표기는 컴퓨터가 지원하는 32비트(기종에 따라서는 16비트나 64비트가 될 수도 있다.) 정수를 기수로 하여 나타낸다. 즉 $b = 2^{32}$를 기수로 하여 $32n$비트의 큰 정수 A를 다음과 같이 n자리수로 표기한다.

$$A = \sum_{i=0}^{n-1} A_i b^i, \quad A_i \in [0, b-1] \ \ for \ \ i = 0, \cdots, n-1.$$

특별한 언급이 없는 한 법 N은 n자리수로 가정하고 모듈라 감소는 $2n$자리의 수를 대상으로 한다.

여러 알고리듬의 구현은 SPARC20/60MHz와 ULTRASPARC 167MHz의 경우는 C 언어로 작성하여 gcc로 컴파일하였고, Pentium 90MHz와 Pentium Pro 200NHz의 경우는 C 언어만으로 작성된 것과, C 언어에 일부 핵심부분을 inline assembly로 구현한 것, 그리고 C 언어에 컴파일러가 제공하는 64비트 data type인 (_int64)를 이용한 것의 세 종류로 작성하여 MSVC/C++ 4.2로 컴파일하였다. 테이블에 사용된 표기를 정리하면 다음과 같다:

- S20-C: SPARC20 60MHz에서 C언어만으로 구현.
- US-C: ULTRASPARC 167MHz에서 C언어만으로 구현.
- P-C, P-D, P-A: 각각 Pentium 90MHz에서 C 언어, _int64 사용, inline assembly 사용시의 구현 결과.
- PP-C, PP-D, PP-A: 각각 Pentium Pro 200NHz에서 C 언어, _int64 사용, inline assembly 사용시의 구현 결과.

제 2 절 모듈라 연산 알고리듬

모듈라 곱셈 및 멱승 알고리듬들을 간략히 살펴본다.

2.1 큰 수의 곱셈

큰 수의 곱셈은 손으로 하는 것과 같은 일반적인 곱셈 방법에 Karatsuba의 알고리듬[6]을 적용하는 것이 유용하다. Karatsuba의 알고리듬은 n자리의 정수 A, B에 대하여 식

$$\begin{aligned} AB &= (A_1 b^{n/2} + A_0)(B_1 b^{n/2} + B_0) \\ &= A_1 B_1 b^n - ((A_1 - A_0)(B_1 - B_0) - A_1 B_1 - A_0 B_0) b^{n/2} + A_0 B_0 \end{aligned}$$

을 이용한 것으로 한 번의 적용으로 필요한 단정도 곱셈의 횟수를 n^2번에서 $3(n/2)^2 = (3/4)n^2$번으로 줄인다. 이때 A, B가 큰 수일 경우 세 번 필요한 각각의 $n/2$자리의 곱셈에 대해서도 Karatsuba의 알고리듬을 반복 적용할 수 있다. 특정 n에 대해 여러번 반복하여 Karatsuba의 알고리듬을 적용할 수는 있지만, 매 과정에서 필요한 단정도 곱셈의 횟수는 줄어드는 반면 여러번의 큰 수의 덧셈과 뺄셈이 필요하므로 Karatsuba의 알고리듬을 몇 번이나 반복하여 적용할 지는 구현결과를 보아 판단하여야 한다. 제곱의 경우는 곱셈 메트릭스의 대칭성을 이용하면 곱셈에 비해 대략 20 ~ 30% 정도 속도를 향상시킬 수 있다.

표1은 구현한 큰 수의 제곱과 곱셈에서 Karatsuba의 아이디어를 몇 번이나 반복하는 것이 효과적인지를 나타낸 것이다. 당연하게도 자리수 n이 커질수록 Karatsuba의 depth는 증가한다. 그리고 C 언어로 작성된 경우가 inline assembly로 작성된 경우보다 Karatsuba의 알고리듬에 더 효과적이다. 아래 표에서 / 좌측은 제곱셈에서, / 우측은 곱셈에서의 최적의 Karatsuba 알고리듬 반복횟수이다.

n	S20-C	US-C	P-C	P-D	P-A	PP-C	PP-D	PP-A
512	2/3	2/3	1/2	0/1	0/0	1/2	0/0	0/0
768	3/3	2/3	2/2	0/1	0/1	1/3	0/1	0/0
1024	3/4	3/4	2/3	0/1	1/1	2/3	0/1	0/1
1536	4/4	3/4	3/4	1/2	1/2	3/3	0/2	1/1
2048	4/5	4/5	3/4	1/2	2/2	3/4	1/2	1/2

표 1: 제곱 / 곱셈에서 최적의 Karatsuba 알고리듬 반복횟수

2.2 모듈라 감소 알고리듬

가장 널리 사용되는 모듈라 감소 알고리듬의 하나인 Montgomery 알고리듬[13]은 수체계의 변환을 통해 모듈라 감소가 쉽게 되는 수체계에서 연산한 후 원래의 수체계로 역변환시키는 방법을 이용한다. 즉 모듈라 곱셈이 환 $\mathbf{Z}_N$에서의 연산이지만 Montgomery는 N과 서로소이면서 모듈라 감소가 쉬운 수 $R = b^n$를 이용하여 환 $R\mathbf{Z}_N$에서 모듈라 곱셈을 비교적 빠르게 할 수 있는 알고리듬을 고안하였다. 따라서 $AB \bmod N$을 계산하기 위해서는 우선 A, B를 환 $R\mathbf{Z}_N$의 대응되는 수로 변환시켜 이 수체계에서 모듈라 곱셈을 한 후 그 결과를 다시 원래의 수체계 $\mathbf{Z}_N$으로 역변환시키면 된다. 이러한 수체계의 변환에는 한 번의 모듈라 감소에 해당하는 연산이 필요하므로 Montgomery 알고리듬은 단지 몇 번의 모듈라 곱셈이 필요한 경우는 비효율적이다. 또한 Montgomery 알고리듬은 감소시키고자 하는 수의 크기에 상관없이 같은 계산량이 필요하므로 $2|N|$비트 보다 작은 수를 모듈라 감소시킬 때에도 불리하다 (모듈라 감소 알고리듬들의 비교 분석에 대해서는 참고문헌 [2] 참조). 그러나 모듈라 멱승과 같이 수체계의 변환에 드는 계산량이 거의 무시될 수 있는 경우에는 고전적인 모듈라 감소법에 비해 상당히 효율적이다.

한편 법 N에 대해 $M_0 = b^{n+2} \bmod N$을 미리 계산하여 저장해 두면, $n+3$자리수 $C = \sum_{i=0}^{i=n+2} C_i b^i$가 주

어질 때 다음과 같이 간단히 C를 한 자리 모듈라 감소시킬 수 있다.

$$C \leftarrow \sum_{i=0}^{i=n+1} C_i b^i + C_{i+2} M_0.$$

위에서 오른쪽의 계산 결과 마지막에 한 비트 carry가 생길 확률은 $1/(2b)$ 미만이며, 이때는 마지막 carry를 무시하고 C에 M_0를 더해 주면 된다. 이 아이디어의 단점은 $2n$자리수 C가 주어졌을 때 그 상위 $n-2$자리는 M_0를 이용하여 모듈라 감소시킬 수 있지만 마지막 두자리는 고전적 알고리듬 등의 다른 방법으로 모듈라 감소시켜야 한다는 것이다. 그러나 이 알고리듬도 효율면에서는 거의 Montgomery 알고리듬과 유사하며, Montgomery에서 필요한 수체계 변환 등이 필요없이 법 N에 의존하는 하나의 사전계산값만을 이용하므로 대부분의 응용에서 유용하게 사용될 수 있다. 위의 알고리듬을 $L1$ 알고리듬이라고 하자.

두 개의 사전계산값을 이용하는 L2 알고리듬은 n자리수의 곱셈이 $2n$자리의 큰 수를 n자리 모듈라 감소시키는 것보다 빠르다는 사실을 이용한 것이다. 일반적으로도 큰 수의 곱셈은 Karatsuba 방식을 쓰면 모듈라 감소보다 훨씬 빠르게 계산될 수 있고, 그 차이는 법 N이 커질 수록 점점 더 벌어지게 된다. $M_1 = b^{n+n/2} \bmod N$ (n이 짝수라 가정)을 미리 계산하여 저장해 두면, $2n$자리수 $C = \sum_{i=0}^{i=2n-1} C_i b^i$는 다음과 같이 곱셈을 이용해 상위 $n/2$자리를 모듈라 감소시킬 수 있다:

$$C \leftarrow \sum_{i=0}^{i=n+n/2-1} C_i b^i + \sum_{i=n+n/2}^{i=2n-1} C_i b^i \times M_1$$

이 때에도 $1/2$ 미만의 확률로 carry가 생길 수 있지만, 마찬가지로 M_1를 한 번 더해 줌으로써 처리할 수 있다. 또한 Karatsuba depth가 큰 경우에는 $M_2 = b^{n+n/4} \bmod N$, $M_3 = b^{n+n/8} \bmod N$ 등을 미리 계산하여 저장함으로써 차상위 $n/2$자리와 차차상위 $n/4$자리도 동일한 방법으로 모듈라감소시킬 수 있다. 그리고 남은 $n/2$, $n/4$ 또는 $n/8$자리의 모듈라감소에는 L1 알고리듬을 이용한다.

만약 법 N의 상위 2자리가 $b-1$이라면 $M_0 = b^n \bmod N$로 두어도 M_0가 두자리 모듈라 감소되었기 때문에 위와 같은 효과를 나타낸다. 그리고 이 경우에는 일반적인 N에서는 불가능한 마지막 두자리까지 완전하게 모듈라 감소시킬 수 있다. 또한 매 자리의 모듈라감소에서도 M_0가 $n-2$자리수이므로 $C_k M_0$ 계산시 2회의 32비트 곱셈이 줄어든다. 그러므로 N의 상위 2자리가 $b-1$로 특수한 형태일 경우 L2 알고리듬은 Montgomery 알고리듬보다 매우 우수하다.

본 구현에서 inline assembly를 사용한 경우는 32비트 정수와 큰 수 M의 곱셈시 M의 자리수가 8, 16, 32인 경우(N이 256, 512, 1024비트)에는 loop을 제거한 코드를 사용하였다. 그래서 16 또는 32자리인 특수한 형태의 N일 경우 L2 알고리듬에서 $C_k M_0$ 계산시 2회의 32비트 곱셈이 줄어드는 효과가 거의 없어졌다. 다시말해서 N이 32자리인 경우 Montgomery는 32비트수와 32자리와의 곱셈은 loop이 없는 코드로 수행되지만, L2 알고리듬의 경우는 32비트수와 30자리와의 곱셈이 이루어지므로 loop이 있는 코드로 수행되기 때문에 속도가 조금 느려진다. 또한 assembly의 경우 Karatsuba depth가 상대적으로 작기 때문에 두번째 아이디어에 의한 효과도 상대적으로 작아서 결과적으로 L2 알고리듬와 Montgomery 알고리듬의 차이가 별로 나지 않는다.

2.3 윈도우 멱승 알고리듬

기본적인 멱승 방법인 이진 멱승 알고리듬(Binary algorithm)은 지수의 상위 비트부터 참조하여 제곱을 반복하면서 지수가 1이면 밑을 곱해주는 멱승법으로, $M^E \bmod N$을 계산하는데 지수 E가 m비트이면 약 m번의 제곱과 $0.5m$번의 곱셈이 필요하다. 밑 M가 고정되지 않은 경우에는 필요한 제곱셈의 횟수를 줄일 수는 없지만 적당한 사전계산을 통해 여러개의 비트를 동시에 처리함으로써 필요한 곱셈의 횟수는 줄일 수 있다. 가장 널리 쓰이는 윈도우 알고리듬(Window algorithm)이 그것인데, 우선 적당한 윈도우 크기 w에 대해 $\{M^1, M^3, \cdots, M^{2^w-1}\}$을 계산하여 저장한다. 이제 E를 2진수로 나타내어 상위비트부터 확인하면서 1로 시작하고 1로 끝나며 크기가 w비트 이하인 윈도우를 잡아, 제곱을 반복하면서 각 윈도우에 대해 그 윈도우값 k로부터 미리 계산해 둔 $M^k \bmod N$를 곱하여 멱승 연산을 수행한다.

윈도우 크기가 w일때 비트길이 m인 지수에 대한 윈도우 멱승 알고리듬의 성능, 즉 멱승에 필요한 평균 모

돌라 곱셈의 수는 다음 식과 같이 주어진다 [7, Sect.1.2.3].

$$Win\_Mul(m,w) = m(1+\frac{1}{w+1}) + PC - \frac{1}{PC} - w.$$

여기서 PC는 사전계산에 필요한 모듈라 곱셈의 수로 $PC = 2^{w-1}$으로 주어지며, 이는 또한 윈도우 멱승 알고리듬의 메모리 요구량이 되기도 한다 (즉 사전 계산량을 저장하는데 PC개의 m비트 수에 대한 메모리가 필요하다). 위의 곱셈수를 최소화하는 윈도우 크기 w는 지수 E의 비트길이에 따라 정해지는데 E의 비트길이를 m이라 할때 최적의 w는 표 2와 같이 주어진다.

m	$m \leq 14$	$m \leq 59$	$m \leq 210$	$m \leq 629$	$m \leq 1735$	$m \leq 4537$
w	2	3	4	5	6	7

표 2: 지수의 비트길이 m에 따른 최적의 윈도우 크기 w

2.4 사전계산을 이용한 LL 멱승 알고리듬

대부분의 이산대수문제에 근거한 공개키 시스템들은 고정된 밑 G와 랜덤수 R에 대해 $G^R \bmod P$의 계산을 필요로 한다. 이와같이 밑이 고정된 경우는 G에 대한 사전계산테이블을 이용하면 윈도우 알고리듬에 비해 훨씬 효율적으로 멱승 연산을 할 수 있다. 대표적인 것으로 BGMW 알고리듬[1]과 LL 알고리듬[8]이 있는데, 후자의 방법이 간단하면서도 좀 더 효율적이고 메모리-계산량의 타협이 훨씬 용이하다. 본 논문에서도 LL 알고리듬을 구현하였다.

LL 알고리듬은 지수 R을 일정한 비트길이 b크기의 $h \times v$ 메트릭스 형태로 배열하였을 때 v개의 각 열에 대해 2^h개의 가능한 모든 비트 조합을 지수로 하고 $G_i = G^{2^{ia}} \bmod P$ $(a = \lceil \frac{m}{h} \rceil,\ 0 < i < h)$를 밑으로 하는 사전계산값들을 이용한다 ($b$는 $b = \lceil \frac{n}{v} \rceil$로 주어진다). 이 알고리듬은 $2^h v$개의 사전계산값이 필요하며, 이 때 평균 곱셈수는 다음과 같이 주어진다:

$$LL\_Mul(m,h,v) = \frac{2^h - 1}{2^h}a + b - 2 \text{ (여기서 } a = \lceil \frac{m}{h} \rceil,\ b = \lceil \frac{a}{v} \rceil).$$

자세한 내용은 참고문헌 [8]을 참조하기 바란다. 표 3에 대표적인 h, v값에 대한 메모리 요구량 및 평균 곱셈수를 나타내었다. 비교를 위해 윈도우 멱승 알고리듬의 성능도 함께 나타내었다.

Config.	Storage	160	256	512	768	1024	1536	2048
Window	2^{w-1}	195.9	309.6	608.3	903.7	1196.3	1781.4	2361.0
4 × 4	64	45.5	74.0	150.0	226.0	302.0	454.0	606.0
5 × 4	128	37.0	61.4	123.8	186.2	248.6	373.4	498.2
6 × 4	256	31.6	51.3	104.7	156.0	209.3	314.0	420.7
7 × 4	512	26.8	44.7	90.4	135.1	180.9	271.3	362.7
8 × 4	1024	22.9	37.9	77.8	117.7	157.5	237.2	317.0

표 3: LL 알고리듬의 메모리 요구량 및 계산량

2.5 다항 멱승 연산

ElGamal형 디지탈서명의 검증에는 $Y^S G^R \bmod P$와 같은 두 멱승의 곱을 계산해야 한다.이러한 형태의 멱승 역시 윈도우 멱승 방법을 응용하면 각각을 독립적으로 멱승한 후 곱하는 것에 비해 훨씬 효율적으로 계산할 수 있다. 즉 일정한 크기의 윈도우에 대해 Y와 G를 밑으로 하는 작은 멱승들을 사전계산하여 저장한 후 S와 R를 동시에 상위비트부터 확인하며 윈도우를 만들어 해당하는 사전계산값을 곱해주는 것이다. 이러한 멱승

법은 각각을 독립적으로 윈도우 멱승하는 것과 비교하여 필요한 곱셈의 횟수는 비슷하지만 제곱의 횟수는 거의 반으로 줄일 수 있다.

윈도우 크기 w에 대해 사전계산 테이블은 다음과 같은 값들을 포함한다:

$$T[i][j] = Y^i G^j \bmod P \ (0 < i, j < 2^w \ \& \ \text{i,j 중 적어도 하나는 홀수})$$

이 사전계산에 필요한 곱셈수는 $PC = b^w - b^{w-1}$ ($b = 2^2 = 4$)이며, 이를 이용해 $Y^S G^R \bmod P$ ($|S| = |R| = m$)을 계산하는데 필요한 평균 곱셈수는 다음과 같이 주어진다 [7, Sect.1.2.4]:

$$N_a(m, w, b) = m(1 + \frac{1}{\overline{w}_a}) + PC - \overline{w}_d - 1.$$

여기서 $\overline{w}_d$와 $\overline{w}_a$는 다음과 같이 주어진다:

$$\begin{aligned} \overline{w}_d &= w - \frac{1}{b-1} + \frac{1}{PC}, \\ \overline{w}_a &= w + \frac{1}{b-1}. \end{aligned}$$

표 4에 m의 크기에 따른 최적 윈도우 크기, 메모리 요구량 및 평균 곱셈수를 정리하였다.

지수 크기	160	256	512	768	1024	1536	2048
Window	2	2	3	3	3	3	3
Storage	12	12	48	48	48	48	48
ModMul	237.8	375.0	709.9	1042.7	1375.5	2041.1	2706.7

표 4: $Y^S G^R \bmod N$의 계산을 위한 윈도우 알고리듬의 성능

제 3 절 공개키 암호 알고리듬

3.1 Diffie-Hellman 키분배

모든 사용자들에게 공통되는 공개변수로 소수 P와 기본원소 G가 공개된다. 두 사용자 i, j가 공통의 비밀키를 공유하고자 할 때 각자 난수 K_i/K_j를 선택하여 $X_i = G^{K_i} \bmod P$ / $X_j = G^{K_j} \bmod P$를 계산하여 공개된 통신선로를 통해 상호 교환한다(과정 1). 그리고 각자는 상대로부터 얻은 수 X_i/X_j를 이용하여 $K_{ij} = X_j^{K_i} \bmod P$ / $K_{ji} = X_i^{K_j} \bmod P$를 계산하면(과정 2) 두 사용자는 비밀키 $K = K_{ij} = K_{ji} = G^{K_i K_j} \bmod P$를 공유하게 된다. 이 키분배 과정에서 주고 받는 메세지들이 인증된 것이 아니므로 이를 그대로 실제 응용에 적용하면 전혀 안전하지 않게 되므로 통상 다른 채널을 통해 주고 받는 메세지 및 공유키를 인증하는 방식을 취한다. 인터넷의 IP 계층 보안 표준 그룹인 IPSEC에서 표준화하고 있는 키관리 표준 ISKMP/Oakley[12, 5, 15]에서는 이미 형성된 비밀채널로 MAC을 이용하여 인증하는 방법, 디지탈서명을 이용하는 방법, 그리고 공개키 암호를 이용하는 방법 등 다양한 인증 방법을 지원한다.

본 논문에서는 인증을 제외하고 과정 1과 과정 2에서 필요한 멱승 연산에 걸리는 시간을 측정하였다. 이 때 사용된 소수법 P와 기본원소 G는 ISKMP/Oakley에서 제안한 것을 사용하였다. Oakley에서는 원주율 $\pi = 3.141592\cdots$를 이용하여 i를 증가시키면서

$$P = 2^k - 2^{k-64} + 2^{64}[2^{k-128-2}\pi + i] - 1$$

에 대해 P와 $(P-1)/2$가 모두 소수가 되는 최초의 P를 찾아서 이 P와 $G = 2$를 사용하도록 권하고 있다. 이 P는 최상위 및 최하위 두 자리가 모두 1로 채워지도록 하여 모듈라 감소를 보다 효율적으로 할 수 있다. 즉 Montgomery 알고리듬에서는 n번의 단정도 곱셈이 줄어들고 L2 알고리듬에서는 $2n$번의 곱셈이 줄어 든다. 표 5에 이와같이 생성된 768, 1024, 1536 및 2048비트의 소수 P를 나타내었다 (표에서 Offset은 위 식에서 i에 해당). 768 및 1024비트의 P는 참고문헌 [15]에 제시된 것이고, 1536 및 2048비트의 P는 같은 방법에 의해 저자들이 찾은 소수들이다.

$\|P\|$	Offset	P
768	149686	FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1 29024E08 8A67CC74
		020BBEA6 3B139B22 514A0879 8E3404DD EF9519B3 CD3A431B 302B0A6D F25F1437
		4FE1356D 6D51C245 E485B576 625E7EC6 F44C42E9 A63A3620 FFFFFFFF FFFFFFFF
1024	129093	FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1 29024E08 8A67CC74
		020BBEA6 3B139B22 514A0879 8E3404DD EF9519B3 CD3A431B 302B0A6D F25F1437
		4FE1356D 6D51C245 E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
		EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE65381 FFFFFFFF FFFFFFFF
1536	9148	FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1 29024E08 8A67CC74
		020BBEA6 3B139B22 514A0879 8E3404DD EF9519B3 CD3A431B 302B0A6D F25F1437
		4FE1356D 6D51C245 E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
		EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D C2007CB8 A1637F05
		98DA4836 1C55D39A 69163FA8 FD24CF5F 83655D23 DCA3AD96 1C62F356 208552BB
		9ED52907 7096966D 670C354E 4ABC9804 F1746C08 CA1839F7 FFFFFFFF FFFFFFFF
2048	5008324	FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1 29024E08 8A67CC74
		020BBEA6 3B139B22 514A0879 8E3404DD EF9519B3 CD3A431B 302B0A6D F25F1437
		4FE1356D 6D51C245 E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
		EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D C2007CB8 A1637F05
		98DA4836 1C55D39A 69163FA8 FD24CF5F 83655D23 DCA3AD96 1C62F356 208552BB
		9ED52907 7096966D 670C354E 4ABC9804 F1746C08 CA18163C 32905E46 2E36CE3B
		E39E772C 180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9 DE2BCBF6 95581718
		3995497C EA956AE5 15D22618 98FA0510 15728E5A 8AF72FF0 FFFFFFFF FFFFFFFF

표 5: ISAKMP/Oakley의 Diffie-Hellman 키분배용 소수법 P

3.2 RSA 서명

RSA의 각 사용자는 두 큰 소수 P, Q를 선택하여 $N = PQ$와 $\phi(N) = (P-1)(Q-1)$를 구하고, $\phi(N)$과 서로소인 정수 e를 임의로 선택한 후 Euclidean 알고리듬을 이용하여 $d = e^{-1} \bmod \phi(N)$을 구한다. 공개키는 N, e가 되고 d가 비밀키가 된다. 메세지 M에 대한 RSA 서명은 $S = M^d \bmod N$과 같이 생성되고, 검증은 $S^e \bmod N$이 M과 같은지를 확인하는 과정이 된다. 이 때 공개키 e는 확인과정의 속도향상을 위해 안전도의 문제가 없는 범위에서 작은 숫자를 공통으로 사용한다. 일반적으로 $e = 2^{16} + 1$을 주로 사용한다.

한편 서명생성은 일반적인 모듈라멱승을 사용할 수도 있지만, 사용자는 N의 소인수 P, Q를 알고 있으므로 다음 값들을 사전에 계산하여 안전하게 저장해 두고,

$$d_p = d \bmod P,\ d_q = d \bmod Q,$$
$$W_p = Q(Q^{-1} \bmod P),\ W_q = P(P^{-1} \bmod Q),$$

서명 S를 다음과 같이 CRT(Chinese Remainder Theorem)를 이용하여 계산하면 $S = M^d \bmod N$으로 계산하는 것에 비해 대략 $3 \sim 4$배 정도 속도를 향상시킬 수 있다:

$$S = W_p(M^{d_p} \bmod P) + W_q(M^{d_q} \bmod Q) \bmod N.$$

3.3 KCDSA 서명

KCDSA는 한국 디지탈서명 표준안으로 상정된 확인서를 이용한 부가형 디지탈서명 알고리듬이다 (KCDSA: Korean Certificate-based Digital Signature Algorithm). 표 6에 KCDSA의 시스템 및 사용자 변수를 정리하였다. 소수 P, Q는 안전성을 위해 $(P-1)/2$가 소수이거나 최소한 Q보다 큰 소수들의 곱으로 구성되도록 선

택할 것을 권고하고 있다 [9]. 그리고 여기에 사용될 해쉬함수 h는 $|Q|$비트 길이의 출력을 내는 해쉬함수로 현재 표준화되고 있는 해쉬함수를 사용할 것이 권고 된다. 이 해쉬함수는 Q가 128비트에서 32비트 단위로 증가하여 최대 256비트 까지를 취할 수 있는 점을 감안하여 Q와 같은 길이를 내는 가변 출력길이 해쉬함수가 될 것이다. 서명의 생성 및 검증 과정은 표 7에 정리 하였다.

변 수	조 건 / 값	설 명
P	$\|P\| = 512 + 256i\ (0 \le i \le 6)$	소수
Q	$\|Q\| = 128 + 32j\ (0 \le j \le 4)$	$P-1$의 소인수
G	$G \neq 1\ \&\ G^Q = 1 \bmod P$	기본원소
h	출력길이 $= \|Q\|$	충돌회피성 해쉬함수
X	$0 < X < Q$	서명자의 비공개 서명키
Y	$Y = G^{X^{-1}} \bmod P$	서명자의 공개 검증키
Z	$Z = h(Cert\_Data)$	서명자 인증정보의 해쉬값
C	$C = \{Cert\_Data \parallel Sign_{CA}(Z)\}$	서명자의 공개키 확인서

표 6: KCDSA의 시스템 및 사용자 변수

비공개 서명키 : X, 공개 검증키 : $Y = G^{X^{-1}} \bmod P$	
서명 생성	서명 검증
$K \in_r \mathbf{Z}_Q^*,\ H = h(Z \parallel M)$ $R = h(G^K \bmod P)$ $E = R \oplus H \bmod Q$ $S = X(K - E) \bmod Q$	$H' = h(Z \parallel M')$ $E' = R' \oplus H' \bmod Q$ $R' = h(Y^{S'} G^{E'} \bmod P)$?

표 7: KCDSA의 서명 생성 및 검증 과정

비교를 위해 미국의 디지탈서명 표준인 DSA[16]를 간략히 소개한다. DSA는 160비트의 Q를 사용하고 P는 512비트부터 64비트 단위로 증가하여 최대 1024비트까지의 소수를 취할 수 있다. 비밀키 X에 대한 공개키는 $Y = G^X \bmod P$로 주어진다. 메세지 M에 대한 서명은 우선 해쉬값 $H = h(M)$을 구한 후 (해쉬함수는 SHA-1[17] 사용) 다음 식과 같이 계산된 $\{R, S\}$이다:

$$\begin{aligned} R &= (G^K \bmod P) \bmod Q \text{ with } K \in_r \mathbf{Z}_Q^*, \\ S &= K^{-1}(RX + H) \bmod Q. \end{aligned}$$

서명의 검증은 다음 식이 만족되는 지를 검사함으로써 이루어진다:

$$R = (Y^{S^{-1}R} G^{S^{-1}H} \bmod P) \bmod Q \ ?$$

KCDSA와 비교하면 서명의 생성 및 검증 과정에서 mod Q로 역원을 계산하는 과정이 더 필요하나, 대부분의 환경에서 이는 속도에 거의 영향을 미치지 않는다.

제 4 절 구현결과 비교/분석

본 논문에서 구현한 모듈라 연산 라이브러리는 32비트 컴퓨터를 기본으로 하여 기수를 $b = 2^{32}$으로 잡았다. 구현은 전체적으로 C 언어로 하였는데 PC의 경우는 C 언어만을 사용한 것(C)과 Visual C/C++ 컴파일러가 제공하는 64비트 데이타형인 _int64를 사용한 것(D), 그리고 핵심부분은 inline assembly로 구현한 것(A)의 세 종류로 분류하여 속도를 측정하였다. Workstation의 경우는 C 언어만로 구현하였다.

구현 결과를 표 9~ 15에 제시하였다. 표 9는 가장 기본적인 연산인 모듈라 곱셈과 제곱에 대한 기종별 속도를 측정한 것이고, 표 10, 11은 고정 밑 G에 대한 멱승의 성능을, 표 12는 RSA 서명의 성능을, 그리고 표 13, 14, 15는 KCDSA 서명의 성능을 각각 나타낸다. 구현결과는 전반적으로 L2 알고리듬이 Montgomery 알고리듬보다 우수하였고, 그 차이는 최적화가 덜 된 C 언어로의 구현에서 특히 두드러지게 나타난다. 그러나 L2알고리듬은 기종별/언어별 반복 적용 회수 등에 대한 최적화가 아직 완전히 조율되지 않은 탓에 부분적으로 예상치 않은 결과를 보이기도 한다. 향후 이를 조율하는 작업을 진행할 예정이다.

모듈라곱셈에서 $|N|$이 클수록 L2 알고리듬이 효과적이지만 $|N|$이 작고 특히 어셈블리어 구현에서는 L2 알고리듬이 Montgomery 알고리듬보다 오히려 열등한 경우도 있었다. 이 경우는 L2 알고리듬에서 n자리수와 $n/2$자리수를 곱하는 부분을 생략한 L1 알고리듬이 보다 효율적이나 따로 데이타를 뽑지는 않았다. 그리고 L2 알고리듬과 Montgomery 알고리듬 모두 특별한 형태의 법(법의 최상위 두 자리와 최하위 자리를 모두 1로 채운 경우)을 사용한 경우 그 효과가 미미하였다 (표 9참조; GG: 일반적인 법, FF: 특수한 형태의 법).

LL 멱승 알고리듬의 $h \times v$ 배열에서 h가 클수록 많은 메모리를 사용하지만 속도는 빨라진다 (표 3및 표 10, 11참조). 전반적으로 LL 알고리듬이 윈도우 멱승법에 비해 3 ~ 6배 정도의 빠른 속도를 보여준다. 여기서 사용한 P와 G는 ISAKMP/Oakley에서 제안한 것들인데 LL 알고리듬의 경우는 $G = 2$와 같이 밑을 작은 수로 택한 효과는 전혀 없으며, 표에서 윈도우 알고리듬을 이용한 경우(Win에 해당하는 열)는 DH 키분배의 과정 2의 성능을 보여주기 위한 것으로 (즉 $(G^{R_i})^{R_j} \bmod P$ 형태의 계산) 역시 G의 크기에는 상관이 없다.

RSA 서명의 생성에서는 CRT를 이용한 경우가 그렇지 않은 경우에 비해 대략 3배 정도 빠른 것을 알 수 있다 (표 12참조). KCDSA의 서명 생성시 (표 13~ 15참조) LL 멱승 알고리듬을 이용하는 경우의 성능은 $|Q| = 160$의 경우 표 10에 나타난 멱승 시간과 거의 유사할 것이다.

본 논문의 구현결과를 RSA Data Security사의 상용 제품인 BSAFE 3.0에 대해 RSADS가 그 홈페이지에 올려 놓은 Benchmark값들과 비교해 보기로 한다. 표 8에 DH, RSA 서명 및 DSA/KCDSA 서명에 대해 두 구현의 속도를 비교하여 놓았다. 대상 기종 및 사용 알고리듬, 구현언어 등은 모두 동일하므로 표의 값들은 두 구현결과의 성능을 그대로 반영한다고 볼 수 있다. 표에서 볼 수 있듯이 본 논문의 구현결과가 전반적으로 RSADS사의 것 보다 우수함을 알 수 있다.

구현 사	RSA Data Security					Future Systems				
방식	DH	RSA 서명		DSA 서명		DH	RSA 서명		KCDSA 서명	
$\|N\|$		생성	검증	생성	검증		생성	검증	생성	검증
512	70	24	2.7	29	52	54	19	1.9	18	23
768	n/a	66	5.3	53	100	190	59	4.1	42	51
1024	470	140	8.6	86	170	368	116	6.1	64	77
2048	n/a	930	31	300	600	2910	769	24	245	300

표 8: Pentium 90MHz에서 RSADS사의 BSAFE 3.0과의 성능 비교 (단위: msec)

- 주:

1. Pentium 90MHz에서 부분적인 어셈블리어를 사용, C로 구현한 결과임. Montgomery / 윈도우 멱승 알고리듬 사용.
2. DH는 법 크기의 지수에 대한 일반적인 멱승(곧 CRT를 사용하지 않은 RSA 서명 생성)으로 DH 키분배의 과정 1 혹은 2에 소요되는 시간이며 과정 1의 경우 사전계산방식을 이용하면 3 ~ 6배 정도의 속도 향상을 기대할 수 있음.
3. DSA/KCDSA에서 Q의 크기는 160비트이며, RSA 서명 생성은 CRT를 이용한 것이고 RSA 서명 검증시의 공개 검증 승수는 $e = 2^{16} + 1$을 사용한 것임.
4. RSA사에서 구현한 DSA는 메세지의 해쉬과정이 포함되지 않은 것이며, 본 논문의 KCDSA 구현에서는 단문의 메세지를 SHA-1으로 해쉬하는 과정이 포함된 것이나 그 차이는 거의 없을 것임. 또한 DSA의 서명 생성 및 검증에는 mod Q로 역원을 계산하는 것이 필요한 반면, KCDSA에서는 이것이 필요없으나 역원의 계산 역시 멱승에 비해 무시할 수 있는 정도이므로 두 구현은 거의 같은 방식을 구현한 것으로 볼 수 있음.
5. RSADS사의 DSA 서명 검증 시간이 본 논문의 KCDSA 검증 시간에 비해 거의 두 배 정도의 차이가 나는 것은 모듈라 연산의 성능 차이에다 RSADS사가 서명 검증에 다항 멱승 연산법을 사용하지 않고 각각의 멱승을 곱해서 계산한 때문임.

제 5 절 결론

본 논문에서는 모듈라 멱승을 이용하는 공개키 암호시스템인 Diffie-Hellman, RSA 및 KCDSA의 고속 구현에 필요한 알고리듬 및 이들의 구현 결과를 소개하였다. 본 논문에서 소개한 모듈라 연산의 구현은 PC의 경우 핵심 부분을 어셈블리어로 구현하였는데, 이들 암호시스템의 구현을 통한 실험 결과 RSADS사의 상용 제품인 BSAFE 3.0보다 그 속도가 우수함을 알 수 있었다.

참고 서적

[1] E.F.Brickell, D.M.Gordon, K.S.McCurley and D.Wilson, Fast exponentiation with precomputation, In *Advances in Cryptology-EUROCRYPT'92*, LNCS 658, Springer-Verlag, 1993, pp.200-207.

[2] A.Bosselaers, R.Govaerts and J.Vandewalle, Comparison of three modular reduction functions, In *Advances in Cryptology-CRYPTO'93*, LNCS 773, Springer-Verlag, 1994, pp.175-186.

[3] W.Diffie and M.E.Hellman, New directions in cryptography, *IEEE Trans. Inform. Theory*, 22(6), 1976, pp.644-654.

[4] T.ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Trans. Inform. Theory*, IT-31, 1985, pp.469-472.

[5] D. Harkins and D. Carrel, The Resolution of ISAKMP with Oakley, Internet-Draft: draft-ietf-ipsec-isakmp-oakley-04.txt, July 1997.

[6] D.E.Knuth, *The Art of Computer Programming*, Vol. 2, Addison-Wesley, 1981.

[7] C.H.Lim, *Computational methods for speeding up public key cryptosystems*, Ph.D Thesis, Dept. Electronic and Electrical Engineering, POSTECH, Mar. 1996.

[8] C.H.Lim and P.J.Lee, More flexible exponentiation with precomputation, In *Advances in Cryptology - Crypto'94*, LNCS 839, Springer-Verlag, pp.95-107.

[9] C.H.Lim and P.J.Lee, A key recovery attack on discrete log based schemes using a prime order subgroup, In *Advances in Cryptology - Crypto'97*, LNCS 1294, Springer-Verlag, pp.249-263.

[10] 임 채훈, 이 필중, 강신각, 박 성준, 확인서 이용 부가형 디지탈 서명 방식 표준(안), CISC'97 자료집.

[11] C.H.Lim, H.S.Hwang and P.J.Lee, Fast modular reduction with precomputation, *Proc. of JW-ISC'97*, Nov. 1997, pp.65-79.

[12] D. Maughan, M. Schertler, M. Schneider and Jeff Turner, Internet Security Association and Key Management Protocol (ISAKMP), IPSEC Working Group INTERNET-DRAFT: draft-ietf-ipsec-isakmp-08.txt, July 1997.

[13] P.L.MontGomery, Modular multiplication without trial division, *Math. Comp.*, 44, 1985, pp.519-521.

[14] R.L.Rivest, A.Shamir and L.Adleman, "A method for obtaining digital signatures and public key cryptosystems," *Commun. ACM*, 21(2), 1978, pp.120-126.

[15] H. K. Orman, The OAKLEY Key Determination Protocol, IPSEC Working Group INTERNET-DRAFT: draft-ietf-ipsec-oakley-02.txt, July 1997.

[16] NIST, Digital signature standard, *FIPS PUB 186*, 1994.

[17] NIST, Secure hash standard, *FIPS PUB 180-1*, Department of Commerce, Washington D.C., Apr. 1995.

$\|N\|$	Type	Alg.	M/S	S20-C	US-C	P-C	P-D	P-A	PP-C	PP-D	PP-A
512	GG	Mo	Sqr	1.11	0.374	0.401	0.222	0.0886	0.128	0.0722	0.0244
			Mul	1.22	0.419	0.465	0.244	0.101	0.146	0.0792	0.0268
		L2	Sqr	0.999	0.359	0.413	0.214	0.0953	0.126	0.0684	0.0258
			Mul	1.14	0.411	0.475	0.244	0.102	0.146	0.0758	0.0279
	FF	Mo	Sqr	1.09	0.369	0.402	0.222	0.0876	0.124	0.0720	0.0247
			Mul	1.21	0.421	0.470	0.244	0.0996	0.147	0.0784	0.0271
		L2	Sqr	0.951	0.343	0.402	0.190	0.0886	0.121	0.0622	0.0253
			Mul	1.09	0.393	0.454	0.220	0.102	0.139	0.0684	0.0270
768	GG	Mo	Sqr	2.33	0.808	0.851	0.452	0.202	0.282	0.149	0.0515
			Mul	2.64	0.905	0.994	0.508	0.242	0.309	0.166	0.0615
		L2	Sqr	2.05	0.720	0.829	0.415	0.207	0.272	0.138	0.0543
			Mul	2.30	0.825	0.962	0.464	0.247	0.299	0.155	0.0627
	FF	Mo	Sqr	2.34	0.794	0.853	0.447	0.202	0.279	0.147	0.0529
			Mul	2.63	0.905	0.982	0.508	0.239	0.323	0.168	0.0615
		L2	Sqr	1.97	0.700	0.797	0.388	0.194	0.265	0.128	0.0522
			Mul	2.25	0.805	0.928	0.442	0.234	0.303	0.145	0.0608
1024	GG	Mo	Sqr	4.01	1.37	1.44	0.776	0.303	0.453	0.251	0.0819
			Mul	4.44	1.52	1.65	0.882	0.340	0.514	0.283	0.0888
		L2	Sqr	3.27	1.16	1.35	0.692	0.318	0.447	0.227	0.0830
			Mul	3.62	1.31	1.57	0.795	0.356	0.489	0.258	0.0888
	FF	Mo	Sqr	4.04	1.38	1.46	0.778	0.299	0.465	0.252	0.0819
			Mul	4.46	1.52	1.65	0.871	0.337	0.531	0.283	0.0907
		L2	Sqr	3.19	1.14	1.31	0.683	0.322	0.429	0.221	0.0830
			Mul	3.64	1.29	1.53	0.757	0.352	0.484	0.245	0.0888
1536	GG	Mo	Sqr	8.75	2.96	3.15	1.66	0.721	1.01	0.543	0.197
			Mul	9.61	3.26	3.53	1.85	0.839	1.14	0.603	0.222
		L2	Sqr	6.41	2.30	2.73	1.40	0.711	0.907	0.478	0.197
			Mul	7.28	2.62	3.11	1.61	0.832	1.01	0.543	0.222
	FF	Mo	Sqr	8.73	2.95	3.11	1.66	0.720	1.01	0.543	0.197
			Mul	9.54	3.29	3.53	1.85	0.831	1.14	0.596	0.220
		L2	Sqr	6.51	2.29	2.73	1.38	0.693	0.896	0.470	0.194
			Mul	7.24	2.59	3.11	1.55	0.814	1.01	0.517	0.216
2048	GG	Mo	Sqr	15.0	5.14	5.39	2.90	1.21	1.70	0.942	0.328
			Mul	16.2	5.55	5.99	3.15	1.33	1.87	1.02	0.351
		L2	Sqr	10.2	3.67	4.32	2.25	1.10	1.43	0.806	0.304
			Mul	11.3	4.13	4.99	2.49	1.22	1.62	0.873	0.328
	FF	Mo	Sqr	15.1	5.16	5.38	2.91	1.21	1.77	0.944	0.328
			Mul	16.1	5.57	5.98	3.18	1.31	1.89	1.02	0.347
		L2	Sqr	10.0	3.62	4.32	2.18	1.09	1.43	0.781	0.304
			Mul	11.2	4.07	4.99	2.46	1.21	1.62	0.875	0.320

표 9: 모듈라 제곱과 모듈라 곱셈의 수행 속도 (단위: msec)

Algorithm		LL/Montgomery						LL/L2					
기종	$\|P\|$	8×4	7×4	6×4	5×4	4×4	Win	8×4	7×4	6×4	5×4	4×4	Win
S20-C	512	32.4	35.0	41.4	49.8	59.0	223	26.4	29.6	35.4	42.0	50.9	195
	768	67.4	78.1	89.8	107	127	462	53.7	63.3	73.7	87.3	105	388
	1024	115	133	155	173	203	808	86.0	99.1	118	133	152	628
	1536	247	278	339	388	428	1775	175	199	243	275	313	1265
	2048	425	495	583	650	821	3030	274	323	381	434	543	2035
US-C	512	11.7	12.4	14.4	17.4	20.7	77.2	9.49	10.6	12.5	15.2	18.3	70.1
	768	23.4	27.1	31.4	36.9	43.9	157	19.2	22.4	26.3	30.9	37.0	137
	1024	39.7	45.8	53.2	59.5	68.3	276	31.3	36.4	42.8	47.8	55.0	230
	1536	85.7	95.5	115	131	146	595	62.7	70.5	85.9	98.2	111	457
	2048	145	169	197	225	280	1036	99.0	116	136	156	195	731
P-C	512	14.7	14.2	15.7	19.0	23.2	80.6	12.9	12.4	14.4	17.5	21.3	76.1
	768	30.4	29.8	35.0	39.3	49.2	174	26.4	25.5	30.3	35.0	43.5	161
	1024	51.2	48.1	57.2	67.2	80.0	293	42.7	40.6	49.6	58.8	71.0	259
	1536	109	107	121	140	179	629	88.4	88.3	101	118	153	543
	2048	192	181	210	233	286	1098	140	140	163	181	224	880
P-D	512	7.96	7.63	8.57	10.2	12.3	44.1	6.64	6.11	7.03	8.44	10.3	38.0
	768	16.1	15.3	17.9	20.5	25.6	92.0	13.6	12.6	14.9	17.0	21.4	79.2
	1024	27.5	25.6	30.9	35.8	42.7	156	22.1	20.6	25.2	29.7	35.5	133
	1536	56.6	56.5	63.2	74.0	93.9	336	43.2	43.5	49.6	58.1	74.7	275
	2048	101	96.1	112	124	152	584	73.4	68.6	81.1	91.1	112	439
P-A	512	3.15	3.06	3.45	4.15	5.04	17.4	2.92	2.91	3.31	4.00	4.95	17.8
	768	7.49	7.22	8.53	9.59	11.9	41.7	6.59	6.51	7.69	8.88	11.3	40.3
	1024	10.8	10.0	12.0	13.9	16.6	59.6	9.85	9.69	11.9	13.9	16.6	63.0
	1536	25.8	25.1	28.3	33.4	42.4	144	22.9	22.4	25.5	30.2	39.2	140
	2048	40.1	40.6	47.5	51.8	64.3	245	32.2	34.3	40.0	45.0	54.9	220
PP-C	512	3.90	4.52	5.04	6.13	7.45	26.3	3.41	3.95	4.53	5.48	6.80	24.8
	768	8.27	9.52	10.9	12.3	15.6	56.8	7.34	8.58	9.97	11.4	14.5	53.8
	1024	13.3	15.4	18.6	21.7	25.8	93.5	11.4	13.0	16.1	18.8	22.9	85.8
	1536	28.3	33.6	38.4	45.0	56.8	203	23.4	28.4	32.2	37.8	48.8	178
	2048	49.6	56.6	66.3	73.5	90.4	352	38.9	45.1	52.5	59.0	72.6	288
PP-D	512	2.14	2.49	2.78	3.36	4.09	14.6	1.73	1.97	2.27	2.78	3.39	12.4
	768	4.49	5.12	5.96	6.79	8.45	30.8	3.61	4.22	4.82	5.58	7.10	25.9
	1024	7.60	8.46	10.0	12.0	14.1	51.0	5.97	6.74	8.21	9.69	11.7	43.6
	1536	16.0	18.3	20.6	24.4	30.8	109	12.7	14.8	16.5	19.6	25.1	91.7
	2048	27.5	31.3	36.4	39.9	49.3	189	20.9	24.1	28.8	32.2	39.5	156
PP-A	512	0.74	0.863	0.964	1.15	1.40	5.10	0.67	0.794	0.892	1.07	1.33	5.03
	768	1.64	1.90	2.20	2.48	3.13	11.0	1.46	1.69	1.99	2.27	2.88	10.6
	1024	2.38	2.68	3.21	3.75	4.41	16.6	2.17	2.45	2.99	3.52	4.24	16.2
	1536	5.90	6.69	7.58	8.88	11.2	39.8	5.25	6.06	6.81	8.23	10.5	38.5
	2048	9.33	10.7	12.4	13.7	16.9	65.8	7.92	9.17	10.8	12.0	14.9	60.4

표 10: 160비트의 랜덤 지수 R에 대한 $G^R \bmod P$의 수행 속도 (단위: msec)

Algorithm		LL/Montgomery						LL/L2					
기종	$\|P\|$	8×4	7×4	6×4	5×4	4×4	Win	8×4	7×4	6×4	5×4	4×4	Win
S20-C	512	95.7	112	132	155	186	674	85.0	98.2	115	135	165	591
	768	308	358	406	483	593	2130	261	300	345	414	504	1830
	1024	699	808	948	1093	1335	4990	555	645	743	868	1073	3920
	1536	2240	2585	2975	3520	4318	15700	1665	1930	2243	2645	3228	11680
	2048	5100	5865	6828	8023	9828	36000	3485	4005	4668	5475	6715	24240
US-C	512	34.0	39.0	45.5	53.6	65.2	233	30.5	35.4	41.5	48.6	59.4	214
	768	107	124	140	168	206	737	93.7	108	123	146	180	647
	1024	241	278	322	373	459	1670	200	233	269	313	383	1383
	1536	770	878	1015	1213	1483	5390	601	690	795	945	1155	4160
	2048	1757	2003	2338	2743	3353	12280	1270	1453	1688	1978	2428	8720
P-C	512	43.3	41.9	48.8	63.3	68.6	250	42.1	39.1	46.6	56.4	64.5	239
	768	137	135	155	185	227	797	125	126	144	172	210	732
	1024	305	298	348	403	499	1793	275	270	311	371	453	1613
	1536	988	963	1098	1303	1565	5660	833	838	963	1140	1373	5000
	2048	2225	2155	2515	2953	3625	12850	1813	1758	2048	2405	2953	10430
P-D	512	23.6	22.3	26.4	32.0	36.4	136	20.2	19.3	23.0	27.4	31.6	119
	768	71.3	70.8	80.2	96.0	118	426	63.2	60.8	69.8	82.3	102	371
	1024	163	157	184	215	264	954	141	135	157	184	225	824
	1536	531	501	570	679	816	3050	427	411	474	563	680	2500
	2048	1180	1153	1333	1565	1910	6970	972	878	1018	1195	1470	5220
P-A	512	9.63	9.20	10.7	12.8	14.9	54.3	9.69	9.20	10.9	13.0	15.1	56.2
	768	33.7	33.3	38.1	44.9	55.5	190	33.0	32.2	35.9	43.8	53.8	186
	1024	64.1	61.2	71.7	83.5	102	368	65.1	64.4	74.0	86.5	107	388
	1536	231	227	261	306	368	1305	222	216	251	291	357	1263
	2048	508	474	556	653	798	2910	444	433	509	590	728	2610
PP-C	512	11.9	13.3	15.7	19.1	22.0	82.0	11.1	12.6	14.6	18.0	20.7	77.4
	768	37.4	43.3	49.1	58.6	72.2	260	35.6	41.3	47.0	55.7	69.3	246
	1024	84.1	99.2	113	130	160	574	77.4	87.2	101	119	146	531
	1536	265	307	348	417	499	1813	236	270	312	366	444	1610
	2048	599	688	798	935	1140	4120	503	563	673	783	963	3400
PP-D	512	6.61	7.38	8.66	10.5	12.1	45.3	5.62	6.23	7.36	8.89	10.3	39.2
	768	20.7	23.1	26.6	31.8	39.2	140	17.4	20.0	22.7	27.5	34.1	119
	1024	45.4	52.3	60.0	70.3	85.8	313	39.0	44.7	52.3	60.9	73.9	271
	1536	143	165	190	223	269	988	122	140	162	192	231	836
	2048	326	371	433	509	625	2225	272	309	365	425	523	1840
PP-A	512	2.27	2.53	2.96	3.57	4.14	15.9	2.19	2.44	2.92	3.49	4.06	15.9
	768	7.50	8.52	9.90	11.7	14.4	50.5	7.08	8.10	9.32	11.1	13.7	48.9
	1024	14.4	16.6	19.2	22.3	27.4	101	14.2	16.3	18.9	22.3	27.2	102
	1536	52.8	59.8	69.6	82.5	99.1	357	51.0	57.9	67.7	79.5	96.1	349
	2048	112	128	149	174	213	787	102	117	137	160	197	723

표 11: $|P|$비트의 지수 랜덤 R에 대한 $G^R \bmod P$의 수행 속도 (단위: msec)

Alg.	$\|N\|$		S20-C	US-C	P-C	P-D	P-A	PP-C	PP-D	PP-A
Mo	512	Sign1	684	236	253	139	56.3	82.9	45.7	16.0
		Sign2	198	70.0	74.9	44.2	18.9	24.7	15.1	6.05
		Verify	20.8	7.13	7.64	5.25	1.89	2.56	1.64	0.578
	768	Sign1	2160	740	787	421	189	254	139	49.5
		Sign2	610	211	233	126	59.0	73.7	43.1	16.8
		Verify	43.5	15.0	15.8	9.43	4.15	5.31	3.27	1.22
	1024	Sign1	4910	1690	1777	954	372	580	313	100
		Sign2	1383	484	508	278	116	168	91.7	32.1
		Verify	73.9	25.4	26.6	15.8	6.07	8.87	5.42	1.63
	1536	Sign1	15820	5390	5650	3050	1320	1833	988	357
		Sign2	4365	1490	1623	858	388	508	289	103
		Verify	159	54.8	56.6	33.5	14.6	18.9	11.3	4.09
	2048	Sign1	36600	12300	12970	6920	2885	4120	2225	786
		Sign2	9968	3423	3610	1950	769	1155	631	206
		Verify	269	93.4	96.6	56.6	24.3	31.7	19.3	6.81
L2	512	Sign1	629	224	256	133	57.5	80.0	43.2	16.7
		Sign2	209	75.5	83.1	47.7	21.2	26.8	15.5	6.68
		Verify	21.0	7.16	7.64	5.24	1.89	2.56	1.65	0.583
	768	Sign1	1875	664	760	385	193	252	127	50.5
		Sign2	588	210	243	122	62.1	76.6	41.9	17.6
		Verify	44.1	15.0	15.8	9.36	4.10	5.31	3.24	1.23
	1024	Sign1	3980	1420	1647	858	388	537	281	101
		Sign2	1333	484	544	272	125	177	85.9	33.5
		Verify	74.2	25.4	26.9	16.0	6.08	8.85	5.47	1.63
	1536	Sign1	11770	4190	5050	2610	1305	1647	880	357
		Sign2	3783	1345	1620	790	405	503	272	104
		Verify	157	54.5	56.7	33.4	14.4	18.9	11.3	4.08
	2048	Sign1	24440	8790	10490	5270	2690	3460	1895	737
		Sign2	8208	3035	3515	1798	796	1155	576	218
		Verify	270	93.9	96.6	56.6	24.3	31.7	19.1	6.62

표 12: RSA의 서명 생성 및 검증 속도 (단위: msec)

- 주:

1. 검증 속도는 공개 검증 승수가 $e = 2^{16} + 1$일 때의 속도이며, 공개법 N은 랜덤한 두 소수의 곱으로 구성된 것임 ($N = PQ$).
2. Sign1: exponentiation without CRT, Sign2: exponentiation with CRT
3. 위의 수치들은 랜덤한 M에 대해 $M^d \bmod N$ (서명 생성), $M^e \bmod N$(서명 검증)의 계산에 소요되는 시간임 (CRT를 사용한 서명 생성은 본문 참조).

기종	|Q|		Montgomery					L2				
			512	768	1024	1536	2048	512	768	1024	1536	2048
S20-C	128	Sign	179	391	665	1415	2420	165	333	530	1065	1600
		Verify	219	486	820	1735	3070	206	414	650	1275	2090
	160	Sign	224	471	790	1720	3040	199	407	638	1310	2030
		Verify	278	599	990	2170	3830	246	504	785	1595	2540
	192	Sign	265	563	965	2100	3660	241	490	765	1565	2370
		Verify	331	714	1203	2625	4430	296	607	945	1895	2990
	224	Sign	304	660	1140	2435	4240	282	557	898	1815	2840
		Verify	378	837	1395	2930	5210	333	691	1133	2195	3480
	256	Sign	346	759	1273	2710	4780	316	649	1013	2060	3150
		Verify	426	926	1600	3405	6010	395	814	1260	2495	3880
US-C	128	Sign	62.1	132	225	482	830	58.3	119	188	368	583
		Verify	76.7	168	282	594	1033	74.8	145	230	464	737
	160	Sign	76.4	165	279	600	1037	72.6	144	231	460	740
		Verify	95.2	202	351	720	1267	91.2	183	293	574	897
	192	Sign	90.5	195	334	722	1237	86.4	173	282	558	883
		Verify	113	244	407	892	1513	107	215	337	688	1093
	224	Sign	106	225	388	830	1437	100	201	325	646	1020
		Verify	132	276	475	1008	1783	122	246	403	792	1267
	256	Sign	120	257	438	948	1627	114	229	367	734	1167
		Verify	149	314	544	1162	1960	140	288	455	906	1437

표 13: WorkStation에서 KCDSA의 서명 생성 및 검증 속도(단위: msec)

- 주:

1. 이 구현에 사용된 공개변수 P, Q는 KCDSA 표준안의 소수생성 방법에 따라 $P = 2QR+1$ (R, Q도 소수) 형태로 생성된 것임. 768, 1024비트의 경우에 대한 수치예는 참고문헌 [10]의 부록에 나타나 있음.
2. 특수한 형태의 소수 P에 대해서도 속도를 측정해 보았으나 일반적인 소수법에 비해 속도 향상 효과는 미미하여 여기에 싣지는 않았음. 표 10의 데이타가 이러한 소수법에 대한 결과이니 그 차이는 이 표를 참조하기 바람.
3. 서명 생성 및 검증 과정에서 메세지는 단문으로 해쉬함수(SHA-1)의 한 블럭이내에 처리될 수 있도록 했으므로 표의 데이타는 실제로 모듈라 연산에 소요되는 시간으로 볼 수 있음.

기종	|Q|		Montgomery					L2				
			512	768	1024	1536	2048	512	768	1024	1536	2048
P-C	128	Sign	67.9	141	235	504	880	66.6	137	215	450	697
		Verify	83.7	177	305	616	1080	83.9	171	275	562	877
	160	Sign	82.4	174	295	638	1097	83.7	168	275	538	880
		Verify	101	220	370	792	1357	99.7	207	335	692	1097
	192	Sign	96.8	211	355	758	1280	99.7	198	325	658	1043
		Verify	121	253	435	922	1630	123	247	405	814	1300
	224	Sign	114	241	414	880	1517	114	232	375	768	1227
		Verify	140	305	505	1098	1887	143	290	469	968	1520
	256	Sign	128	274	465	988	1703	130	266	430	868	1390
		Verify	161	342	574	1230	2123	165	327	534	1086	1760
P-D	128	Sign	37.2	75.4	126	274	467	33.9	67.4	113	231	357
		Verify	46.3	92.6	160	335	585	43.4	84.9	141	285	448
	160	Sign	44.9	92.6	160	341	587	42.7	84.6	140	280	440
		Verify	56.5	111	198	423	732	52.5	107	179	351	568
	192	Sign	53.4	111	190	407	697	51.8	102	168	341	530
		Verify	68.0	138	234	494	868	63.5	122	206	423	660
	224	Sign	62.7	129	220	473	805	58.9	118	198	396	613
		Verify	76.6	160	272	587	1017	72.7	145	242	489	768
	256	Sign	70.4	147	250	538	925	67.2	135	225	451	695
		Verify	87.5	181	311	659	1117	82.8	165	275	544	852
P-A	128	Sign	14.9	34.2	49.4	118	195	15.2	34.9	51.6	115	177
		Verify	18.0	44.3	62.6	149	249	19.3	43.5	64.8	144	224
	160	Sign	18.3	42.1	60.4	146	245	18.9	42.9	63.6	144	224
		Verify	22.7	51.3	77.0	178	300	23.6	52.7	82.4	178	278
	192	Sign	21.7	50.6	73.6	175	296	22.7	50.6	75.8	175	266
		Verify	27.6	62.9	92.2	217	355	28.9	64.2	96.8	217	330
	224	Sign	25.4	58.4	85.6	204	342	25.8	58.4	89.0	204	308
		Verify	31.6	72.7	107	251	414	31.6	73.5	113	249	385
	256	Sign	28.6	66.4	96.6	230	385	29.8	67.8	101	230	351
		Verify	35.6	82.7	123	290	478	37.9	84.2	125	285	443

표 14: Pentium 90MHz에서 KCDSA의 서명 생성 및 검증 속도(단위: msec)

기종	$\|Q\|$		Montgomery					L2				
			512	768	1024	1536	2048	512	768	1024	1536	2048
PP-C	128	Sign	21.3	45.2	74.8	165	276	21.4	44.1	71.5	147	231
		Verify	27.0	56.8	93.0	205	348	26.5	55.9	88.5	183	293
	160	Sign	26.5	55.9	94.8	201	348	26.5	54.8	89.7	179	288
		Verify	33.6	70.5	115	245	433	33.2	66.8	110	223	360
	192	Sign	31.2	66.6	115	245	409	31.7	64.8	106	216	342
		Verify	38.9	81.4	142	301	507	38.9	81.3	132	271	427
	224	Sign	36.9	75.5	132	278	488	37.0	75.5	123	257	397
		Verify	45.9	95.2	163	341	592	46.4	94.1	150	315	494
	256	Sign	42.2	88.4	150	319	543	42.2	86.3	140	285	458
		Verify	51.6	108	181	399	678	52.1	109	172	348	561
PP-D	128	Sign	11.9	24.6	41.5	89.0	152	11.3	22.5	36.9	77.6	129
		Verify	14.5	30.9	52.1	112	184	13.8	28.3	45.1	94.8	155
	160	Sign	14.8	30.4	51.3	110	187	13.7	27.7	45.9	94.8	158
		Verify	18.2	37.1	62.1	134	233	16.9	34.6	57.7	116	197
	192	Sign	17.7	36.7	61.3	133	223	16.6	33.5	54.9	114	191
		Verify	21.7	45.0	78.4	165	281	20.9	40.8	67.5	140	239
	224	Sign	20.9	42.4	71.1	152	262	19.3	38.2	63.9	132	216
		Verify	24.8	52.8	88.2	189	316	23.5	47.6	79.3	163	275
	256	Sign	23.2	48.1	81.0	172	297	21.9	44.0	73.0	151	246
		Verify	28.6	59.7	100	216	365	26.9	54.4	90.0	186	310
PP-A	128	Sign	4.23	8.69	13.5	32.0	52.7	4.33	9.24	13.1	31.9	49.2
		Verify	5.21	11.1	16.9	41.0	66.3	5.39	11.3	16.4	40.4	60.6
	160	Sign	5.13	11.0	16.6	39.6	65.2	5.39	11.1	16.6	39.0	60.6
		Verify	6.31	13.4	20.2	50.8	81.3	6.57	14.2	20.7	49.2	74.4
	192	Sign	6.13	13.0	19.8	48.0	79.0	6.57	13.4	19.8	47.2	73.1
		Verify	7.48	16.4	24.6	59.7	96.3	8.02	17.0	24.5	58.5	91.7
	224	Sign	7.20	15.1	23.1	55.7	90.4	7.38	15.3	23.9	55.7	84.6
		Verify	9.10	19.0	28.7	69.5	112	9.10	19.1	28.0	67.5	103
	256	Sign	8.20	17.2	26.0	62.5	104	8.56	17.9	26.0	62.5	96.0
		Verify	10.1	21.1	31.8	79.2	125	10.4	22.1	33.0	76.5	119

표 15: Pentium Pro 200MHz에서 KCDSA의 서명 생성 및 검증 속도(단위: msec)